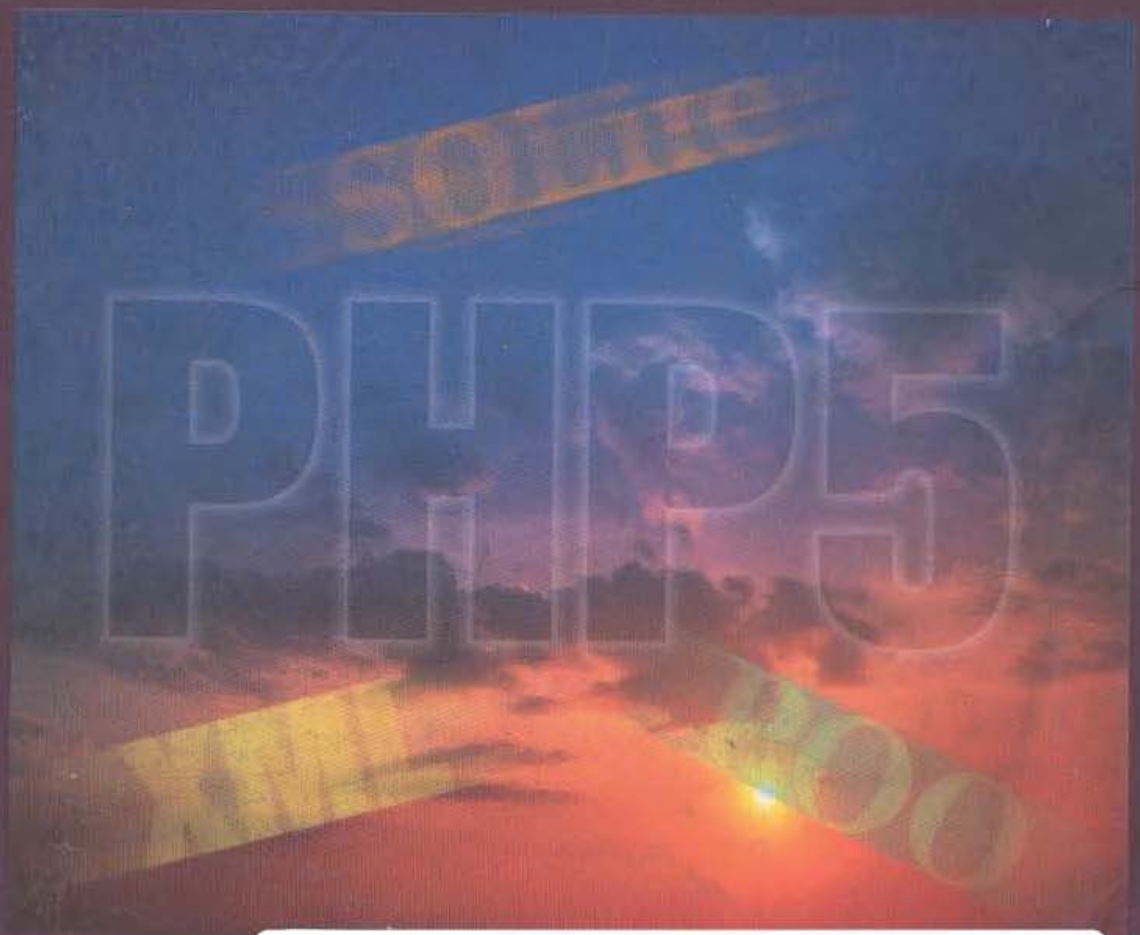
PHP5

a través de ejemplos



Abraham Gutiérrez Rodríguez
Ginés Bravo Garcia

Incluye CD-ROM

Alfaomega Ra-Ma®

ÍNDICE

APÉNDICES

INTRODUCCIÓN

PHP (acrónimo de *PHP: Hypertext Pre-Processor*) es un lenguaje de programación, relativamente nuevo (su antecesor, PHP/FI, data de finales de 1994), concebido principalmente como herramienta para el desarrollo de aplicaciones Web. PHP nos permite diseñar páginas dinámicas de servidor, es decir, generar páginas bajo petición capaces de responder de manera inteligente a las demandas del cliente y que nos permitan la automatización de gran cantidad de tareas. Si tuviéramos que definir PHP en una sola línea, podríamos decir que es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor.

Aunque existe una multitud de lenguajes y entornos de desarrollo concebidos para Internet, PHP se ha convertido en uno de los lenguajes, del lado servidor, más ampliamente utilizados para el desarrollo de páginas dinámicas junto con ASP, JSP, ColdFusion y Perl. En los últimos años, el número de servidores que utilizan PHP se ha disparado. De hecho, según datos de Netcraft (http://www.netcratf.com/) a fecha de Agosto de 2004 son casi 17 millones de dominios los que usan PHP.

En PHP se combinan muchas características que contribuyen notablemente a su masiva utilización; entre otras, está el hecho de ser un *software* de libre distribución y multiplataforma (existen versiones de PHP para U*ix, Win32, Mac OS X, etc.) que sigue la filosofía *Open Source*. También ha contribuido a su éxito el hecho de haberse convertido en el complemento ideal para el popular tándem Linux-Apache en el desarrollo de sitios Web. Pero lo más destacable del lenguaje y una de las características que más han influido en su popularización es la sencillez de uso que presenta a los programadores principiantes (se puede desarrollar aplicaciones sencillas en un corto intervalo de tiempo) combinada con las posibilidades

avanzadas que proporciona al programador profesional (comunicación con bases de datos, comunicación vía *sockets*, generación de gráficos, etc.).

UN POCO DE HISTORIA

Primeros comienzos de PHP

PHP comenzó siendo un conjunto de *scripts* escritos en Perl que permitían a su creador, Rasmus Lerdorf, el control de los accesos a sus páginas personales. A este conjunto de *scripts* les denominó como *Personal Home Page Tools*. Poco a poco, Ramus fue completando las funcionalidades básicas de su herramienta escribiendo programas en C. En 1995 decidió liberar el código fuente escrito en C para que cualquiera pudiera utilizarlo e, incluso, colaborar en su mejora. De este modo nació PHP/FI. A finales de 1997 se libera PHP/FI 2.0, pasando de ser el proyecto de una sola persona al desarrollo de un equipo. Tuvo un seguimiento estimado de varios miles de usuarios en todo el mundo, con aproximadamente 50.000 dominios informando que lo tenían instalado, lo que sumaba alrededor del 1% de los dominios de Internet.

En junio de 1998 se liberó oficialmente PHP 3.0, anunciado como sucesor oficial de PHP/FI 2.0, si bien había sido completamente reescrito por Andi Gutmans y Zeev Zuraski. Una de las mejores características de PHP 3.0 que atrajo a docenas de desarrolladores a unirse y enviar nuevos módulos de extensión era su gran extensibilidad, además de proveer a los usuarios finales de una sólida infraestructura para muchísimas bases de datos, protocolos y APIs. En su apogeo, PHP 3.0 estaba instalado en aproximadamente un 10% de los servidores Web en Internet.

Evolución de PHP

El siguiente paso en la evolución de PHP consistió en la reescritura de su núcleo, dando lugar a un nuevo motor denominado **Zend** (acrónimo de los apellidos Zeev y Andi). PHP 4.0, basado en este motor, y acoplado con un gran rango de nuevas características adicionales, fue oficialmente liberado en mayo de 2000. Además de la mejora de ejecución de esta versión, PHP 4.0 incluía otras características clave, como el soporte para la mayoría de los servidores Web, sesiones HTTP de forma nativa, ciertas facilidades de orientación a objetos, compatibilidad con las expresiones regulares de Perl, *buffers* de salida, encriptación, formas más seguras de controlar las entradas de usuario y muchas nuevas construcciones de lenguaje, etc.

La última y actual versión de PHP, liberada en Julio de 2004, es la 5.0. Está basada en el nuevo motor **Zend 2**, el cual ha vuelto a ser reescrito por completo. Entre sus características y novedades más resaltables está el completo soporte para la programación orientada a objetos (que a buen seguro satisfará a los más apasionados y ortodoxos seguidores de este paradigma de la programación). También incorpora la gestión de excepciones, una nueva librería de XML (`libxml2`), soporte nativo para el sistema gestor de base de datos SQLite, y mejoras en la gestión de las cadenas de caracteres. PHP 5.0 soporta también MySQLi, una nueva ampliación de MySQL (está diseñada para trabajar con la versión 4.1.2 o superior), la cual, además de la interfaz habitual, encierra una interfaz basada en objetos.

¿QUÉ SE PUEDE HACER CON PHP?

Aunque principalmente se utiliza para programar *scripts* que van a ser ejecutados en servidores Web, no hay que olvidar que puede utilizarse como cualquier otro lenguaje (Perl, C, Python, Shell, etc.) para escribir programas que se ejecuten desde la línea de comandos, es decir, sin la necesidad de que se ejecute conjuntamente con un servidor Web. De todas formas, es en el entorno Web donde ha conseguido su mayor aceptación, y es que PHP no sólo nos permite realizar todas las acciones propias de un *script* CGI tradicional (procesamiento de formularios, manipulación de *cookies*, generación de páginas con contenidos dinámicos...), sino que también nos proporciona las siguientes posibilidades:

- Soporte para múltiples sistemas operativos: Unix (entre otras, Linux, HP-UX, Solaris y OpenBSD), Microsoft Windows, Mac OS X, RISC OS. Actualmente está en preparación para las plataformas IBM OS/390 y AS/400.

- Soporte para múltiples servidores Web: Apache, Microsoft Internet Information Server, Personal Web Server, Netscape e iPlanet, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd y muchos otros.

- Soporte para más de 25 gestores de bases de datos: Adabas D, Ingres, Oracle, dBase, InterBase, Ovrimos, Empress, FrontBase, PostgreSQL, mSQL, Solid, Hyperwave, Direct MS-SQL, Sybase, IBM DB2, Informix, Unix dbm y MySQL, entre otras.

- Soporte para ODBC y extensiones DBX.

- Soporte para comunicarse con otros servicios usando protocolos tales como LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (en Windows) y muchos otros.

- Puede utilizar objetos Java de forma transparente, como objetos PHP.
- La extensión de CORBA puede ser utilizada para acceder a objetos remotos.
- PHP soporta WDDX para intercambio de datos entre lenguajes de programación en Web.
- Generación de resultados en múltiples formatos como XHTML, XML, ficheros de imágenes, ficheros PDF y películas Flash...
- *Parser* de documentos XML, soporte de los estándares SAX y DOM. Manejo de XSLT para transformar documentos XML.
- Manejo de expresiones regulares POSIX Extended o Perl.
- Funciones de comercio electrónico, como Cybercash, CyberMUT, VeriSign Payflow Pro y CCVS para las pasarelas de pago.
- Otras extensiones muy interesantes son las funciones del motor de búsquedas mnoGoSearch, funciones para pasarelas de IRC, utilidades de compresión (gzip, bz2), conversión de calendarios, traducción...

Y un sinfín de posibilidades que van en aumento cada día.

COMPARATIVA ENTRE ASP Y PHP

El otro lenguaje utilizado para el diseño de páginas dinámicas de servidor y ampliamente extendido es ASP (*Active Server Pages*). Es un lenguaje derivado de Visual Basic (aunque se puede programar con otros lenguajes como VBScript y JScript) desarrollado por Microsoft. Evidentemente, se emplea principalmente sobre plataformas que funcionan bajo sistemas operativos Windows (aunque desde hace poco tiempo existe un *software* de SUN, *Sun ONE Active Server Pages*, anteriormente conocido como Chili Soft ASP, que permite trabajar con ASP en plataformas Unix/Linux).

PHP es en las plataformas Unix lo que ASP en las plataformas Windows. De hecho, ambos lenguajes se insertan en documentos HTML haciendo uso de emulación de etiquetas (otros lenguajes como Perl deben generar toda la página HTML de respuesta). Pero lo cierto es que, a pesar de sus semejanzas, las diferencias entre ambos lenguajes son muchas. Diferencias que hacen que la balanza se vaya inclinando hacia PHP como una mejor solución para implementar aplicaciones Web.

La principal diferencia es que ASP es una tecnología propietaria de Microsoft, mientras que PHP sigue la filosofía *Open Source*. Esto hace que ASP esté pensado para funcionar principalmente sobre plataformas Microsoft, a pesar de que existan soluciones –con un coste económico elevado– como *Sun ONE ASP* que permiten su utilización sobre Unix/Linux. Sin embargo, PHP nos permite que sin ningún problema podamos migrar nuestras aplicaciones Web entre todos los sistemas operativos y servidores en los que funciona.

La filosofía de producto comercial de ASP influye además en que gran cantidad del *software* adicional necesario para complementar una aplicación Web supone un coste económico adicional, por ejemplo, *ASPEncrypt* (necesario para la encriptación), *ServerObject's Qmail* (gestor de correo electrónico) o *Artisans SA-FileUp* (necesario para la gestión de descargas de ficheros). Sin embargo, en PHP todas estas opciones están incluidas de forma gratuita.

Finalmente, la comunicación de errores en ASP y su solución por parte de Microsoft es muchísimo más lenta que en PHP; las revisiones del *software* y los parches a los errores encontrados tienen un tiempo de periodo de desarrollo largo. Hay que tener en cuenta que la filosofía *Open Source* de PHP hace que a lo largo del mundo existan gran cantidad de equipos comprobando el producto, lo cual permite actualizar el producto con nuevas versiones y revisiones que solventan sus problemas de una forma mucho más rápida.

A nivel técnico, se pueden dar muchas razones a favor de PHP: entre ellas, una mayor rapidez de ejecución o una gestión de memoria más acertada. ASP, debido a su propia construcción (basada en una arquitectura COM), nunca podrá llegar a ser tan rápido como PHP. Todas las operaciones en ASP están controladas por objetos COM (*Response*, *Request*, *ADO*, *File System*...). Sin embargo, PHP está construido de forma modular; esto quiere decir que todos sus componentes se ejecutan en el mismo espacio de memoria que PHP. De este modo, el código PHP puede ejecutarse más rápidamente al no sufrir la sobrecarga impuesta por la comunicación con los diferentes objetos COM y procesos que soporta ASP. Además, cada compilador de ASP se ejecuta en su propio proceso, de modo que, cuando nos encontramos el comienzo de una etiqueta ASP, se produce un cambio de contexto para salir del *parser* HTML y elegir el compilador apropiado, volviendo a realizar un salto de contexto al encontrar la etiqueta de cierre ASP para volver de nuevo al *parser* HTML.

Algunos de los problemas de ASP se resuelven en parte con la última versión del producto ASP.NET. Si bien, hay que tener en cuenta que este producto no puede ser considerado como un lenguaje de programación de páginas Web, sino más bien, como una herramienta de desarrollo de aplicaciones Web (una de las principales realizaciones es la flexibilidad a la hora de elegir el lenguaje de programación a utilizar, ASP.NET trabaja con lenguajes de *script* como VBScript, JScript, PerlScript y Python y con lenguajes compilados como VB, C#, C, Cobol, Smalltalk y Lisp).

CAPÍTULO 1

INSTALACIÓN

Tal y como se ha comentado en la introducción, la configuración en la que se encuentra de forma más habitual el lenguaje PHP suele ser conjuntamente con Apache como servidor Web y MySQL como gestor de base de datos. De hecho, está muy extendido el término **LAMP**, cuyas iniciales hacen referencia a **L**inux, **A**pache, **M**ySQL y **P**HP (aunque también podría ser Perl o Python). Por todo ello, en este capítulo comentaremos la instalación de estos tres productos en las plataformas Linux y Windows.

En este capítulo no se aborda la instalación de SQLite puesto que está disponible por defecto en la distribución de PHP 5.0. De todas formas, en el capítulo de bases de datos y dentro de la sección de SQLite se incluye un apartado sobre su instalación en sistemas U*ix.

1.1 MODO DE FUNCIONAMIENTO

El intérprete PHP puede ser ejecutado por el servidor Web de dos modos distintos: como módulo del propio servidor (interfaz SAPI, *Server Application Programming Interface*) o como programa externo a éste (modo CGI, *Common Gateway Interface*).

A grandes rasgos, ejecutar un programa CGI le supone a la máquina donde se está ejecutando el servidor Web llamar al sistema operativo para que realice las siguientes tareas básicas: cargar el programa en memoria, anotarlo en la lista de tareas, lanzar su ejecución, esperar a que termine y, por fin, descargarlo de memoria y de la lista de tareas. Es de perogrullo: tantas veces un cliente pida la ejecución de

un programa CGI, tantas veces se repetirán estas acciones en la máquina servidora... Es fácil imaginar que, si el número de peticiones es medio-alto, el rendimiento general de ésta se verá proporcionalmente degradado, no ya sólo por el número de tareas simultáneas a realizar, sino por la ocupación de la memoria física del equipo servidor con las copias del programa CGI ejecutándose.

Por otra parte, podemos pensar en un SAPI como un protocolo que permite acceder directamente a las funciones internas del servidor; por tanto, a través del SAPI podemos añadir nuevas funcionalidades a un servidor Web (por ejemplo, acceso a bases de datos, autenticación de usuarios, cacheo de páginas, generación de imágenes al vuelo, etc.). Lo más interesante de esta característica es que todas estas nuevas funcionalidades se van a ejecutar de forma más rápida y eficiente ya que lo van a hacer en el espacio de memoria del propio servidor. Esto significa que las ejecuciones de los programas las hará el propio servidor Web y, por tanto, serán mucho más rápidas y eficaces (no habrá creación de procesos nuevos ni ocupación extra de la memoria física con copias de éstos...).

Según el fabricante de servidores, existen varios tipos de API's: **Apache API** para el servidor Apache, **ISAPI** de Microsoft para su producto *Internet Information Server*, **NSAPI** de AOL para el servidor *iPlanet* (antes, *Netscape Enterprise Server*) y **WSAPI**, que es el SAPI del servidor *Website Pro* de O'Reilly[1].

Además del bajo rendimiento, la versión CGI tiene innumerables problemas de seguridad asociados. Si, por la razón que fuera, se necesitara instalar PHP en su versión CGI (por ejemplo, para que los procesos PHP se ejecuten bajo determinados usuarios), se recomienda la lectura de los siguientes documentos del CERT (*Computer Emergency Response Team*):

http://www.cert.org/advisories/CA-1996-11.html

http://www.cert.org/advisories/CA-96.11-interpreters_in_cgi_bin_dir.html

1.2 INSTALACIÓN EN WINDOWS

PHP está portado para las versiones de Windows de 32 bits, esto es, Windows 95/98/Me y Windows NT/2000/XP.

---

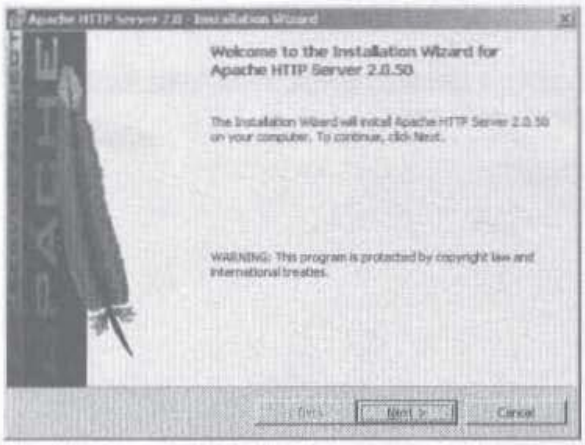
[1] La función `php_sapi_name()` devuelve el tipo de SAPI que hay entre el servidor Web y el intérprete PHP. Si PHP se está ejecutando como CGI, devolverá la cadena "cgi".

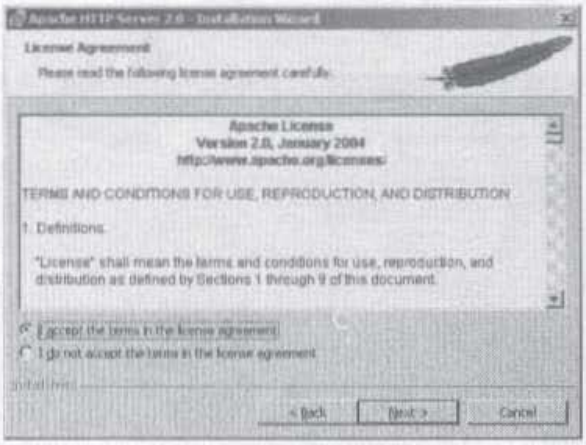
1.2.1 Instalación de Apache en Windows

De las dos ramas de desarrollo del servidor Apache 1.3.x y 2.x, vamos a instalar esta última por las ventajas que ofrece en cuanto a rendimiento y amplia gama de posibilidades; de hecho, la versión 2.0 es competitiva con el servidor IIS de Microsoft en cuanto a estabilidad y rendimiento. De momento, la versión 2.x sólo se recomienda para Windows NT/2000/XP/2003, plataformas claramente orientadas a trabajar como servidores.

En el momento de la edición de este capítulo, la última versión liberada y estable para Windows es la 2.0.50; por tanto, bajaremos de la dirección http://httpd.apache.org/download.cgi el fichero `apache_2.0.50-win32-x86-no_ssl.msi`. El nombre del fichero nos permite saber dos cosas: que será ejecutado por Microsoft System Installer[2] (extensión `msi`), y que no incorpora las librerías Secure Sockets Layer necesarias para instalar un servidor seguro (con encriptación): sufijo `_no_ssl`.

Antes de lanzar la ejecución del fichero `msi` desde el explorador de archivos, si hubiera un servidor Apache ya instalado (y además estuviera funcionando), lo primero que habría que hacer es detener su ejecución. La instalación de Apache en Windows es muy sencilla, basta seguir las indicaciones y rellenar algunos campos de formulario:



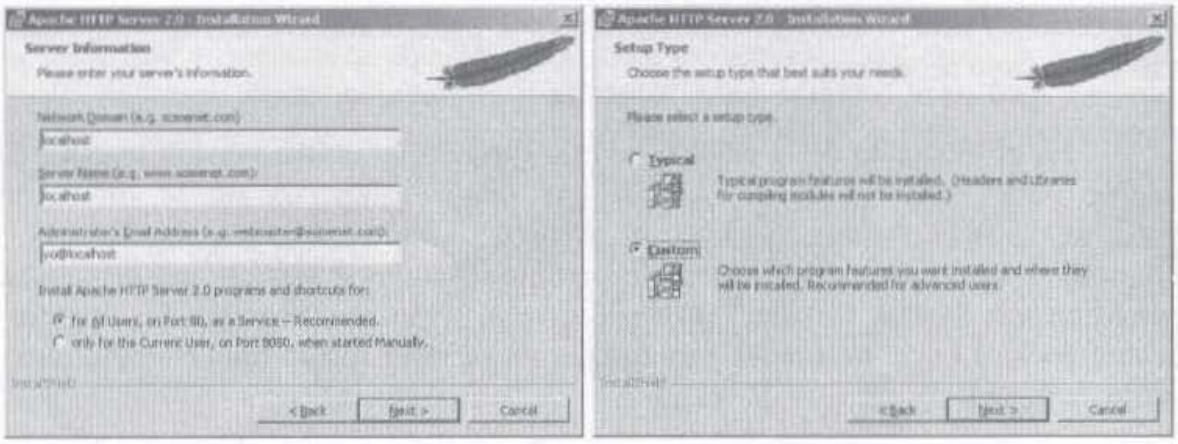


Después de la presentación y la aceptación de los términos de la licencia de uso de este paquete *software*, aparecerá otra ventana donde se nos pide información tal como el nombre del dominio donde va a estar *colgado* el servidor, el nombre con

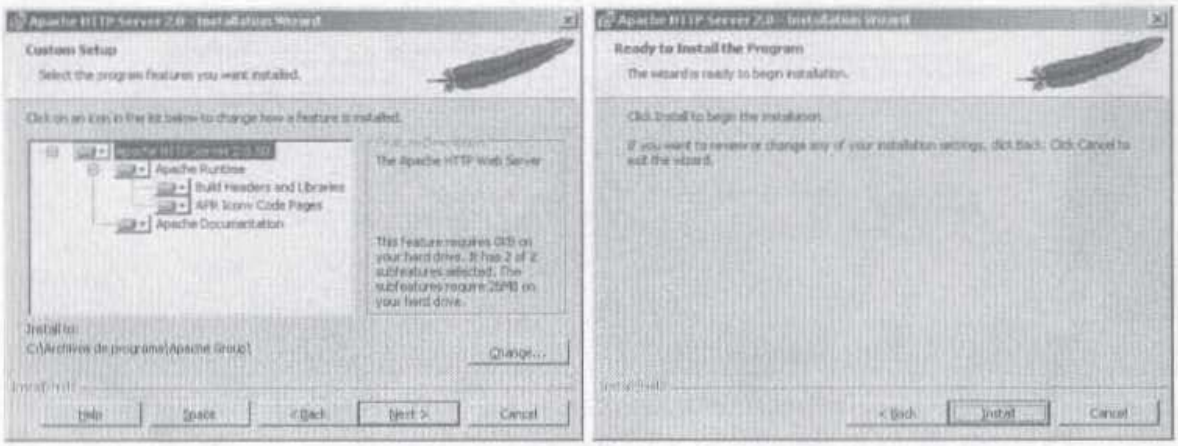
---

[2] Se necesita la versión 1.10 como mínimo. Para NT puede bajarse de http://www.microsoft.com/downloads/release.asp?ReleaseID=17344 y para Win-9x de la dirección http://www.microsoft.com/downloads/release.asp?ReleaseID=17343.

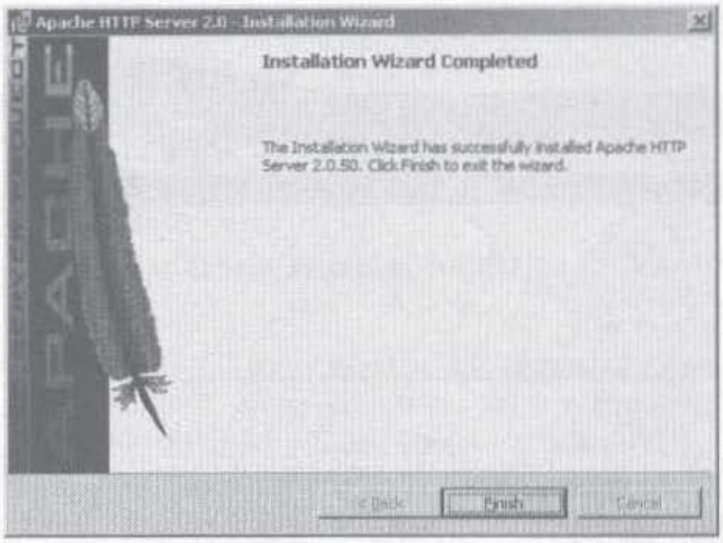
el que queremos que se dé a conocer y la dirección de correo del administrador. En la misma pantalla se nos pide además si queremos que este paquete se instale como servicio de Windows (se ejecutará de manera automática al arrancar el equipo) en el puerto estándar (*well-known*) 80, o bien, que se instale como una copia para el usuario desde el que se está realizando la instalación y en el puerto 8080.



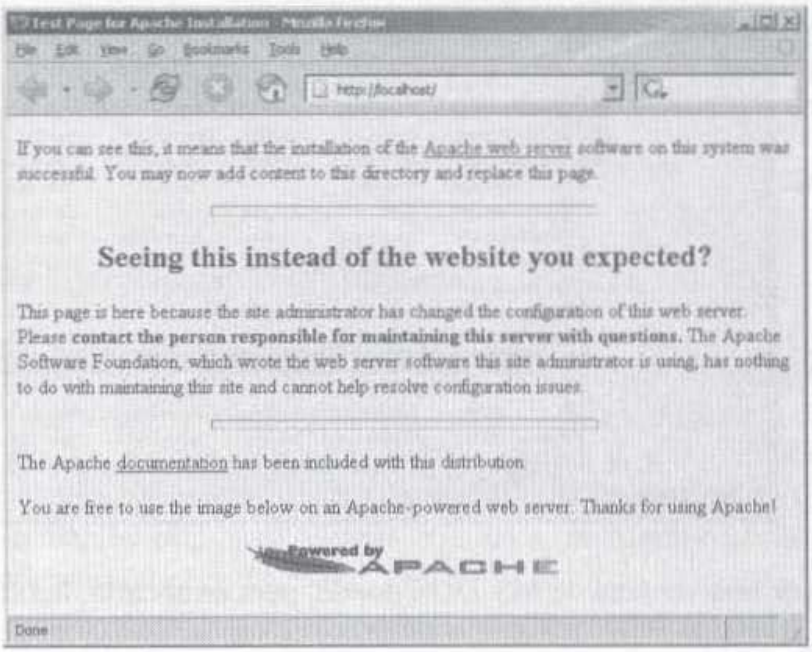
En la siguiente pantalla, preguntará si queremos hacer la instalación típica (se instala todo, excepto el código fuente) o personalizada. El directorio por omisión donde se instalan todos los ficheros de Apache es `c:\Archivos de programa\Apache Group2\Apache2`; allí se crearán, entre otros, los directorios `conf`, `htdocs`, `bin`, `cgi-bin`, `libexec`, `logs`, etc.



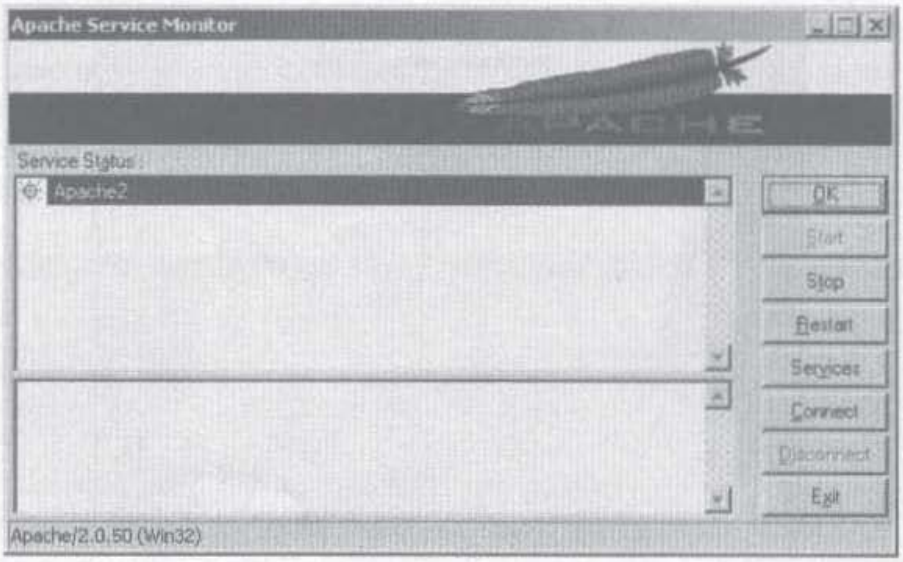
Una vez indicados los datos necesarios, procederemos a instalar el paquete pulsando en el botón preparado al efecto. Al terminar esta operación, deberemos ver la siguiente pantalla:



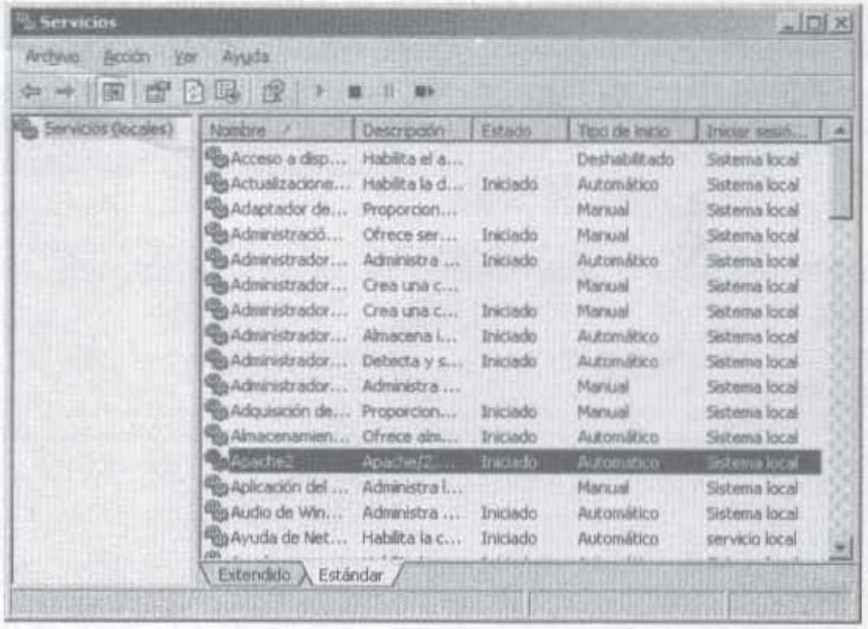
Para comprobar que, efectivamente, hemos hecho una instalación correcta, arrancamos el servidor (ya está disponible desde el menú `Inicio -> Archivos de Programa -> Apache HTTP Server 2.0.50 -> Control Apache Server -> Start`) y, con un navegador, nos conectamos a cualquiera de los URLs http://localhost/ o http://127.0.0.1/. Si la instalación ha sido correcta, veremos la siguiente imagen:



Las operaciones de arrancar y parar el servidor las podemos realizar de tres formas distintas. La primera de ellas es a través del programa monitor que viene con la instalación de Apache (`Inicio -> Programas -> Apache HTTP Server 2.0.50 -> Control Apache Server -> Monitor Apache Servers`) que nos mostrará la siguiente pantalla:



Si está instalado como servicio de Windows, podemos hacerlo desde el entorno gráfico del sistema operativo (`Inicio -> Configuración -> Panel de Control -> Herramientas Administrativas -> Servicios`):



O desde una ventana de MS-DOS donde, para arrancarlo, habrá que ejecutar el comando:

```
net start apache2
```

Y, para pararlo, el comando:

```
net stop apache2
```

1.2.2 Instalación de PHP en Windows

Los pasos básicos a realizar para la instalación de PHP en plataformas Windows son:

1. Obtención de la última distribución PHP

En la dirección http://www.php.net/downloads.php encontraremos las versiones binarias del paquete (válidas para todas las versiones Windows de 32bits: Win-9x/Me, Win-NT/2000/XP). De las dos versiones que allí hay, `zip` y `msi`, nos bajaremos la primera puesto que en ella se incluye tanto la versión CGI como la versión SAPI para Apache. La última versión de PHP 5 estable que encontramos es la 5.0.1 (almacenada en el fichero `php-5.0.1-Win32.zip`).

2. Descomprimir el fichero bajado

Una vez tengamos el fichero en nuestro disco duro, el siguiente paso es descomprimirlo. Nuestra recomendación es hacerlo bajo un directorio cuyo nombre sea el nombre completo de la distribución usada (en este caso, `c:\php-5.0.1-Win32`) dado que PHP es un proyecto que está en permanente evolución y, por tanto, aparecen de forma continua nuevas versiones. Albergar la distribución en una carpeta del disco duro que tenga como parte del nombre el número de distribución evitará mezclar versiones y configuraciones diferentes.

3. Configuración de PHP: Librerías y `php.ini`

Es recomendable que todos los ficheros de PHP (ejecutables, librerías, extensiones y ficheros de configuración) estén bajo el directorio donde hemos desempaquetado la distribución. En nuestro caso, y para ambos modos de funcionamiento de PHP (módulo y CGI), tendremos debajo de `c:\php-5.0.1-Win32` los ejecutables `php.exe` y `php-cgi.exe`, la librería `php5ts.dll`, y el fichero de configuración `php.ini`.

Para obtener este último, en la misma carpeta `c:\php-5.0.1-Win32`, hacemos una copia de `php.ini-recommended` sobre `php.ini`, y lo editamos según nuestras necesidades. Por ejemplo, para que PHP sea capaz de encontrar las librerías dinámicas que ofrecen funcionalidades adicionales (acceso a base de datos, criptografía, generación de documentos en formato PDF, etc.) necesitamos primero modificar la directiva **`extension_dir`** con la ruta donde están todas las librerías

dinámicas, y luego descomentar la línea correspondiente a la funcionalidad que queramos. Concretamente, ahora que PHP5 no incorpora por defecto el interfaz de acceso a un servidor MySQL[3], las modificaciones que habría que hacer a `php.ini` serían:

```
extension_dir = "c:/php-5.0.1-Win32/ext/"
extension=php_mysqli.dll
```

Además, abría que copiar la librería dinámica `libmysqli.dll`, que se encuentra en la carpeta de la distribución de PHP, en el directorio `c:\windows\system32`.

4. Modificación del fichero `httpd.conf` de Apache

Si el servidor Web elegido es Apache, hay que indicarle un par de cosas para que ejecute el intérprete PHP cuando se le pidan páginas con extensión '`.php`'. Con esa intención modificamos su fichero de configuración `httpd.conf` (recomendamos guardar una copia de respaldo del fichero de configuración) con cualquier editor de texto (bloc de notas, por ejemplo).

Es de resaltar que, según estemos hablando de directorios de Windows o de nombres de directorios en los ficheros de configuración de Apache, la notación cambia: en el primer caso, se usará la barra inversa o *backslash* ("\"), mientras que en el segundo será la barra normal o *slash* ("/").

Después de realizar cualquier modificación en el fichero `httpd.conf`, es conveniente ejecutar desde una ventana de MS-DOS la instrucción `"c:\archivos de programa\apache group\apache2\bin\apache.exe" -t`, para asegurarnos de que los cambios están bien hechos, ya que esta instrucción comprueba la sintaxis del fichero de configuración `httpd.conf`.

Para que Apache sepa dónde encontrar el fichero de configuración de PHP (`php.ini`) modificamos la directiva `PHPIniDir`:

```
PHPIniDir "c:/php-5.0.1-Win32/"
```

A continuación, veremos qué pasos son los necesarios para hacer la instalación de PHP, bajo Apache en Windows, de las dos formas que comentamos al principio del capítulo: como módulo o como CGI.

---

[3] Hay dos interfaces para trabajar contra servidores MySQL: `mysql` y `mysqli`. Utilizamos el último porque da soporte para las versiones 3.22 hasta la 5.0, mientras que la `mysql` sólo llega hasta la versión 4.0.

6a) Modo SAPI: modificación del fichero `httpd.conf` de Apache

Para que se ejecute como módulo de Apache, sólo hay que asegurarse de que las siguientes líneas están en el fichero de configuración de Apache:

```
LoadModule php5_module "c:/php-5.0.1-Win32/php5apache2.dll"
AddType application/x-httpd-php .php .phtml
```

PHP ofrece una característica muy útil que consiste en poder visualizar el programa fuente de cualquier *script* PHP con la sintaxis del lenguaje resaltada en distintos colores. Para conseguir esto, necesitamos hacer dos cosas: que el fichero tenga la extensión `phps` y añadir la línea de abajo a `httpd.conf`:

```
AddType application/x-httpd-php-source .phps
```

6b) Modo CGI: modificación del fichero `httpd.conf` de Apache

Para que se ejecute como un programa aparte, después de descomprimir el fichero empaquetado en el directorio `c:\php-5.0.1-Win32`, hay que asegurarse de que tenemos las siguientes líneas en el fichero `httpd.conf` de configuración de Apache:

```
ScriptAlias /php/ "c:/php-5.0.1-Win32/"
AddType application/x-httpd-php .php .phtml
Action  application/x-httpd-php "/php-5.0.1-win32/php-cgi.exe"
```

En modo de configuración, si queremos visualizar el programa fuente de cualquier *script* PHP con la sintaxis del lenguaje resaltada en distintos colores, necesitaremos crear otro fichero que contenga la función `show_source()`. Por ejemplo, para ver el programa `mi_script.php`, escribimos el programa:

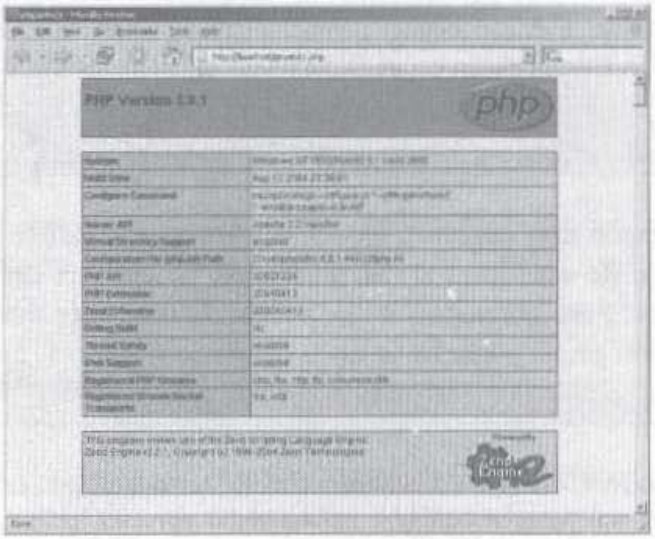
```
<?
show_source("mi_script.php");
?>
```

7. Comprobación de la instalación

Para verificar que PHP se está ejecutando, arrancamos Apache, editamos un fichero que contenga las tres líneas de abajo debajo de la carpeta `"c:\archivos de programa\apache group\apache2\htdocs"` con el nombre 'prueba.php', por ejemplo, y con un navegador nos conectamos a esa página (`http://localhost/prueba.php`).

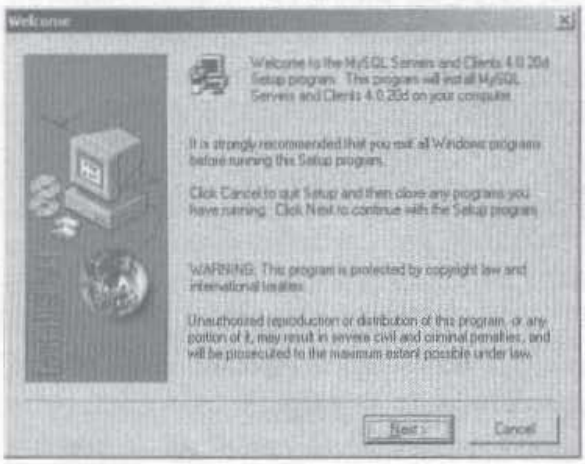
```
<?
phpinfo()
?>
```

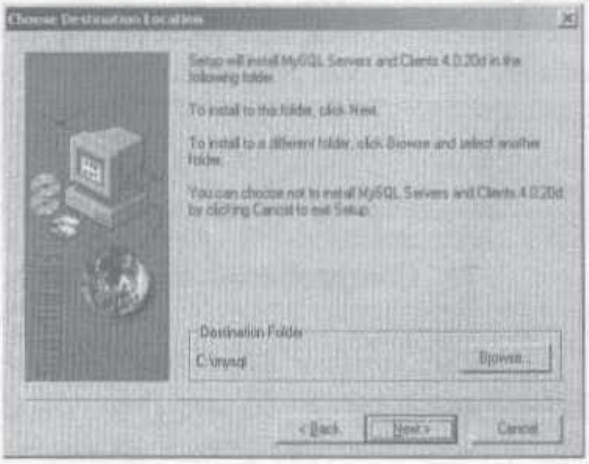
El resultado, si todo ha ido bien, sería el que aparece en la siguiente imagen:

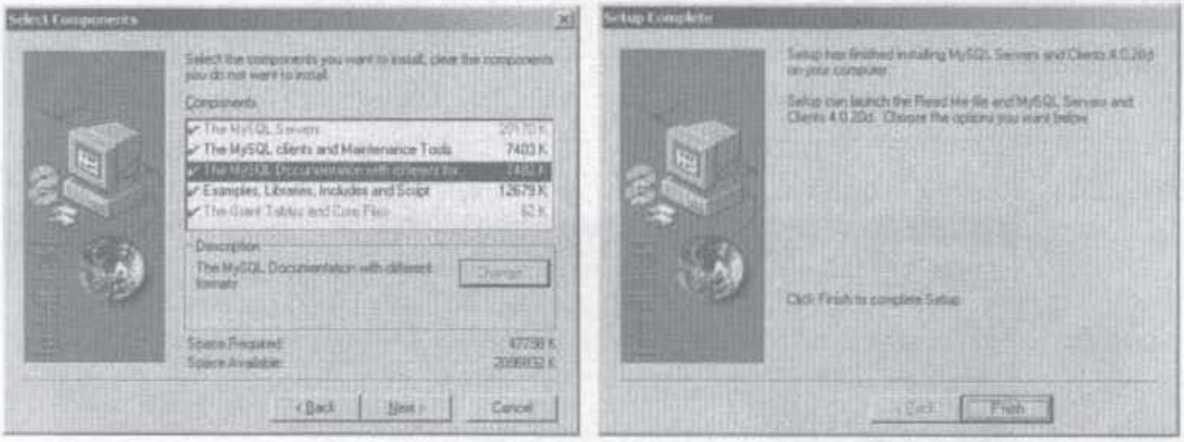


1.2.3 Instalación de MySQL en Windows

Lo primero que hacemos es bajarnos la última distribución estable de MySQL de la dirección http://www.mysql.com/downloads/mysql.html (la encontrada en el momento de la edición de este capítulo fue la 4.0.20) y descomprimirla en un directorio temporal. Luego, nos cambiamos a dicho directorio temporal y ejecutamos el programa `setup.exe`, que es el que hará la verdadera instalación de este paquete:





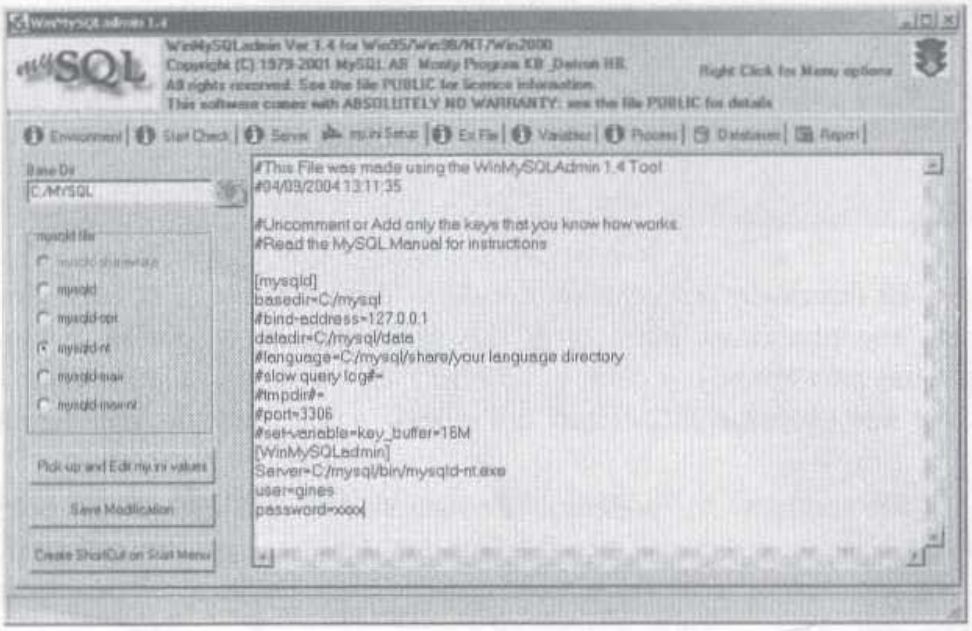


El directorio por omisión donde se instala MySQL es `c:\mysql`. En esta carpeta encontraremos, entre otros, los directorios `bin` (donde se almacenan los programas para arrancar y parar el servidor), `data` (donde se almacenan las bases de datos) y `doc` (documentación muy útil relativa a la instalación, mantenimiento, etc).

El servidor lo podemos arrancara con cualquiera de los comandos `mysqld.exe`, `mysqld-nt.exe` o `WinMySQLAdmin.exe`. Sin embargo, la primera vez que arrancamos el servidor, dado que tiene que inicializar bases de datos, tablas, índices, etc., utilizaremos `WinMySQLAdmin.exe` porque la primera vez que se ejecuta nos pedirá, por medio de la ventana de abajo, el nombre y clave del usuario del sistema bajo el que va a ejecutarse el servidor. También creará el fichero de configuración `c:\windows\my.ini`. Una vez que está arrancado, aparecerá un icono en forma de semáforo en la zona de notificación de la barra de tareas.

`WinMySQLAdmin.exe` es una herramienta gráfica gracias a la cual podremos realizar de manera sencilla e intuitiva la administración de nuestro gestor de base de datos. Con ella podremos conocer datos sobre el equipo servidor donde se está ejecutando, los mensajes que se produjeron al arrancar el servicio, mensajes de error que hayan ocurrido, variables del sistema, procesos en ejecución, bases de

datos existentes, informes y, sobre todo, podremos editar el fichero de configuración.



Si, por el motivo que fuese, necesitáramos instalar la aplicación en un directorio distinto a `c:\mysql`, deberemos incluir en el fichero de configuración `c:\my.cnf` las siguientes líneas:

```
[mysqld]
basedir=UNIDAD:/ruta-instalacion/
datadir=UNIDAD:/ruta-datos/
```

Para evitar que desde fuera de nuestra máquina sepan que tenemos levantado un servidor MySQL podemos poner la directiva `bind-address=127.0.01` en el fichero de configuración `my.ini`, o añadirla como opción en la línea de comandos. De cualquiera de las dos formas, el servidor MySQL sólo atenderá en el interfaz de red local.

Por último, para arrancar y parar manualmente el proceso servidor de MySQL deberemos:

	Arrancar	Parar
9x/Me	`mysqld`	`mysqladmin -u root shutdown`
NT/2000	instalación del servicio: `mysqld-nt --install`	eliminación del servicio: `mysqld-nt --remove`

	Arrancar	Parar
(servicio)	`net start mysql`	`net stop mysql`
NT/2000 (consola)	`mysqld-nt --standalone --console`	`mysqladmin -u root shutdown`

En caso de problemas, el fichero `c:\mysql\data\mysql.err` nos puede ser de gran utilidad ya que es en ese fichero donde el servidor escribe los mensajes de error.

1.3 INSTALACIÓN EN LINUX

Unix es el nombre genérico con el que se denomina a sistemas operativos tales como GNU/Linux, Solaris (Sun Microsistems), True64 Unix (Compaq), HP-UX (HP), AIX (IBM), etc.); esto es, con este término no nos referimos a un único sistema operativo, sino a una amplia familia. Tanto es así, que es frecuente encontrarse con expresiones del tipo **`*ix`** o **`u*ix`** cuando queremos referirnos a Unix.

De entre todas estas versiones de Unix elegimos GNU/Linux por varios motivos: fiabilidad, estabilidad, robustez (como todos los Unix), pero, sobre todo, por su precio: es gratuito. La versión más extendida de Linux/GNU es la que corre en arquitecturas Intel, aunque existen versiones que se pueden instalar y ejecutar en máquinas Alpha de Compaq, McIntosh de Apple, Sparc de Sun Microsystems, *mainframes* S/390 de IBM, etc.

Es habitual que las distribuciones de muchas aplicaciones vengan ya precompiladas (en formato binario o en formato RPM, *RedHad Package Manager*) para distintas arquitecturas. Pero, precisamente por la amplia diversidad de U*ix que podemos encontrarnos, vamos a ver la instalación de estos paquetes de la forma más genérica posible: desde los fuentes.

1.3.1 Instalación de Apache en Linux

Mientras que en Windows se recomienda instalar la versión de Apache perteneciente a la rama `2.0`, en Linux instalaremos la versión `1.3`, dado que la integración con PHP no está suficientemente estable.

Los pasos a realizar consisten en bajarse el fichero `apache_1.3.31.tar.gz` que contiene toda la distribución en código fuente de la dirección http://www.apache.org/dist/httpd/, descomprimirlo en un directorio adecuado, configurar el paquete y compilarlo:

```
$ cd /usr/local/install
$ tar zxvf /usr/local/distrib/apache_1.3.31.tar.gz
$ cd apache_1.3.31
$ ./configure --enable-module=so
$ make
```

Por último, la instalación efectiva hay que hacerla como superusuario o, al menos, con un usuario que tenga los suficientes derechos de escritura en directorios del sistema:

```
# make install
```

Si la instalación ha ido bien, veremos el siguiente mensaje:

```
+--------------------------------------------------------+
| You now have successfully built and installed the      |
| Apache 1.3 HTTP server. To verify that Apache actually |
| works correctly you now should first check the         |
| (initially created or preserved) configuration files   |
|                                                        |
|   /usr/local/apache/conf/httpd.conf                    |
|                                                        |
| and then you should be able to immediately fire up     |
| Apache the first time by running:                      |
|                                                        |
|   /usr/local/apache/bin/apachectl start                |
|                                                        |
| Thanks for using Apache.        The Apache Group       |
|                                 http://www.apache.org/ |
+--------------------------------------------------------+
```

Antes de arrancar el servidor, podemos modificar algunas de las directivas del fichero de configuración `httpd.conf` para que queden con los valores:

```
ServerAdmin webmaster@localhost
ServerName localhost
Port 80
```

Para comprobar que la configuración del servidor Apache es correcta, ejecutamos:

```
# /usr/local/apache/bin/apachectl configtest
Syntax OK
```

Para ejecutar el servidor:

```
# /usr/local/apache/bin/apachectl start
./apachectl start: httpd started
```

1.3.2 Instalación de MySQL en Linux

A partir de PHP5, por problemas relacionados con las licencias de uso, el soporte para MySQL ya no viene integrado por defecto. Por esto, en Linux, hay que instalar MySQL antes que PHP con el fin de indicarle a PHP dónde se encuentran las cabeceras y librerías de MySQL.

Las acciones a seguir son: bajar el paquete `mysql-4.0.20.tar.gz` de la dirección http://www.mysql.com/downloads/mysql.html, configurar, compilar la distribución y, por último, como superusuario, hacer la instalación:

```
$ cd /usr/local/install
$ gzip -d < /usr/local/distrib/mysql-4.0.20.tar.gz | tar xvf -
$ cd mysql-4.0.20
$ ./configure --prefix=/usr/local/mysql \
              --with-mysqld-user=mysql  \
              --with-unix-socket-path=/tmp/mysql.sock
$ make
$ su
# make install
# cp support-files/my-medium.cnf /etc/my.cnf
# chmod 644 /etc/my.cnf
# strip /usr/local/bin/mysqld
```

Por seguridad, el usuario que ejecute el servidor MySQL no será `root`, sino uno cualquiera que no tenga privilegios especiales. Por esto, como superusuario, primero creamos un grupo (`mysql`) y, luego, un usuario (`mysql`), que pertenecerá al grupo recién creado:

```
# groupadd mysql
# useradd -c "Servidor MySQL" -d/dev/null -g mysql -s/bin/false mysql
```

Luego, asignamos los dueños oportunos a los directorios donde se ha instalado la distribución:

```
# chown -R root  /usr/local/mysql
# chown -R mysql /usr/local/mysql/var
# chgrp -R mysql /usr/local/mysql
```

Las tareas que quedan ahora son de puesta a punto de la base de datos: con el *script* `mysql_install_db` creamos la tabla `mysql` donde se definirán qué usuarios y qué máquinas (`hosts`) podrán conectarse y, en general, los permisos de acceso; la otra tabla que se crea se llama `test` y, como su propio nombre indica,

podremos usarla para hacer pruebas. Luego arrancaremos el demonio servidor de MySQL, comprobaremos que, efectivamente, MySQL se está ejecutando y, a continuación, cambiaremos la palabra clave por defecto del superusuario de la base de datos:

```
# scripts/mysql_install_db --user=mysql
# cd /usr/local/mysql/bin
# ./mysqld_safe  --user=mysql &
# /usr/local/mysql/bin/mysqlshow -p
+-----------------+
| Databases       |
+-----------------+
| mysql           |
| test            |
+-----------------+
# ./mysqladmin -u root password 'una.clave'
```

Necesitamos ahora que el servidor MySQL se ejecute de manera automática cuando se levante el sistema operativo. Para ello, debemos situar el *script* que ejecuta el demonio adecuadamente en la estructura de directorios del comando `init` (`/etc/rc.d`)[4].

```
# cp support-files/mysql.server /etc/rc.d/init.d/mysqld
# chkconfig --add mysqld
# /sbin/chkconfig --list mysqld
mysqld          0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

Si alguna vez necesitamos arrancar o parar el demonio de forma manual, podemos hacer:

```
# /etc/rc.d/init.d/mysqld start
# /etc/rc.d/init.d/mysqld stop
```

Una buena medida de seguridad es que el demonio servidor se ejecute como el usuario `mysql` en lugar de hacerlo como `root`; para ello sustituimos en el fichero `/etc/rc.d/init.d/mysqld` la línea:

```
$bindir/mysqld_safe --datadir=$datadir --pid-file=$pid_file &
```

Por esta otra:

```
$bindir/mysqld_safe --datadir=$datadir --pid-file=$pid_file --user=mysql &
```

[4] En una distribución *RedHat*, esta tarea se resuelve fácilmente con el comando `chkconfig`.

1.3.3 Instalación de PHP en Linux

Instalaremos PHP como módulo de Apache (DSO, *Dynamic Shared Object*) porque tiene, entre otras, la ventaja de no necesitar hacer una recompilación de Apache en caso de cambiar a una versión superior de PHP.

A la hora de configurar PHP, necesitamos que éste le *informe* a Apache de que va a usarlo. Si disponemos de la distribución y el código fuente de Apache, podremos usar la opción `--with-apache=/camino/distribucion/apache`. Si, por el contrario, nos encontramos con que Apache ya está instalado y además no tenemos acceso al código fuente para poder recompilarlo, necesitaremos contar con el *script* `apxs`, que suele venir con la distribución de Apache, y usar la opción de configuración `--with-apxs=/camino/script/apxs`. Además, como hemos comentado anteriormente, tenemos que incluir el soporte para MySQL.

```
$ cd /usr/local/install
$ tar zxvf /usr/local/distrib/php-5.0.1.tar.gz
$ cd php-5.0.1
$ ./configure --with-mysql=/usr/local/mysql \
              --with-apxs=/usr/local/apache/bin/apxs
```

Si todo ha ido bien, obtendremos el mensaje siguiente:

```
+--------------------------------------------------------------------+
| License:                                                           |
| This software is subject to the PHP License, available in this     |
| distribution in the file LICENSE.  By continuing this installation |
| process, you are bound by the terms of this license agreement.     |
| If you do not agree with the terms of this license, you must abort |
| the installation process at this point.                            |
+--------------------------------------------------------------------+

Thank you for using PHP.
```

Seguiremos con la compilación y la instalación:

```
$ make
$ su
# make install
```

Este último comando hace efectiva la instalación en los directorios del sistema y se encarga de *notificarle* a Apache de que cargue PHP como módulo. Esto lo hace añadiendo al fichero `/usr/local/apache/conf/httpd.conf` la línea[5]:

[5] Probablemente, esta línea ya estará puesta en `httpd.conf`: se habrá añadido al ejecutar '`make install`'.

```
LoadModule php5_module  libexec/libphp5.so
```

Queda indicarle nosotros mismos a Apache cuándo tiene que ejecutar PHP, esto es, sobre qué ficheros solicitados al servidor tendrá que pasarlos por el intérprete PHP. Lo normal será que queramos que se haga para todos los ficheros con extensión `php`; por ello, creamos un tipo MIME especial con la línea:

```
AddType application/x-httpd-php .php
```

Si en este fichero estuviera la directiva `ClearModuleList`, tendríamos que añadir justo a continuación esta otra línea:

```
AddModule mod_php5.c
```

Por fin, copiamos el fichero de configuración que viene como ejemplo con la distribución al directorio donde el intérprete de PHP espera encontrarlo:

```
# cp php.ini-recommended /usr/local/lib/php.ini
```

1.4 FICHERO DE CONFIGURACIÓN `PHP.INI`

El fichero de configuración, por omisión, se encuentra en el directorio `/usr/local/lib` en Unix[6]. Por su parte, en Windows este fichero suele estar den la misma carpeta donde está la distribución, o en los directorios `C:\WINDOWS` o `C:\WINNT`, según el caso.

Algunas de las directivas más importantes las describimos a continuación:

Directiva	Valor	Explicación
`register_globals`	`on`	Crea automáticamente las variables
`display_errors`	`on`	Muestra los errores en el navegador (útil mientras depuramos un *script*)
`safe_mode`	`on`	Modo seguro: sólo permite accesos a ficheros cuyo dueño sea el mismo que el del proceso
`open_basedir`		Limita al directorio especificado las acciones sobre ficheros

[6] Puede especificarse otra situación en el momento de hacer la configuración inicial con la opción `--with-config-file-path=/directorio/deseado`.

Directiva	Valor	Explicación
`max_execution_time`	`30`	Tiempo de ejecución máxima para un *script* (previene ataques de denegación de servicio o *DoS*)
`Memory_limit`	`8M`	Memoria máxima asignada a un *script* (previene *DoS*)
`SMTP`		Servidor SMTP al que se le entregarán los mensajes que se deseen enviar (sólo Windows)
`sendmail_from`		Dirección de correo del remitente (sólo Windows)

1.5 PAQUETES INTEGRADOS

En los apartados anteriores, hemos descrito los procedimientos necesarios para la instalación manual de Apache, PHP5 y MySQL, tanto en Linux como en Windows. Sin embargo, dada la aceptación y uso de estas aplicaciones, existen *kits* de instalación automática que nos simplifican todas las tareas anteriores, ya que nos instalan las tres aplicaciones (y algunas más como Perl, PHPMyAdmin, Webalizer, etc.) a partir de un solo ejecutable. Algunos de estos paquetes son los siguientes:

- Xampp: http://www.apachefriends.org/en/xampp.html
- FoxServ: http://sourceforge.net/projects/foxserv/
- Apache2Triad: http://apache2triad.sourceforge.net/
- Wamp5: http://www.en.wampserver.com/download.php/
- Winlamp: http://www.datacaptech.com/winlamp.php
- AppServ: http://www.appservnetwork.com/
- NuShpere: http://nusphere.com/

CAPÍTULO 2

FUNDAMENTOS DEL LENGUAJE PHP

La sintaxis de PHP es muy parecida a otros lenguajes de programación muy extendidos como son los lenguajes C, JAVA, Perl o, incluso, el lenguaje de *script* JavaScript. Por tanto, todos aquéllos que estén familiarizados con estos lenguajes encontrarán una gran facilidad a la hora de seguir la estructura sintáctica de las instrucciones y sentencias que presenta el lenguaje PHP.

El vocabulario de PHP, relativamente pequeño, es fácil de comprender y nos da un amplio número de posibilidades, antes no disponibles. Además, PHP nos proporciona un conjunto de herramientas compactas propias que realzan las interacciones entre los usuarios y las páginas HTML, permitiéndonos dar servicio a las peticiones más habituales de una forma sencilla.

Finalmente, PHP ofrece, como veremos, las características básicas de un lenguaje orientado a objetos, pero sin las complejas realizaciones que acompañan a estos lenguajes como Java y C++. Nos permite la definición básica de clases, objetos y la utilización de la herencia.

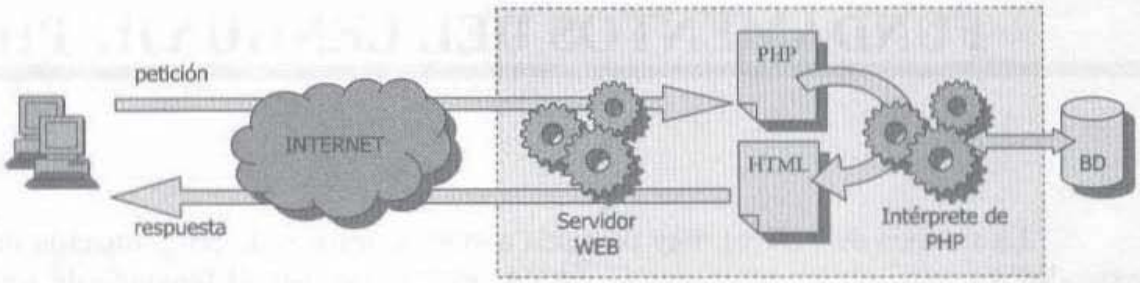
2.1 FORMATO DEL CÓDIGO PHP

A continuación vamos a ir introduciendo las bases sintácticas de la programación en PHP, comentando sus aspectos más importantes y viendo algunos

ejemplos que nos permitan entender la utilización de las estructuras, tipos de datos, sentencias e instrucciones que definen el lenguaje.

2.1.1 Delimitadores

PHP está muy relacionado con el lenguaje de hipertextos HTML; tanto es así, que el código PHP aparece normalmente insertado dentro de un documento HTML. El documento PHP, una vez interpretado correctamente en el servidor, genera una página HTML que será enviada al cliente, tal y como muestra el siguiente gráfico:



Para diferenciar ambos lenguajes dentro del mismo documento, se utilizan etiquetas de comienzo y final del código PHP. Las etiquetas más habituales para delimitar los bloques de código PHP son las siguientes:

```
<?php
        instrucciones PHP
?>
```

Existen otros posibles formatos de etiquetas, menos utilizados que los anteriores, si bien no todas estas opciones están disponibles en nuestro sistema por defecto. Su utilización será correcta dependiendo de las características de configuración seleccionadas durante el proceso de instalación del intérprete de PHP:

```
<?
        instrucciones PHP
?>
```

> **NOTA**: Debe estar activada la directiva `short_open_tag` en el fichero de configuración `php.ini`.

```
<%
    instrucciones PHP
%>
```

> **NOTA:** Debe estar activada la directiva `aspt_tags` en el fichero de configuración `php.ini`. Este tipo de etiquetas son las utilizadas por ASP (Active Server Pages). El soporte para estas etiquetas se añadió en la versión 3.0.4 de PHP.

Finalmente, podemos introducir el código PHP dentro del documento HTML haciendo uso de la etiqueta `<script>`; para ello, además, deberemos indicar el lenguaje a utilizar:

```
<script language="php">
    instrucciones PHP
</script>
```

En los ejemplos del libro utilizaremos las etiquetas `<?php... ...?>` por ser las más ampliamente utilizadas y no necesitar ninguna configuración por defecto.

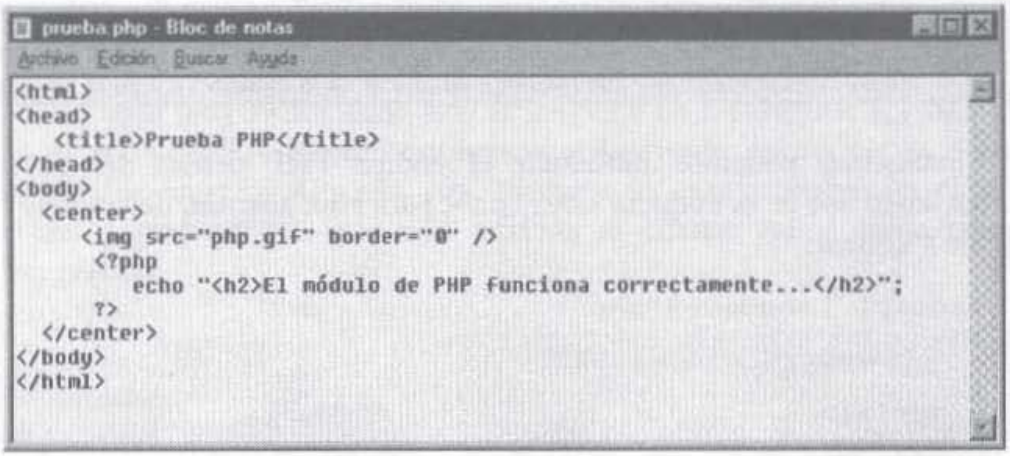
2.1.2 Extensión de los ficheros en PHP

La extensión de los ficheros que se utilizan en PHP es muy importante, ya que, dependiendo de dicha extensión, el servidor Web utilizado decide si el documento solicitado debe ser procesado por el intérprete de PHP o no. Además, las extensiones que indican al servidor HTTP que el fichero contiene códigos PHP son las siguientes:

`.php3`	Código PHP 3.x
`.php4`	Código PHP 4.x
`.php`	Indica código PHP (esta extensión será la que utilicemos a la hora de guardar nuestros programas PHP)
`.phps`	Utilizada para ver la sintaxis del código resaltado en colores
`.phtml`	Extensión en desuso

En general, PHP 3.0 es compatible con PHP 4.0, por lo que no tiene mucha importancia el poner una extensión u otra, ya que el intérprete las procesa casi de igual forma.

Ahora estamos en situación de poder hacer nuestra primera página PHP y comprobar el correcto funcionamiento del intérprete. Para ello, editaremos y guardaremos el siguiente código en un fichero llamado `prueba.php`[1]:



Una vez interpretado correctamente por el procesador PHP el documento anterior, el código que le llega al navegador y, por tanto, lo que éste visualiza es:

Como se puede observar en el ejemplo anterior, el código PHP se halla inmerso dentro de un documento HTML que da formato a la página. Habitualmente encontraremos código PHP insertado en cualquier parte del documento HTML y aparecerá tantas veces como sea necesario. Cada uno de los bloques de código PHP, al ser procesado, generará una salida en el documento HTML resultado, justo en la posición en que dicho bloque está.

Esto no tiene por qué ser siempre así, es decir, el código PHP también puede formar parte de un documento que no contenga ninguna etiqueta HTML. De esta

[1] En los ejemplos iniciales haremos uso de la función `echo`, que nos permite mostrar información en el cuerpo del documento HTML que será interpretado por el navegador del cliente.

forma, el código de abajo visualizará el mismo texto pero sin ningún formato adicional:

```
<?php
echo "<center><img src=php.gif border=0>";
echo "<h2>El módulo de PHP funciona correctamente...</h2></center>";
?>
```

En definitiva, la idea a recordar es que el servidor Web (salvo casos muy concretos) siempre va a devolver código HTML al cliente. Cuando solicitamos una página .php, el servidor invocará al intérprete PHP y éste comenzará a "traducir" el documento seleccionado, es decir, leerá el documento, devolviendo al servidor el texto que se encuentre tal y como está, exceptuando a aquél que se encuentre entre las etiquetas <?php y ?> , que lo analizará y ejecutará por ser código PHP. Si alguna de las instrucciones PHP genera texto, el intérprete lo entregará al servidor Web para que éste lo envíe al cliente.

2.1.3 Comentarios

Para introducir comentarios dentro del código, PHP ofrece la posibilidad de hacerlo de tres formas distintas que también son utilizadas en otros lenguajes. Las siguientes porciones de código nos muestran la utilización de estos diferentes tipos de comentarios:

```
<?php
   // Comentario inicial
   echo "Primer tipo de comentarios";  // Comentario tipo C, C++
   # Comentario final tipo shell de unix
?>
```

Este primer tipo de comentario sólo se puede utilizar para comentar una única línea de código. Se puede usar el doble carácter "//" (utilizado en los lenguajes C, C++, Java, Javascript, etc.) o el carácter almohadilla "#" (Perl, Shell).

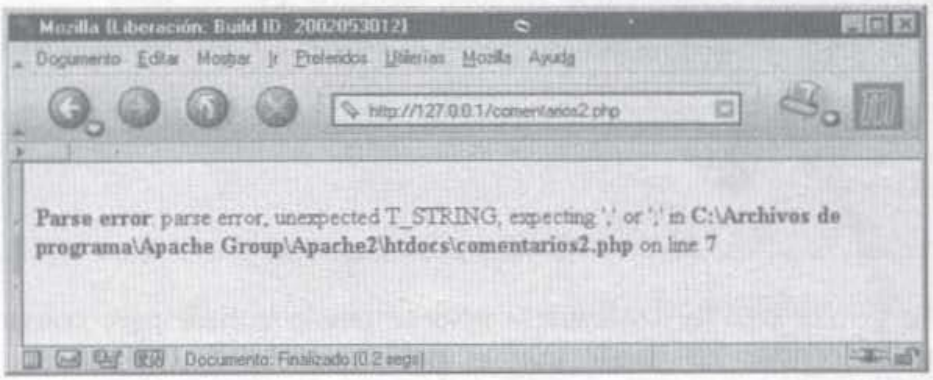
```
<?php
   /* Comentario inicial */
   echo "Segundo tipo de comentarios";  /* Comentario tipo C, C++ */
   /*
      Comentario
      Final
   */
?>
```

Este segundo tipo de comentario es de tipo multilínea, es decir, nos permite comentar varias líneas de código fuente. Por tanto, podemos extender nuestros comentarios a más de una línea de código, si bien, como se puede ver en el ejemplo, también se puede utilizar para realizar comentarios de una sola línea.

Hay que tener cuidado con los comentarios de tipo multilínea, ya que no pueden anidarse, es decir, no se pueden poner comentarios dentro de otros comentarios. También es importante tener en cuenta que este tipo de comentarios siempre tiene que cerrarse con la secuencia de caracteres "*/". El siguiente ejemplo nos muestra cómo se interpretaría un código PHP en el que existe un comentario multilínea que, a su vez, contiene otro comentario de tipo multilínea:

```
<?php
   echo "Comentario... "
        /* Este comentario es usado para comprobar
           /* que dos comentarios multilinea anidados
              no pueden darse
           */
         ya que se llega a un error */
?>
```

Tal y como muestra la imagen, se nos presenta un mensaje de error en el explorador, indicándonos la dirección del fichero dentro del servidor y la línea donde se encuentra el error: la primera serie de caracteres "*/" cierra la apertura del primer comentario de forma que el intérprete de PHP intenta procesar la última línea de comentarios introducida en el código y, al no reconocer la primera palabra, genera el error.



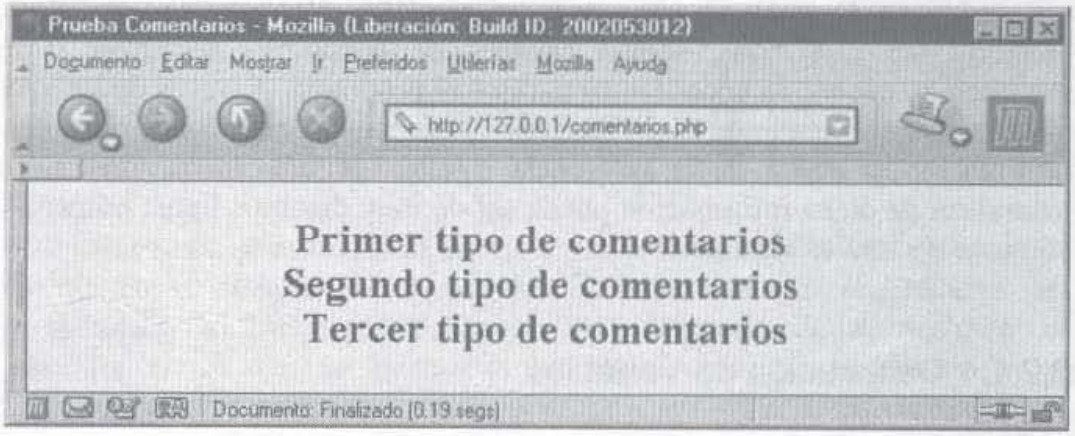
Dentro del código PHP podremos usar indistintamente cualquier tipo de los tres para introducir comentarios. No obstante, no es recomendable mezclar distintos tipos de comentario en un mismo archivo, sino elegir una sintaxis de las anteriormente citadas y utilizarla en todo el documento.

El siguiente código nos muestra la utilización de los diversos tipos de comentarios dentro de un documento PHP:

```
<HTML>
 <HEAD>
   <TITLE>Prueba Comentarios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>
```

```
        <?php
            // Comentario inicial
            echo "Primer tipo de comentarios";  // Comentario tipo C, C++
            // Comentario final
        ?>
        <BR>
        <?php
            /* Comentario inicial */
            echo "Segundo tipo de comentarios";  /* Comentario tipo C, C++ */
            /*
              Comentario
              Final
            */
        ?>
        <BR>
        <?php
            # Comentario inicial
            echo "Tercer tipo de comentarios";  # Comentario tipo shell
            # Comentario Final
        ?>
    </H2>
  </CENTER>
 </BODY>
</HTML>
```

Como podemos observar, sólo se muestra la información impresa por la función `echo` y no el resto de comentarios incluidos en el código:



2.1.4 Fin de línea

PHP ignora cualquier carácter de espaciado presente en el código, incluyendo espacios en blanco, tabuladores y saltos de línea, excepto si se encuentran dentro de una cadena de texto.

El fin de sentencia se marca en todas las instrucciones con el carácter de punto y coma, ";", o bien, aprovechando la etiqueta de cierre, "?>", puesto que se

asume que el final de la inclusión de código limita la introducción de nuevas instrucciones. Por tanto, los códigos siguientes son correctos:

```
<?php
   echo "Finalizando con la etiqueta de cierre"
?>
<?php
   echo "Finalizando con punto y coma";
?>
```

2.2 SINTAXIS BÁSICA

Para comenzar a programar en PHP, es necesario conocer más detalles de su sintaxis como son: tipos de variables que puede aceptar el lenguaje, definición de constantes y tipo y uso de los operadores. En este apartado veremos en profundidad la sintaxis básica de PHP.

2.2.1 Variables

Una variable es el nombre que se le da a una posición de la memoria del ordenador en la cual se almacena información. Conociendo esta posición de memoria (su dirección), somos capaces de encontrar, actualizar o recuperar información cuando la necesitemos durante el programa. A través de las variables se pueden asignar nombres significativos a las posiciones donde se almacena la información y poder hacer referencia a ellas de una forma más sencilla. La naturaleza de dicha información puede ser de muy distintos tipos: números enteros, números decimales, caracteres...

2.2.1.1 Declaración de Variables

En los lenguajes de programación, al definir o declarar una variable, es habitual expresar la naturaleza del tipo de información que se va a *guardar*. Además, durante todo el programa, cualquier valor asignado a esa variable se espera que sea del tipo de datos utilizado en su definición, provocando un error cuando se intenta asignar un tipo de datos diferente.

En el caso de PHP, no es necesario declarar las variables antes de su utilización. Las variables se crean en el instante en que son utilizadas por primera vez; para su inicialización, se utiliza el operador de asignación (=); a partir de ese instante, podremos recuperar su contenido simplemente referenciando la variable por su nombre. Por otra parte, las variables no tienen asociada la naturaleza del tipo

de información que *almacenan* (este tipo de lenguajes se denominan débilmente *tipados*). De hecho, una variable podrá almacenar durante todo su tiempo de vida diferentes tipos de informaciones.

2.2.1.2 Nombrado de Variables

En PHP todos los nombres de las variables van precedidos por el símbolo de dólar, `$`, seguido, al menos, por una letra (de la a a la z o de la A a la Z) o un guión bajo ("_") para, después, continuar por cualquier combinación de letras (en mayúsculas o minúsculas), de dígitos (del 0 al 9) y de guiones bajos ("_"). La siguiente tabla muestra algunos ejemplos de identificadores válidos e inválidos:

Válidos	Inválidos	Motivo
`$Valor_actual`	`$Valor actual`	Contiene un espacio
`$NumeroDeDatos`	`$#Datos`	Contiene el carácter no válido "#"
`$N`	`$2Saldos`	Empieza por número
`$n`	`$Prueba,valor`	Contiene el carácter noválido ","

Finalmente, cabe destacar que, como el intérprete PHP distingue entre mayúsculas y minúsculas, los nombres de variables con diferencias en mayúsculas o minúsculas en caracteres iguales componen nombres de variables distintas; por tanto, los identificadores `$n` y `$N` son diferentes, es decir, serían variables distintas.

2.2.1.3 Variables predefinidas

PHP ofrece una gran cantidad de variables predefinidas[2] a cualquier *script* que se ejecute en su sistema. Estas variables guardan información relativa del entorno de ejecución del intérprete y del propio PHP. Dependen de factores como el tipo de servidor en el que se ejecuta el intérprete de PHP, la versión y su configuración, entre otras cuestiones.

Estas variables se pueden clasificar, por tanto, en dos grandes grupos: por una parte, las que el entorno, el servidor Web, le pasa al intérprete de PHP y, por otra parte, las que el intérprete pone a disposición del programador. La siguiente tabla muestra alguna de las variables del entorno más utilizadas:

Variable	Significado
`SERVER_NAME`	Indica el nombre del equipo servidor sobre el que se ejecuta el *script*.
`SERVER_PORT`	Indica el puerto del equipo servidor que usa el servidor Web para la comunicación.

[2] Para ver una lista de todas las variables predefinidas se puede usar la función `phpinfo()`.

Variable	Significado
SERVER_SOFTWARE	Indica qué *software* está siendo utilizado en el equipo servidor.
REMOTE_PORT	Contiene el puerto que utiliza el peticionario para comunicarse con el servidor Web.
REMOTE_ADDR	Contiene la dirección remota desde la que se realiza la petición.
DOCUMENT_ROOT	Indica el directorio raíz del documento bajo el que se ejecuta el *script*.
HTTP_REFERER	Contiene la dirección de la página (en caso de haberla) desde la que el navegador saltó a la página actual.

Estas variables se importan en el espacio de nombres global de PHP desde el entorno en el que se esté ejecutando el intérprete PHP. La mayoría es proporcionada por el intérprete de comandos en el que se está ejecutando PHP. Otras variables de entorno son las de CGI, para las que no importa si PHP se está ejecutando como un módulo del servidor o con un intérprete CGI.

La siguiente tabla muestra algunas de las variables que el PHP ofrece al programador para facilitar su tarea:

Variable	Variable[3]	Significado
argv		*array* con los argumentos que se pasan al *script*.
argc		Indica el número de argumentos que se pasan al *script*.
PHP_SELF		Indica el nombre del fichero que contiene el *script* en ejecución, salvo que se ejecute PHP como intérprete de la línea de comandos, en cuyo caso no estará disponible.
HTTP_COOKIES_VARS	_COOKIES	*array* asociativo con las variables pasadas a través de *cookies*.
HTTP_GET_VARS	_GET	*array* asociativo con las variables pasadas a través del método GET.
HTTP_POST_VARS	_POST	*array* asociativo con las variables pasadas a través del método POST.
HTTP_ENV_VARS	_ENV	*array* asociativo con las variables pasadas desde el entorno.

[3] Introducidas a partir de la versión 4.1.0.

Variable	Variable[3]	Significado
HTTP_SERVER_VARS	_SERVER	*array* asociativo con las variables pasadas desde el servidor.
HTTP_SESSION_VARS	_SESSION	*array* asociativo con las variables de sesión.
HTTP_POST_FILES	_FILES	*array* asociativo con información relativa a los ficheros recibidos a través del método POST.

2.2.2 Tipos de datos

PHP soporta tres tipos de datos simples: `integer`, `float`[4] y `string`; y dos tipos de datos compuestos: `array` y `object`. Además, hace uso de un tipo lógico o `boolean`, aunque no aparece definido como tal en la sintaxis del lenguaje.

2.2.2.1 Enteros

Las variables de tipo `integer` pueden contener números enteros que varían entre un rango de -2 billones y +2 billones, y se pueden especificar usando diferentes sintaxis. Los *enteros* se pueden representar en formato decimal (base 10), octal (base 8) o hexadecimal (base 16). Un número entero en formato decimal puede incluir cualquier secuencia de dígitos que no comience por `0` (cero). Los números enteros en formato octal comienzan por un cero y pueden incluir cualquier secuencia de dígitos entre el 0 y el 7. Para designar a un hexadecimal, hay que poner `0x` (o `0X`) delante del entero. Los enteros hexadecimales incluyen dígitos del 0 al 9, junto con letras desde la `a` (o `A`) hasta la `f` (o `F`). He aquí algunos ejemplos:

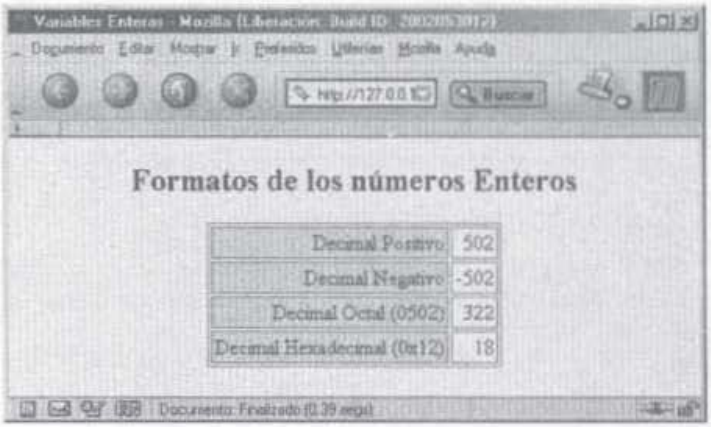
Formato	Valores
Decimal	-33 2139
Octal	071 03664
Hexadecimal	0x7b8 0x395

El siguiente código nos muestra la asignación de valores de tipo *entero* a una variable en los diferentes formatos numéricos que PHP puede manejar para este tipo de datos:

[4] Sustituya a `double` desde la versión 4.2.0.

```
<HTML>
 <HEAD>
   <TITLE>Variables Enteras</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Formatos de los números Enteros</H2>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
        <TR ALIGN="right">
           <TD BGCOLOR="yellow">Decimal Positivo</TD>
           <TD>
            <?php
              $num = 502;    //número decimal
               echo $num;    //mostramos el valor de $num
            ?>
           </TD>
        </TR>
        <TR ALIGN="right">
           <TD BGCOLOR="yellow">Decimal Negativo</TD>
           <TD>
            <?php
              $num = -502;  //número decimal negativo
               echo $num;    //mostramos el valor de $num
            ?>
           </TD>
        </TR>
        <TR ALIGN="right">
           <TD BGCOLOR="yellow">Decimal Octal (0502)</TD>
           <TD>
            <?php
              $num = 0502;  //número octal
               echo $num;    //mostramos el valor de $num
            ?>
           </TD>
        </TR>
        <TR ALIGN="right">
           <TD BGCOLOR="yellow">Decimal Hexadecimal (0x12)</TD>
           <TD>
            <?php
              $num = 0x12;  //número hexadecimal
               echo $num;    //mostramos el valor de $num
            ?>
           </TD>
        </TR>
     </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en la siguiente imagen, la función `echo` muestra por defecto siempre la información en decimal, a pesar de que internamente ésta se haya almacenado en la variable con un formato distinto. Ya veremos más adelante diferentes funciones PHP que nos permiten formatear la información a visualizar.



2.2.2.2 Números en coma flotante

Con las variables de tipo `float` representamos números en coma flotante o, lo que es igual, números con valores decimales. Son, por tanto, números decimales con parte fraccionaria que se pueden expresar en forma estándar o con notación científica. En notación científica se utiliza la letra `E` o `e` para designar el exponente. Ambos, el número decimal y el exponente, pueden ser positivos o negativos, como se muestra en los ejemplos siguientes:

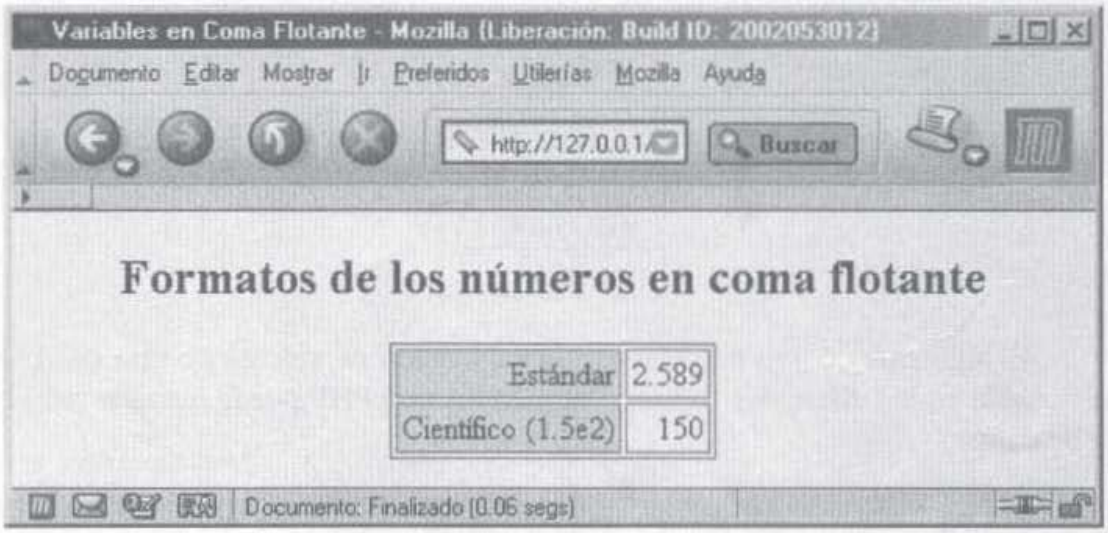
Valores
3405.673
-1.958
8.3200e+11
8.3200e11

El siguiente código nos muestra la asignación de valores de tipo `double` a una variable en los diferentes formatos numéricos que PHP puede manejar para este tipo de datos:

```
<HTML>
 <HEAD>
   <TITLE>Variables en Coma Flotante</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Formatos de los números en coma flotante</H2>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="right">
         <TD BGCOLOR="yellow">Estándar</TD>
         <TD>
          <?php
            $num = 2.589; //formato estándar
             echo $num;    //mostramos el valor de $num
```

```
            ?>
          </TD>
        </TR>
        <TR ALIGN="right">
          <TD BGCOLOR="yellow">Científico (1.5e2)</TD>
          <TD>
           <?php
            $num = 1.5e2; //formato cientifico
             echo $num;    //mostramos el valor de $num
           ?>
          </TD>
        </TR>
      </TABLE>
    </CENTER>
  </BODY>
</HTML>
```

Como podemos observar de nuevo en la siguiente imagen, la función `echo` muestra por defecto siempre la información en decimal, a pesar de que internamente ésta se haya almacenado en la variable con un formato distinto:



2.2.2.3 Cadenas

Con el tipo `string` representamos cadenas de caracteres. Una cadena está formada por cero o más caracteres encerrados entre dobles comillas (") o entre comillas simples ('). Siempre se debe utilizar el mismo tipo de comilla para rodear cada cadena. Los siguientes son ejemplos de cadenas:

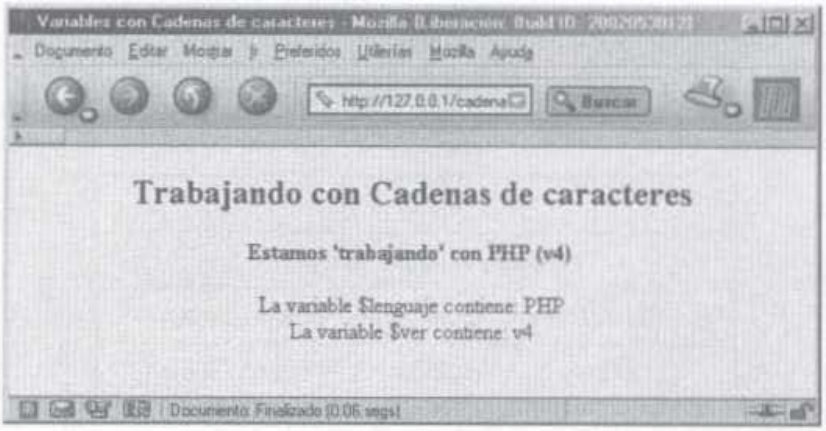
Cadenas
"saludos"
'saludos'
"#12-6"
"¡Sonríe por favor!"
'incluye "dobles" comillas'
"incluye \"dobles\" comillas"

Como ocurre en el penúltimo ejemplo, en algunos casos se puede entremezclar el uso de los dos tipos de entrecomillado, principalmente para insertar una cadena literal dentro de otra. El último ejemplo hace uso del carácter "\" para poder introducir comillas dobles dentro de un texto entrecomillado con comillas dobles a su vez. Los dos últimos ejemplos serían totalmente compatibles.

Cuando utilizamos comillas dobles, podemos incluir dentro de la cadena nombres de variables que serán evaluados (sustituidos por sus respectivos valores) a la hora de mostrar la información. Si introducimos nombres de variables dentro de una cadena encerrada entre comillas simples, la variable no será evaluada. El siguiente ejemplo muestra la asignación de cadenas de caracteres a variables y su funcionamiento diferenciado según el tipo de comillas utilizado:

```
<HTML>
 <HEAD>
   <TITLE>Variables con Cadenas de caracteres</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con Cadenas de caracteres</H2>
     <?php
       $lenguaje="PHP";
       $ver="v4";
       echo "<B>Estamos 'trabajando' con $lenguaje ($ver) </B><BR><BR>";
       echo 'La variable $lenguaje contiene: ';
       echo $lenguaje ;
       echo "<BR>";
       echo 'La variable $ver contiene: ';
       echo $ver;
     ?>
   </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra el resultado de visualizar el código anterior:



Las cadenas de caracteres, además de texto normal y variables, pueden contener caracteres especiales como la tabulación o el retorno de carro que tienen funcionalidades incorporadas. La siguiente tabla nos muestra los más utilizados:

Código de escape	Significado
\b	espacio hacia atrás (*backspace*)
\f	cambio de página (*form feed*)
\n	cambio de línea (*line feed*)
\r	retorno de carro (*carriage return*)
\t	tabulación
\\	barra inversa (*backslash*)
\'	comilla simple
\"	comilla doble
\$	carácter $

Al igual que ocurre con las variables, si la cadena que contiene los caracteres especiales utiliza comillas simples, éstos no se evaluarán y simplemente se mostrarán. El siguiente ejemplo muestra la utilización de este tipo de caracteres para mostrar el mismo resultado que el ejemplo anterior:

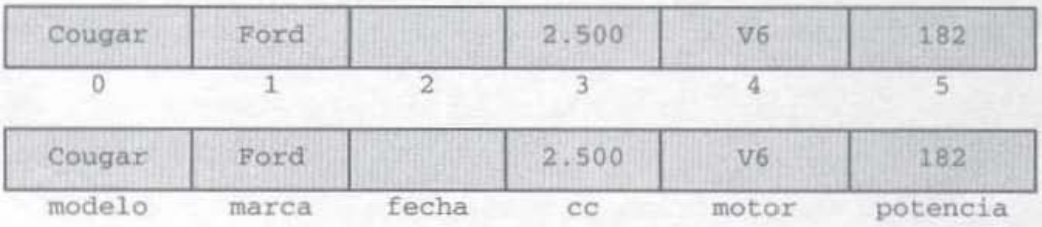
```
<HTML>
 <HEAD>
   <TITLE>Variables con Cadenas de caracteres</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con Cadenas de caracteres</H2>
     <?php
       $lenguaje="PHP";
       $ver="v4";
       echo "<B>Estamos \"trabajando\" con $lenguaje ($ver) </B><BR><BR>";
       echo "La variable \$lenguaje contiene: $lenguaje <BR>";
       echo "La variable \$ver contiene: $ver";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Dada la importancia de las cadenas en la programación en PHP, veremos con más detalle estas variables en un capítulo posterior.

2.2.2.4 *Arrays*

Los *arrays* o matrices son estructuras que permiten el almacenamiento de un conjunto de datos bajo un mismo nombre; son una construcción tradicional de los lenguajes de programación. Podemos definir un *array* o matriz como un conjunto ordenado de elementos identificados por un índice (la posición del elemento dentro de esta colección ordenada), de modo que en cada posición marcada por un índice el *array* contiene un valor. Se pueden construir tantos índices como se quiera, aunque el uso habitual de los *arrays* es en forma de matriz unidimensional. La longitud del *array* se modifica de forma dinámica siempre que le añadimos un nuevo elemento.

En el caso de PHP, los *arrays* pueden estar compuestos de elementos de diferente naturaleza y su primer índice o posición es la 0. Además, existen unos *arrays* especiales en PHP denominados **asociativos** en los que el índice es un valor de tipo `string`, de modo que cada posición está definida por el par (clave, valor), pudiendo acceder al contenido (valor) a través de la clave. La siguiente imagen nos muestra dos posibles estructuras de *arrays* como un conjunto de elementos cada uno de los cuales tiene asociada una posición o una clave:



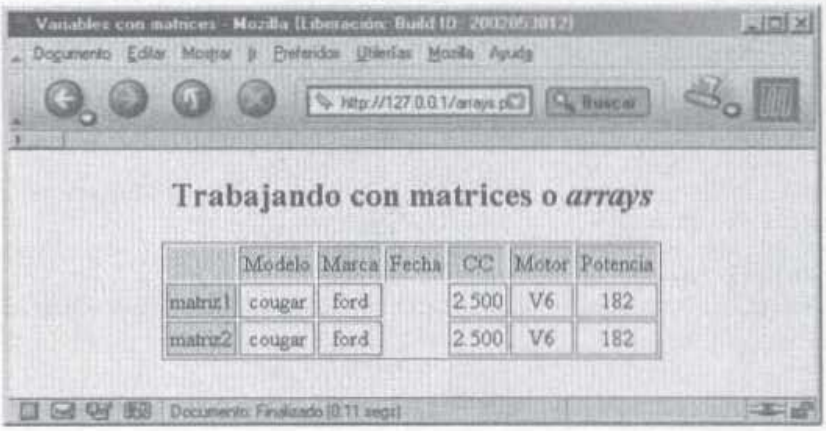
Podemos observar que en ambos *arrays* los índices son de diferente naturaleza y que pueden existir posiciones o claves sin contenido asociado. Para manejar el *array* de forma global, utilizamos el nombre asignado a la variable que lo contiene y, para manejar cada uno de sus elementos, tendremos que hacer referencia a su posición o clave dentro del conjunto global.

A modo de introducción y para familiarizarnos con su uso, vamos a ver un ejemplo en el que se muestra cómo definir y manejar los *arrays* de la forma más sencilla. Si bien, dada la importancia que estos componentes tienen en la programación con PHP, los veremos en detalle más adelante.

```
<HTML>
 <HEAD>
   <TITLE>Variables con matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con matrices o <I>arrays</I></H2>
     <?php
       $matriz1[0]="cougar";
       $matriz1[1]="ford";
       // la tercera posición del array esta vacía
       // por eso le asignamos una cadena sin contenido
       $matriz1[2]="";
       $matriz1[3]="2.500";
       $matriz1[4]="V6";
       // para añadir el último elemento a una matriz
       // no es necesario poner el número de índice
       $matriz1[]=182;

       // creamos la matriz asociativa
       $matriz2['modelo']="cougar";
       $matriz2['marca']="ford";
       // para marca una posición sin contenido también
       // se puede utilizar <null>
       $matriz2['fecha']=null;
       $matriz2['cc']="2.500";
       $matriz2['motor']="V6";
       $matriz2['potencia']=182;
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
        <TR ALIGN="center" BGCOLOR="yellow">
          <TD></TD>
          <TD>Modelo</TD>
          <TD>Marca</TD>
          <TD>Fecha</TD>
          <TD>CC</TD>
          <TD>Motor</TD>
          <TD>Potencia</TD>
        </TR>
        <TR ALIGN="center">
          <TD BGCOLOR="yellow">matriz1</TD>
           <?php
             echo "<TD> $matriz1[0] </TD>";
             echo "<TD> $matriz1[1] </TD>";
             echo "<TD> $matriz1[2] </TD>";
             echo "<TD> $matriz1[3] </TD>";
             echo "<TD> $matriz1[4] </TD>";
             echo "<TD> $matriz1[5] </TD>";
           ?>
        </TR>
        <TR ALIGN="center">
          <TD BGCOLOR="yellow">matriz2</TD>
           <?php
             echo "<TD>". $matriz2['modelo'] ."</TD>";
             echo "<TD>". $matriz2['marca'] ."</TD>";
             echo "<TD>". $matriz2['fecha'] ."</TD>";
             echo "<TD>". $matriz2['cc'] ."</TD>";
             echo "<TD>". $matriz2['motor'] ."</TD>";
             echo "<TD>". $matriz2['potencia'] ."</TD>";
```

```
            ?>
        </TR>
      </TABLE>
    </CENTER>
  </BODY>
</HTML>
```

El resultado se muestra en la siguiente figura:



2.2.2.5 Objetos

Si se tiene un conocimiento de la programación orientada a objetos, se entenderá mejor el concepto de *objeto*; no obstante, a modo de introducción, diremos que un objeto es una estructura que mantiene en sí misma tanto sus características básicas (denominadas *propiedades*), como las posibles funcionalidades que admite (denominadas *métodos*). En capítulos posteriores veremos con más detalle este tipo de datos, así como la forma en que se declaran, crean y utilizan dentro de PHP.

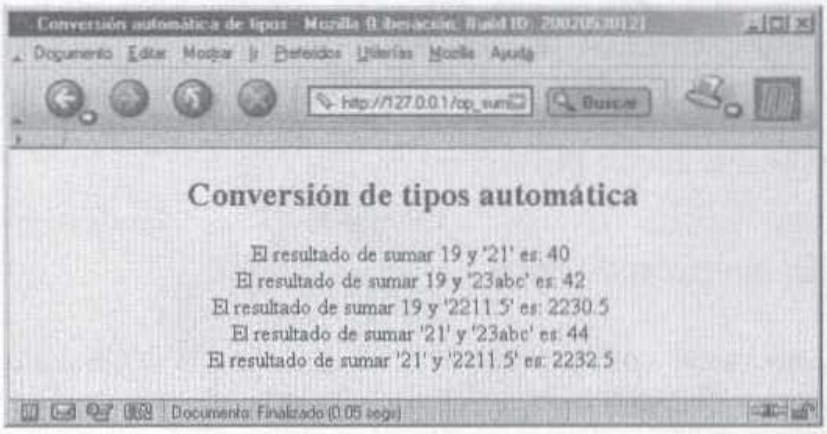
2.2.2.6 Conversión de tipos

Como antes comentamos, PHP es un lenguaje débilmente tipado, es decir, puede contener distintos tipos o valores a lo largo de su vida. Por tanto, en PHP el tipo de una variable se determina por el contexto. Así, una variable a la que se le asigna un valor entero será de tipo `integer`; si, seguidamente, sobre esa misma variable se le asigna una cadena de caracteres, la variable en cuestión pasará a ser de tipo *string*. Además, cuando operamos con variables de distinto tipo, el intérprete de PHP tiende a homogeneizar sus diferentes tipos en función de la operación que se pretende realizar y de los operandos que forman parte de ella.

Un caso muy claro es el que se produce con el operador suma "+". Veámoslo con un ejemplo:

```
<HTML>
 <HEAD>
   <TITLE>Conversión automática de tipos</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Conversión de tipos automática</H2>
     <?php
       $numero=19;
       $cadena1="21";
       $cadena2="23abc";
       $cadena3="2211.5";
       $suma=$numero+$cadena1;
       echo "El resultado de sumar $numero y '$cadena1' es: $suma <BR>";
       $suma=$numero+$cadena2;
       echo "El resultado de sumar $numero y '$cadena2' es: $suma <BR>";
       $suma=$numero+$cadena3;
       echo "El resultado de sumar $numero y '$cadena3' es: $suma <BR>";
       $suma=$cadena1+$cadena2;
       echo "El resultado de sumar '$cadena1' y '$cadena2' es: $suma <BR>";
       $suma=$cadena1+$cadena3;
       echo "El resultado de sumar '$cadena1' y '$cadena3' es: $suma <BR>";
?>
   </CENTER>
 </BODY>
</HTML>
```

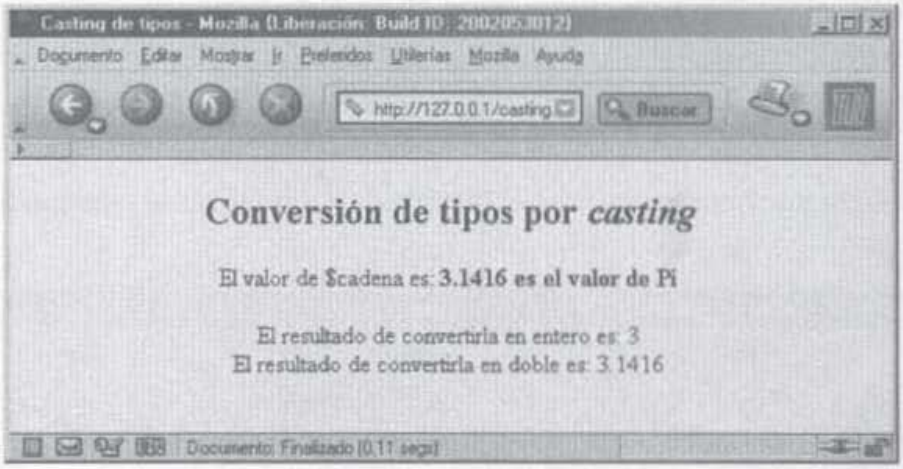
El intérprete intentará convertir las cadenas en valores numéricos para que éstos se puedan sumar correctamente (una cadena que no pueda convertirse a un valor numérico es evaluada con el valor 0 para poder operar aritméticamente con ella). El resultado que obtenemos lo podemos ver en la siguiente imagen:



Además de la conversión automática realizada por el intérprete de PHP, es posible convertir las variables de un tipo a otro cuando el programador lo desee. Se trata de obligar a que una variable sea evaluada con un tipo concreto. Es similar al conversión de tipos que se da en el lenguaje C y que se denomina comúnmente como *casting*. Para ello se escribe entre paréntesis el tipo deseado (`integer`, `float`, `string`, `boolean`, `array`, `object`) antes de la variable a la que se pretende convertir. En el siguiente ejemplo se ven algunas conversiones de tipos:

```
<HTML>
 <HEAD>
   <TITLE>Casting de tipos</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Conversión de tipos por <I>casting</I></H2>
     <?php
       $cadena="3.1416 es el valor de Pi";
       echo "El valor de \$cadena es: <B> $cadena </B><BR><BR>";
       $aux=(integer)$cadena;
       echo "El resultado de convertirla en entero es: $aux <BR>";
       $aux=(double)$cadena;
       echo "El resultado de convertirla en doble es: $aux <BR>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en la siguiente imagen, el *casting* de la cadena inicial a entero nos devuelve un valor de tipo `integer` en un caso y de tipo `float` en el otro. De igual forma, podríamos haber convertido la cadena inicial en cualquiera de los restantes tipos con los que trabaja PHP.



Como podremos ver en el siguiente epígrafe, existen además un conjunto de funciones que permiten al programador modificar el tipo de las variables con las que trabaja siempre que lo desee.

2.2.3 Otros componentes asociados a las variables

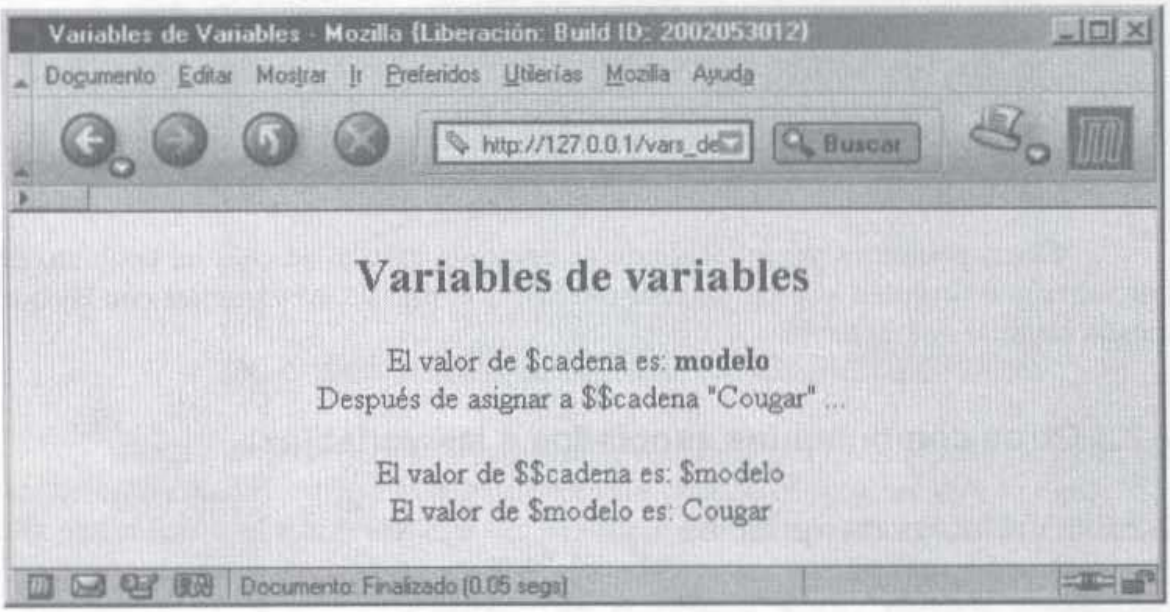
2.2.3.1 Variables de variables

Consiste en reutilizar el valor de las variables como nombre de otra variable al mismo tiempo. Las variables de variables se pueden establecer y usar

dinámicamente; de hecho, habitualmente se utilizan en *scripts* en los que se genera código de forma dinámica. El siguiente ejemplo muestra la utilización de este tipo de variables:

```
<HTML>
 <HEAD>
   <TITLE>Variables de Variables</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Variables de variables</H2>
     <?php
       $cadena="modelo";
       echo "El valor de \$cadena  es: <B> $cadena </B><BR>";
       echo 'Después de asignar a $$cadena "Cougar" ...<BR><BR>';
       $$cadena="Cougar";
       echo "El valor de \$\$cadena es: $$cadena <BR>";
       echo "El valor de \$modelo  es: $modelo <BR>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra el resultado de visualizar el ejemplo anterior:



2.2.3.2 Funciones para variables

PHP proporciona un conjunto de funciones de gran utilidad a la hora de tratar con las variables. Estas funciones son:

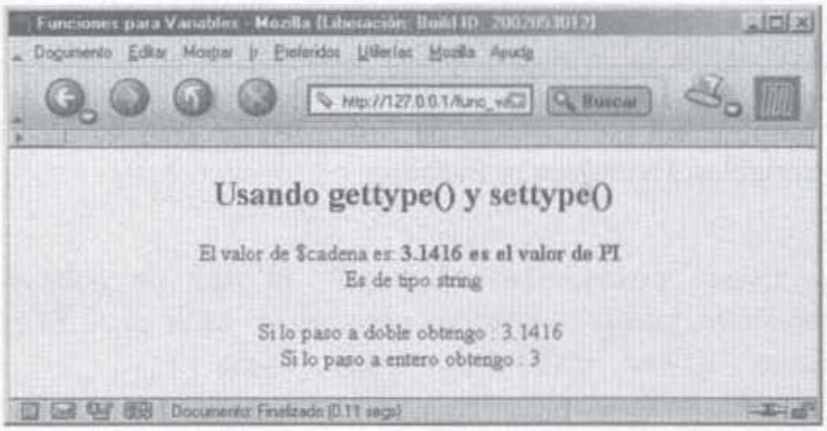
- `gettype (variable)`: Devuelve el tipo de dato pasado como parámetro, pudiendo devolver como valor: `integer`, `float`, `string`, `array`, `class`, `object` y `unknown type`.

- `settype (variable, tipo)`: Establece el tipo de dato a guardar en una variable. Tiene dos argumentos: el nombre de la variable y el tipo que se quiere establecer. Con esta función podemos, pues, realizar conversiones de un tipo de datos a otro. Devolverá valor `true` si ha tenido éxito; en caso contrario, devolverá `false`.

El siguiente ejemplo muestra la utilización de ambas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Funciones para Variables</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Usando gettype() y settype() </H2>
     <?php
       $cadena="3.1416 es el valor de PI";
       echo "El valor de \$cadena  es: <B> $cadena </B><BR>";
       echo "Es de tipo ".gettype($cadena)."<BR><BR>";
       settype($cadena,"double");
       echo "Si lo paso a doble obtengo : $cadena <BR>";
       settype($cadena,"integer");
       echo "Si lo paso a entero obtengo : $cadena <BR>";
     ?>
   </CENTER>
 </BODY></HTML>
```

El resultado obtenido se puede ver en la siguiente imagen:



- `isset (variable)`: Indica si una variable ha sido inicializada con un valor, en cuyo caso devuelve `true` y, en caso contrario, devuelve `false`.

- `unset(variable)`: Destruye una variable liberando los recursos dedicados a dicha variable. Es necesario indicar como parámetro el nombre de la variable a destruir.

- `empty(variable)`: Devuelve valor `true` si la variable aún no ha sido inicializada, o bien, tiene un valor igual a 0 o es una cadena vacía y, en caso contrario, devuelve `false`.

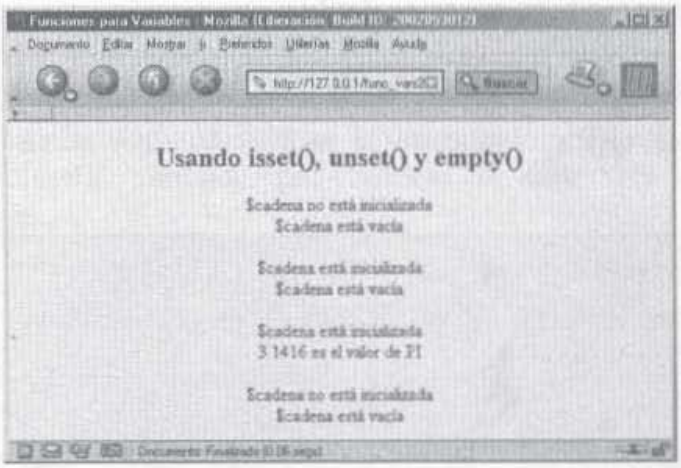
El siguiente ejemplo nos muestra la utilización de las funciones anteriores:

```
<HTML>
 <HEAD>
   <TITLE>Funciones para Variables</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Usando isset(), unset() y empty()</H2>
     <?php
       echo '$cadena ';
       echo (isset($cadena))?'está ':'no está ';
       echo "inicializada<BR>";
       echo (empty($cadena))?'$cadena está vacía':$cadena;
       echo "<BR><BR>";
       $cadena="";
       echo '$cadena ';
       echo (isset($cadena))?'está ':'no está ';
       echo "inicializada<BR>";
       echo (empty($cadena))?'$cadena está vacía':$cadena;
       echo "<BR><BR>";
       $cadena="3.1416 es el valor de PI";
       echo '$cadena ';
       echo (isset($cadena))?'está ':'no está ';
```

```
        echo "inicializada<BR>";
        echo (empty($cadena))?'$cadena está vacía':$cadena;
        echo "<BR><BR>";
        unset($cadena);
        echo '$cadena ';
        echo (isset($cadena))?'está ':'no está ';
        echo "inicializada<BR>";
        echo (empty($cadena))?'$cadena está vacía':$cadena;
    ?>
  </CENTER>
 </BODY>
</HTML>
```

NOTA: En este ejemplo se hace uso del operador condicional que se explica posteriormente en el apartado de operadores.

El resultado de visualizar el ejemplo anterior se muestra en la siguiente imagen:



- `is_int(variable)`, `is_integer(variable)`, `is_long(variable)`: Estas funciones devuelven `true` si la variable pasada como argumento es de tipo `integer`; en caso contrario, devuelven `false`.

- `is_float(variable)`, `is_real(variable)`, `is_double(variable)`: Estas funciones devuelven `true` si la variable pasada como argumento es de tipo `float`; en caso contrario, devuelven `false`.

- `is_numeric(variable)`: Esta función devuelve `true` si la variable pasada como argumento es un número o una cadena numérica; en caso contrario, devuelve `false`.

- `is_bool(variable)`: Esta función devuelve `true` si la variable pasada como argumento es de tipo lógico; en caso contrario, devuelve `false`.

- `is_array(variable)`: Esta función devuelve `true` si la variable pasada como argumento es de tipo *array*; en caso contrario, devuelve `false`.

- `is_string(variable)`: Esta función devuelve `true` si la variable pasada como argumento es de tipo *string*; en caso contrario, devuelve `false`.

- `is_object(variable)`: Esta función devuelve `true` si la variable pasada como argumento es de tipo `object`; en caso contrario, devuelve `false`.

El siguiente ejemplo muestra la utilización de algunas de las funciones anteriores:

```
<HTML>
 <HEAD>
   <TITLE>Funciones para Variables</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Comprobando el tipo de las variables</H2>
     <?php
       $cadena="Hola a todos";
       echo 'La variable $cadena contiene <B>'.$cadena.'</B><BR><BR>';
       echo 'La variable $cadena ';
       echo (is_integer($cadena))?'es':' no es';
       echo ' de tipo entero <BR>';
       echo 'La variable $cadena ';
       echo (is_float($cadena))?'es':' no es';
       echo ' de tipo real <BR>';
       echo 'La variable $cadena ';
       echo (is_string($cadena))?'es':' no es';
       echo ' una cadena de caracteres <BR>';
     ?>
   </CENTER>
 </BODY>
</HTML>
```

NOTA: En este ejemplo se hace uso del operador condicional que se explica posteriormente en el apartado de operadores.

El resultado de visualizar el ejemplo anterior se muestra en la siguiente imagen:

- `intval(variable,base)`, `floatval(variable)`, `strval(variable)`: Estas funciones sirven para realizar conversiones de tipos (*casting*), de modo que convierten a `integer`, `float` o `string`, respectivamente, el valor de la variable que se le pasa como parámetro. Estas funciones no pueden utilizarse para convertir *arrays* u objetos. En particular, la función `intval()` puede recibir además un segundo parámetro que representa la base a utilizar en la conversión (10-decimal, 8-octal y 16-hexadecimal), tomando por defecto la base 10 de los números decimales.

El siguiente ejemplo muestra la utilización de estas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Casting de tipos</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Conversión de variables</H2>
     <?php
       $cadena=" 38E6";
       echo "El valor de \$cadena es: <B> $cadena </B><BR><BR>";
       $aux=intval($cadena);
       echo "El resultado de convertirla en entero es: $aux <BR>";
       $aux=intval($cadena,8);
       echo "El resultado de convertirla en entero octal: $aux <BR>";
       $aux=intval($cadena,16);
       echo "El resultado de convertirla en entero hexadecimal: $aux <BR>";
       $aux=floateval($cadena);
       echo "El resultado de convertirla en float es: $aux <BR>";
       $aux=strval($cadena);
       echo "El resultado de convertirla en cadena es: $aux <BR>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se puede ver en la siguiente imagen:

2.2.4 Constantes

Una constante es una variable que mantiene el mismo valor durante toda la ejecución del programa. Se puede asegurar que la constante mantiene siempre el mismo valor; en ninguna parte del *script* se puede cambiar el valor de una constante una vez que se define. De hecho, los intentos de cambio provocan errores.

2.2.4.1 Funciones para constantes

Para trabajar con constantes, PHP nos proporciona las siguientes funciones:

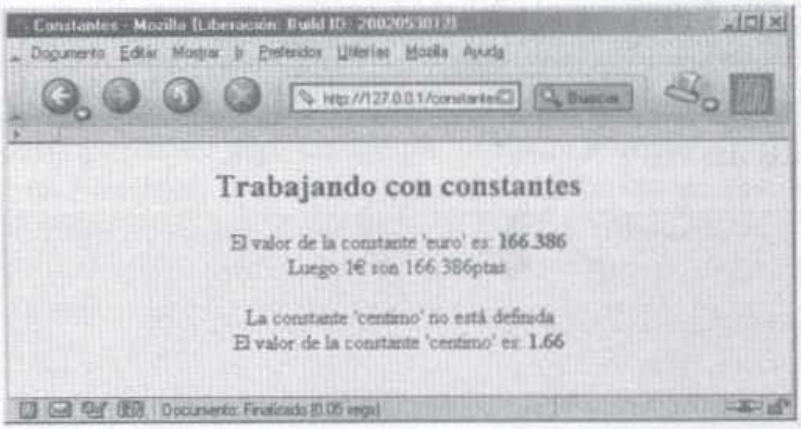
- `define(constante, valor)`: Esta función nos permite crear una constante asignándole su nombre y su valor a través de los parámetros que recibe.

- `defined(constante)`: Esta función devuelve `true` si la constante pasada como argumento está definida y, por tanto, existe; en caso contrario, devuelve `false`.

Para hacer referencia a las constantes, no es necesario anteponer el carácter `$` al nombre, ya que no se trata de una variable. Bastará con hacer referencia a ellas con el nombre con el que se definieron. En caso de necesitar redefinir el valor de una variable, habrá que volver a utilizar la función `define()`. Veamos un ejemplo del uso de las variables:

```
<HTML>
 <HEAD>
   <TITLE>Constantes</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con constantes</H2>
     <?php
       define("euro",166.386);
       echo "El valor de la constante 'euro' es: <B>". euro ."</B><BR>";
       echo "Luego 1€ son ".euro."ptas.<BR><BR>";
       echo "La constante 'centimo' ";
       echo (defined("centimo"))?"está".centimo:"no está";
       echo " definida<BR>";
       define("centimo",1.66);
       echo "El valor de la constante 'centimo' es: <B>";
       echo centimo ."</B><BR><BR>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen nos muestra el resultado:



2.2.4.2 Constantes predefinidas

El propio intérprete de PHP tiene un conjunto de constantes predefinidas y siempre disponibles para el programador. No se podrán definir constantes propias de nuestros programas con los mismos nombres ya que se tomará por defecto la constante predefinida. Algunas de estas constantes son:

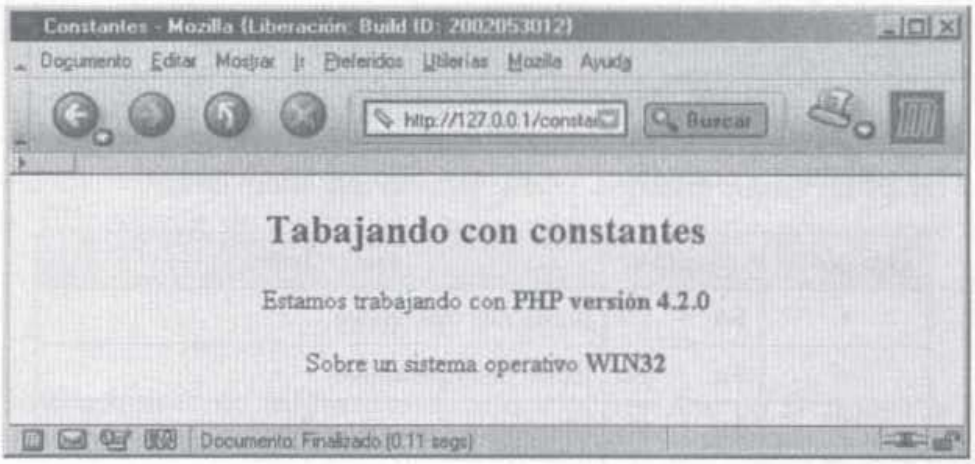
Constante	Significado
PHP_VERSION	Cadena que representa la versión del intérprete de PHP en uso.
PHP_OS	Cadena con el nombre del sistema operativo en el que se está ejecutando el intérprete de PHP.

Constante	Significado
TRUE	Verdadero.
FALSE	Falso.
E_ERROR	Información sobre errores distintos a los de interpretación del cual no es posible recuperarse.
E_PARSE	Informa que el intérprete encontró una sintaxis inválida en el archivo de comandos. Finaliza la ejecución.
E_NOTICE	Informa que se produjo algo incorrecto que puede provenir o no de un error. La ejecución continúa.
E_WARNING	Denota un error que no impide que continúe la ejecución.
E_ALL	Conjunto con todos los errores que se han producido.

EL siguiente ejemplo muestra la utilización de dos de estas constantes predefinidas del PHP:

```
<HTML>
 <HEAD>
   <TITLE>Constantes</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Tabajando con constantes</H2>
     <?php
       echo "Estamos trabajando con <B>PHP versión ";
       echo PHP_VERSION."</B><BR><BR>";
       echo "Sobre un sistema operativo <B>".PHP_OS."</B>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen nos muestra el resultado:



2.2.5 Expresiones

Las expresiones son la base principal de PHP, que es, en sí, un lenguaje orientado a expresiones, ya que casi todo en él es una expresión. La forma más ajustada de definir qué es una expresión es: "cualquier cosa que tiene o produce un valor".

Una expresión puede ser algo tan simple como un número o una variable, o puede incluir muchas variables, constantes, operadores y funciones conjuntamente. Las expresiones como grupo, si son correctas, son evaluadas a un valor simple. Este valor resultante debe ser clasificado dentro de uno de los tipos de datos que maneja PHP (`integer`, `float`, `string`, `boolean`, `array`, `object`).

Las expresiones más básicas son las variables y las constantes. Otro tipo de expresiones lo forman las expresiones de comparación que se evalúan a 0 ó 1 correspondiendo con el valor `false` o `true`, respectivamente. También tenemos las expresiones que surgen como resultado de la combinación de operador y operandos. Así, podríamos continuar enumerando todos los posibles tipos de expresiones.

2.2.6 Operadores

Los operadores en PHP son muy parecidos a los de otros lenguajes como C, Java y JavaScript. Se utilizan para determinar un valor, o bien, para obtener un valor final a partir de uno o más operandos. Podemos encontrar varios tipos de operadores, que se clasifican según su uso en los grupos que veremos a continuación.

2.2.6.1 Operadores Aritméticos

Estos operadores funcionan igual que en la aritmética básica y se pueden aplicar a las variables y constantes numéricas. Son los siguientes:

Operador	Ejemplo	Descripción
+	$a + $b	Suma dos operandos
-	$a - $b	Resta dos operandos
*	$a * $b	Multiplica dos operandos
/	$a / $b	Divide dos operandos
%	$a % $b	Resto de la división entera

Veamos un ejemplo que utiliza algunos de estos operadores:

```
<HTML>
 <HEAD>
   <TITLE>Operadores</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Operadores aritméticos</H2>
     <?php
       define("euro",166.386);
       echo "El valor de la constante 'euro' es: <B>". euro
."pts.</B><BR><BR>";
       echo "1000€ son ". (euro*1000) ."pts<BR>";
       echo "1000pts. son ". intval((1000/euro)*100)/100 ."€<BR><BR>";
       echo "157€ son ". intval(157/50) ." billetes de 50€";
       echo " y ". (157%50) ."€";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado obtenido es el siguiente:

2.2.6.2 Operadores de Asignación

El operador de asignación más utilizado es "="; su función básica es asignar un valor a una variable, para que de este modo se pueda conservar dicho valor en memoria. El operador de asignación es un operador binario que siempre toma la forma:

```
variable = expresión
```

Este operador de asignación hace que el variable de la izquierda tome el valor de la expresión de la derecha. Este operador lo hemos utilizado básicamente hasta ahora para asignar valores a las variables con las que hemos trabajado. PHP soporta otros operadores de asignación, que realmente son una combinación del operador de asignación básico con operadores aritméticos y con el operador de concatenación de cadenas. La siguiente tabla resume los operadores de asignación:

Operador	Ejemplo	Descripción
=	$a = $b	$a toma el valor de $b
+=	$a += $b	Equivale a $a = $a + $b
-=	$a -= $b	Equivale a $a = $a - $b
*=	$a *= $b	Equivale a $a = $a * $b
/=	$a /= $b	Equivale a $a = $a / $b
%=	$a %= $b	Equivale a $a = $a % $b
.=	$a .= $b	Equivale a $a = $a . $b

El siguiente ejemplo muestra la utilización de algunos de estos operadores aplicados a un ejemplo anterior:

```
<HTML>
 <HEAD>
   <TITLE>Operadores</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Operadores de asignación</H2>
     <?php
       define("euro",166.386);
       // en la variable $texto vamos a concatenar todo el texto a mostrar
       // por pantalla
       $texto = "El valor de la constante 'euro' es: <B>";
       $texto .= euro;
       $texto .= "pts.</B><BR><BR>";
       $texto .= "1000€ son ";
```

```
        $valor = euro*1000;       //calculamos el valor de 1000€
        $texto .= $valor;
        $texto .= "pts<BR>";
        $texto .= "1000pts. son ";
        $valor =1000/euro;        //obtenemos el valor de 1000pts en €
        $valor *= 100;          //lo multiplicamos por 100
        $valor = intval($valor); //eliminamos los decimales que no queremos
        $valor /= 100;     //lo dividimos por 100 para obtener el valor final
        $texto .= $valor;
        $texto .= "€<BR><BR>";
        $texto .= "157€ son ";
        $valor = intval(157/50);
        $texto .= $valor ." billetes de 50€ y ". (157%50) ."€";
        echo $texto;
    ?>
  </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en la siguiente imagen, el resultado obtenido es exactamente igual que el del ejemplo original:

2.2.6.3 Operadores de cadenas

A lo largo de los ejemplos anteriores hemos utilizado el *operador de concatenación* de cadenas representado por un punto (" . "); como hemos comprobado, devuelve como resultado la concatenación de sus operandos izquierdos y derechos; también hemos visto el *operador de concatenación y asignación*, representado por un punto seguido del signo de igualdad (" .="). Como su funcionalidad es la vista hasta el momento, no es necesario profundizar más en estos operadores.

2.2.6.4 Operadores de Incremento y Decremento

Al igual que otros lenguajes de programación como C, PHP soporta operadores específicos para incrementar y decrementar el valor de las variables.

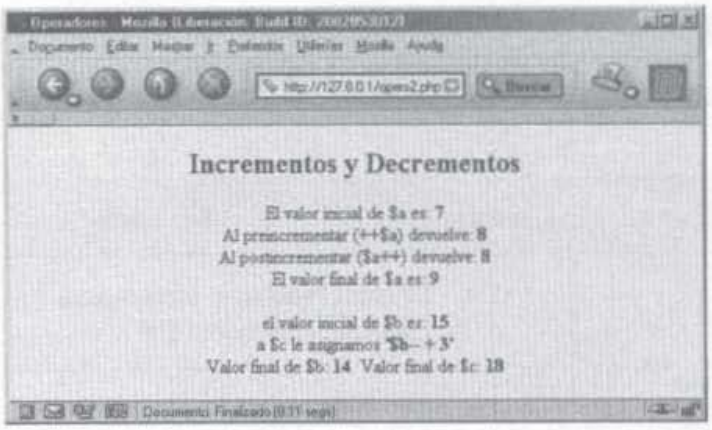
Además, el momento de su ejecución dependerá de la posición en la que aparezcan respecto de la variable a la que son aplicados. La siguiente tabla muestra las posibilidades de estos operadores:

Operador	Ejemplo	Descripción
++	++$a	**Preincremento**: Incrementa $a en uno y después devuelve $a
	$a++	**Postincremento**: Devuelve $a y después incrementa en uno $a
--	--$a	**Predecremento**: Decrementa $a en uno y después devuelve $a
	$a--	**Posdecremento**: Devuelve $a y después decrementa en uno $a

Por ejemplo:

```
<HTML>
<HEAD>
   <TITLE>Operadores</TITLE>
</HEAD>
<BODY>
<CENTER>
<H2>Incrementos y Decrementos</H2>
<?php
  $a=7;
  echo "El valor inicial de \$a es: <B>$a</B><BR>";
  echo "  Al preincrementar (++\$a) devuelve: <B>".++$a."</B><BR>";
  echo "  Al postincrementar (\$a++) devuelve: <B>".$a++."</B><BR>";
  echo "El valor final de \$a es: <B>$a</B><BR>";

  $b=15;
  echo "<br>el valor inicial de \$b es: <b>$b</b><br>";
  $c=$b-- + 3;
  echo "a \$c le asignamos <b>'\$b-- + 3'</b><br>";
  echo "Valor final de \$b: <b>$b</b>. Valor final de \$c: <b>$c</b><br>";
?>
</CENTER></BODY></HTML>
```

El resultado correspondiente se visualiza en la siguiente imagen:



Cabe destacar cómo los operadores de postincremento y posdecremento devuelven el mismo valor que reciben y posteriormente lo modifican.

2.2.6.5 Operadores de Comparación

Los operadores de comparación se utilizan básicamente para comparar expresiones. Las expresiones que utilizan operadores de comparación habitualmente realizan preguntas sobre los dos valores contenidos en los operandos. La respuesta a dicha pregunta puede ser `true` o `false`.

Dos símbolos de igual conforman el operador de igualdad ("=="). Cuando se utiliza el operador de igualdad entre dos operandos, lo que se pretende determinar es si los valores contenidos en los operandos son iguales. Un error habitual suele ser confundir el operador de igualdad con el operador de asignación. La lista completa de operadores de comparación se muestra en la siguiente tabla:

Oper.	Ejemplo	Devuelve `true` cuando
`==`	`$a == $b`	Los operandos son iguales.
`!=`	`$a != $b`	Los operandos son distintos.
`===`	`$a === $b`	Los operandos son idénticos: iguales y del mismo tipo.
`!==`	`$a !== $b`	Los operandos no son iguales o del mismo tipo.
`<`	`$a < $b`	El operando de la izquierda es menor que el operando de la derecha.
`>`	`$a > $b`	El operando de la izquierda es mayor que el operando de la derecha.
`<=`	`$a <= $b`	El operando de la izquierda es menor o igual que el operando de la derecha.
`>=`	`$a >= $b`	El operando de la izquierda es mayor o igual que el operando de la derecha.

Los operadores de comparación se utilizan en PHP principalmente en la toma de decisiones.

2.2.6.6 Operadores a nivel de bit

Todos los datos, en el nivel más bajo, se almacenan en memoria como bits, es decir, se almacenan utilizando el sistema de numeración binario, el cual representa cualquier valor como una ristra de ceros y unos. Dependiendo de su colocación, un bit puesto a 1 representa un valor igual a 2 elevado a *n*, siendo *n* la posición de dicho bit dentro del número comenzando por la derecha que es la posición 0.

Por ejemplo: el entero `10` se representa con el número binario `1010` y necesita un mínimo de 4 bits para almacenarse en memoria. El valor decimal del número `1010` binario se calcula del siguiente modo:

```
1 x 23 + 0 x 22 + 1 x 21 + 0 x 20 = 10
```

PHP utiliza 32 bits para almacenar en memoria los valores enteros. Por tanto, una vez en memoria el número `10`, se almacenaría de la siguiente forma: `00000000000000000000000000001010`. Lo habitual es escribirlo como `1010` eliminando los ceros de la izquierda que no son significativos. Se pueden introducir los valores enteros como números decimales, octales o hexadecimales y PHP los convertirá en binarios cuando los almacene. A pesar de ello, PHP por defecto siempre muestra la representación decimal de los números almacenados en memoria.

Por esto, PHP nos permite acceder a la representación binaria de los valores numéricos a través de los operadores de bit. La siguiente tabla nos muestra dichos operadores:

Oper.	Ejemplo	Descripción
&	`$a & $b`	**Y:** Activa sólo los bits que están activos simultáneamente, tanto para `$a` como para `$b`.
|	`$a | $b`	**O:** Activa los bits que están activos en `$a`, tanto como o en `$b`.
^	`$a ^ $b`	**Xor:** Activa los bits que están activos en `$a` o en `$b`, pero no en ambos a la vez.
~	`~ $a`	**CA1:** Devuelve el complemento a uno del operando.
<<	`$a << $b`	Desplaza los bits de `$a`, `$b` posiciones a la izda.
>>	`$a >> $b`	Desplaza los bits de `$a`, `$b` posiciones a la dcha.

El siguiente ejemplo muestra algunos de estos operadores en acción:

```
<HTML>
 <HEAD>
   <TITLE>Operadores</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Operadores de Bits</H2>
     <?php
       $a=3;  // 0011 en binario
       $b=9;  // 1001 en binario
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center">
         <TD BGCOLOR="yellow"> </TD>
         <TD BGCOLOR="yellow">2<SUP>3</SUP></TD>
         <TD BGCOLOR="yellow">2<SUP>2</SUP></TD>
         <TD BGCOLOR="yellow">2<SUP>1</SUP></TD>
         <TD BGCOLOR="yellow">2<SUP>0</SUP></TD>
         <TD BGCOLOR="yellow">valor decimal</TD>
       </TR>
       <TR ALIGN="center">
         <TD BGCOLOR="yellow">$a</TD>
           <?php
            echo "<TD>".(($a&8)>>3)."</TD>";
            echo "<TD>".(($a&4)>>2)."</TD>";
            echo "<TD>".(($a&2)>>1)."</TD>";
            echo "<TD>".($a&1)."</TD>";
            echo "<TD>$a</TD>";
           ?>
         </TD>
       </TR>
       <TR ALIGN="center">
         <TD BGCOLOR="yellow">$b</TD>
           <?php
            echo "<TD>".(($b&8)>>3)."</TD>";
            echo "<TD>".(($b&4)>>2)."</TD>";
            echo "<TD>".(($b&2)>>1)."</TD>";
            echo "<TD>".($b&1)."</TD>";
            echo "<TD>$b</TD>";
           ?>
        </TR>
       <TR ALIGN="center">
         <TD BGCOLOR="yellow">$a & $b</TD>
           <?php
             $aux=$a&$b;
            echo "<TD>".(($aux&8)>>3)."</TD>";
```

```
            echo "<TD>".(($aux&4)>>2)."</TD>";
            echo "<TD>".(($aux&2)>>1)."</TD>";
            echo "<TD>".($aux&1)."</TD>";
            echo "<TD>$aux</TD>";
           ?>
        </TR>
       <TR ALIGN="center">
          <TD BGCOLOR="yellow">$a | $b</TD>
           <?php
             $aux=$a|$b;
            echo "<TD>".(($aux&8)>>3)."</TD>";
            echo "<TD>".(($aux&4)>>2)."</TD>";
            echo "<TD>".(($aux&2)>>1)."</TD>";
            echo "<TD>".($aux&1)."</TD>";
            echo "<TD>$aux</TD>";
           ?>
        </TR>
       <TR ALIGN="center">
          <TD BGCOLOR="yellow">$a ^ $b</TD>
           <?php
             $aux=$a^$b;
            echo "<TD>".(($aux&8)>>3)."</TD>";
            echo "<TD>".(($aux&4)>>2)."</TD>";
            echo "<TD>".(($aux&2)>>1)."</TD>";
            echo "<TD>".($aux&1)."</TD>";
            echo "<TD>$aux</TD>";
           ?>
        </TR>
       <TR ALIGN="center">
          <TD BGCOLOR="yellow">~$a</TD>
           <?php
             $aux=~$a;
            echo "<TD>".(($aux&8)>>3)."</TD>";
            echo "<TD>".(($aux&4)>>2)."</TD>";
            echo "<TD>".(($aux&2)>>1)."</TD>";
            echo "<TD>".($aux&1)."</TD>";
            echo "<TD>$aux</TD>";
           ?>
        </TR>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

Y el resultado obtenido es:

2.2.6.7 Operadores lógicos

Los operadores lógicos (también llamados operadores *booleanos*) se utilizan conjuntamente con expresiones que devuelven valores lógicos. Con estos operadores se pueden combinar varias condiciones y evaluarlas en una sola expresión. La sintaxis de estos operadores es la siguiente:

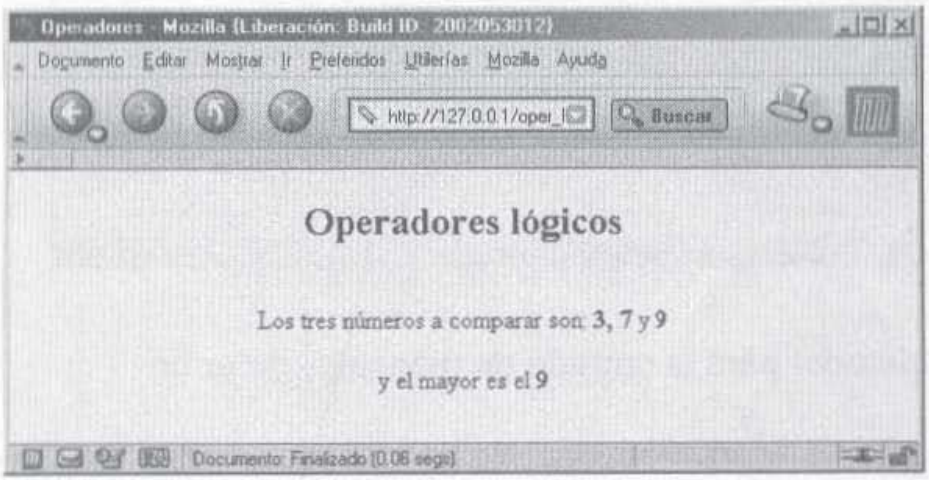
Oper.	Ejemplo	Devuelve TRUE cuando
&&	`$a && $b`	`$a` y `$b` son ambos `true`.
and	`$a and $b`	
||	`$a || $b`	`$a` o `$b` son `true`.
or	`$a or $b`	
!	`! $a`	`$a` es `false`, niega el valor lógico de la variable.
xor	`$a xor $b`	`$a` es `true` o `$b` es `true`, pero no lo son los dos a la vez.

Veamos el ejemplo que mostraba el mayor de tres valores haciendo uso de este tipo de operadores:

```
<HTML>
 <HEAD>
   <TITLE>Operadores</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Operadores lógicos</H2>
     <?php
       $a=3;
       $b=7;
       $c=9;
       echo "<BR>Los tres números a comparar son: ";
       echo "<B>$a, $b </B>y<B> $c</B><BR><BR>";
```

```
        echo " y el mayor es el <B>";
        echo ($a>$b)&&($a>$c)?$a:"";
        echo ($b>$a)&&($b>$c)?$b:"";
        echo ($c>$a)&&($c>$b)?$c:"";
        echo "</B>";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en la siguiente imagen, el resultado coincide con el ejemplo base utilizado:



2.2.6.8 Operador de ejecución

PHP ofrece el operador de ejecución representado por el apóstrofo invertido (“`”). Ante este operador, PHP toma la cadena que hay dentro de los dos apóstrofos invertidos como un comando o programa externo y lanza su ejecución.

Un ejemplo de este operador sería el siguiente:

```
<HTML>
 <HEAD>
   <TITLE>Operadores</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Operador de ejecución</H2>
</CENTER>
     <?php
        $entorno = `set`;
        echo "<pre>$entorno</pre>";
     ?>
</BODY>
</HTML>
```

El resultado se puede observar en la siguiente imagen:

2.2.6.9 Operador para la omisión de mensajes de error

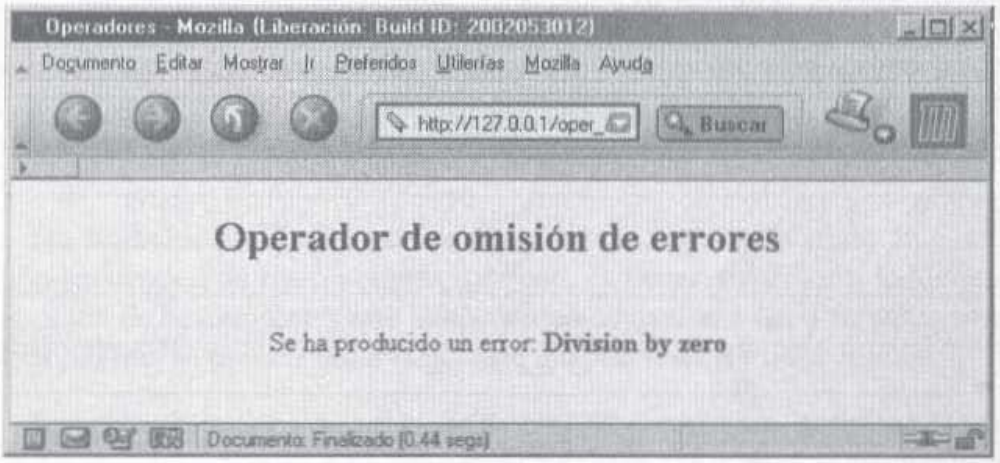
Se trata de un operador especialmente útil pero, a veces, poco utilizado. Sirve para omitir los mensajes de error que pueden surgir al utilizar una instrucción. Se representa con el carácter "@". Hay que tener cuidado especial con su uso, pues evita la aparición de los mensajes de error, pero no las acciones que se puedan haber producido a consecuencia del error, es decir, podía darse el caso de que el mensaje que avisa de la producción de un error crítico no se visualizara, con lo que la ejecución del *script* cesaría sin que el usuario hubiera tenido ningún tipo de notificación.

Normalmente, su uso está asociado a la utilización de la directiva `track_errors` del fichero de configuración `php.ini` que nos permite obtener el mensaje de error generado en la variable `$php_errormsg`. El siguiente ejemplo muestra su utilización conjunta para controlar la generación de los errores a través del *script*:

```
<HTML>
 <HEAD>
   <TITLE>Operadores</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Operador de omisión de errores</H2>
     <?php
       $a = 10;
```

```
        $b = 0;
        // la división por 0 genera un error con este operador
        // la ejecución continúa sin problemas
        @$resultado=$a/$b;
        $texto1="El resultado de $a/$b es: $resultado<BR>";
        $texto2="Se ha producido un error:<B> $php_errormsg </B>";
        // comprobamos si se ha producido un error
        // y si es así mostramos el mensaje de error
        echo "<BR>";
        echo (empty($resultado))? $texto2: $texto1;
    ?>
  </CENTER>
</BODY></HTML>
```

La imagen siguiente muestra el resultado de interpretar el *script*:



2.2.6.10 Precedencia de los operadores

Cuando una expresión está formada por más de un operador del mismo tipo, PHP la evalúa de izquierda a derecha, pero, cuando creamos expresiones que utilizan más de un operador diferente, hay que tener cuidado porque PHP no siempre evalúa estas expresiones de la misma forma. Cada parte de una expresión se evalúa en un orden basado en la precedencia predefinida para cada operador. Por tanto, *la precedencia de operadores* especifica cómo se agrupan las expresiones para ser evaluadas. Siempre podemos modificar la precedencia relativa de los operadores encerrando entre paréntesis las expresiones que queremos que se evalúen de una forma diferenciada.

Veámoslo con un ejemplo sencillo: ante la expresión `$x = 7 + 5 * 3`, podríamos pensar que, después de evaluarla, en `$x` obtendríamos el valor `36`; sin embargo, cuando PHP interpreta la expresión, da mayor importancia a la operación de multiplicación `5 * 3` respecto a la de la suma `7 + 5`. Es, por tanto, el resultado de la multiplicación lo que se suma con el primer operando `7 + 15`, almacenando el resultado final `22` en `$x`. Si se quería evaluar primero la suma `7 + 5` para después

multiplicar este resultado intermedio por 3, dando como resultado 36, se tendría que haber utilizado los paréntesis para modificar la precedencia: `x = (7 + 5) * 3`.

La siguiente tabla muestra los operadores en orden de menor a mayor precedencia:

Operador
or
xor
and
= += -= *= /= .= %= |= ^= ~= <<= >>=
? :
||
&&
|
^
&
== != ===
< <= > >=
<< >>
* / %
+ - .
! ~ ++ -- (int) (double) (string) (array) (objet) @

CAPÍTULO 3

ESTRUCTURAS DE CONTROL

No todos los problemas que se nos plantean tienen una solución basada en la ejecución secuencial de instrucciones; por eso, es necesario dotar a los lenguajes de programación de herramientas que los permitan adaptarse a las diferentes situaciones o condiciones que se pueden dar a la hora de intentar resolver un problema.

Las estructuras de control o sentencias de control nos permiten modificar el flujo de ejecución básico del programa, es decir, gracias a ellas la ejecución no tiene por qué ser totalmente secuencial, vamos a poder controlar el flujo lógico de cualquier programa. Estas estructuras nos permiten bifurcar el flujo del programa y así ejecutar unas partes u otras de código según se cumplan una serie de condiciones, hacer que un determinado código no llegue a ejecutarse nunca o que lo haga tantas veces como queramos.

A continuación enumeramos las distintas estructuras de control que nos podemos encontrar en un programa PHP. Son comunes en cuanto a concepto en la mayoría de los lenguajes de programación de alto nivel y casi idénticas a las que presentan lenguajes como C, C++, Java y también Perl o Bash.

3.1 SENTENCIAS CONDICIONALES

Son las estructuras de control más sencillas, se basan en el uso de la sentencia `if...else` y en las diferentes formas que ésta puede presentar. Utilizando estas sentencias, somos capaces de hacer que el programa elija entre dos caminos de ejecución diferentes en función de la evaluación de una expresión lógica.

3.1.1 if

Es una de las más utilizadas e importantes en la mayoría de los lenguajes de programación. Su sintaxis es la siguiente:

```
if (condicion) {
        [sentencias]
}
```

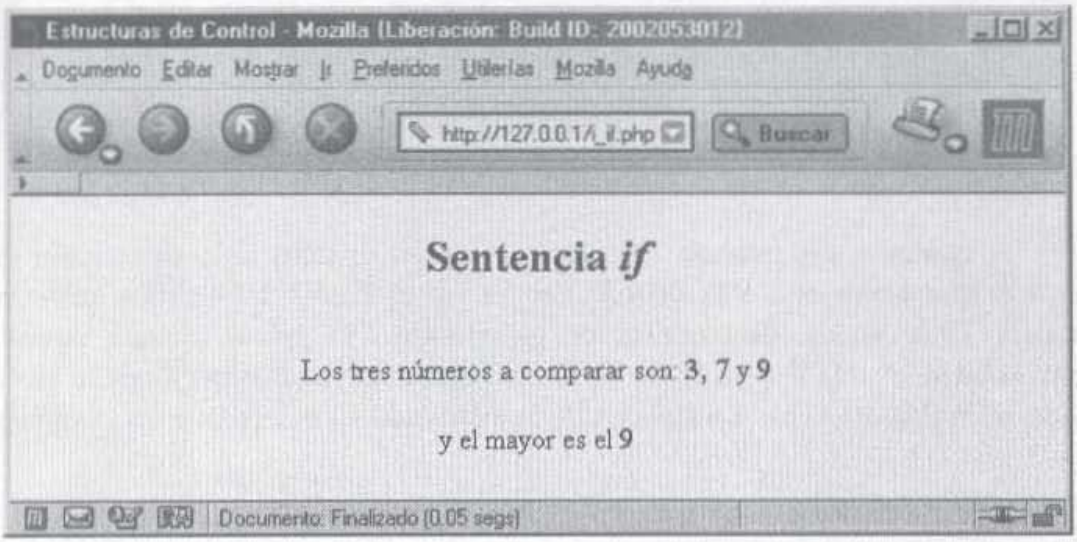
El intérprete de PHP lo que hace es evaluar la condicion, que debe ser una expresión lógica y, si resulta verdadera, se ejecutarán las sentencias comprendidas entre las llaves "{" y "}" y, si es falsa, PHP ignorará las sentencias y seguirá con la ejecución normal del programa. es decir, nos permite tomar decisiones en torno a una condición.

> **NOTA:** En caso de que dentro del cuerpo de la sentencia if sólo haya una sentencia se podrá prescindir del uso de las llaves "{" y "}". Aun así, es recomendable ponerlas siempre.

En el siguiente ejemplo podemos observar la utilización de la sentencia if para saber cuál es el mayor de un conjunto de tres valores. Para descubrir dicho valor, utilizamos tres sentencias if, cada una de las cuales, a través de una expresión lógica, pregunta si un valor determinado es mayor que los otros dos:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>if</I></H2>
     <?php
        $a=3;
        $b=7;
        $c=9;
        echo "<BR>Los tres números a comparar son: ";
        echo "<B>$a, $b </B>y<B> $c</B><BR><BR>";
        echo " y el mayor es el <B>";
        if (($a>$b)&&($a>$c)){
           echo $a;
        }
        if (($b>$a)&&($b>$c)){
           echo $b;
        }
        if (($c>$a)&&($c>$b)){
           echo $c;
        }
        echo "</B>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Como podemos observar, la *expresión* a evaluar puede ser tan compleja como se necesite. En este ejemplo el resultado obtenido es el siguiente:



Las sentencias `if` se pueden anidar, es decir, podemos poner dentro de un bloque `if` otras sentencias `if`, lo cual proporciona una flexibilidad completa para ejecuciones condicionales. Haciendo uso de `if`*s* sanidados, el siguiente código da el mismo resultado que el mostrado en el ejemplo anterior:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>if</I></H2>
     <?php
        $a=3;
        $b=7;
        $c=9;
        echo "<BR>Los tres números a comparar son: ";
        echo "<B>$a, $b </B>y<B> $c</B><BR><BR>";
        echo " y el mayor es el <B>";
        if ($a>$b){
           if($a>$c){
              echo $a;
           }
        }
        if ($b>$a){
           if($b>$c){
              echo $b;
           }
        }
        if ($c>$a){
           if($c>$b){
              echo $c;
           }
        }
```

```
        echo "</B>";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

3.1.2 `if...else`

A menudo nos interesa ejecutar un código distinto si la evaluación de la expresión que acompaña a la instrucción `if` no es cierta. Utilizamos entonces la sentencia `if...else`; ésta consta de un bloque `if` que se ejecuta cuando la `expresion` se evalúa a `true` y de un bloque `else` cuyas instrucciones se ejecutan cuando se evalúa a `false`. La sintaxis de la instrucción `if...else` es la siguiente:

```
if (expresion) {
    sentencias
} else {
    sentencias
}
```

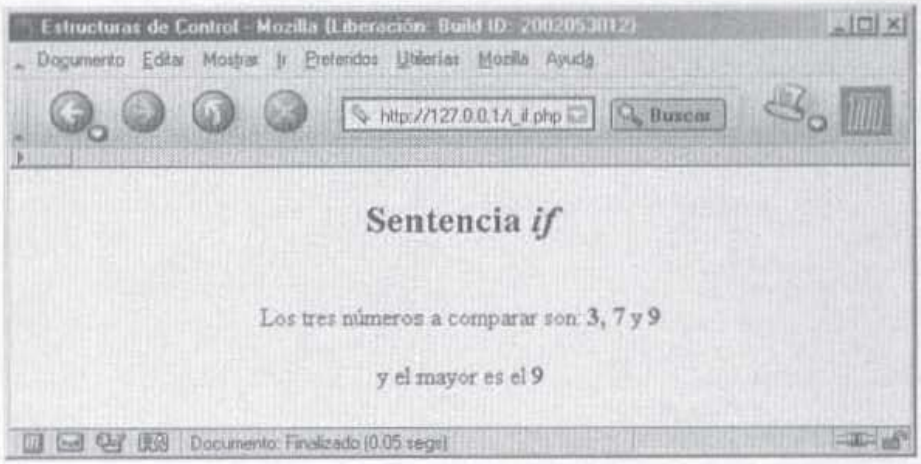
Siguiendo con el ejemplo anterior, pero haciendo uso de la sentencia `if...else`, tendríamos esta posible solución:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>if</I></H2>
     <?php
       $a=3;
       $b=7;
       $c=9;
       echo "<BR>Los tres números a comparar son: ";
       echo "<B>$a, $b </B>y<B> $c</B><BR><BR>";
       if ($a>$b){
           $elmayor=$a;
       }
       else{
           $elmayor=$b;
       }
       if($elmayor<$c){
          $elmayor=$c;
       }
       echo " y el mayor es el <B>"$elmayor</B>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Otra posible solución viene dada por el siguiente código:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>if</I></H2>
     <?php
       $a=3;
       $b=7;
       $c=9;
       echo "<BR>Los tres números a comparar son: ";
       echo "<B>$a, $b </B>y<B> $c</B><BR><BR>";
        if ($a>$b){
            if($a>$c){
               $elmayor=$a;
            }
            else {
               $elmayor=$c;
            }
        }
        else{
            if($b>$c){
               $elmayor=$b;
            }
            else {
               $elmayor=$c;
            }
        }
       echo " y el mayor es el <B>$elmayor</B>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Como se puede observar en la siguiente imagen, el resultado de la ejecución de todos los ejemplos anteriores es totalmente equivalente:



3.1.3 if...elseif

Hay muchas ocasiones en que se quiere evaluar una nueva comprobación utilizando una sentencia `if` dentro del cuerpo de una sentencia `else`; para estos casos se puede utilizar la sentencia `elseif` que nos permite combinar ambas sentencias en una sola. La sintaxis de la sentencia `if...elseif` es la siguiente:

```
if (expresión) {
   sentencias
} elseif (expresión) {
          sentencias
           }
 [elseif (expresión) {
          sentencias
           }]...
 [else {
          sentencias
           }]
```

La sentencia `elseif` se ejecuta si, y sólo si, se evalúa a `true` y la expresión `if` precedente y cualquier expresión `elseif` precedente se evalúan a `false`. Su comportamiento, por lo demás, coincide con el de las sentencias `if` anteriores.

El siguiente ejemplo nos muestra un posible uso de esta estructura aplicado al ejemplo anterior:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>if</I></H2>
     <?php
       $a=3;
       $b=7;
       $c=9;
       echo "<BR>Los tres números a comparar son: ";
       echo "<B>$a, $b </B>y<B> $c</B><BR><BR>";
       if ($a>$b){
           if($a>$c){
              $elmayor=$a;
           }
           else {
              $elmayor=$c;
           }
        }
        elseif($b>$c){
              $elmayor=$b;
```

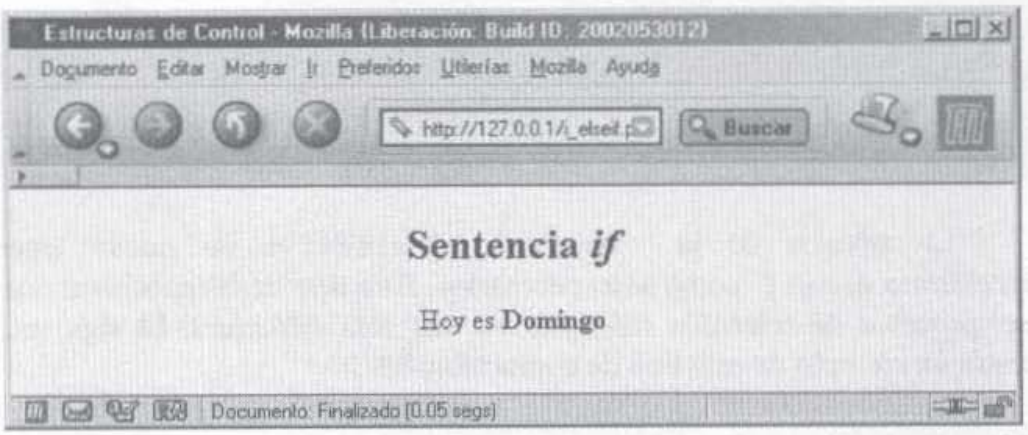
```
            }
            else {
               $elmayor=$c;
            }
        echo " y el mayor es el <B>$elmayor</B>";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

La sintaxis de la sentencia `if-elseif-else` puede tener tantos componentes `elseif` como sean necesarios. Este tipo de composición nos permite hacer procesos de selección múltiples en una sola estructura. El siguiente código muestra un ejemplo de este tipo de composiciones[1]:

```
<HTML>
 <HEAD>
  <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
  <CENTER>
   <H2>Sentencia <I>if</I></H2>
   <?php
      echo "Hoy es <B>";
      $dia = date("D");

      if ($dia == "Mon") {
          echo "Lunes";
      } elseif ($dia == "Tue") {
          echo "Martes";
      } elseif ($dia == "Wed") {
          echo "Miercoles";
      } elseif ($dia == "Thu") {
          echo "Jueves";
      } elseif ($dia == "Fri") {
          echo "Viernes";
      } elseif ($dia == "Sat") {
          echo "Sabado";
      } else {
          echo "Domingo";
      }       echo "</B>";
   ?>
  </CENTER>
 </BODY>
</HTML>
```

[1] Este ejemplo hace uso de la función `date` llamada con el argumento "`D`" que nos devuelve el día de la semana en curso pero con tres caracteres y en notación inglesa, es decir, nos devuelve: `Mon`, `Tue`, `Wed`, . . ., `Sun`.

El resultado obtenido es el siguiente:



PHP ofrece una sintaxis alternativa para algunas de sus estructuras de control. La forma básica de la sintaxis alternativa para la instrucción `if` es cambiar el carácter "`{`" por el carácter "`:`" y el carácter "`}`" por la palabra `endif`, con lo que quedaría de la siguiente forma:

```
if (expresión):
    sentencias
    ...
[elseif (expresión):
    sentencias]
    ...
[else:
    sentencias]
    ...
endif
```

3.1.4 Expresión condicional (`if` compacto)

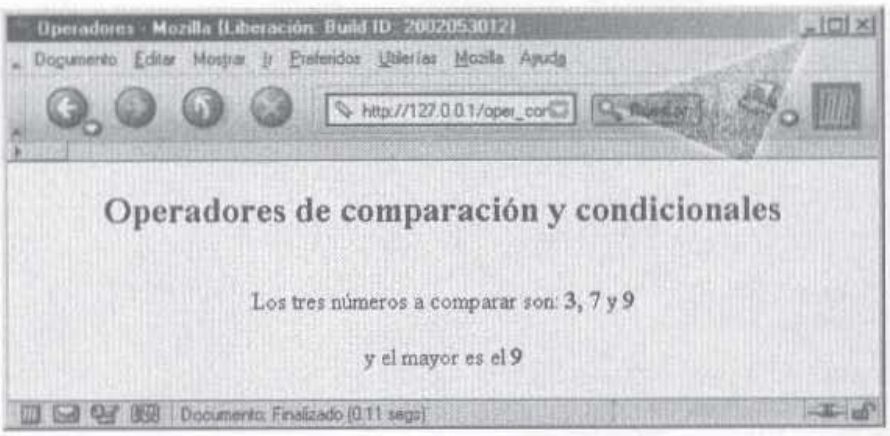
PHP utiliza dos operadores ("?" y ":") para formar expresiones condicionales que devolverán uno de dos posibles valores basándose en el valor lógico obtenido al evaluar una expresión. La sintaxis de estos operadores es:

```
<expresion1> ? <expresion2> : <expresion3>;
```

De este modo, considerando `expresion1`, si ésta se evalúa a `true`, entonces se devolvería la evaluación de `expresion2`; en caso contrario, el valor devuelto sería el resultado de evaluar la `expresion3`. El siguiente ejemplo nos muestra su uso combinado con operadores de comparación:

```
<HTML>
 <HEAD>
   <TITLE>Operadores</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Operadores de comparación y condicionales</H2>
     <?php
       $a=3;
       $b=7;
       $c=9;
       echo "<BR>Los tres números a comparar son: ";
       echo "<B>$a, $b </B>y<B> $c</B><BR><BR>";
       // obtenemos el mayor de $a y $b
       $elmayor=($a>$b)?$a:$b;
       echo " y el mayor es el <B>";
       // mostramos el mayor de los tres valores
       echo ($elmayor<$c)?$c:$elmayor;
       echo "</B>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado que muestra el navegador es el siguiente:



3.1.5 switch

La sentencia `switch` se utiliza para comparar un dato con un conjunto de posibles valores. Esta tarea se puede realizar utilizando

múltiples sentencias `if` o con una sentencia `if...elseif` múltiple, pero la sentencia `switch` es mucho más legible y nos permite especificar un conjunto de sentencias por defecto, en el caso de que el dato no tenga un valor con que compararlo (equiparable a la cláusula `else` de la sentencia `if`). La sintaxis de la sentencia `switch` es la siguiente:

```
switch ($variable) {
  case valor1:  [sentencias;]
                [break;]
  case valor2:  [sentencias;]
                [break;]
  case valorN:  [sentencias;]
                [break;]
 [ default:  sentencias;   ]
}
```

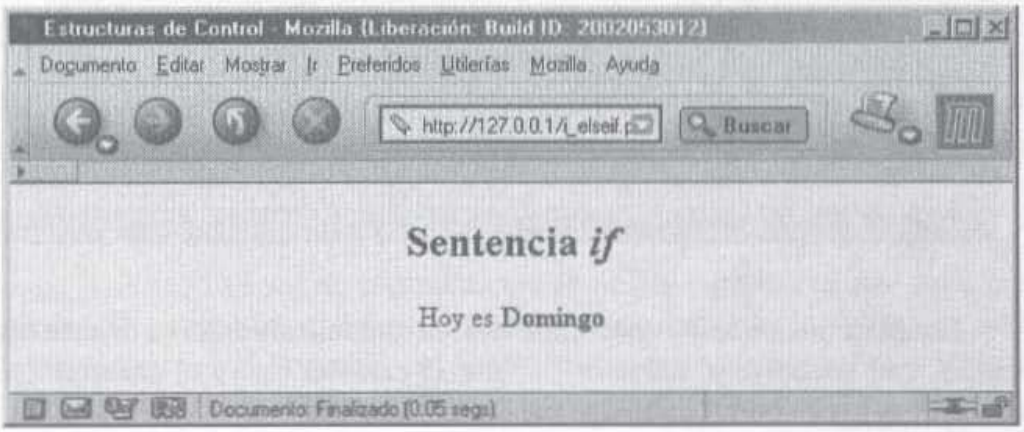
En el caso de que el valor de la variable no coincida con ninguna cláusula `case`, se ejecutará la opción `default`. Esta opción se puede omitir del `switch` y, entonces, no habría ninguna acción por defecto. La sentencia `break` dentro de cada una de las cláusulas `case` permite que, una vez encontrado el valor buscado, no se sigan realizando más comparaciones; fuerza la finalización de la sentencia `switch`. Si no se utiliza la sentencia `break` dentro de las cláusulas `case`, la ejecución de una cláusula continúa por la siguiente.

El siguiente ejemplo nos muestra la utilización de esta cláusula para resolver el ejemplo anterior:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>if</I></H2>
     <?php
        echo "Hoy es <B>";
        $dia = date("D");
        switch($dia){
          case "Mon": $texto="Lunes";
                      break;
          case "Tue": echo "Martes";
                      break;
          case "Wed": echo "Miércoles";
                      break;
          case "Thu": echo "Jueves";
                      break;
          case "Fri": echo "Viernes";
                      break;
```

```
        case "Sat": echo "Sábado";
                    break;
        default:    echo "Domingo";
      }
      echo "</B>";
   ?>
  </CENTER>
 </BODY>
</HTML>
```

Como podemos observar, el resultado es idéntico al del ejemplo visto con la sentencia `if`:



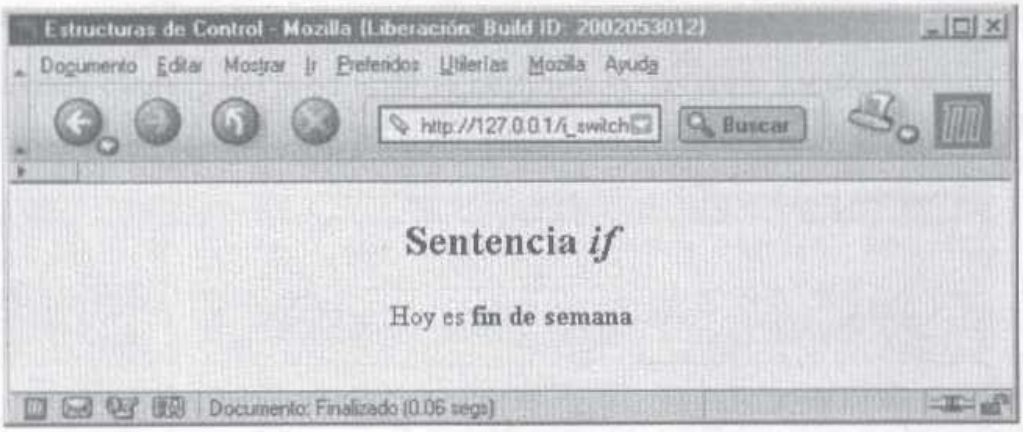
Existen ocasiones en que es interesante prescindir de la utilización de la sentencia `break` dentro de las cláusulas `case`. También es habitual que alguna de las cláusulas `case` no tengan asociada ninguna sentencia, ocasionando que el control pase a la cláusula `case` inmediatamente posterior.

El siguiente ejemplo nos muestra la utilización de ambas situaciones:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>if</I></H2>
     <?php
        echo "Hoy es <B>";
        $dia = date("D");
        switch($dia){
          case "Mon":
          case "Tue":
          case "Wed":
          case "Thu":
          case "Fri": echo "un día de diario";
                      break;
             default: echo "fin de semana";
```

```
        }
        echo "</B>";
    ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado que obtenemos se muestra en la siguiente imagen:



También nos podemos encontrar con la sintaxis alternativa de esta estructura de control, que sustituye el carácter "{" por el carácter ":" y el carácter "}" por la palabra `endswitch`, como podemos ver a continuación:

```
switch ($variable):
   case valor1: [sentencias;]
                [break;]
   case valor2: [sentencias;]
                [break;]

   case valorN: [sentencias;]
                [break;]
   [default: sentencias;]
endswitch;
```

3.2 SENTENCIAS DE BUCLES

La utilización de bucles dentro de un *script* sirve para muchos propósitos. Un uso sencillo, pero habitual, es utilizar los bucles para contar. También se utilizan para recorrer objetos o estructuras compuestas por más de un elemento, como ocurre con las estructuras de tipo *array*.

3.2.1 `for`

Esta instrucción nos permite realizar un conjunto de instrucciones un determinado número de veces. Es una de las estructuras de control sintácticamente más complejas de PHP.

La sintaxis de esta estructura de control es la siguiente:

```
for ([exp_inicialización];[exp_condición];[exp_bucle]){
     [sentencias]
}
```

Las tres expresiones cerradas entre paréntesis son opcionales, pero es necesario escribir los caracteres punto y coma (";") que las separan aunque las expresiones se omitan, para que cada expresión permanezca en el lugar apropiado. Vamos a detallar las opciones que nos permite esta sentencia:

- `exp_inicialización` normalmente se utiliza para inicializar y declarar la variable o variables que se van utilizar como contadores del bucle; sólo se ejecuta una vez al principio del bucle. Si hay más de una variable, se separan por comas. Suele ser de la forma:

```
$variable = valor_inicial
```

- `exp_condición` define una o más condiciones que han de cumplirse (evaluarse a `true`) para poder ejecutar las sentencias encerradas entre las llaves. Mientras estas condiciones sean ciertas, se estarán ejecutando las sentencias delimitadas entre llaves. La expresión se evalúa antes de cada iteración y, si no se cumple la condición, ya no se ejecutan las sentencias. Es muy importante comprobar la corrección de esta expresión porque, si inicialmente no se cumple el cuerpo de la sentencia, no llegaría a ejecutarse nunca y si, al contrario, siempre se evalúa a `true`, estaríamos ante un caso de bucle infinito. Suele tener la forma:

```
$variable <= límite
```

- Finalmente, la expresión `exp_bucle` modifica el valor de la variable o variables (separadas por coma) utilizadas como contadores del bucle. Se ejecuta cada vez que finaliza una iteración. No sigue un patrón fijo; algunas de las formas que presenta son:

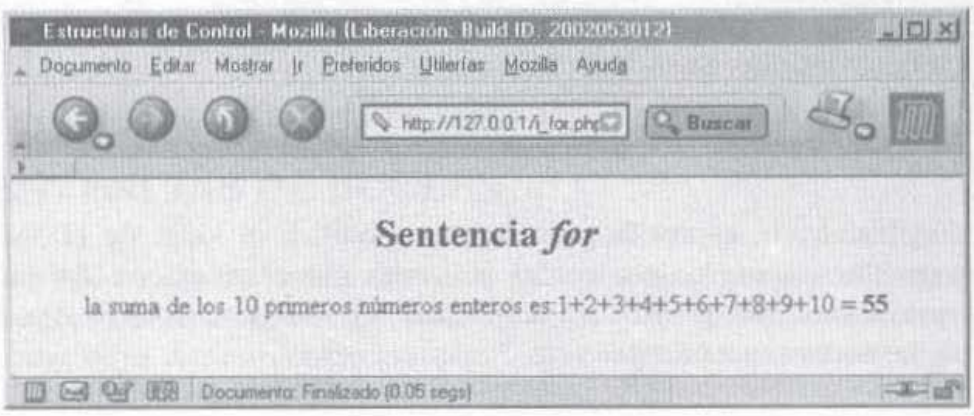
```
$variable++
$variable--
$variable+=valor ...
```

> **NOTA:** En caso de que dentro del cuerpo del bucle `for` sólo haya una sentencia, se podrá prescindir del uso de las llaves "{" y "}". Aun así, es recomendable ponerlas siempre.

El siguiente ejemplo nos muestra su funcionamiento para imprimir la suma de los 10 primeros números:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>for</I></H2>
     <?php
        echo "la suma de los 10 primeros números enteros es:";
        $suma=0;
        for($i=1;$i<=10;$i++){
           $suma+=$i;
           echo "$i";
           if ($i==10)
              echo " = ";
           else
              echo "+";
        }
        echo "<B>$suma</B>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado obtenido se visualiza en la siguiente imagen:

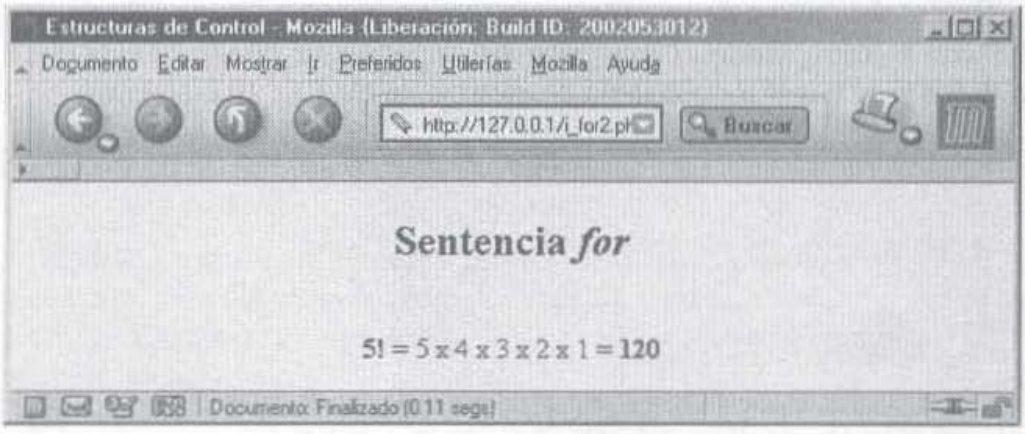


El ejemplo consta de un bucle `for` que se repite 10 veces, sumando en cada una de ellas el valor de la variable contador `$i` al contenido de la variable acumulador `$suma`. El operador condicional se utiliza para diferenciar el número `10` del resto a la hora de mostrarlo por pantalla.

De igual forma que el contador puede irse incrementando en cada iteración del bucle, también se puede ir decrementando. El siguiente ejemplo muestra este tipo de bucle para calcular el factorial de un número dado:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>for</I></H2>
     <?php
        $numero=5;
        echo "<BR><B>$numero!</B> = ";
        $factorial=1;
        for($i=$numero;$i>=1;$i--){
          $factorial*=$i;
          echo $i;
          echo ($i==1) ? " = " : " x " ;
        }
        echo "<B>$factorial</B>";
    ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



Como se dijo anteriormente en la definición del bucle `for`, no siempre son necesarias las tres expresiones presentes en su sintaxis; el siguiente ejemplo nos lo muestra basándose en el código anterior:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>for</I></H2>
     <?php
        $numero=5;
        echo "<BR><B>$numero!</B> = ";
        $factorial=1;
        for($continuar=($numero>1);$continuar;){
          echo $numero;
          $factorial*=$numero;
         $numero--;
          if($numero==0){
             echo " = ";
             $continuar=false;
          }
          else {
             echo " x ";
          }
        }
        echo "<B>$factorial</B>";
    ?>
   </CENTER>
 </BODY>
</HTML>
```

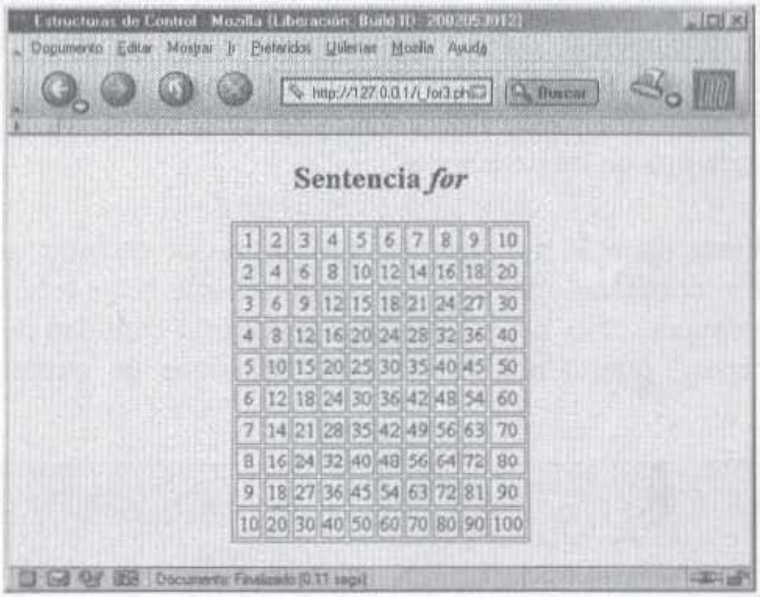
Como vemos, lo importante en los bucles es que dentro del bloque de sentencias haya, al menos, una que modifique algún componente de la expresión de evaluación pues, si no ocurriera así, nos encontraríamos con un bucle infinito.

Dentro del cuerpo de una sentencia `for` podemos utilizar cualquier otro tipo de sentencias e instrucciones. Como ejemplo, el siguiente código utiliza dos bucles `for` anidados para construir una tabla de multiplicar de 10 x 10 elementos:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>for</I></H2>
     <?php
       echo "<TABLE BORDER='1'>";
       for($fila=1;$fila<=10;$fila++){
```

```
            echo "<TR ALIGN='CENTER'>";
            for($col=1;$col<=10;$col++){
                echo "<TD>".($fila*$col)."</TR>";
            }
            echo "</TR>";
        }
        echo "</TABLE>";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



Finalmente, se debe comentar que también se puede encontrar la siguiente sintaxis alternativa para la instrucción `for`:

```
for (exp1; exp2; exp3):
    sentencias;
endfor;
```

3.2.2 `foreach`

Esta sentencia nos permite recorrer las estructuras de tipo *array* de una forma sencilla, obteniendo en cada paso de la iteración uno de sus elementos componentes. También se suele utilizar para *traducir* tablas de tipo *hash*, tal y como veremos más adelante.

> **NOTA:** PHP3 no tiene incluida esta sentencia, por lo que, para conseguir el mismo resultado, hay que combinar la estructura de control `while` con las funciones `reset`, `list` y `each`, tal y como veremos más adelante.

Tiene dos sintaxis. La primera de ellas es la siguiente:

```
foreach (nombre_array as $variable) {
        sentencias;
}
```

Lo que hace este bucle es recorrer cada uno de los elementos del *array* que tiene por nombre `nombre_array`, asignando en cada paso el valor del elemento actual del *array* al contenido de la variable `$variable`. El bucle hace uso de un puntero interno que apunta a la posición actual del *array* (comenzando por la primera y siguiendo en orden ascendente) y que va actualizando de forma automática en cada una de las iteraciones.

El siguiente ejemplo nos muestra dos formas de recorrer un *array*. En la primera se utiliza un bucle `for` y en la segunda un bucle `foreach`. Como podemos observar en el segundo caso, no es necesario conocer la cantidad de elementos que conforman el *array*, puesto que la sentencia mantiene un puntero interno para recorrerlo:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>foreach</I></H2>
     <?php
       // creamos el array y llenamos de datos
       $matriz[0]="cougar";
       $matriz[1]="ford";
       // posición sin contenido
       $matriz[2]=null;
       $matriz[3]="2.500";
       $matriz[4]="V6";
       $matriz[5]=182;
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
           <?php
             for($i=0;$i<6;$i++){
                 echo "<TD> $matriz[$i] </TD>";
             }
           ?>
       </TR>
       <TR ALIGN="center" BGCOLOR="yellow">
           <?php
             foreach($matriz as $valor){
```

```
                echo "<TD> $valor </TD>";
            }
        ?>
    </TR>
  </TABLE>
 </CENTER>
</BODY>
</HTML>
```

Como podemos observar en la siguiente imagen, ambas sentencias muestran de igual forma el contenido del *array*:

La segunda sintaxis que nos ofrece la sentencia `foreach` es la siguiente:

```
foreach (nombre_array as $clave => $valor){
    sentencias
}
```

Con esta nueva sintaxis podemos realizar la misma operación que con la anterior, pero, además, nos permite conocer en todo instante la posición exacta (el índice) del componente actual dentro de la estructura del *array*, a través de la variable `$clave`, es decir, se realiza el mismo procedimiento de asignación de valores a la variable `$valor` y, además, la clave (o índice) del elemento actual se asigna a la variable `$clave` en cada iteración. Con `foreach` no hace falta inicializar el puntero interno del *array* a la primera posición ya que se hace automáticamente.

Veamos un ejemplo haciendo uso de esta sintaxis:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>foreach</I></H2>
     <?php
       // creamos el array y llenamos de datos
       $matriz[0]="cougar";
       $matriz[1]="ford";
       $matriz[2]=null;
       $matriz[3]="2.500";
       $matriz[4]="V6";
       $matriz[6]=182;
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
         <TD>posición</TD>
         <TD>contenido</TD>
       </TR>
           <?php
             foreach($matriz as $clave => $valor){
                 echo "<TR ALIGN='center'>";
                 echo "<TD> $clave </TD>";
                 echo "<TD> $valor </TD>";
                 echo "</TR>";
             }
           ?>
       </TR>
     </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado de este ejemplo nos los muestra la siguiente imagen:

Como podemos observar, aunque una posición del *array* esté vacía (la posición 2), el recorrido no la salta, si bien una posición no utilizada (posición 5) no aparece en este tipo de recorridos.

Finalmente, se debe decir que esta sentencia se puede aplicar también a un *array* de tipo ***asociativo***, en el que el índice de cada elemento no es de tipo numérico. El siguiente ejemplo muestra este tipo de utilización:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>foreach</I></H2>
     <?php
       // creamos la matriz asociativa
       $matriz['modelo']="cougar";
       $matriz['marca']="ford";
       // posición sin contenido
       $matriz['fecha']=null;
       $matriz['cc']="2.500";
       $matriz['motor']="V6";
       $matriz['potencia']=182;
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
         <TD>clave</TD>
         <TD>contenido</TD>
       </TR>
           <?php
             foreach($matriz as $key => $valor){
                 echo "<TR ALIGN='center'>";
                 echo "<TD> $key </TD>";
                 echo "<TD> $valor </TD>";
                 echo "</TR>";
             }
           ?>
       </TR>
     </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:

3.2.3 while

La sentencia `while` actúa de forma muy parecida a la sentencia `for`, pero se diferencia de ésta en que no incluye en su declaración la inicialización de la variable de control del bucle ni su incremento o decremento. Por tanto, dicha variable se deberá declarar antes del bucle *while* y su incremento o decremento se deberá realizar dentro del cuerpo de dicho bucle. La sintaxis de la instrucción `while` es la siguiente:

```
while (condición){
   sentencias;
}
```

Con esta instrucción se va a poder ejecutar un conjunto de instrucciones un indeterminado número de veces, siempre y cuando el resultado de comprobar la `condición` sea verdadera (debe ser una expresión que se evalúe a un valor lógico). Si la `condición` se evalúa a `true`, se ejecutan las sentencias del cuerpo del bucle; después de ejecutarlas, se volverá a evaluar la `condición`, de forma que, si ésta sigue cumpliéndose, se volverán a ejecutar las sentencias. Esto se repite hasta que la `condición` se evalúa a `false`, en cuyo caso no se ejecutarán las sentencias del cuerpo del bucle y continuará la ejecución del *script* por la instrucción siguiente a la sentencia `while`.

> **NOTA:** En caso de que dentro del cuerpo del bucle `while`, sólo haya una sentencia, se podrá prescindir del uso de las llaves "{" y "}". Aun así, es recomendable ponerlas siempre.

Veamos la utilización de esta sentencia con el ejemplo del sumatorio de los 10 primeros números enteros:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>while</I></H2>
     <?php
        echo 'la suma de los 10 primeros números enteros es: ";
        $suma=0;
        $i=1;
        while($i<=10){
          $suma+=$i;
          echo ($i==10) ? "$i = " : "$i+" ;
          $i++;
        }
        echo "<B>$suma</B>";
    ?>
   </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en el código, la inicialización de la variable que se utiliza para controlar el final del bucle, `$i`, se realiza fuera de él, mientras que su actualización forma parte de las instrucciones del cuerpo de la sentencia `while`. Es muy importante comprobar la corrección de la `condición` porque, si inicialmente no se cumple, el cuerpo de la sentencia no llegaría a ejecutarse nunca y si, por el contrario, siempre se evalúa a `true`, estaríamos ante un caso de bucle infinito.

De igual forma que ocurría con el bucle `for`, dentro del cuerpo de un bucle `while` puede encontrarse cualquier otro tipo de instrucción. El siguiente ejemplo nos muestra el anidamiento de bucles `while` para resolver el problema de la tabla de multiplicación de 10 x 10 elementos:

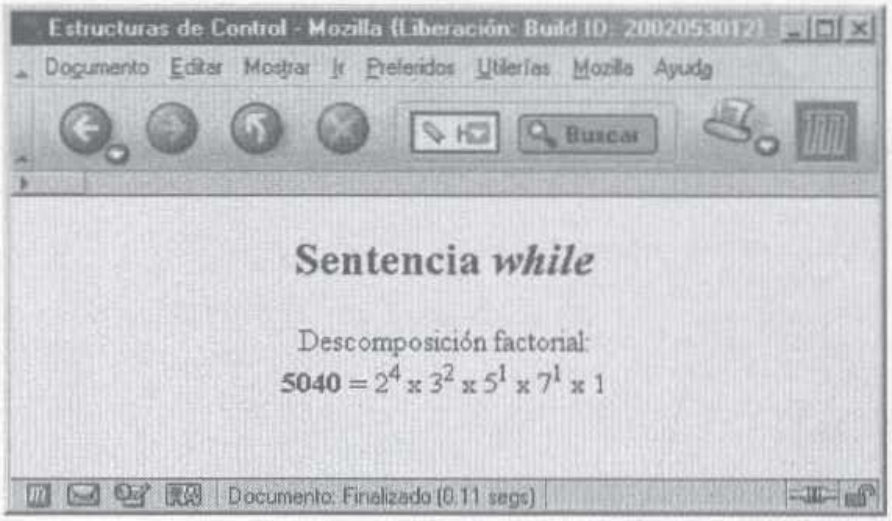
```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>while</I></H2>
     <?php
       echo "<TABLE BORDER='1'>";
       $fila=1;
       while($fila<=10){
          echo "<TR ALIGN='CENTER'>";
          $col=1;
          while($col<=10){
              echo "<TD>".($fila*$col)."</TD>";
              $col++;
          }
          echo "</TR>";
          $fila++;
       }
```

```
      echo "</TABLE>";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

No siempre la inicialización y actualización de las variables que controlan la finalización del bucle son tan claras como en los ejemplos anteriores. El siguiente código, que nos permite calcular la descomposición factorial de un número, contiene un bucle `while` en el que la actualización de la variable depende de varios factores:

```
<HTML>
 <HEAD>
  <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
  <CENTER>
   <H2>Sentencia <I>while</I></H2>
   <?php
     $numero=5040;
     // primer divisor a probar
     $divisor=2;
     // utilizamos cociente para poder modificar el numero inicial
     $cociente=$numero;
     // solo hayamos la descomposición para valores > 1
     $calcular=($cociente>1);
     $potencia=0;

     echo "Descomposición factorial:<BR>";
     echo "<B>$numero</B> = ";
     while($calcular){
        if($cociente%$divisor==0){
          $cociente/=$divisor;
          $potencia++;
        }
        else{
          // solo se muestran los factores válidos
          if ($potencia>0){
              echo "$divisor<SUP>$potencia</SUP> x ";
          }
          // si el cociente es 1 hemos acabado
          if ($cociente==1){
              $calcular=false;
          }
          $divisor++;
          $potencia=0;
        }
     }
     echo "1";
   ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



Finalmente, se debe comentar que también se puede encontrar la siguiente sintaxis alternativa para esta sentencia:

```
while (condición):
        sentencias;
endwhile;
```

3.2.4 do...while

Esta sentencia funciona exactamente igual que el bucle `while`, excepto que la condición no se comprueba hasta que se ha realizado una iteración (la `condición` se comprueba al final de cada iteración). Esto garantiza que, al menos, el cuerpo del bucle se realiza una vez, aunque la expresión `condición` se evalúe a `false`. Como siempre, `condición` debe ser una expresión que se evalúe a un valor lógico.

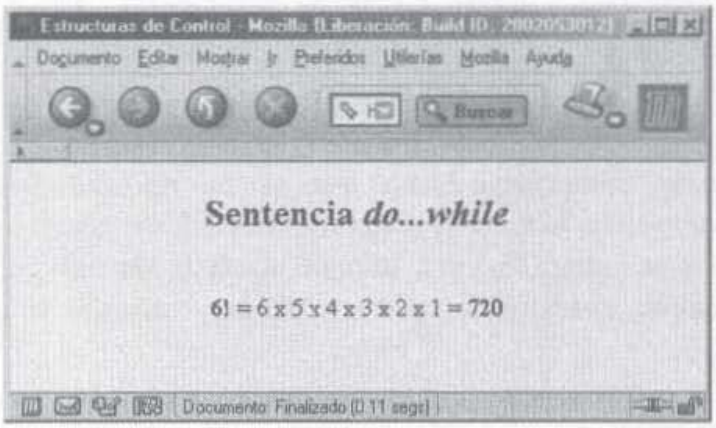
La sintaxis del bucle `do...while` es la siguiente:

```
do {
   sentencias;
} while (condición);
```

El siguiente ejemplo nos muestra un uso básico de este tipo de sentencias para calcular el factorial de un número dado:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>do...while</I></H2>
     <?php
        $numero=6;
        echo "<BR><B>$numero!</B> = ";
        $factorial=1;
        do{
          echo $numero." x ";
          $factorial*=$numero;
         $numero--;
        }while($numero>1);
        echo "1 = <B>$factorial</B>";
    ?>
   </CENTER>
 </BODY>
</HTML>
```

Como se observa en el código, la condición de salida del bucle se evalúa después de que se hayan ejecutado las instrucciones que forman el cuerpo del mismo al menos una vez. El resultado obtenido se muestra en la siguiente imagen:



3.2.5 `break` y `continue`

Cuando explicamos la sentencia `switch`, vimos que era habitual utilizar la instrucción `break` para interrumpir su ejecución una vez que encontrábamos la cláusula `case` que coincidía con el valor origen de la comparación. A veces, cuando utilizamos bucles, también se nos plantea la necesidad de finalizarlos antes de que sus condiciones no se cumplan; para ello, utilizaremos las sentencias `break`. Asimismo, es posible que nos interese que la ejecución del cuerpo del bucle no

contemple siempre las mismas instrucciones, es decir, que, al alcanzar una cierta posición dentro del cuerpo del bucle, las sentencias restantes no se ejecuten y que vuelva a evaluarse de nuevo la expresión del bucle (si existe) continuando con la siguiente iteración; para ello, utilizaremos la instrucción `continue`.

Las instrucciones `break` y `continue` se pueden utilizar dentro de los cuerpos de todas las sentencias de control de bucles, desde el `for` hasta el `do..while`.

El siguiente ejemplo nos muestra la utilización de la instrucción `break` para finalizar la ejecución de un bucle:

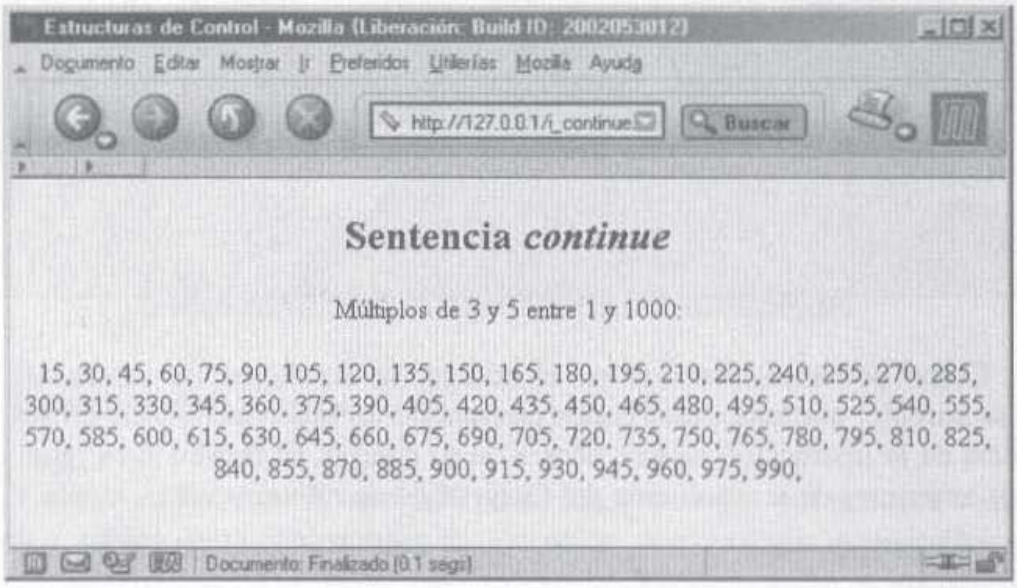
```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>break</I></H2>
     <?php
        $numero=6;
        echo "<BR><B>$numero!</B> = ";
        $factorial=1;
        while (1){
          echo $numero." x ";
          $factorial*=$numero;
          $numero--;
          if ($numero==1){
             break;
          }
        }
        echo "1 = <B>$factorial</B>";
    ?>
   </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en el código, la condición de salida del bucle nunca se cumple; por tanto, estaríamos ejecutando un bucle infinito de no ser por la existencia de la instrucción `break` dentro de su cuerpo. El resultado es igual que el de casos anteriores de la resolución del factorial de un número dado.

El siguiente ejemplo, que obtiene los números múltiplos de 3 y 5 comprendidos entre el 1 y el 1.000, nos muestra la utilización de la instrucción `continue` para saltar la ejecución de algunas instrucciones dentro de un bucle:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>continue</I></H2>
     <?php
        echo "Múltiplos de 3 y 5 entre 1 y 1000:<BR><BR>";
        for($i=1;$i<1000;$i++){
          if (($i%3 != 0)||($i%5 != 0)) {
             continue;
          }
          echo "$i, ";
        }
     ?>
   </CENTER>
 </BODY>
</HTML>
```

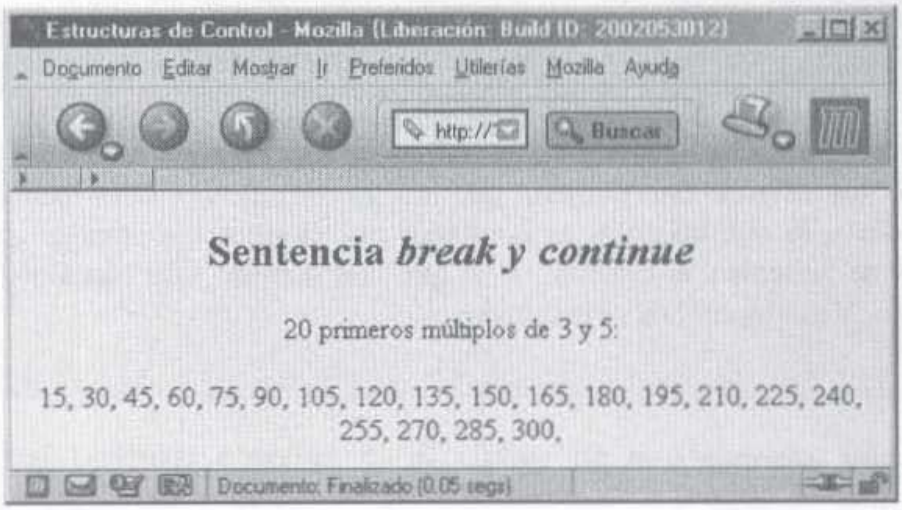
Como podemos observar en el código, sólo cuando encontramos un número divisible por 3 y por 5, éste se muestra por pantalla a través de la instrucción `echo`; en el resto de las ocasiones, la instrucción `continue` hace que la ejecución salte a la siguiente iteración del bucle sin tener en cuenta las instrucciones que aparecen por debajo de ella. El resultado se muestra en la imagen de la página siguiente.



El siguiente ejemplo muestra el uso combinado de ambas sentencias:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencia <I>break y continue</I></H2>
     <?php
        echo "20 primeros múltiplos de 3 y 5:<BR><BR>";
        $multiplos=0;
        $i=0;
        while(1){
          $i++;
          if ((($i%3 !== 0)||($i%5 !== 0)) {
             continue;
          }
          echo "$i, ";
          $multiplos++;
          if($multiplos>=20){
            break;
          }
        }
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Gracias a la sentencia `continue` podemos discriminar los números que son múltiplos de 3 y 5 de los que no lo son. La sentencia `break` finaliza el bucle una vez que hemos encontrado los 20 primeros. El resultado se muestra en la imagen de la página siguiente.

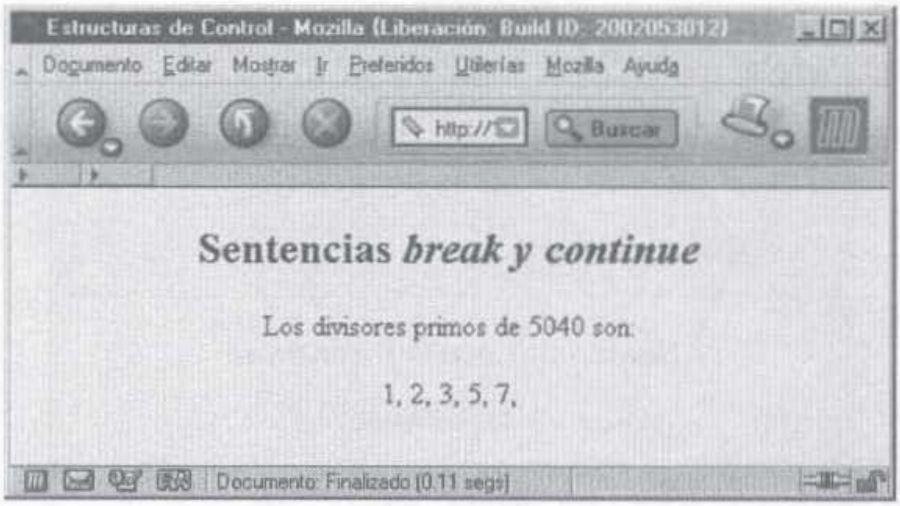


Finalmente, se debe comentar que ambas sentencias admiten un parámetro opcional con el que podemos indicar el número de estructuras de control que deben saltarse en caso de que en el código existan anidamientos de bucles. Esto nos permite decidir el nivel de ruptura que queremos dentro del bucle cuando se ejecuta alguna de estas dos sentencias. Las sentencias `break` y `continue`, por defecto, sólo saltan un nivel de anidamiento, el del bucle más interno que las contiene.

El siguiente ejemplo muestra el uso de estos parámetros opcionales en la sentencia `continue`:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencias <I>break y continue</I></H2>
     <?php
       $numero=5040;
       echo "Los divisores primos de $numero son:<BR><BR>";
       for($i=1;$i<=$numero;$i++){
          // comprobamos si es un divisor
          if($numero%$i==0){
             // comprobamos si es primo
             for($j=2;$j<$i;$j++){
               if($i%$j==0){
                 continue 2;
               }
             }
             //si llega hasta aquí es que es primo
             echo $i.", ";
          }
       }
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en el código de este ejemplo, una vez que hemos encontrado un divisor del número que no es primo, no tiene sentido seguir procesándolo y lo que hacemos es continuar por el siguiente número gracias a la utilización de la sentencia `continue 2` que nos permite salir hasta el bucle `for` más externo. Veamos cuál es su resultado:

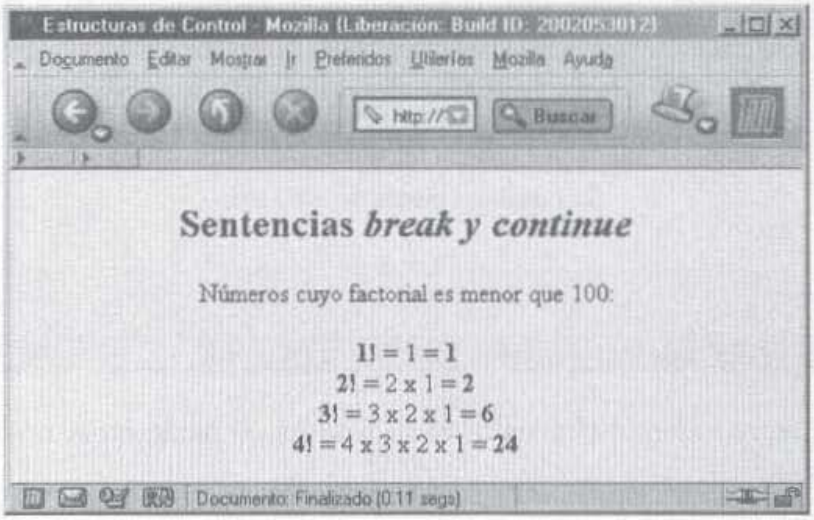


El siguiente ejemplo muestra el uso de estos parámetros opcionales en la sentencia `break`:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencias <I>break y continue</I></H2>
     <?php
       echo "Números cuyo factorial es menor que 100:<BR><BR>";
       // recorremos todos los números desde el 1
       $numero=1;
       while (1){
          $factorial=1;
          $texto = "<B>$numero!</B> = ";
          // obtenemos el factorial
          for($i=$numero;$i>1;$i--){
             $factorial*=$i;
             $texto.="$i x ";
             if($factorial>100){
                break 2;
             }
          }
          $texto.= "1 = <B>$factorial</B><BR>";
          $numero++;
          echo $texto;
       }
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en el código de este ejemplo, una vez que encontramos que el valor de la variable `$factorial` es mayor de `100`, debemos acabar el procesamiento de los dos bucles y, para eso, utilizamos la sentencia

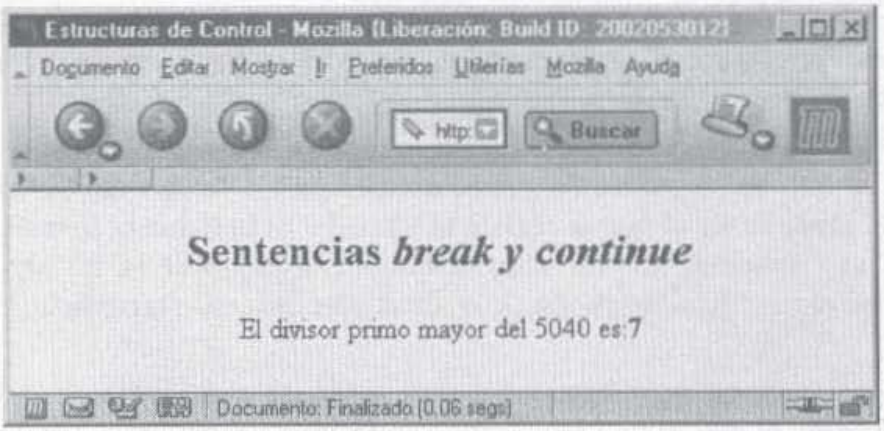
`break 2` que fuerza que finalice el bucle `while`. La siguiente imagen muestra el resultado del ejemplo:



Es muy habitual el uso combinado de ambas sentencias. El siguiente ejemplo nos muestra cómo una pequeña variación sobre el ejemplo que calculaba los divisores primos de un número dado nos permite encontrar el divisor primo mayor de un número:

```
<HTML>
 <HEAD>
   <TITLE>Estructuras de Control</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Sentencias <I>break y continue</I></H2>
     <?php
       $numero=5040;
       echo "El divisor primo mayor del $numero es: ";
       // recorremos todos los números desde el dado
       for($i=$numero;$i>=1;$i--){
          // comprobamos si es un divisor
          if($numero%$i==0){
             // comprobamos si es primo
             for($j=2;$j<$i;$j++){
               if($i%$j==0){
                  continue 2;
               }
             }
             //si llega hasta aquí es que es primo
             echo "<B>$i</B>";
             break;
          }
       }
     ?>
   </CENTER>
 </BODY>
</HTML>
```

La imagen siguiente nos muestra el resultado:



3.3 INCLUSIÓN DE FICHEROS

Las funciones que abordamos en esta sección deben incluirse dentro de las estructuras de control del lenguaje, puesto que pueden influir en gran manera en el flujo de ejecución e interpretación de los *scripts* que las contengan (sobre todo si los ficheros a incluir contienen a su vez código PHP). Deben ser consideradas más bien como construcciones del lenguaje, no como simples funciones.

Esta facilidad del lenguaje se utiliza principalmente para la definición de librerías comunes a varios *scripts*, permitiendo, de este modo, una reutilización y mantenimiento del código más óptimos. La naturaleza del fichero externo a incluir puede ser de cualquier tipo PHP, HTML, TXT, etc.

3.3.1 `include()`

Esta función incluye y evalúa un fichero externo cada vez que es interpretada. Ambos pasos, la inclusión del fichero externo y su posterior evaluación, se realizan cada vez que el flujo del programa llega a una línea que contenga la llamada a esta función. En caso de que el fichero a incluir no exista o su referencia sea errónea, la función genera un aviso o *warning*, continuando con la ejecución por la siguiente instrucción.

Cuando un archivo se incluye con `include()`, el intérprete sale del modo PHP y entra en modo HTML al principio del archivo referenciado, y vuelve de nuevo al modo PHP al acabar de leer dicho archivo. Por esta razón, cualquier código dentro del archivo referenciado que debiera ser ejecutado como código PHP debe ser encerrado dentro de etiquetas válidas de comienzo y fin de PHP.

Todas las funciones y variables definidas con anterioridad a la llamada a la función `include()` son accesibles para el código presente en el fichero importado. De igual forma, todos los elementos definidos en el código PHP del fichero incluido estarán disponibles para el *script* llamante una vez se haya terminado la lectura y ejecución del fichero incluido.

El ámbito de las variables que se definen en los *scripts* del fichero incluido depende del lugar desde el que se realiza la llamada. Si la llamada se realiza desde el cuerpo de una función, las variables tendrán ámbito local (a no ser que estén definidas con el modificador `global`); en cualquier otro caso tendrán ámbito global.

El siguiente ejemplo muestra la utilización de esta función. Consiste en un *script* PHP desde el que se incluyen otros tres ficheros; el primero, `cabecera.inc`, sólo contiene texto HTML:

```
<H3>cabecera</H3>
```

El segundo, `cuerpo.inc`, que sólo contiene código PHP, se encarga de calcular la exponenciación de dos valores que se le proporcionan:

```
<?php
  $resultado=1;
  for($i=1;$i<=abs($exponente);$i++){
    $resultado*=$base;
  }
  if($exponente<0){
    $resultado=1/$resultado;
  }
  echo "$base<SUP>$exponente</SUP> = ";
  echo "<B>$resultado</B>";
?>
```

Y el tercero, `pie.inc`, contiene texto HTML y código PHP:

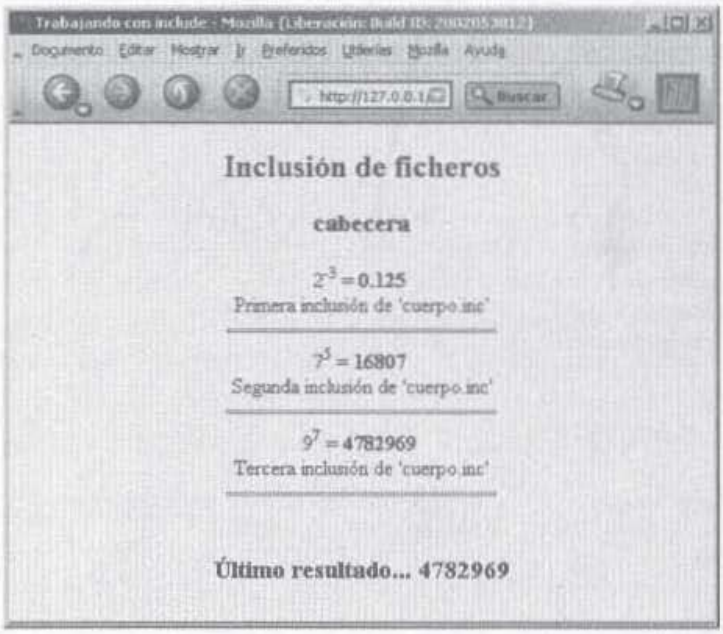
```
<BR><H3>Último resultado... <?=$resultado?></H3>
```

El *script* desde el que se realizan las llamadas a la función `include()` es el siguiente:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con include</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
      <H2>Inclusión de ficheros</H2>
      <?php
        // se incluye una cabecera html
        include("cabecera.inc");
```

```
        // el fichero "cuerpo.inc" hace uso
        // de las variables $base y $exponente
        $base=2;
        $exponente=-3;
        include("cuerpo.inc");
        echo "<br>Primera inclusión de 'cuerpo.inc'<hr Width='40%'>";
        $base=7;
        $exponente=5;
        include("cuerpo.inc");
        echo "<br>Segunda inclusión de 'cuerpo.inc'<hr Width='40%'>";
        $base=9;
        $exponente=7;
        include("cuerpo.inc");
        echo "<br>Tercera inclusión de 'cuerpo.inc'<hr Width='40%'>";

        // se incluye un fichero que hace uso
        // de variables definidas en 'cuerpo.inc'
        include("pie.inc")
      ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



Como podemos observar, las variables definidas en el cuerpo del fichero principal `$base` y `$exponente` son visibles dentro del *script* contenido en `cuerpo.inc`, que las utiliza para realizar los cálculos. De igual forma, la variable `$resultado` definida en este último *script* está disponible para ser utilizada en el fichero `pie.inc`.

La función `include()` puede utilizarse en combinación con otras estructuras de control de flujo. El contenido del fichero se incluirá y evaluará sólo

cuando se interprete la llamada. Si el flujo de ejecución del *script* nunca alcanza la línea en la que se realiza la llamada, el fichero nunca será incluido. Cuando esta función se utiliza dentro de estructuras de bucle o condicionales, siempre debe aparecer entre llaves, pues, en otro caso, podría obtenerse un resultado no deseado.

El siguiente ejemplo nos muestra esta situación. Para ello utilizamos dos ficheros `factorial.inc`, que contienen el código necesario para calcular el factorial de un número:

```
<?php
   if(!isset($numero))
     $numero=10;
   echo "<BR><B>$numero!</B> = ";
   $factorial=1;
   do{
     echo $numero." x ";
     $factorial*=$numero;
    $numero--;
   } while($numero>1);
   echo "1 = <B>$factorial</B>";
?>
```

Y `divisores.inc` calculará los divisores primos de un número:

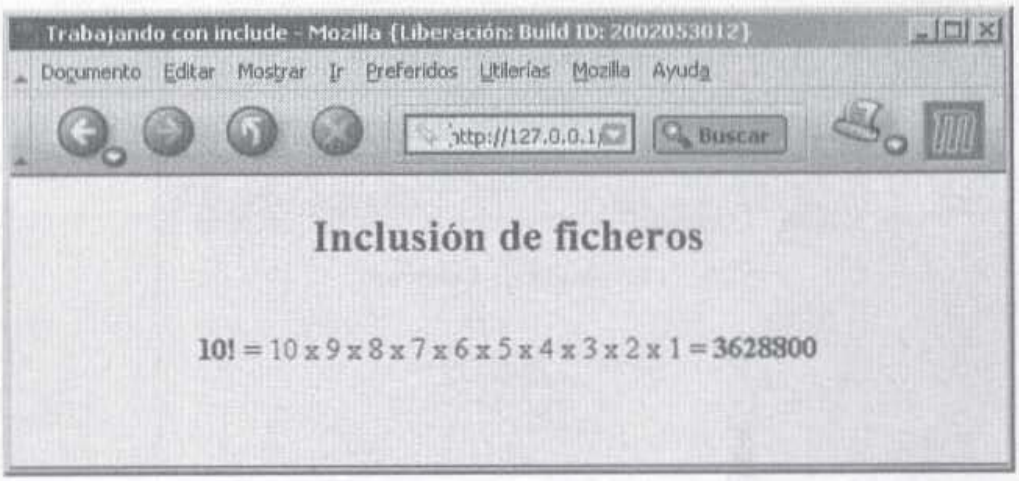
```
<?php
   if(!isset($numero))
     $numero=10;
  echo "Los divisores primos de $numero son:<BR><BR>";
  for($i=1;$i<=$numero;$i++){
     // comprobamos si es un divisor
     if($numero%$i==0){
        // comprobamos si es primo
        for($j=2;$j<$i;$j++){
          if($i%$j==0){
            continue 2;
          }
        }
        //si llega hasta aquí es que es primo
        echo $i.", ";
     }
  }
?>
```

El contenido del *script* principal es el siguiente:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con include</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
       <H2>Inclusión de ficheros</H2>
```

```
        <?php
          $operacion="factorial";
          if($operacion=="divisores_primos"){
            include("divisores.inc");
          }
          else {
            include("factorial.inc");
          }
        ?>
    </CENTER>
  </BODY>
</HTML>
```

Como podemos observar en la imagen siguiente, sólo se ha cargado e interpretado uno de los dos ficheros incluidos:



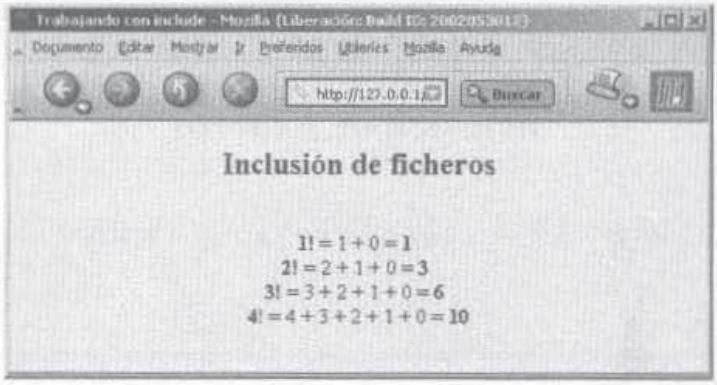
El siguiente ejemplo nos muestra otra posibilidad de uso combinado de la función `include()` con estructuras de control de bucles. Para ello, hace uso de cuatro ficheros de inclusión, denominados `fichero1.inc` a `fichero4.inc`, cuyo contenido es un *script* encargado de calcular el sumatorio de un número dado:

```
<?php
    if(!isset($numero))
      $numero=10;
    echo "<BR><B>$numero!</B> = ";
    $sumatorio=0;
    do{
      echo $numero." + ";
      $sumatorio+=$numero;
     $numero--;
    }while($numero>0);
    echo "0 = <B>$sumatorio</B>";
?>
```

Por su parte, el *script* principal define dinámicamente el nombre del fichero a incluir haciendo uso de un bucle:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con include</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
       <H2>Inclusión de ficheros</H2>
       <?php
         $fichero="";
         for($i=1;$i<5;$i++){
           $fichero="fichero".$i.".inc";
           $numero=$i;
           include($fichero);
         }
       ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



PHP permite que desde los ficheros incluidos se devuelva un valor; para ello utiliza el mismo mecanismo que la devolución de valores de una expresión (este mismo mecanismo será utilizado por las *funciones*, como se verá más profundamente en un capítulo posterior). Se puede devolver cualquier tipo de valores, incluyendo listas y objetos, pero sólo un único valor (para devolver múltiples valores, deberemos utilizar un *array*).

Para poder hacerlo, se utiliza la palabra reservada **`return`** acompañada de una expresión. En el instante en que aparece dentro del cuerpo de un *script* una sentencia con esta palabra reservada, el *script* deja de ejecutarse para devolver el flujo de ejecución al punto del programa principal donde se llamó a la función `include()`. Si después de `return` hay más líneas de código, dichas líneas no se procesarán nunca.

> **NOTA:** *En PHP 3 la sentencia `return` no puede aparecer dentro de un bloque de* sentencias a menos que éste sea un bloque de función.

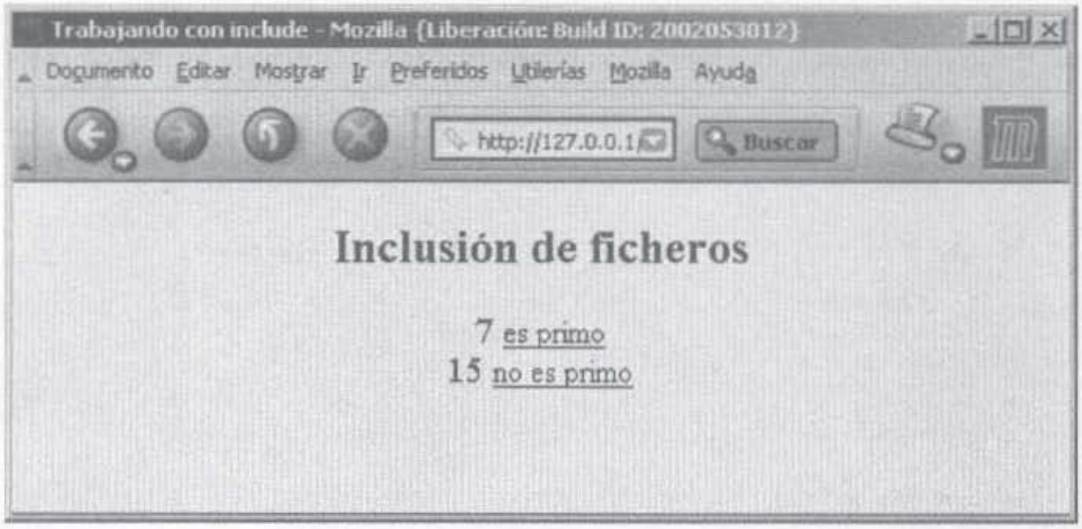
El siguiente ejemplo nos muestra la devolución de valores desde un fichero incluido y su recuperación en el *script* que realiza la llamada. El fichero a incluir `es_primo.inc`; se encarga de comprobar si un número dado es primo o no, devolviendo una variable *booleana* con el resultado. Su contenido es el siguiente:

```
<?php
  if(!isset($numero))
    $numero=10;

  $es_primo=true;
  for($i=2;$i<$numero;$i++){
    // comprobamos si tiene divisores
    // distintos de él mismo y la unidad
    if($numero%$i==0){
      $es_primo=false;
      break;
    }
  }
  return $es_primo;
?>
```

El código del *script* encargado de incluir el fichero es el siguiente:

```
<HTML>
 <HEAD>
  <TITLE>Trabajando con include</TITLE>
 </HEAD>
 <BODY>
  <CENTER>
      <H2>Inclusión de ficheros</H2>
      <?php
        $primo;
        $texto;

        // probamos si 7 es primo
        $numero=7;
        $texto = "<BIG>".$numero." </BIG><U>";
        $primo=include("es_primo.inc");
        if(!$primo) $texto .= "no ";
        $texto .= "es primo</U><BR>";
        echo $texto;

        // probamos si 15 es primo
        $numero=15;
        $texto = "<BIG>".$numero." </BIG><U>";
        $primo=include("es_primo.inc");
        if(!$primo) $texto .= "no ";
        $texto .= "es primo</U><BR>";
        echo $texto;
      ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



Como podemos observar, no es necesario que se conozcan las características de implementación de los *scripts* incluidos (nombres de la variables utilizadas); simplemente es necesario saber la naturaleza de los valores devueltos.

Por último, se debe comentar que el fichero pasado como parámetro a la función `include()` puede especificarse como una URL; incluso es posible incluir el resultado de la interpretación de un *script* en un servidor (dentro de la URL se pueden pasar variables al *script* remoto utilizando las mismas reglas que admite el método GET del protocolo HTTP).

> **NOTA:** Esta operación sólo está disponible en versiones superiores a PHP 4.0. Para poder utilizar esta funcionalidad, es necesario que la directiva `allow_url_fopen` del fichero de configuración `php.ini` esté activada.

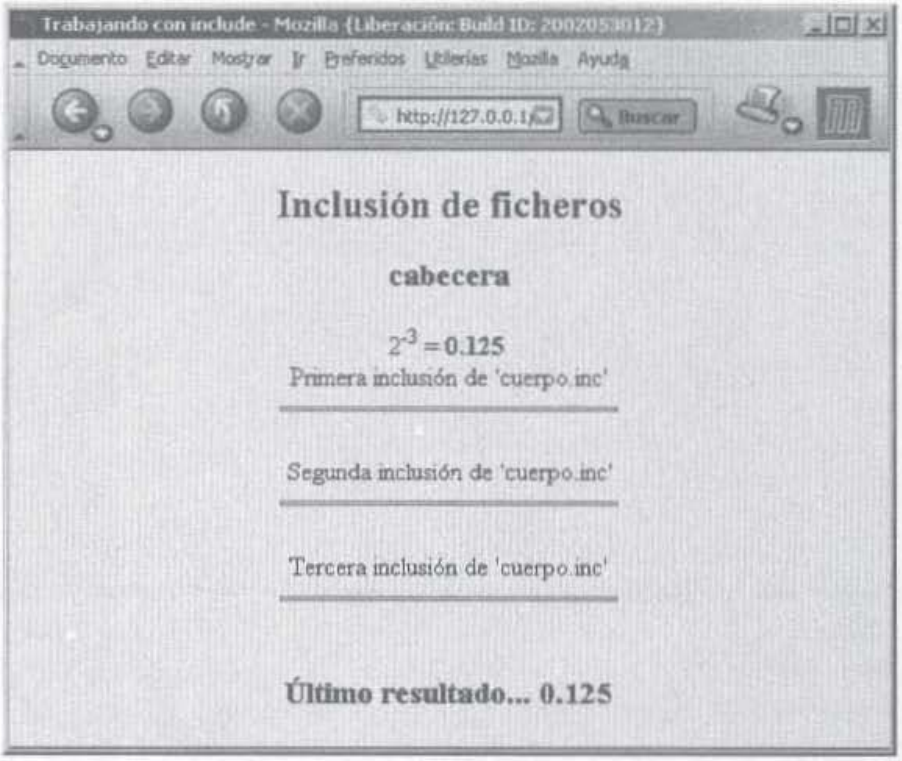
3.3.2 `include_once()`

Su funcionamiento es idéntico al de la función `include()`, con la única salvedad de que esta función sólo cargará y evaluará cada *script* una vez como máximo. Con esta función nos aseguramos de que un fichero sólo se ha cargado una vez a lo largo de la interpretación de nuestro *script*, evitando, de este modo, los errores producidos por la redefinición de funciones o la reasignación de valores a variables.

El siguiente ejemplo nos muestra su funcionamiento:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con include</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
        <H2>Inclusión de ficheros</H2>
        <?php
          // se incluye una cabecera html
          include_once("cabecera.inc");

          // el fichero "cuerpo.inc" hace uso
          // de las variables $base y $exponente
          $base=2;
          $exponente=-3;
          include_once("cuerpo.inc");
          echo "<br>Primera inclusión de 'cuerpo.inc'<hr Width='40%'>";
          $base=7;
          $exponente=5;
          include_once("cuerpo.inc");
          echo "<br>Segunda inclusión de 'cuerpo.inc'<hr Width='40%'>";
          $base=9;
          $exponente=7;
          include_once("cuerpo.inc");
          echo "<br>Tercera inclusión de 'cuerpo.inc'<hr Width='40%'>";

          // se incluye un fichero que hace uso
          // de variables definidas en 'cuerpo.inc'
          include_once("pie.inc")
        ?>
   </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en la siguiente imagen, el resultado es totalmente diferente al *script* original en el que hacíamos uso de la función `include()`:



NOTA: Esta función distingue entre mayúsculas y minúsculas; su funcionamiento puede no ser el esperado si el intérprete de PHP se ejecuta sobre un sistema operativo que no hace tales distinciones como Windows.

3.3.3 `require()`

Esta función se comporta en líneas generales como la función `include()`, pero con las siguientes salvedades:

1. `require()` sólo incluye el fichero referenciado, es decir, no lo interpreta. Su comportamiento es equivalente a la directiva `#include` del lenguaje C.

2. `require()` no puede ser utilizado con estructuras condicionales o de control de bucles porque el contenido del fichero referenciado se incluye antes de que se evalúe la sentencia que lo contiene y se incluye siempre aunque la condición que lo contiene no se cumpla. No obstante, si la línea en la que aparece `require()` no se ejecuta, tampoco se ejecutará el código del archivo al que esta llamada haga referencia.

3. En caso de que el fichero referenciado por `require()` no exista, se genera un *error fatal* que no nos permite continuar con la ejecución del *script*.

4. Se desaconseja pasar variables como parámetros de la función `require()`, si bien este tipo de construcción funciona.

3.3.4 `require_once()`

De igual forma que `include()` cuenta con la función `include_once()`, para evitar la carga en más de una ocasión de un fichero, `require()` hace lo propio con la función `require_once()`, la cual tiene un comportamiento equivalente.

CAPÍTULO 4

CADENAS

El tratamiento de cadenas es de gran importancia en el uso de PHP, puesto que, en la mayoría de los casos, el objetivo final del procesamiento de un fichero PHP está relacionado con la generación de documentos (casi siempre documentos HTML). Por ello, existe un amplio conjunto de funciones para el manejo de cadenas, de las que veremos las principales, agrupadas en torno al tipo de función que realizan.

4.1 DELIMITACIÓN DE CADENAS

Como vimos en el capítulo segundo, una cadena está formada por cero o más caracteres encerrados entre dobles comillas ("") o entre comillas simples (''). Es obligatorio utilizar siempre el mismo tipo de comilla para rodear cada cadena, aunque en algunos casos se puede entremezclar el uso de los dos tipos de entrecomillado, principalmente para insertar una cadena literal dentro de otra (aunque para este tipo de acciones también se utiliza el enmascarado [*escaped*] de las comillas, es decir, anteponer a las comillas el carácter "\").

Se debe recordar que, cuando utilizamos comillas dobles, podemos incluir dentro de la cadena nombres variables o caracteres especiales que serán evaluados (sustituidos por sus respectivos valores) a la hora de mostrar la información, cosa que no ocurre si introducimos nombres de variables dentro de una cadena encerrada entre comillas simples.

Veamos un ejemplo en el que veremos el comportamiento de PHP ante cadenas con comillas dobles o simples, y cómo insertar una comilla doble dentro de una cadena delimitada por comillas dobles:

```
<HTML>
<HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
</HEAD>
<BODY>
<CENTER>
<H2>Comillas dobles y simples</H2>
   <?php
      $cadena='un tesoro';
      echo "la cadena '$cadena' contiene: $cadena<br><br>";
      echo "la cadena \"$cadena\" contiene: $cadena<br><br>";
      echo 'la cadena "$cadena" contiene: $cadena<br><br>';
   ?>
</CENTER>
</BODY>
</HTML>
```

4.2 VISUALIZACIÓN DE CADENAS

Comenzamos con las funciones que nos permiten visualizar y formatear cadenas de caracteres, permitiéndonos gestionar, de este modo, la salida de los datos por pantalla:

- `echo(cadena)`, `print(cadena)`: No son funciones propiamente dichas, sino construcciones del lenguaje. Ambas muestran información por la salida estándar; no soportan ningún atributo de formato de salida y sólo

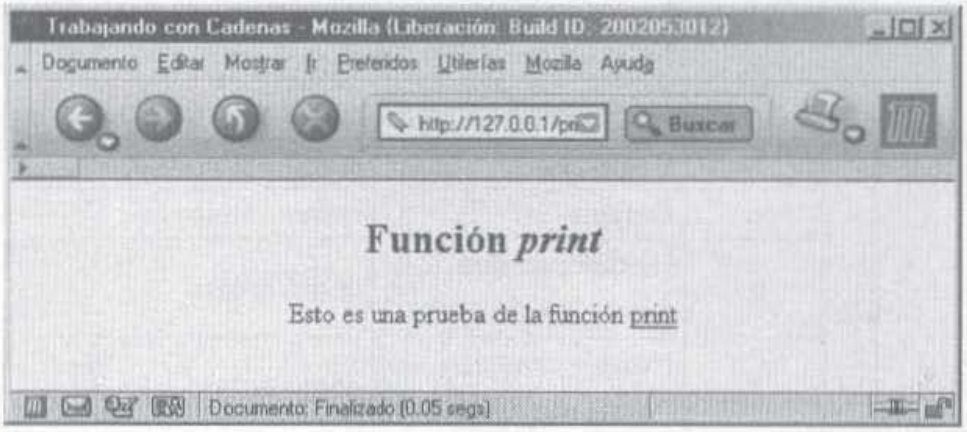
admiten una única cadena como argumento en su llamada (cuando necesitamos pasar más de un argumento a cualquiera de estas funciones, deberemos utilizar las comillas, como ocurre en los ejemplos precedentes).

Como su uso nos es conocido, de los capítulos anteriores, no profundizaremos más en ella.

El siguiente ejemplo nos muestra su funcionamiento básico:

```
<HTML>
<HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
</HEAD>
<BODY>
<CENTER>
<H2>Función <I>echo</I> & <I>print</I></H2>
    <?php
       $funcion1 = "print";
       $funcion2 = "echo";
       print "Esto es una prueba de la función <U>$funcion1</U><br>";
       echo "Esto es una prueba de la función <U>$funcion2</U><br>";
    ?>
</CENTER>
</BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



❑ `printf (formato [, argumentos])`: Imprime información por la salida estándar soportando diferentes formatos de salida. Admite múltiples tipos de argumentos a visualizar.

Imprime una cadena cuya estructura depende del formato descrito en el argumento *formato*. Esta cadena está formada por una ristra de caracteres, algunos de los cuales se mostrarán directamente, mientras que otros, los que

van precedidos por el carácter "%", conocidos como *especificaciones de conversión*, son utilizados para formatear la información.

Cada *especificación de conversión* se compone de los siguientes elementos (en orden):

1. Un *carácter de relleno* –opcional–, que se utilizará para rellenar y ajustar el resultado hasta lograr el tamaño de cadena correcto, por ejemplo, el espacio en blanco para caracteres (valor por defecto) o el cero para valores numéricos.

2. Un *carácter de alineamiento* –opcional–, para indicar si el resultado debe alinearse a la izquierda (carácter "-") o a la derecha (valor por defecto).

3. Un *indicador de tamaño* –opcional–, que indica el tamaño mínimo de caracteres que ocupará el argumento tras la conversión.

4. Un *indicador de precisión* –opcional–, formado por un punto (".") seguido del número de dígitos decimales que deberán mostrar los números en punto flotante. No tiene efecto con otros tipos de datos.

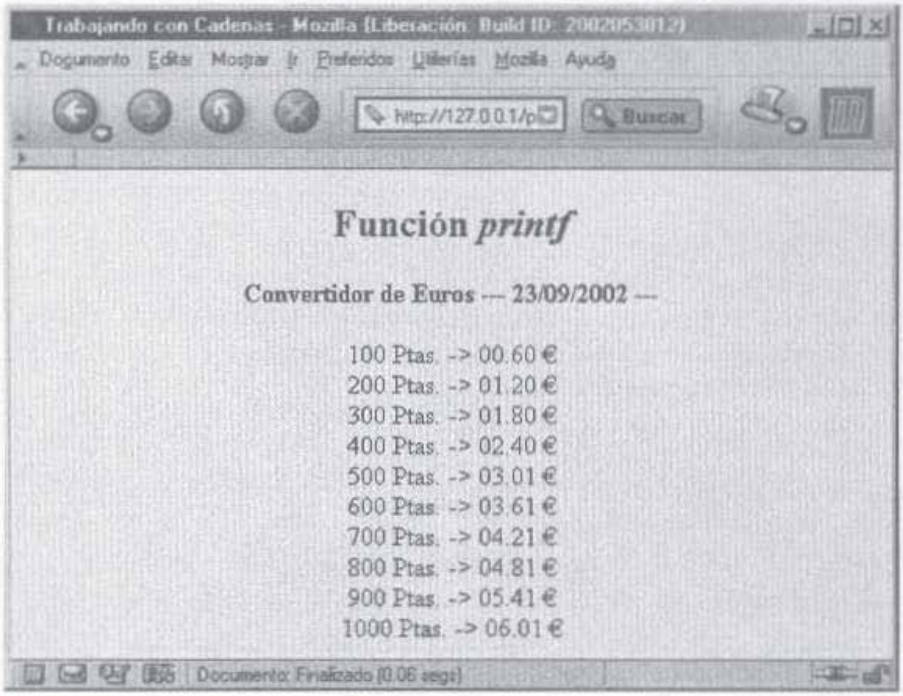
5. Finalmente, un *identificador de tipo de datos*, que especifica cómo se deberá tratar el dato. Los tipos disponibles son:

Símbolo	Tipo de dato	Tratamiento
`%`	carácter %	carácter %
`d`	Decimales	como entero decimal
`b`	Binarios	como entero binario
`o`	Octales	como entero
`x`	Hexadecimales (letras minúsculas)	como entero
`X`	Hexadecimales (letras mayúsculas)	como entero
`c`	Caracteres ASCII	como carácter
`f`	Punto flotante (signo decimal)	como `float` o decimales
`e`	Punto flotante (notación exponencial)	como `float` o decimales
`s`	Cadenas	como *string*

El siguiente ejemplo nos muestra la utilidad de esta función para formatear resultados en pantalla:

```
<HTML>
<HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
</HEAD>
<BODY>
<CENTER>
<H2>Función <I>printf</I></H2>
<?php
   $euro=166.386;
   $anyo=2002;
   $mes=9;
   $dia=23;
   printf ("%s--- %02d/%02d/%04d ---</B><BR><BR>",
         "<B>Convertidor de Euros ", $dia, $mes, $anyo);
   for ($i=100;$i<1100;$i+=100)
      printf("%4d Ptas. -> %02.2f €%s",$i,$i/$euro,"<BR>");
?>
</CENTER>
</BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



- `sprintf(formato [, argumentos])`: Es muy parecida a `printf()`, pero se diferencia de ésta en que devuelve una cadena de caracteres, la cual lo habitual es almacenarla en una variable

> **NOTA:** No se debe abusar del uso de estas funciones cuando no haya que aplicar un formato específico a los datos o el formato no sea importante, porque se obtienen mejores resultados haciendo uso de la función `echo` y concatenando las cadenas con el operador ".".

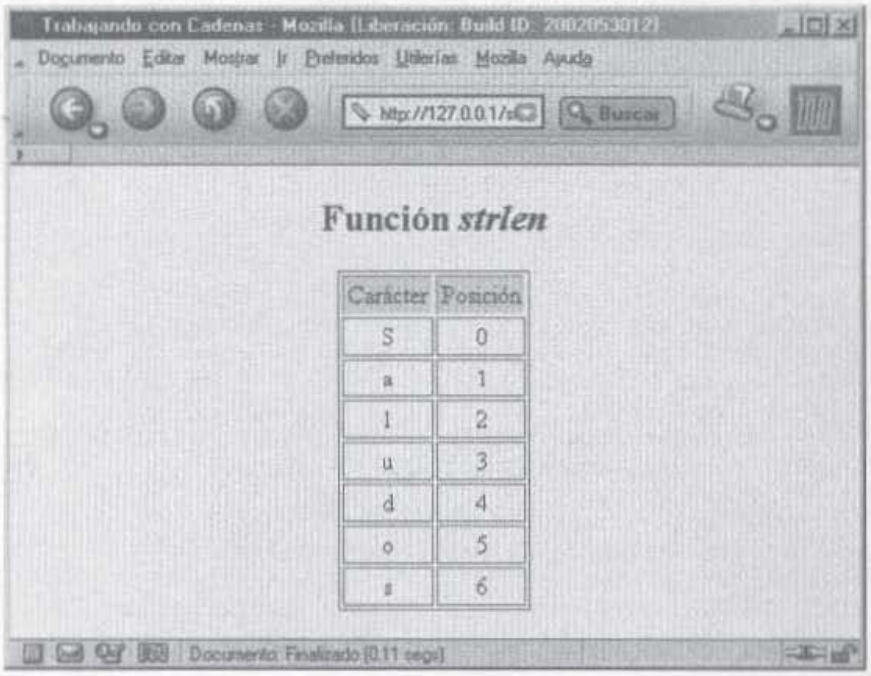
4.3 ACCESO AL CONTENIDO

Podemos acceder a cada uno de los caracteres que componen una cadena haciendo referencia a la posición que ocupan dentro de ella, de igual modo a como referenciamos los diferentes componentes de una *matriz* o *array*. Para ello, se hace indispensable conocer el tamaño de la cadena a recorrer.

- `strlen(cadena)`: Devuelve la longitud de la cadena pasada como argumento. El siguiente ejemplo nos muestra un modo básico de recorrer los diferentes componentes de una cadena:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Función <I>strlen</I></H2>
     <?php
       $cadena="Saludos";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR BGCOLOR='yellow'>\n";
       echo "<TD>Carácter</TD><TD>Posición</TD>\n</TR>\n";
       for($i=0;$i<strlen($cadena);$i++){
           echo "<TR ALIGN='center'>";
           echo "<TD>".$cadena[$i]."</TD><TD>".$i."</TD></TR>\n";
       }
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado sería:



4.4 BÚSQUEDA EN CADENAS

Una de las operaciones más habituales sobre las cadenas es la búsqueda de caracteres y subcadenas componentes, para su posterior tratamiento. Dentro de este tipo de funciones destacan las siguientes:

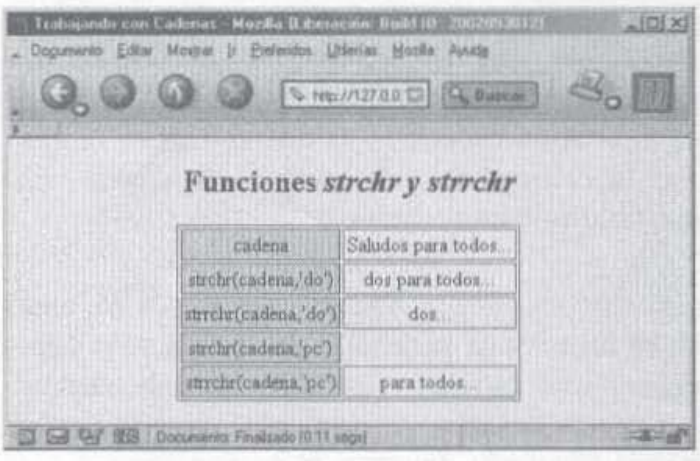
- `strstr(cadena, cadBusq)`, `strchr(cadena, cadBusq)`: Busca la aparición de una cadena dentro de otra y devuelve la subcadena comprendida entre la primera aparición de la cadena buscada (incluyéndola) hasta el final de la cadena en la que se realiza la búsqueda. En caso de no encontrar la cadena, devuelve una cadena vacía. La búsqueda diferencia entre mayúsculas y minúsculas.

- `strrchr(cadena, cadBusq)`: Busca la aparición de un carácter (aunque se utilice una cadena de búsqueda sólo tiene en cuenta su primer carácter) en una cadena y devuelve la subcadena comprendida entre la última aparición del carácter (incluido) hasta el final de la cadena en la que se realiza la búsqueda. En caso de no encontrar el carácter en la cadena, se devuelve una cadena vacía. La búsqueda diferencia entre mayúsculas y minúsculas. Si el elemento a buscar no es una cadena, se convierte a entero y se aplica como el valor ordinal de un carácter.

El siguiente código nos muestra el funcionamiento de ambas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>strchr y strrchr</I></H2>
     <?php
       $cadena="Saludos para todos...";
       $car1="do";
       $car2="pc";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena</TD>";
       echo "<TD>$cadena</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strchr(cadena,'$car1')</TD>";
       echo "<TD>".strchr($cadena,$car1)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strrchr(cadena,'$car1')</TD>";
       echo "<TD>".strrchr($cadena,$car1)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strchr(cadena,'$car2')</TD>";
       echo "<TD>".strchr($cadena,$car2)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strrchr(cadena,'$car2')</TD>";
       echo "<TD>".strrchr($cadena,$car2)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El navegador nos mostraría:



- `stristr(cadena, cadBusq)`: El comportamiento de esta función es igual al de la función `strstr()`, salvo que ésta no diferencia entre mayúsculas y minúsculas.

El siguiente ejemplo nos muestra el diferente comportamiento de estas dos funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>strstr y stristr</I></H2>
     <?php
       $cadena="Saludos para todos...";
       $car="Pa";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena</TD>";
       echo "<TD>$cadena</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strstr(cadena,'$car')</TD>";
       echo "<TD>".strstr($cadena,$car)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>stristr(cadena,'$car')</TD>";
       echo "<TD>".stristr($cadena,$car)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Veamos el resultado:

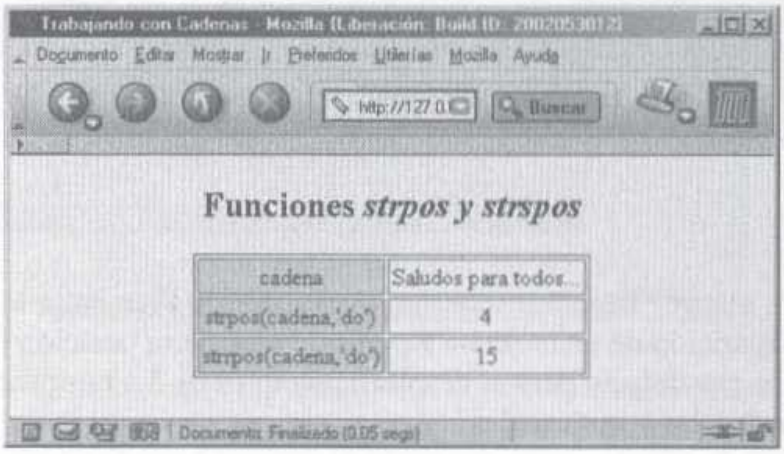
- `strpos(cad1, cad2 [, desplz])`: Encuentra la primera posición de aparición de una cadena a partir de una cierta posición (si no se indica, el valor por defecto para el desplazamiento es 0). La búsqueda diferencia entre mayúsculas y minúsculas.

- `strrpos(cadena, carácter)`: Devuelve la posición de la última aparición de un carácter determinado en una cadena (aunque se utilice una cadena de búsqueda sólo tiene en cuenta su primer carácter). En caso de no encontrar el carácter en la cadena, se devuelve `false`. La búsqueda diferencia entre mayúsculas y minúsculas. Si el elemento a buscar no es una cadena, se convierte a entero y se aplica como el valor ordinal de un carácter.

Veamos un ejemplo del funcionamiento de ambas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>strpos y strspos</I></H2>
     <?php
       $cadena="Saludos para todos...";
       $car="do";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena</TD>";
       echo "<TD>$cadena</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strpos(cadena,'$car')</TD>";
       echo "<TD>".strpos($cadena,$car)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strrpos(cadena,'$car')</TD>";
       echo "<TD>".strrpos($cadena,$car)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado que obtenemos es el siguiente:



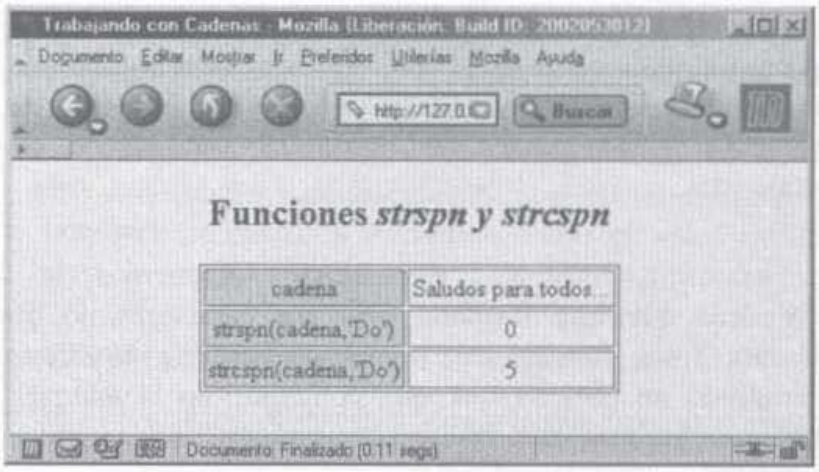
- `strspn(cadena, máscara)`: Obtenemos la longitud de la subcadena más larga que está formada sólo por caracteres contenidos en una máscara. Una vez encontrado un carácter que no está contenido en la máscara, se abandona la búsqueda. La búsqueda diferencia entre mayúsculas y minúsculas.

- `strcspn(cadena, máscara)`: Obtenemos la longitud de la subcadena que está formada sólo por caracteres no contenidos en una máscara (esta función es la complementaria a `strspn()`). Una vez encontrado un carácter que está contenido en la máscara, se abandona la búsqueda. La búsqueda diferencia entre mayúsculas y minúsculas.

Veamos un ejemplo con el funcionamiento de ambas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>strspn y strcspn</I></H2>
     <?php
       $cadena="Saludos para todos...";
       $car="Do";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena</TD>";
       echo "<TD>$cadena</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strspn(cadena,'$car')</TD>";
       echo "<TD>".strspn($cadena,$car)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strcspn(cadena,'$car')</TD>";
       echo "<TD>".strcspn($cadena,$car)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El navegador mostraría:



4.5 COMPARACIÓN DE CADENAS

Otra operación común con cadenas es compararlas para saber cuál es mayor. Dentro de este tipo de funciones destacan las siguientes:

- `strcmp(cad1, cad2)`: Compara dos cadenas y devuelve un valor menor que 0, si la segunda cadena es mayor que la primera; mayor que 0, en caso contrario, y 0, si ambas cadenas son iguales. La comparación distingue entre mayúsculas y minúsculas.

- `strcasecmp(cad1, cad2)`: Se comporta igual que `strcmp()`, excepto en que no diferencia mayúsculas de minúsculas.

Veamos un ejemplo con el funcionamiento de ambas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>strcmp y strcasecmp</I></H2>
     <?php
       $cad1="Atención";
       $cad2="atención";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena1</TD>";
       echo "<TD>$cad1</TD></TR>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena2</TD>";
       echo "<TD>$cad2</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strcmp(cadena1,cadena2)</TD>";
       echo "<TD>".strcmp($cad1,$cad2)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strcasecmp(cadena1,cadena2)</TD>";
       echo "<TD>".strcasecmp($cad1,$cad2)."</TD></TR>\n";
```

```
      echo "</TABLE>\n";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:

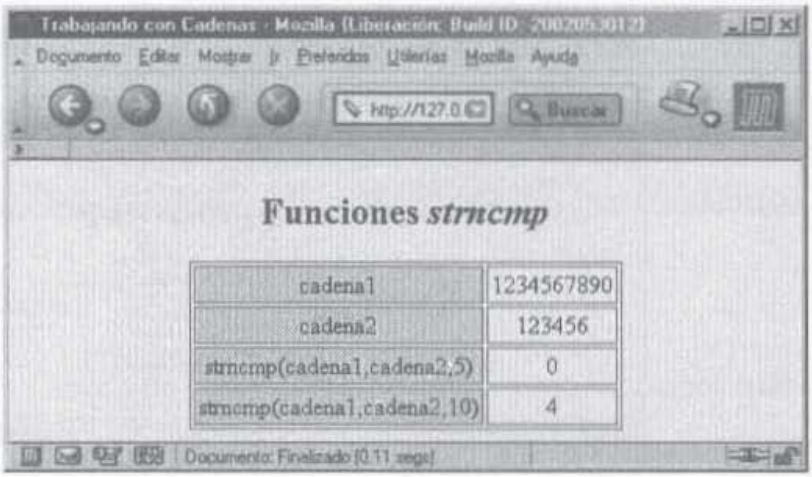
- `strncmp(cad1, cad2, long)`: Funciona como `strcmp()`, sólo que permite comparar los `long` primeros caracteres de dos cadenas. Si alguna cadena es menor que el número de caracteres a comparar, se usará su longitud como `long` para la comparación. Diferencia mayúsculas de minúsculas.

Veamos un ejemplo con el funcionamiento de esta función:

```
<HTML>
 <HEAD>
  <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
  <CENTER>
   <H2>Funciones <I>strncmp</I></H2>
   <?php
    $cad1="1234567890";
    $cad2="123456";
    $long1=5;
    $long2=10;
    echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
    echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena1</TD>";
    echo "<TD>$cad1</TD></TR>\n";
    echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena2</TD>";
    echo "<TD>$cad2</TD></TR>\n";
    echo "<TR ALIGN='center'>";
    echo "<TD BGCOLOR='yellow'>strncmp(cadena1,cadena2,$long1)</TD>";
    echo "<TD>".strncmp($cad1,$cad2,$long1)."</TD></TR>\n";
    echo "<TR ALIGN='center'>";
    echo "<TD BGCOLOR='yellow'>strncmp(cadena1,cadena2,$long2)</TD>";
    echo "<TD>".strncmp($cad1,$cad2,$long2)."</TD></TR>\n";
    echo "</TABLE>\n";
```

```
    ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



- `strnatcmp(cad1, cad2)`: Se comporta igual que `strcmp()`, excepto en que utiliza una comparación "natural" de las cadenas alfanuméricas. Distingue entre mayúsculas y minúsculas.

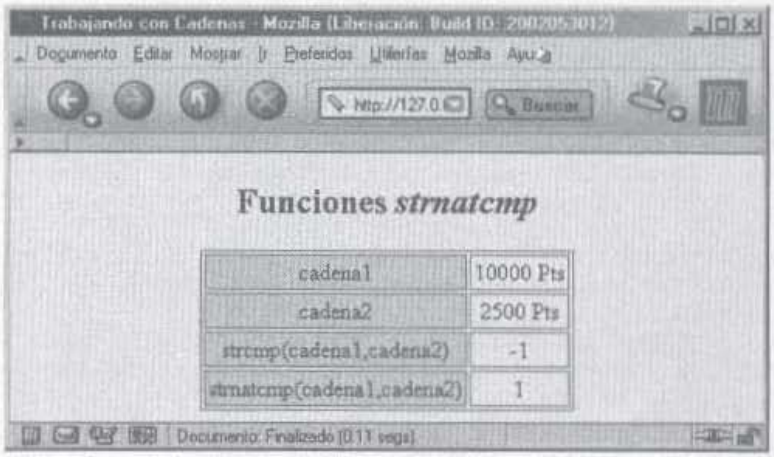
- `strnatcasecmp(cad1, cad2)`: Se comporta igual que `strnatcmp()`, excepto en que no diferencia entre mayúsculas y minúsculas.

Veamos un ejemplo con el funcionamiento diferenciado de `strcmp()` y `strnatcmp()`:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>strnatcmp</I></H2>
     <?php
       $cad1="10000 Pts";
       $cad2="2500 Pts";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena1</TD>";
       echo "<TD>$cad1</TD></TR>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena2</TD>";
       echo "<TD>$cad2</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strcmp(cadena1,cadena2)</TD>";
       echo "<TD>".strcmp($cad1,$cad2)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
```

```
        echo "<TD BGCOLOR='yellow'>strnatcmp(cadena1,cadena2)</TD>";
        echo "<TD>".strnatcmp($cad1,$cad2)."</TD></TR>\n";
        echo "</TABLE>\n";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



4.6 OPERAR CON SUBCADENAS

No siempre nos interesan todos los caracteres que componen una cadena, sino que sólo estamos interesados en trabajar con un conjunto. Para ello contamos con un conjunto de funciones preparadas para extraer subcadenas de una cadena original. Entre las más utilizadas están:

- `substr(cadena, inicio [, tamaño])`: Devuelve la subcadena que se encuentra a partir de una posición dada y llega hasta el final de la cadena original, pudiendo de forma opcional decidir el tamaño de la subcadena a recuperar.

Los argumentos enteros pueden ser números negativos de modo que:

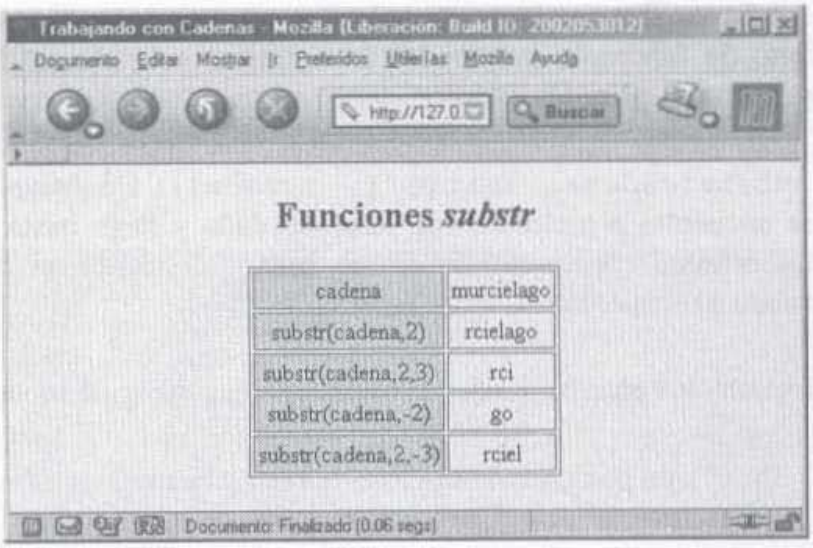
- Una posición de inicio con valor negativo significa que se debe comenzar desde el final de la cadena.

- Un tamaño con valor negativo indica cuántos caracteres del final de la cadena no se tendrán en cuenta.

```
<HTML>
 <HEAD>
  <TITLE>Trabajando con Cadenas</TITLE>
```

```
</HEAD>
<BODY>
  <CENTER>
    <H2>Funciones <I>substr</I></H2>
    <?php
      $cadena="murcielago";
      $ini1 = 2;
      $ini2 = -2;
      $tam1 = 3;
      $tam2 = -3;
      echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
      echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena</TD>";
      echo "<TD>$cadena</TD></TR>\n";
      echo "<TR ALIGN='center'>";
      echo "<TD BGCOLOR='yellow'>substr(cadena,$ini1)</TD>";
      echo "<TD>".substr($cadena,$ini1)."</TD></TR>\n";
      echo "<TR ALIGN='center'>";
      echo "<TD BGCOLOR='yellow'>substr(cadena,$ini1,$tam1)</TD>";
      echo "<TD>".substr($cadena,$ini1,$tam1)."</TD></TR>\n";
      echo "<TR ALIGN='center'>";
      echo "<TD BGCOLOR='yellow'>substr(cadena,$ini2)</TD>";
      echo "<TD>".substr($cadena,$ini2)."</TD></TR>\n";
      echo "<TR ALIGN='center'>";
      echo "<TD BGCOLOR='yellow'>substr(cadena,$ini1,$tam2)</TD>";
      echo "<TD>".substr($cadena,$ini1,$tam2)."</TD></TR>\n";
      echo "</TABLE>\n";
    ?>
  </CENTER>
</BODY>
</HTML>
```

Veamos qué muestra el navegador con el código anterior:



- `substr_replace(cad1, cad2, inicio [, tamaño])`: Devuelve una cadena que es resultado de la sustitución de parte del contenido de una cadena original (indicado por una posición de inicio y opcionalmente por un tamaño) por el contenido de otra cadena. La cadena

original no sufre ninguna modificación. Distingue entre mayúsculas y minúsculas.

Los argumentos enteros pueden ser números negativos de modo que:

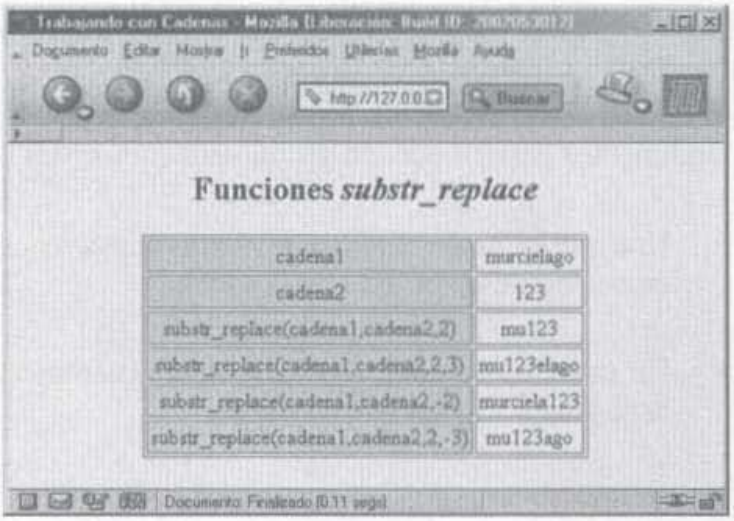
- Una posición de inicio con valor negativo significa que se debe comenzar desde el final de la cadena.

- Un tamaño con valor negativo indica cuántos caracteres del final de la cadena no se tendrán en cuenta.

Por ejemplo:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>substr_replace</I></H2>
     <?php
       $cad1="murcielago";
       $cad2="123";
       $ini1 = 2;
       $ini2 = -2;
       $tam1 = 3;
       $tam2 = -3;
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena1</TD>";
       echo "<TD>$cad1</TD></TR>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena2</TD>";
       echo "<TD>$cad2</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>";
       echo "substr_replace(cadena1,cadena2,$ini1)</TD>";
       echo "<TD>".substr_replace($cad1,$cad2,$ini1)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>";
       echo "substr_replace(cadena1,cadena2,$ini1,$tam1)</TD>";
       echo "<TD>".substr_replace($cad1,$cad2,$ini1,$tam1)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>";
       echo "substr_replace(cadena1,cadena2,$ini2)</TD>";
       echo "<TD>".substr_replace($cad1,$cad2,$ini2)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>";
       echo "substr_replace(cadena1,cadena2,$ini1,$tam2)</TD>";
       echo "<TD>".substr_replace($cad1,$cad2,$ini1,$tam2)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

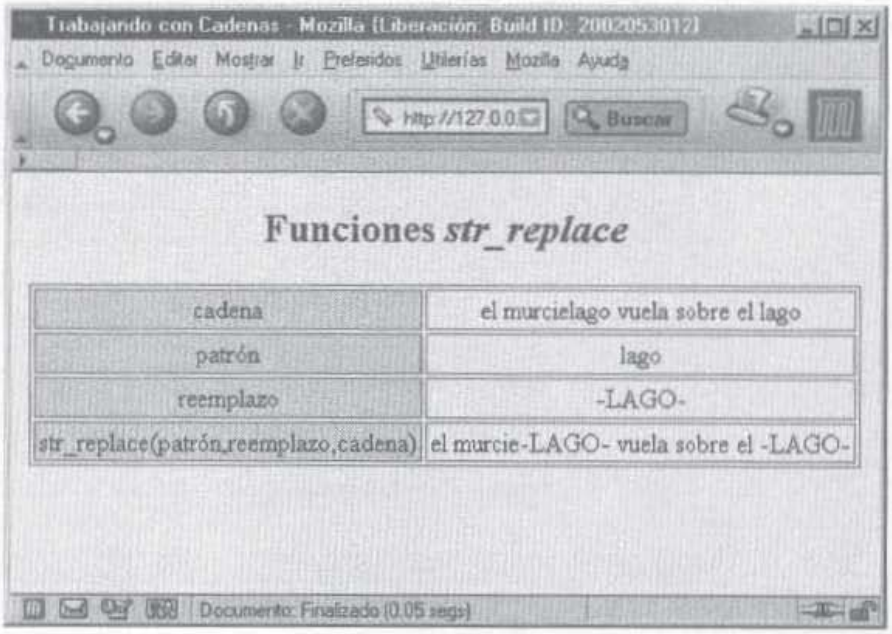
Y el resultado obtenido es:



- `str_replace(cadBus, cadRee, cadena)`: Devuelve una cadena, que es resultado de la sustitución de todas las apariciones de una subcadena, por otro contenido en la cadena original. La cadena original no sufre ninguna modificación. Diferencia entre mayúsculas y minúsculas. Por ejemplo:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>str_replace</I></H2>
     <?php
       $cadena="el murcielago vuela sobre el lago";
       $cadB="lago";
       $cadS="-LAGO-";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena</TD>";
       echo "<TD>$cadena</TD></TR>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>patrón</TD>";
       echo "<TD>$cadB</TD></TR>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>reemplazo</TD>";
       echo "<TD>$cadS</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>";
       echo "str_replace(patrón,reemplazo,cadena)". "</TD>";
       echo "<TD>".str_replace($cadB,$cadS,$cadena)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

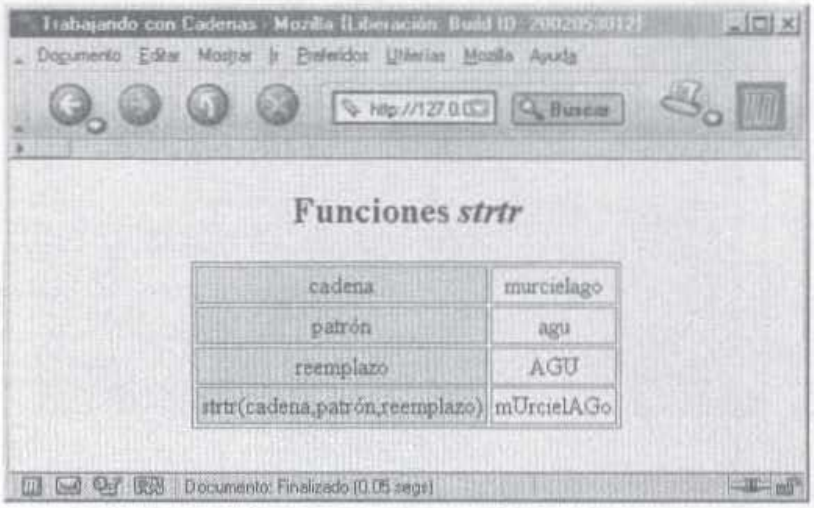
Y el resultado obtenido es:



- `strtr(cadena, cadBus, cadRee)`: Devuelve una cadena, que es resultado de la sustitución de cada una de las apariciones, en la cadena original, de cada uno de los caracteres de una subcadena, por sus correspondientes caracteres dentro de una cadena de sustitución dada. La cadena original no sufre ninguna modificación. Diferencia entre mayúsculas y minúsculas. Por ejemplo:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>strtr</I></H2>
     <?php
       $cadena="murcielago";
       $cadB="agu";
       $cadS="AGU";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena</TD>";
       echo "<TD>$cadena</TD></TR>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>patrón</TD>";
       echo "<TD>$cadB</TD></TR>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>reemplazo</TD>";
       echo "<TD>$cadS</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strtr(cadena,patrón,reemplazo)</TD>";
       echo "<TD>".strtr($cadena,$cadB,$cadS)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se puede observar en la siguiente imagen:



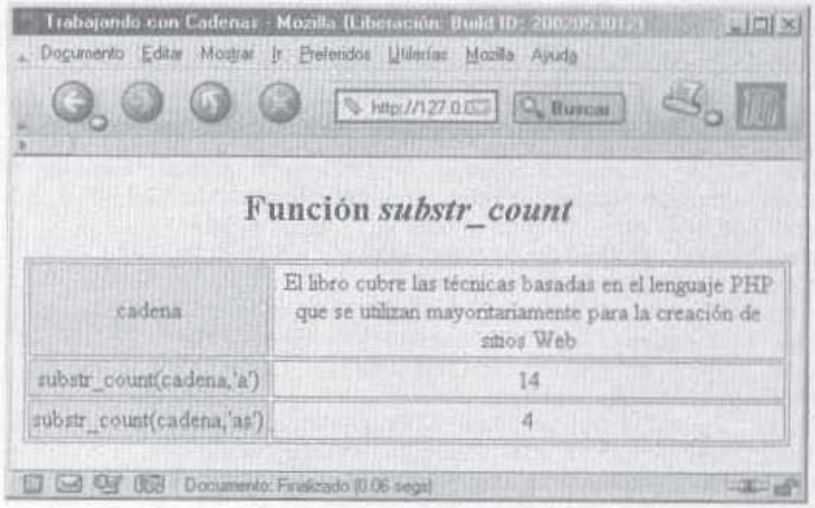
- `strtr (cadena, array_reemplazo)`: Es la forma alternativa que permite esta función. En este caso, el segundo parámetro es un *array asociativo* que tiene información sobre los reemplazos a realizar. Su uso se muestra en el siguiente ejemplo:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>strtr</I></H2>
     <?php
       $cadena="murcielago";
       $matriz['a']="A";
       $matriz['g']="G";
       $matriz['u']="U";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>cadena</TD>";
       echo "<TD>$cadena</TD></TR>\n";
       echo "<TR ALIGN='center'><TD BGCOLOR='yellow'>matriz</TD>";
       echo "<TD>$matriz</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>strtr(cadena,matriz)</TD>";
       echo "<TD>".strtr($cadena,$matriz)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

- `substr_count(cadena, patron)`: Devuelve el número de apariciones de una subcadena dentro de una cadena. Diferencia entre mayúsculas y minúsculas. Por ejemplo:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Función <I>substr_count</I></H2>
     <?php
       $patron1="a";
       $patron2="as";
       $cadena="El libro cubre las técnicas basadas en el lenguaje";
       $cadena.=" PHP que se utilizan mayoritariamente para la";
       $cadena.=" creación de sitios Web";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>cadena</TD><TD>$cadena</TD>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>substr_count(cadena,'$patron1')</TD>";
       echo "<TD>".substr_count($cadena,$patron1)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>substr_count(cadena,'$patron2')</TD>";
       echo "<TD>".substr_count($cadena,$patron2)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



4.7 MODIFICACIÓN DEL CONTENIDO

4.7.1 Limpieza de cadenas

- `chop(cadena)`: Devuelve la cadena argumento sin los espacios en blanco y caracteres de fin de línea que se encuentren al final de la cadena.

- `rtrim(cadena)`: Se comporta igual que `chop()`.

- `ltrim(cadena)`: Elimina espacios en blanco que se encuentran al principio de una cadena de texto. Recibe como parámetro una cadena de caracteres y devuelve la misma cadena pero sin los espacios iniciales.

- `trim(cadena)`: Elimina los espacios en blancos que se encuentran al principio y al final de una cadena de texto. Recibe como argumento una cadena de caracteres y devuelve como resultado esa cadena sin los espacios iniciales y finales.

Veamos un ejemplo de estas cuatro funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>chop, rtrim, ltrim y trim</I></H2>
     <?php
       // 6 espacios por delante y 4 por detrás
       $cadena="      abcdefghi    ";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'><TD colspan=2> </TD>";
       echo "<TD BGCOLOR='yellow'>tamaño</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>cadena</TD><TD><pre>$cadena</pre></TD>";
       echo "<TD>".strlen($cadena)."</TD>\n<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>chop(cadena)</TD>";
       echo "<TD><pre>".chop($cadena)."</pre></TD>";
       echo "<TD>".strlen(chop($cadena))."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>rtrim(cadena)</TD>";
       echo "<TD><pre>".rtrim($cadena)."</pre></TD>";
       echo "<TD>".strlen(rtrim($cadena))."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>ltrim(cadena)</TD>";
       echo "<TD><pre>".ltrim($cadena)."</pre></TD>";
       echo "<TD>".strlen(ltrim($cadena))."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>trim(cadena)</TD>";
       echo "<TD><pre>".trim($cadena)."</pre></TD>";
       echo "<TD>".strlen(trim($cadena))."</TD></TR>\n";
```

```
      echo "</TABLE>\n";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado que se obtiene es el siguiente:

4.7.2 Relleno de cadenas

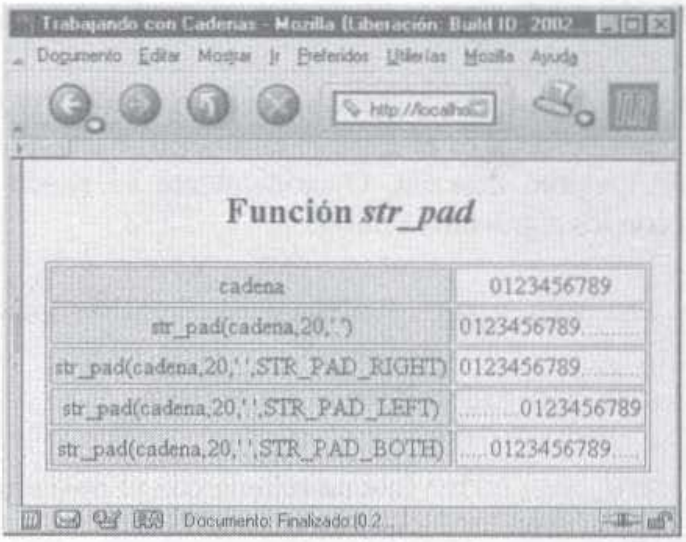
- `str_pad(cadena, long)`: Rellena una cadena con un carácter de relleno (por defecto, es el espacio en blanco) hasta que la cadena resultante tenga la longitud deseada. Opcionalmente se puede indicar el modo de relleno con los siguientes valores:
  - `STR_PAD_RIGHT`: Rellena por la derecha (opción por defecto).
  - `STR_PAD_LEFT`: Rellena por la izquierda.
  - `STR_PAD_BOTH`: Intenta rellena con el mismo número de caracteres por la derecha y por la izquierda.

Veamos un ejemplo de esta función:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
</HEAD>
 <BODY>
   <CENTER>
     <H2>Función <I>str_pad</I></H2>
     <?php
       $cadena="0123456789";
```

```
        $long = 20;
        $car = ".";
        echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
        echo "<TR ALIGN='center'>";
        echo "<TD BGCOLOR='yellow'>cadena</TD><TD>$cadena</TD>\n";
        echo "<TR ALIGN='center'>";
        echo "<TD BGCOLOR='yellow'>str_pad(cadena,$long,'$car')</TD>";
        echo "<TD>".str_pad($cadena,$long,$car)."</TD></TR>\n";
        echo "<TR ALIGN='center'>";
        echo "<TD
BGCOLOR='yellow'>str_pad(cadena,$long,'$car',STR_PAD_RIGHT)</TD>";
        echo "<TD>".str_pad($cadena,$long,$car,STR_PAD_RIGHT)."</TD></TR>\n";
        echo "<TR ALIGN='center'>";
        echo "<TD
BGCOLOR='yellow'>str_pad(cadena,$long,'$car',STR_PAD_LEFT)</TD>";
        echo "<TD>".str_pad($cadena,$long,$car,STR_PAD_LEFT)."</TD></TR>\n";
        echo "<TR ALIGN='center'>";
        echo "<TD
BGCOLOR='yellow'>str_pad(cadena,$long,'$car',STR_PAD_BOTH)</TD>";
        echo "<TD>".str_pad($cadena,$long,$car,STR_PAD_BOTH)."</TD></TR>\n";
        echo "</TABLE>\n";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



4.7.3 Conversión entre mayúsculas y minúsculas

- `strtolower(cadena)`: Convierte una cadena de caracteres a minúsculas. Recibe como argumento una cadena y devuelve la misma cadena con los caracteres alfabéticos en minúsculas.

- `strtoupper(cadena)`: Funciona de forma contraria a la función anterior. Convierte una cadena de caracteres a mayúsculas.

- `ucfirst(cadena)`: Se encarga de convertir a mayúsculas el primer carácter de una cadena de caracteres, siempre que éste sea alfabético.

> **NOTA:** Para hacer un uso correcto de esta función, hemos de prestar atención a las características locales propias de PHP, de modo que tengamos las que necesitamos.

- `ucwords(cadena)`: Se comporta igual que la función anterior, salvo que ésta convierte a mayúsculas el primer carácter de cada palabra que compone la cadena, siempre y cuando sean caracteres alfanuméricos.

Vamos a ver el funcionamiento conjunto de todas estas funciones en el siguiente ejemplo:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>strtoupper, strtolower, ucfirst y ucwords</I></H2>
     <?php
       $frase="saludos PARA todos...";
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='right'>";
       echo "<TD BGCOLOR='yellow'>frase</TD><TD>$frase</TD>\n";
       echo "<TR ALIGN='right'>";
       echo "<TD BGCOLOR='yellow'>strtoupper(frase)</TD>";
       echo "<TD>".strtoupper($frase)."</TD></TR>\n";
       echo "<TR ALIGN='right'>";
       echo "<TD BGCOLOR='yellow'>strtolower(frase)</TD>";
       echo "<TD>".strtolower($frase)."</TD></TR>\n";
       echo "<TR ALIGN='right'>";
       echo "<TD BGCOLOR='yellow'>ucfirst(frase)</TD>";
       echo "<TD>".ucfirst($frase)."</TD></TR>\n";
       echo "<TR ALIGN='right'>";
       echo "<TD BGCOLOR='yellow'>ucwords(frase)</TD>";
       echo "<TD>".ucwords($frase)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

Y obtenemos:

4.7.4 Enmascaramiento de caracteres

Al generar documentos, es habitual que los caracteres que tienen un significado especial (metacaracteres) aparezcan precedidos por una barra invertida ("\"), esto es, aparezcan enmascarados, para que el intérprete de PHP los considere como a otro cualquiera. Para realizar operaciones con caracteres especiales, PHP cuenta con las siguientes funciones:

- `addslashes(cadena)`: Devuelve una cadena de caracteres igual que la original en la que se han *escapado* los caracteres especiales comillas dobles (`"`), comillas simples (`'`) y barra invertida (\).

- `stripslashes(cadena)`: Tiene una función contraria a la función `addslashes()` ya que devuelve la cadena resultante de eliminar las barras invertidas que protegen los caracteres comillas dobles (“), comillas simples (‘) y barra invertida (\).

- `addcslashes(cadena)`: Nos permite enmascarar cualquier carácter que desee el usuario. Al estilo del lenguaje C, los caracteres con código ASCII inferior a 32 y superior a 126 son convertidos a representación octal.

- `stripcslashes(cadena)`: Realiza la operación contraria a `addcslashes()`.

- `quotemeta(cadena)`: Devuelve una cadena de caracteres igual que la original en la que se han *escapado* los caracteres especiales: `.`, `\\`, `+`, `*`, `?`, `[`, `^`, `]`, `(`, `$`, y `)`.

Veamos con un ejemplo el funcionamiento de las funciones anteriores:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>addslashes y stripslashes</I></H2>
     <?php
       $cadena='path=["\usr\$usuario"]|\usr';
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>cadena</TD><TD>$cadena</TD>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>addslashes(cadena)</TD>";
       echo "<TD>".addslashes($cadena)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>addcslashes(cadena,'$|')</TD>";
       echo "<TD>".addcslashes($cadena,"$|")."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>quotemeta(cadena)</TD>";
       echo "<TD>".quotemeta($cadena)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>stripslashes(cadena)</TD>";
       echo "<TD>".stripslashes($cadena)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>stripcslashes(cadena)</TD>";
       echo "<TD>".stripcslashes($cadena)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado obtenido es:

4.7.5 División de cadenas

- `strtok(cadena, divisor)`: Divide una cadena en diferentes subcadenas.

Veamos su utilización con el siguiente ejemplo:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Función <I>strtok</I></H2>
     <?php
       $patron=";";
       $cadena="dato1;dato2;dato3:dato4;dato5";
       $datos = strtok($cadena,$patron);
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>cadena</TD><TD>$cadena</TD>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD COLSPAN='2'
BGCOLOR='yellow'>substr_count(cadena,'$patron')</TD>";
       echo "</TR>\n";
       while($datos){
          echo "<TR ALIGN='center'>";
          echo "<TD BGCOLOR='yellow'>subcadena</TD>";
          echo "<TD>$datos</TD></TR>\n";
          $datos=strtok($patron);
       }
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

La solución se muestra en la siguiente imagen:

- `chunk_split(cadena [, long [, separador]])`: Permite marcar una cadena en porciones diferenciadas de tamaño menor. Podemos indicarle el tamaño de estas porciones e, incluso, el carácter o cadena con el que las separará. No modifica la cadena original.

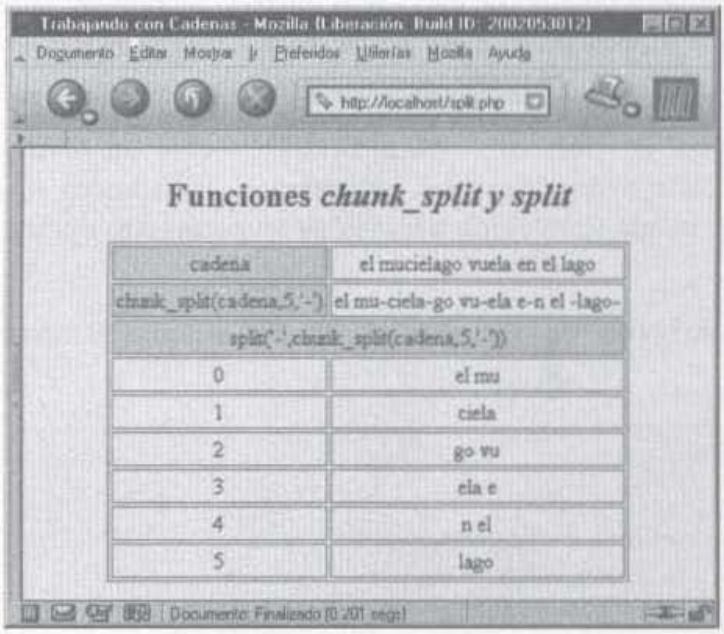
- `split(patron, cadena)`: Devuelve un *array* resultado de dividir una cadena en diferentes subcadenas. El número total de elementos del *array* puede ser elegido a través de un parámetro opcional.

Veamos con un ejemplo el funcionamiento conjunto de las funciones anteriores:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>chunk_split y split</I></H2>
     <?php
       $cadena="el murcielago vuela en el lago";
       $long=5;
       $sep="-";
       $cadena_aux = chunk_split($cadena,$long,$sep);
       $productos = split($sep, $cadena_aux);
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>cadena</TD>";
       echo "<TD>$cadena</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>chunk_split(cadena,$long,'$sep')</TD>";
       echo "<TD>$cadena_aux</TD></TR>\n";
```

```
        echo "<TR ALIGN='center'>";
        echo "<TD BGCOLOR='yellow' COLSPAN='2'>";
        echo "split('$sep',chunk_split(cadena,$long,'$sep'))</TD></TR>\n";
        for($i=0;$i<count($productos)-1;$i++){
            echo "<TR ALIGN='center'>";
            echo "<TD>$i</TD><TD>$productos[$i]</TD></TR>\n";
        }
        echo "</TABLE>\n";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se observa en la siguiente imagen:



- `explode(patron, cadena)`: Devuelve un *array* resultado de dividir una cadena en diferentes subcadenas. El número de elementos del *array* puede ser elegido a través de un parámetro opcional.

- `implode(nexo, cadena)`: Devuelve una cadena resultado de unir todos los elementos de un *array* en el mismo orden en el que aparecen, pero con una cadena en medio de ellos.

Veamos con un ejemplo el funcionamiento conjunto de las funciones anteriores:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones <I>explode e implode</I></H2>
     <?php
       $patron="-";
       $nexo=":";
       $limite=5;
       $cadena="dato1-dato2-dato3-dato4-dato5";
       $datos = explode($patron,$cadena,$limite);
       echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>cadena</TD><TD>$cadena</TD>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD COLSPAN='2'
BGCOLOR='yellow'>explode('$patron',cadena,$limite)</TD>";
       echo "</TR>\n";
       for($i=0;$i<$limite;$i++){
          echo "<TR ALIGN='center'>";
          echo "<TD BGCOLOR='yellow'>subcadena $i</TD>";
          echo "<TD>$datos[$i]</TD></TR>\n";
       }
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>implode('$nexo',datos)</TD>";
       echo "<TD>",implode($nexo,$datos),"</TD></TR>\n";
       echo "</TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se observa en la siguiente imagen:

4.8 RELACIONADAS CON HTML

Gran parte de las cadenas con las que trabaja PHP tienen como función convertirse en partes de un documento HTML que será enviado al usuario. Para facilitar algunas de los problemas que tiene la transformación de texto en código HTML válido, PHP proporciona las siguientes funciones entre otras:

- `htmlspecialchars(cadena)`: Se encarga de convertir los caracteres con un significado especial en HTML en entidades HTML según la tabla de la página siguiente.

Car.	Traducción a entidad HTML
&	`&`
"	`&quot;`
<	`<`
>	`>`

- `htmlentities(cadena)`: Es similar a la función anterior, salvo que esta función traduce todos los caracteres a su correspondiente entidad en HTML.

4.9 OTRAS FUNCIONES

- `chr(entero)`: Recibe como argumento un valor entero correspondiente a un código ASCII y devuelve el carácter asociado a dicho código.

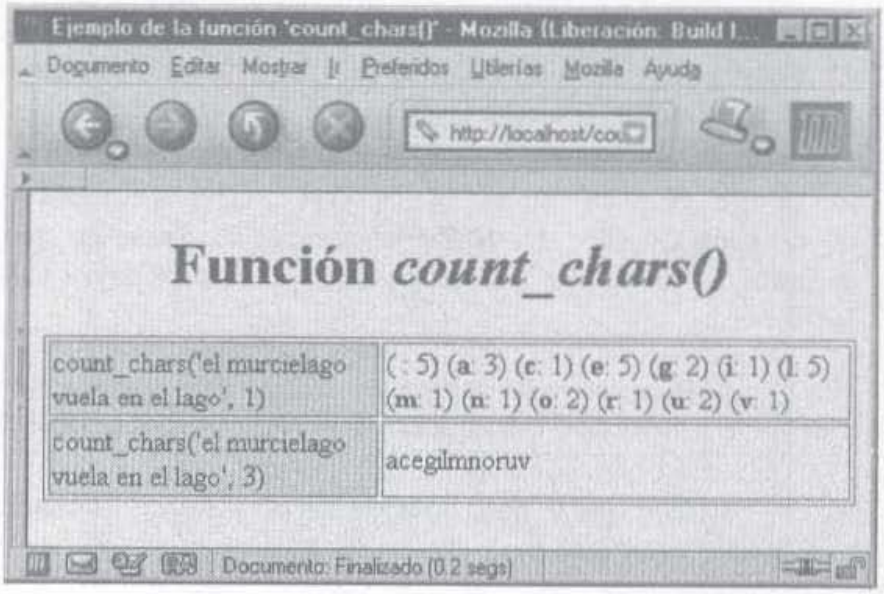
- `count_chars(cadena [, modo])`: Cuenta el número de apariciones de cada carácter dentro de la cadena. El modo en que devuelve esta información depende de un parámetro opcional cuyos valores son:

0	Devuelve una matriz asociativa con el valor del carácter como clave y su frecuencia como valor (valor por defecto)
1	Como el 0, pero sólo los caracteres con frecuencia de aparición superior a 0
2	Como el 0, pero sólo los caracteres con frecuencia de aparición igual a 0
3	Devuelve una cadena que contiene los caracteres utilizados
4	Devuelve una cadena que contiene los caracteres no utilizados

El siguiente ejemplo nos muestra una utilización posible de esta función:

```
<HTML>
<HEAD>
<TITLE>Ejemplo de la función 'count_chars()'</TITLE>
<BODY>
<CENTER>
<H1>Función <I>count_chars()</I></H1>
<TABLE BORDER='1'>
<?
$cadena="el murcielago vuela en el lago";

echo "<TR><TD BGCOLOR='yellow'>count_chars('$cadena', 0)</TD>";
echo "<TD>";
$matriz=count_chars($cadena, 0);
foreach ($matriz as $clave => $valor)
  echo "(<b>", chr($clave), "</b>: $valor) ";
echo "</TD></TR>";

echo "<TR><TD BGCOLOR='yellow'>count_chars('$cadena', 3)</TD>";
echo "<TD>", count_chars($cadena, 3), "</TD></TR>";

?>
</TABLE>
</BODY>
</HTML>
```

El resultado se visualiza en la siguiente imagen:



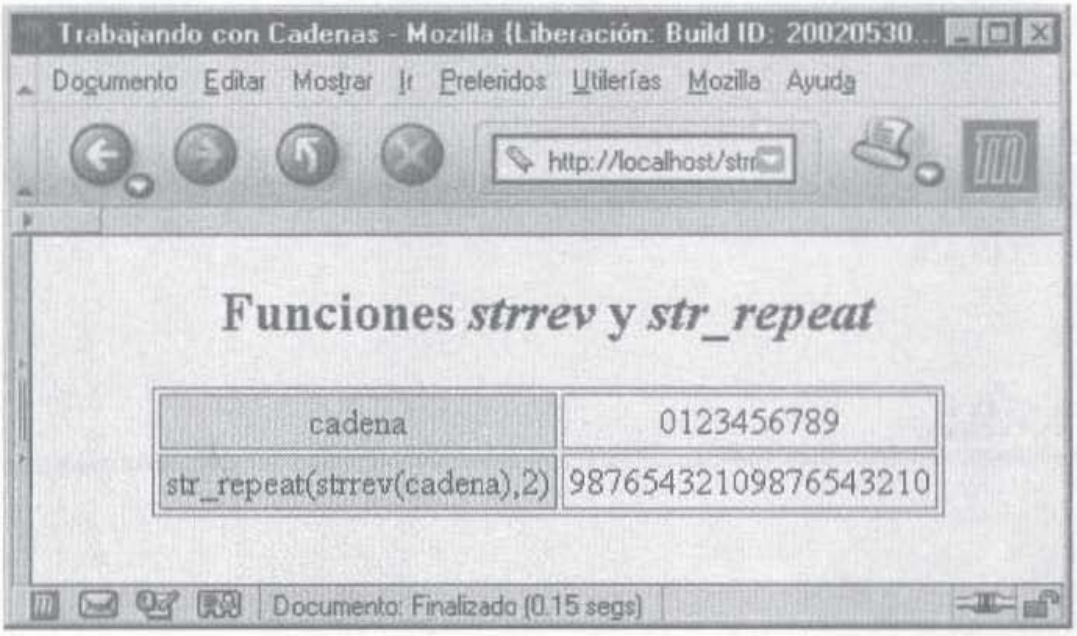
- `strrev(cadena)`: Devuelve la cadena invertida.

- `str_repeat(cadena, num_veces)`: Recibe como argumento una cadena y un número entero para devolver, a continuación, la cadena repetida tantas veces como indique el número entero.

El siguiente ejemplo muestra la utilización de ambas funciones:

```
<HTML>
<HEAD>
   <TITLE>Trabajando con Cadenas</TITLE>
</HEAD>
<BODY>
  <CENTER>
  <H2>Funciones <I>strrev</I> y <I>str_repeat</I></H2>
  <?php
    $cadena="0123456789";
    echo "<TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
    echo "<TR ALIGN='center'>";
    echo "<TD BGCOLOR='yellow'>cadena</TD><TD>$cadena</TD>\n";

    echo "<TR ALIGN='center'>";
    echo "<TD BGCOLOR='yellow'>str_repeat(strrev(cadena),2)</TD>";
    echo "<TD>".str_repeat(strrev($cadena),2)."</TD></TR>\n";

    echo "</TABLE>\n";
  ?>
 </CENTER>
</BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



CAPÍTULO 5

ARRAYS

Los *arrays* o **matrices** forman una parte muy importante de la programación en PHP ya que permiten manejar grupos de valores relacionados: nos permiten almacenar múltiples valores en una sola estructura y, de esta forma, asociarlos bajo una misma denominación. Como veremos, PHP tiene gran cantidad de funciones cuyos parámetros, tanto de llamada como de resultados, son variables de tipo *array*. En especial, son ampliamente utilizados en las funciones ligadas a las bases de datos.

En este capítulo profundizaremos en los distintos tipos de *arrays* que podemos definir, escalares y asociativos, tanto unidimensionales como multidimensionales, y las distintas formas de hacerlo. Veremos las principales funciones para su manejo y tratamiento.

5.1 *ARRAYS* ESCALARES

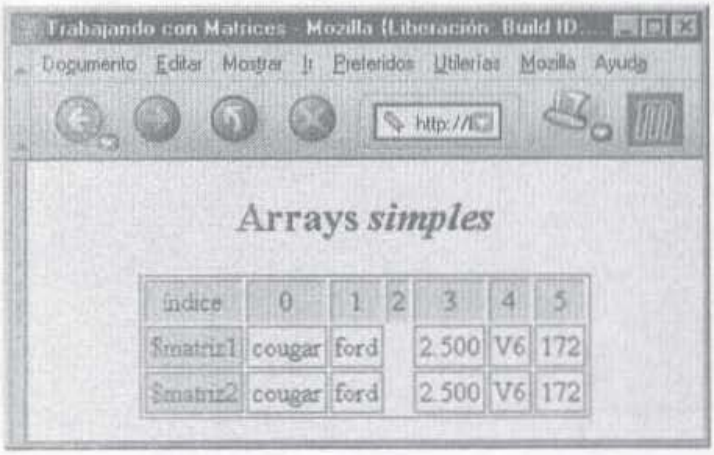
Un *array* escalar, o simple, está formado por un conjunto de valores ordenados respecto a un índice de tipo entero. Este índice indicará la posición del elemento dentro de esta colección ordenada, de modo que, en cada posición marcada por el índice dentro del *array*, haya un valor.

Existen diferentes formas de crear *arrays*. La más sencilla consiste en asignar el valor de cada elemento de manera explícita, es decir, indicando cada uno de los valores que lo componen e, incluso, la posición que ocupan dentro del *array*. El siguiente ejemplo nos muestra estas dos formas equivalentes de definir el mismo *array*:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays <I>simples</I></H2>
     <?php

       $matriz1[0]="cougar";
       $matriz1[1]="ford";
       $matriz1[3]="2.500";
       $matriz1[4]="V6";
       $matriz1[5]=172;

       $matriz2[]="cougar";
       $matriz2[]="ford";
       $matriz2[]="";
       $matriz2[]="2.500";
       $matriz2[]="V6";
       $matriz2[]=172;

     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
         <TD>indice</TD>
         <?php
           for($i=0;$i<=5;$i++)
               echo "<TD>$i</TD>";
         ?>
       <TR ALIGN="center">
         <TD BGCOLOR="yellow">$matriz1</TD>
          <?php
            echo "<TD> $matriz1[0] </TD>";
            echo "<TD> $matriz1[1] </TD>";
            echo "<TD> $matriz1[2] </TD>";
            echo "<TD> $matriz1[3] </TD>";
            echo "<TD> $matriz1[4] </TD>";
            echo "<TD> $matriz1[5] </TD>";
          ?>
       </TR>
       <TR ALIGN="center">
         <TD BGCOLOR="yellow">$matriz2</TD>
          <?php
            echo "<TD> $matriz2[0] </TD>";
            echo "<TD> $matriz2[1] </TD>";
            echo "<TD> $matriz2[2] </TD>";
            echo "<TD> $matriz2[3] </TD>";
            echo "<TD> $matriz2[4] </TD>";
            echo "<TD> $matriz2[5] </TD>";
          ?>
```

```
      </TR>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

Como observamos en el código anterior, un *array* puede combinar elementos de naturaleza distinta: en el ejemplo, valores enteros y cadenas de caracteres (que deben aparecer entrecomillados) e, incluso, elementos vacíos. El resultado se muestra en la siguiente imagen:

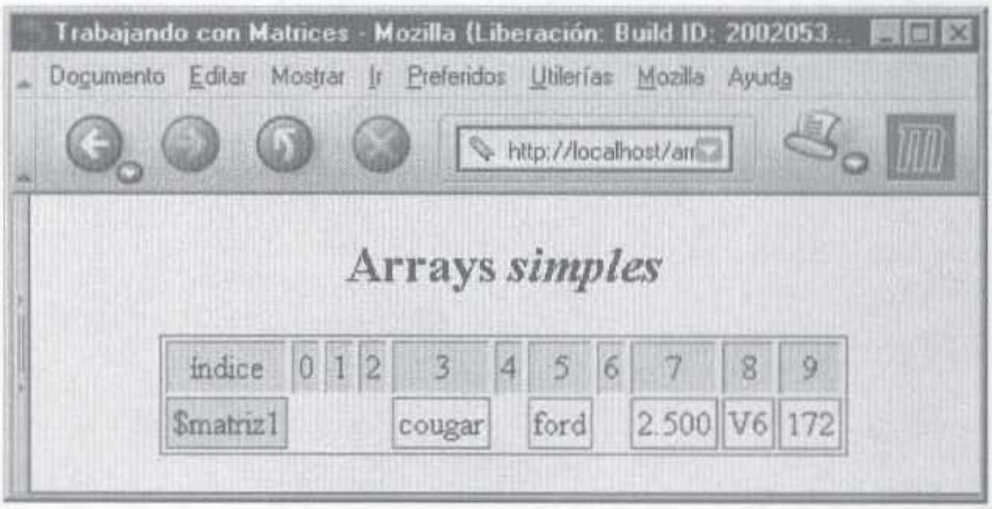


Cuando, al generar el *array*, no indicamos la posición de sus elementos, éstos se van situando secuencialmente respecto a la última asignación realizada sobre el *array*. La asignación numérica de posiciones dentro del *array* no tiene por qué ser secuencial, es decir, podemos definir el orden numérico que nos interese. El siguiente ejemplo nos muestra estas realizaciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays <I>simples</I></H2>
     <?php

       $matriz1[3]="cougar";
       $matriz1[5]="ford";
       $matriz1[7]="2.500";
       $matriz1[ ]="V6";
       $matriz1[ ]=172;

     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
        <TR ALIGN="center" BGCOLOR="yellow">
           <TD>indice</TD>
```

```
            <?php
              for($i=0;$i<=9;$i++)
                  echo "<TD>$i</TD>";
            ?>
        </TR>
        <TR ALIGN="center">
           <TD BGCOLOR="yellow">$matriz1</TD>
            <?php
              echo "<TD> $matriz1[0] </TD>";
              echo "<TD> $matriz1[1] </TD>";
              echo "<TD> $matriz1[2] </TD>";
              echo "<TD> $matriz1[3] </TD>";
              echo "<TD> $matriz1[4] </TD>";
              echo "<TD> $matriz1[5] </TD>";
              echo "<TD> $matriz1[6] </TD>";
              echo "<TD> $matriz1[7] </TD>";
              echo "<TD> $matriz1[8] </TD>";
              echo "<TD> $matriz1[9] </TD>";
            ?>
        </TR>
     </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

No debe confundirnos el hecho de que tengamos valores guardados en la posición 9 del *array* y pensar que hemos definido un *array* de nueve elementos, pues estaríamos en un error, ya que con esta definición nuestro *array* sigue siendo de cinco elementos, tal y como muestra la siguiente imagen:



La otra forma de definir *arrays* es utilizar el constructor `array()` proporcionado por el lenguaje. Este constructor no es una función regular; tiene la siguiente definición:

```
array array (mixed valores,...)
```

El siguiente ejemplo nos muestra cómo utilizar este constructor para definir el mismo *array* que en ejemplos anteriores:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>El constructor <I>array</I></H2>
     <?php
       $matriz1 = array("cougar","ford",null,"2.500","V6",172);
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
        <TR ALIGN="center" BGCOLOR="yellow">
           <TD>índice</TD>
           <?php
             for($i=0;$i<=5;$i++)
                 echo "<TD>$i</TD>";
           ?>
        </TR>
        <TR ALIGN="center">
           <TD BGCOLOR="yellow">$matriz1</TD>
           <?php
             echo "<TD> $matriz1[0] </TD>";
             echo "<TD> $matriz1[1] </TD>";
             echo "<TD> $matriz1[2] </TD>";
             echo "<TD> $matriz1[3] </TD>";
             echo "<TD> $matriz1[4] </TD>";
             echo "<TD> $matriz1[5] </TD>";
           ?>
        </TR>
     </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

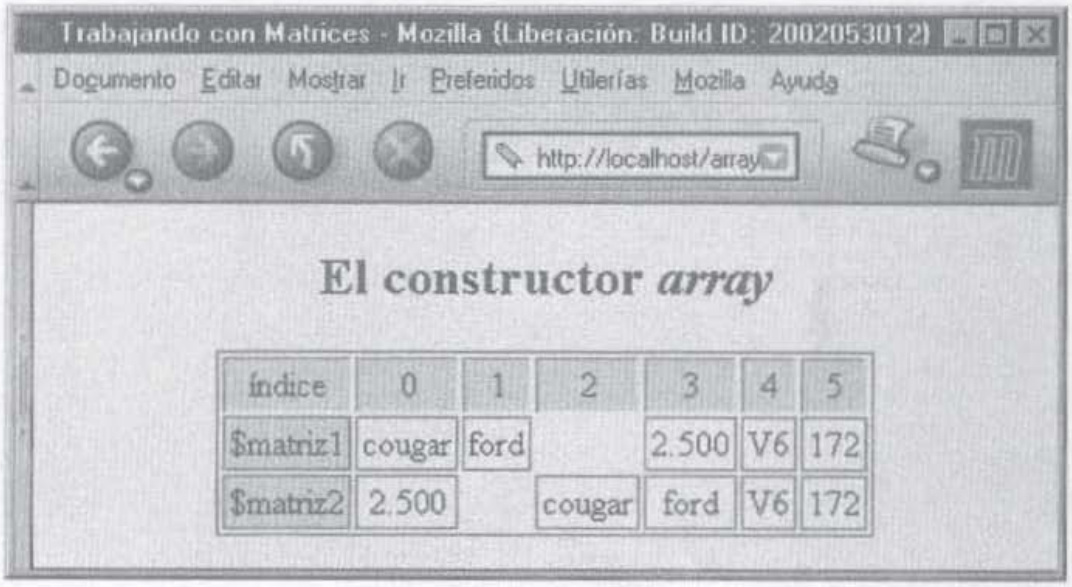
Como podemos observar en la siguiente imagen, los índices se asignan, por defecto, de modo secuencial comenzando por la posición 0:

El constructor `array()` también nos permite asignar los elementos del *array* en el orden en que queramos. Para ello, indicamos el índice, seguido del símbolo "=>" y el valor del elemento. Por ejemplo:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>El constructor <I>array</I></H2>
     <?php
       $matriz1 = array("cougar","ford",null,"2.500","V6",172);
       $matriz2 = array(2=>"cougar","ford",1=>null,0=>"2.500","V6",172);
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
          <TD>Indice</TD>
          <?php
            for($i=0;$i<=5;$i++)
                echo "<TD>$i</TD>";
          ?>
       </TR>
       <TR ALIGN="center">
          <TD BGCOLOR="yellow">$matriz1</TD>
          <?php
            echo "<TD> $matriz1[0] </TD>";
            echo "<TD> $matriz1[1] </TD>";
            echo "<TD> $matriz1[2] </TD>";
            echo "<TD> $matriz1[3] </TD>";
            echo "<TD> $matriz1[4] </TD>";
            echo "<TD> $matriz1[5] </TD>";
          ?>
       </TR>
       <TR ALIGN="center">
          <TD BGCOLOR="yellow">$matriz2</TD>
          <?php
```

```
            echo "<TD> $matriz2[0] </TD>";
            echo "<TD> $matriz2[1] </TD>";
            echo "<TD> $matriz2[2] </TD>";
            echo "<TD> $matriz2[3] </TD>";
            echo "<TD> $matriz2[4] </TD>";
            echo "<TD> $matriz2[5] </TD>";
        ?>
      </TR>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

Como se muestra en la siguiente imagen, los elementos a los que no se les asigna explícitamente un índice toman la posición secuencial relativa a la última asignación de posición dentro del *array*:



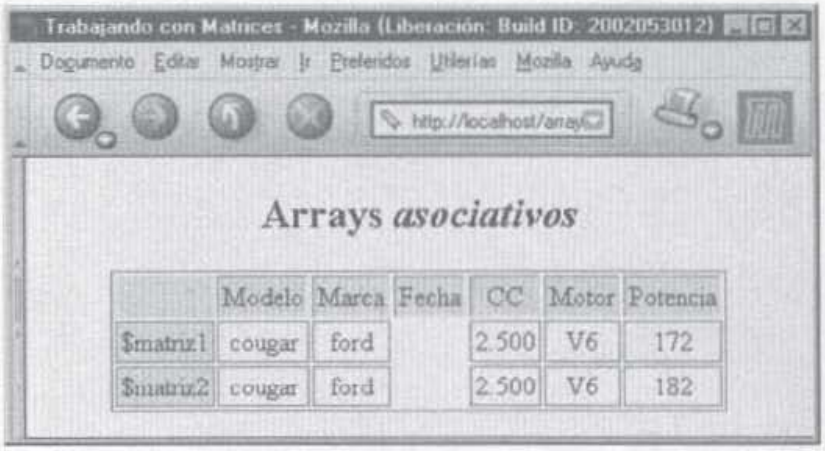
5.2 *ARRAYS* ASOCIATIVOS

A diferencia de los *arrays* simples, los ***arrays* asociativos** (también conocidos como tablas *hash* o *arrays* indexados por cadena) están formados por un conjunto de valores que están ordenados respecto a un índice de tipo *string*, es decir, una cadena de caracteres. De este modo, nuestro *array* va a estar compuesto por pares clave-valor, siendo necesario proporcionar la clave para poder acceder al valor almacenado en el *array*.

De igual forma que ocurre con los *arrays* simples, podemos utilizar el constructor del lenguaje `array()` para definirlos, o bien, hacerlo especificando de forma explícita cada uno de sus componentes. El siguiente ejemplo nos muestra ambas posibilidades:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays <I>asociativos</I></H2>
     <?php
       $matriz1 = array("modelo"=>"cougar","marca"=>"ford",
                        "fecha"=>null,"cc"=>"2.500","motor"=>"V6",
                        "potencia"=>172);

       $matriz2['modelo']="cougar";
       $matriz2['marca']="ford";
       $matriz2['fecha']=null;
       $matriz2['cc']="2.500";
       $matriz2['motor']="V6";
       $matriz2['potencia']=182;
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
        <TR ALIGN="center" BGCOLOR="yellow">
           <TD></TD>
           <TD>Modelo</TD>
           <TD>Marca</TD>
           <TD>Fecha</TD>
           <TD>CC</TD>
           <TD>Motor</TD>
           <TD>Potencia</TD>
        </TR>
        <TR ALIGN="center">
           <TD BGCOLOR="yellow">$matriz1</TD>
           <?php
             echo "<TD>".$matriz1['modelo']."</TD>";
             echo "<TD>".$matriz1['marca']."</TD>";
             echo "<TD>".$matriz1['fecha']."</TD>";
             echo "<TD>".$matriz1['cc']."</TD>";
             echo "<TD>".$matriz1['motor']."</TD>";
             echo "<TD>".$matriz1['potencia']."</TD>";
           ?>
        </TR>
        <TR ALIGN="center">
           <TD BGCOLOR="yellow">$matriz2</TD>
           <?php
             echo "<TD>".$matriz2['modelo']."</TD>";
             echo "<TD>".$matriz2['marca']."</TD>";
             echo "<TD>".$matriz2['fecha']."</TD>";
             echo "<TD>".$matriz2['cc']."</TD>";
             echo "<TD>".$matriz2['motor']."</TD>";
             echo "<TD>".$matriz2['potencia']."</TD>";
           ?>
        </TR>
     </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra los dos *arrays* definidos en el código anterior:



5.3 *ARRAYS* MULTIDIMENSIONALES

PHP nos permite definir *arrays* multidimensionales mediante la combinación de *arrays* unidimensionales (que pueden ser tanto de tipo escalar, como asociativos). Los siguientes ejemplos nos muestran las diferentes formas de definirlos.

- *Array* multidimensional de tipo escalar:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays <I>multidimensionales</I></H2>
     <?php
       $matriz1[0][0] = "peseta";
       $matriz1[0][1] = 166.386;
       $matriz1[1][0] = "dolar";
       $matriz1[1][1] = 0.96;

       $matriz2[0] = array("peseta",166.386);
       $matriz2[1] = array("dolar",0.96);

       $matriz3 = array(array("peseta",166.386),array("dolar",0.96));
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
         <TD></TD>
         <TD>Moneda</TD>
         <TD>Cambio €</TD>
       </TR>
       <?php
         for($i=0;$i<2;$i++){
           echo "<TR ALIGN='center'>";
```

```
                echo "<TD BGCOLOR='yellow'>\$matriz1[$i]</TD>";
                for($j=0;$j<2;$j++){
                    echo "<TD>".$matriz1[$i][$j]."</TD>";
                }
                echo "</TR>";
            }
            for($i=0;$i<2;$i++){
                echo "<TR ALIGN='center'>";
                echo "<TD BGCOLOR='yellow'>\$matriz2[$i]</TD>";
                for($j=0;$j<2;$j++){
                    echo "<TD>".$matriz2[$i][$j]."</TD>";
                }
                echo "</TR>";
            }
            for($i=0;$i<2;$i++){
                echo "<TR ALIGN='center'>";
                echo "<TD BGCOLOR='yellow'>\$matriz3[$i]</TD>";
                for($j=0;$j<2;$j++){
                    echo "<TD>".$matriz3[$i][$j]."</TD>";
                }
                echo "</TR>";
            }
        ?>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

- *Array* multidimensional de tipo asociativo:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays <I>multidimensionales</I></H2>
     <?php
       $matriz1['peseta']['moneda'] = "peseta";
       $matriz1['peseta']['cambio'] = 166.386;
       $matriz1['dolar']['moneda'] = "dolar";
       $matriz1['dolar']['cambio'] = 0.96;

       $matriz2['peseta']=array("moneda"=>"peseta","cambio"=>166.386);
       $matriz2['dolar']=array("moneda"=>"dolar","cambio"=>0.96);

       $matriz3 =
array("peseta"=>array("moneda"=>"peseta","cambio"=>166.386),
                     "dolar"=>array("moneda"=>"dolar","cambio"=>0.96));
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
         <TD></TD>
         <TD>Moneda</TD>
         <TD>Cambio €</TD>
       </TR>
       <?php
         echo "<TR ALIGN='center'>";
         echo "<TD BGCOLOR='yellow'>\$matriz1</TD>";
         echo "<TD>".$matriz1['peseta']['moneda']."</TD>";
         echo "<TD>".$matriz1['peseta']['cambio']."</TD>";
```

```
            echo "</TR>";
            echo "<TR ALIGN='center'>";
            echo "<TD BGCOLOR='yellow'>\$matriz1</TD>";
            echo "<TD>".$matriz1['dolar']['moneda']."</TD>";
            echo "<TD>".$matriz1['dolar']['cambio']."</TD>";
            echo "</TR>";
            echo "<TR ALIGN='center'>";
            echo "<TD BGCOLOR='yellow'>\$matriz2</TD>";
            echo "<TD>".$matriz2['peseta']['moneda']."</TD>";
            echo "<TD>".$matriz2['peseta']['cambio']."</TD>";
            echo "</TR>";
            echo "<TR ALIGN='center'>";
            echo "<TD BGCOLOR='yellow'>\$matriz2</TD>";
            echo "<TD>".$matriz2['dolar']['moneda']."</TD>";
            echo "<TD>".$matriz2['dolar']['cambio']."</TD>";
            echo "</TR>";
            echo "<TR ALIGN='center'>";
            echo "<TD BGCOLOR='yellow'>\$matriz3</TD>";
            echo "<TD>".$matriz3['peseta']['moneda']."</TD>";
            echo "<TD>".$matriz3['peseta']['cambio']."</TD>";
            echo "</TR>";
            echo "<TR ALIGN='center'>";
            echo "<TD BGCOLOR='yellow'>\$matriz3</TD>";
            echo "<TD>".$matriz3['dolar']['moneda']."</TD>";
            echo "<TD>".$matriz3['dolar']['cambio']."</TD>";
            echo "</TR>";
        ?>
      </TABLE>
    </CENTER>
  </BODY>
</HTML>
```

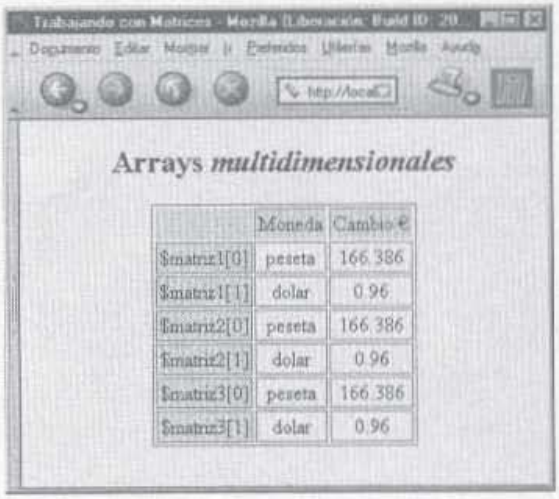
NOTA: En PHP3 no es posible referirse a *arrays* multidimensionales directamente dentro de cadenas, es necesario hacer uso del operador de concatenación para solucionar este problema. En PHP4, sin embargo, todo el problema se soluciona encerrando la referencia al *array* (dentro de la cadena) entre llaves; también se puede hacer uso de la concatenación de cadenas.

- *Array* multidimensional combinado:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays <I>multidimensionales</I></H2>
     <?php
       $matriz1[0]['moneda'] = "peseta";
       $matriz1[0]['cambio'] = 166.386;
       $matriz1[1]['moneda'] = "dolar";
       $matriz1[1]['cambio'] = 0.96;

       $matriz2[0]=array("moneda"=>"peseta","cambio"=>166.386);
       $matriz2[1]=array("moneda"=>"dolar","cambio"=>0.96);

       $matriz3 = array(array("moneda"=>"peseta","cambio"=>166.386),
                        array("moneda"=>"dolar","cambio"=>0.96));
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
```

```
            <TD></TD>
            <TD>Moneda</TD>
            <TD>Cambio €</TD>
        </TR>
        <?php
            for($i=0;$i<2;$i++){
                echo "<TR ALIGN='center'>";
                echo "<TD BGCOLOR='yellow'>\$matriz1</TD>";
                echo "<TD>".$matriz1[$i]['moneda']."</TD>";
                echo "<TD>".$matriz1[$i]['cambio']."</TD>";
                echo "</TR>";
            }
            for($i=0;$i<2;$i++){
                echo "<TR ALIGN='center'>";
                echo "<TD BGCOLOR='yellow'>\$matriz2</TD>";
                echo "<TD>".$matriz2[$i]['moneda']."</TD>";
                echo "<TD>".$matriz2[$i]['cambio']."</TD>";
                echo "</TR>";
            }
            for($i=0;$i<2;$i++){
                echo "<TR ALIGN='center'>";
                echo "<TD BGCOLOR='yellow'>\$matriz3</TD>";
                echo "<TD>".$matriz3[$i]['moneda']."</TD>";
                echo "<TD>".$matriz3[$i]['cambio']."</TD>";
                echo "</TR>";
            }
        ?>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra la salida de los tres ejemplos: observamos que los tres *arrays*, definidos en el código de diferentes modos, son equivalentes:



5.4 RECORRER UN *ARRAY*

Como hemos visto en los ejemplos anteriores, una operación habitual a realizar cuando trabajamos con *arrays* es recorrerlo para obtener sus elementos, para modificarlos o trabajar con ellos.

5.4.1 Recorridos en *arrays* secuenciales

De los ejemplos anteriores podemos destacar que recorrer los diferentes elementos que componen un *array* secuencial es relativamente sencillo, pues, haciendo uso de un bucle e iterando en función del valor del índice, podemos realizar la operación. El problema principal en este tipo de recorridos es conocer a priori el número de elementos que componen el *array*. Precisamente para solventar este problema, PHP proporciona la función `count()`, que devuelve el número de elementos que tiene la variable que recibe como argumento.

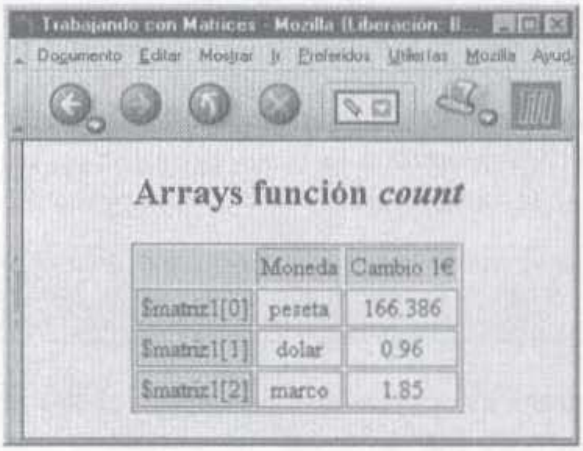
> **NOTA:** Aunque es habitual que el argumento pasado a la función sea un *array*, podemos aplicar la función a cualquier otro tipo de variable, en cuyo caso devolverá el valor 1, si la variable, tiene valor o 0, si ésta aún no ha sido inicializada.

Una vez conocido el número de elementos del *array*, podemos utilizar un bucle para ir recorriendo sus elementos, tal y como muestra el siguiente ejemplo, en el que nos permite recorrer un *array* multidimensional:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays función <I>count</I></H2>
     <?php
       $matriz1[0][0] = "peseta";
       $matriz1[0][1] = 166.386;
       $matriz1[1][0] = "dolar";
       $matriz1[1][1] = 0.96;
       $matriz1[2][0] = "marco";
       $matriz1[2][1] = 1.85;
     ?>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
         <TD></TD>
         <TD>Moneda</TD>
         <TD>Cambio 1€</TD>
       </TR>
       <?php
         for($i=0;$i<count($matriz1);$i++){
           echo "<TR ALIGN='center'>";
           echo "<TD BGCOLOR='yellow'>\$matriz1[$i]</TD>";
```

```
                for($j=0;$j<count($matriz1[$i]);$j++){
                    echo "<TD>".$matriz1[$i][$j]."</TD>";
                }
                echo "</TR>";
            }
        ?>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra el resultado de realizar el recorrido:



Otra función relacionada con el recorrido de *arrays* es `sizeof()`, que obtiene el número de elementos del *array* pasado como argumento en la llamada a la función. El ejemplo anterior podía haberse resuelto haciendo uso de esta función: bastaría con sustituir la línea:

```
for($i=0; $i<count($matriz1); $i++){
```

Por ésta:

```
for($i=0; $i<sizeof($matriz1); $i++){
```

5.4.2 Recorridos en *arrays* no secuenciales

A primera vista, un *array* no secuencial no parece muy fácil de recorrer, puesto que no sólo necesitamos saber el número de elementos que componen el *array*, sino también su posición dentro del *array*. Este mismo problema se presenta con los *arrays* asociativos, puesto que, además del número de elementos que componen el *array*, necesitamos conocer las claves para poder acceder a los valores almacenados. Para poder trabajar con *arrays* estructurados de modo no secuencial, tanto de tipo escalar como asociativos, PHP cuenta con el siguiente conjunto de

funciones (estas funciones también son aplicables a *arrays* estructurados secuencialmente):

- `current(matriz)`: Devuelve el valor de la posición actual del puntero dentro del *array*: todos los *arrays* tienen un puntero interno que apunta a la posición del elemento *actual* con el que se está trabajando en un momento dado. Devuelve `false` cuando el puntero está al final del *array* o cuando el *array* no contiene ningún elemento.

- `pos(matriz)`: Es idéntica a la funcióna anterior

- `key(matriz)`: Devuelve el índice de la posición actual del *array* pasado como argumento: un número, en caso de que el *array* sea de tipo escalar, o una cadena de caracteres, en el caso de que el *array* sea de tipo asociativo.

- `next(matriz)`: Devuelve el valor del elemento siguiente al actual (si existe) y avanza el puntero interno una posición. En caso de que el elemento actual sea el último del *array*, devuelve `false`.

- `prev(matriz)`: Devuelve el valor del elemento anterior al actual (si existe) y retrocede el puntero interno una posición. En caso de que el elemento actual sea el primero del *array*, devuelve `false`.

- `end(matriz)`: Coloca el puntero interno en el último elemento de un *array* escalar.

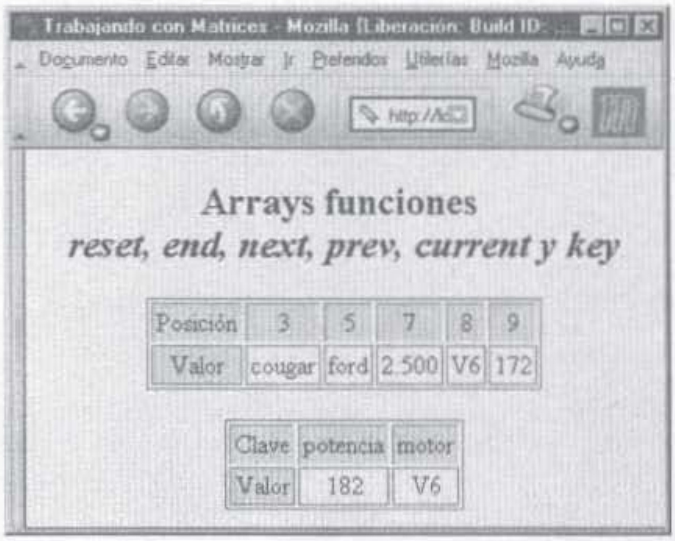
- `reset(matriz)`: Devuelve el valor del primer elemento del *array* y sitúa el puntero interno en su primera posición.

El siguiente ejemplo muestra la utilización de alguna de estas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays funciones <BR>
     <I>reset, end, next, prev, current y key</I></H2>
     <?php
       $matriz1[3]="cougar";
       $matriz1[5]="ford";
       $matriz1[7]="2.500";
       $matriz1[ ]="V6";
```

```
        $matriz1[ ]=172;

        $matriz2['modelo']="cougar";
        $matriz2['marca']="ford";
        $matriz2['cc']="2.500";
        $matriz2['fecha']=null;
        $matriz2['motor']="V6";
        $matriz2['potencia']=182;
    ?>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Posición</TD>
      <?php
          echo "<TD BGCOLOR='yellow'>".key($matriz1)."</TD>";
          while(next($matriz1)){
              echo "<TD BGCOLOR='yellow'>".key($matriz1)."</TD>";
          }
      ?>
      </TR>
      <TR ALIGN="center">
      <TD BGCOLOR="yellow">Valor</TD>
      <?php
          echo "<TD>".reset($matriz1)."</TD>";
          while(next($matriz1)){
              echo "<TD>".current($matriz1)."</TD>";
          }
      ?>
    </TABLE><BR>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Clave</TD>
      <?php
          end($matriz2);
          echo "<TD BGCOLOR='yellow'>".key($matriz2)."</TD>";
          while(prev($matriz2)){
              echo "<TD BGCOLOR='yellow'>".key($matriz2)."</TD>";
          }
      ?>
      </TR>
      <TR ALIGN="center">
      <TD BGCOLOR="yellow">Valor</TD>
      <?php
          end($matriz2);
          echo "<TD>".current($matriz2)."</TD>";
          while(prev($matriz2)){
              echo "<TD>".current($matriz2)."</TD>";
          }
      ?>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en la siguiente imagen, el recorrido del segundo *array* (en orden inverso) se detiene en el instante en que se encuentra un elemento vacío, mientras que el primero se muestra completamente:



- `each(matriz)`: Se usa para recorrer *arrays* (sobre todo los asociativos), pues devuelve un par de valores correspondientes a la clave y al valor asociado a esa clave. Además, avanza el puntero interno hasta el siguiente elemento. Si el puntero interno apunta a la última posición del *array*, la ejecución de esta función devuelve `false`.

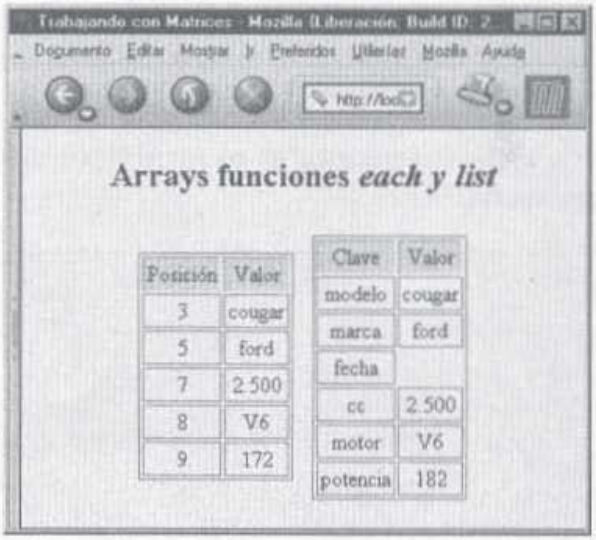
- `list()`: Asigna una lista de variables en una sola operación. Suele utilizarse en combinación con la función anteriormente vista, `each()`.

El siguiente ejemplo muestra el recorrido por dos *arrays* no secuenciales haciendo uso de ambas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
    <H2>Arrays funciones <I>each y list</I></H2>
    <?php
      $matriz1[3]="cougar";
      $matriz1[5]="ford";
      $matriz1[7]="2.500";
      $matriz1[ ]="V6";
      $matriz1[ ]=172;

      $matriz2['modelo']="cougar";
      $matriz2['marca']="ford";
      $matriz2['fecha']=null;
      $matriz2['cc']="2.500";
      $matriz2['motor']="V6";
      $matriz2['potencia']=182;
    ?>
   <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
    <TR ALIGN="center"><TD>
```

```
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Posición</TD><TD>Valor</TD></TR>
      <?php
          while(list($pos,$valor)=each($matriz1)){
              echo "<TR ALIGN='center'><TD>".$pos."</TD>";
              echo "<TD>".$valor."</TD></TR>";
          }
      ?>
    </TABLE></TD><TD>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Clave</TD><TD>Valor</TD></TR>
      <?php
          while(list($clave,$valor)=each($matriz2)){
              echo "<TR ALIGN='center'><TD>".$clave."</TD>";
              echo "<TD>".$valor."</TD></TR>";
          }
      ?>
    </TABLE></TD></TR>
   </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en la siguiente imagen, el recorrido de ambos *arrays* es completo independientemente de que tengan elementos vacíos:



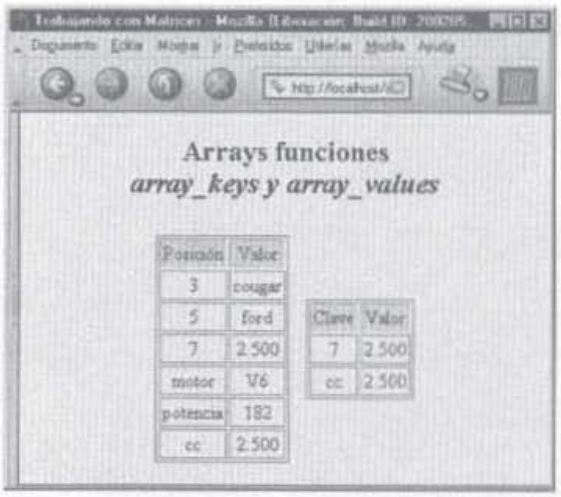
- `array_keys(matriz)`: Devuelve las claves que forman el *array* `matriz`. Admite un parámetro opcional que nos permite seleccionar sólo las claves cuyo valor coincida con un patrón dado.

- `array_values(matriz)` : Devuelve los valores que forman parte del *array* pasado como parámetro.

El siguiente ejemplo muestra el recorrido por dos *arrays* no secuenciales haciendo uso de estas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays funciones<BR><I>array_keys y array_values</I></H2>
     <?php
       $matriz1[3]="cougar";
       $matriz1[5]="ford";
       $matriz1[7]="2.500";
       $matriz1['motor']="V6";
       $matriz1['potencia']=182;
       $matriz1['cc']="2.500";
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center"><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Posición</TD><TD>Valor</TD></TR>
       <?php
           $claves=array_keys($matriz1);
           $valores=array_values($matriz1);
           for($i=0;$i<sizeof($claves);$i++){
               echo "<TR ALIGN='center'><TD>".$claves[$i]."</TD>";
               echo "<TD>".$valores[$i]."</TD></TR>";
           }
       ?>
     </TABLE></TD><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Clave</TD><TD>Valor</TD></TR>
       <?php
           $claves=array_keys($matriz1,"2.500");
           for($i=0;$i<sizeof($claves);$i++){
               echo "<TR ALIGN='center'><TD>".$claves[$i]."</TD>";
               echo "<TD>".$matriz1[$claves[$i]]."</TD></TR>";
           }
       ?>
     </TABLE></TD></TR>
    </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

En la siguiente imagen vemos el resultado del código:



5.5 ORDENAR UN *ARRAY*

Veremos a continuación varias funciones que PHP pone a nuestra disposición para poder ordenar los elementos de un *array*, ya que es frecuente que necesitemos tener los elementos de un *array* ordenados para después, por ejemplo, poder listarlos en orden alfabético.

- `sort(matriz)`: Ordena alfanuméricamente los valores de los elementos de un *array* de menor a mayor. Para ordenar de manera inversa, disponemos de la función `rsort(matriz)`.

NOTA: Si aplicamos estas funciones a *arrays* asociativos, éstos perderán sus claves y se convertirán en *arrays* simples o escalares.

El siguiente ejemplo muestra la utilización de ambas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays funciones <I>sort y rsort</I></H2>
     <?php
       $matriz1[0]="Madrid";
       $matriz1[1]="Zaragoza";
       $matriz1[2]="Bilbao";
       $matriz1[3]="Valencia";
       $matriz1[4]="Lerida";
       $matriz1[5]="Alicante";
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center">
     <TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
```

```
    <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>Array Original</TH>
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Posición</TD><TD>Valor</TD></TR>
      <?php
          while(list($pos,$valor)=each($matriz1)){
              echo "<TR ALIGN='center'><TD>".$pos."</TD>";
              echo "<TD>".$valor."</TD></TR>";
          }
      ?>
    </TABLE></TD>
    <TD>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
    <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>sort()</TH>
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Posición</TD><TD>Valor</TD></TR>
      <?php
          sort($matriz1);
          while(list($pos,$valor)=each($matriz1)){
              echo "<TR ALIGN='center'><TD>".$pos."</TD>";
              echo "<TD>".$valor."</TD></TR>";
          }
      ?>
    </TABLE></TD><TD>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
    <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>rsort()</TH>
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Posición</TD><TD>Valor</TD></TR>
      <?php
          rsort($matriz1);
          while(list($pos,$valor)=each($matriz1)){
              echo "<TR ALIGN='center'><TD>".$pos."</TD>";
              echo "<TD>".$valor."</TD></TR>";
          }
      ?>
    </TABLE></TD>
    </TR>
   </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra el resultado de ordenar el *array* del código ascendente y descendentemente; en ella observamos que ambas funciones, al ordenar los valores del *array*, ordenan en realidad los índices:

Arrays funciones *sort y rsort*

Array Original		sort()		rsort()	
Posición	Valor	Posición	Valor	Posición	Valor
0	Madrid	0	Alicante	0	Zaragoza
1	Zaragoza	1	Bilbao	1	Valencia
2	Bilbao	2	Lerida	2	Madrid
3	Valencia	3	Madrid	3	Lerida
4	Lerida	4	Valencia	4	Bilbao
5	Alicante	5	Zaragoza	5	Alicante

Para evitar este efecto lateral en el que se redefinen los índices, tanto para los *arrays* escalares como asociativos, podemos usar las siguientes funciones:

- `asort(matriz)`: Ordena alfanuméricamente los valores de los elementos de un *array* de mayor a menor, pero manteniendo la relación existente entre los índices y sus valores asociados. Esto es posible debido a que la ordenación se hace sobre los elementos del *array* y no sobre los índices.

- `arsort(matriz)`: realiza la ordenación inversa.

El siguiente ejemplo muestra la utilización de estas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays funciones <I>asort y arsort</I></H2>
     <?php
       $matriz1[0]="Madrid";
       $matriz1[1]="Zaragoza";
       $matriz1[2]="Bilbao";
       $matriz1[3]="Valencia";
       $matriz1[4]="Lerida";
       $matriz1[5]="Alicante";
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center"><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>Array
Original</TH>
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Posición</TD><TD>Valor</TD>
```

```
        <?php
            while(list($pos,$valor)=each($matriz1)){
                echo "<TR ALIGN='center'><TD>".$pos."</TD>";
                echo "<TD>".$valor."</TD></TR>";
            }
        ?>
    </TABLE></TD>
    <TD>

    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
     <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>asort()</TH>
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Posición</TD><TD>Valor</TD></TR>
      <?php
          asort($matriz1);
          while(list($pos,$valor)=each($matriz1)){
              echo "<TR ALIGN='center'><TD>".$pos."</TD>";
              echo "<TD>".$valor."</TD></TR>";
          }
      ?>
    </TABLE></TD><TD>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
     <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>arsort()</TH>
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Posición</TD><TD>Valor</TD></TR>
      <?php
          arsort($matriz1);
          while(list($pos,$valor)=each($matriz1)){
              echo "<TR ALIGN='center'><TD>".$pos."</TD>";
              echo "<TD>".$valor."</TD></TR>";
          }
      ?>
    </TABLE></TD></TR>
   </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

Veamos el resultado obtenido:

- `ksort(matriz)`: Ordena alfanuméricamente las claves de un *array* de menor a mayor manteniendo las correlaciones entre clave y valor asociado. La función que realiza la ordenación inversa es `krsort(matriz)`.

El siguiente ejemplo muestra la utilización de estas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays funciones <I>ksort y krsort</I></H2>
     <?php
       $matriz1[d]="Madrid";
       $matriz1[c]="Zaragoza";
       $matriz1[e]="Bilbao";
       $matriz1[b]="Valencia";
       $matriz1[f]="Lerida";
       $matriz1[a]="Alicante";
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center"><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Posición</TD><TD>Valor</TD></TR>
       <?php
           ksort($matriz1);
           while(list($pos,$valor)=each($matriz1)){
               echo "<TR ALIGN='center'><TD>".$pos."</TD>";
               echo "<TD>".$valor."</TD></TR>";
           }
       ?>
     </TABLE></TD><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Posición</TD><TD>Valor</TD></TR>
       <?php
           krsort($matriz1);
           while(list($pos,$valor)=each($matriz1)){
               echo "<TR ALIGN='center'><TD>".$pos."</TD>";
               echo "<TD>".$valor."</TD></TR>";
           }
       ?>
     </TABLE></TD></TR>
    </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:

Arrays funciones *ksort y krsort*

Array Original	
Posición	Valor
d	Madrid
c	Zaragoza
e	Bilbao
b	Valencia
f	Lerida
a	Alicante

ksort()	
Posición	Valor
a	Alicante
b	Valencia
c	Zaragoza
d	Madrid
e	Bilbao
f	Lerida

krsort()	
Posición	Valor
f	Lerida
e	Bilbao
d	Madrid
c	Zaragoza
b	Valencia
a	Alicante

- `usort(matriz, func_comparar)`: Ordena los valores de un *array* según el criterio de la función pasada por el usuario como argumento. La función hay que pasarla como una cadena de caracteres, es decir, entre comillas (las funciones se verán en profundidad en un capítulo posterior).

- `uksort (array matriz, func_comparar)`: Ordena las claves de un *array* en base a una función pasada por el usuario como argumento, manteniendo las correlaciones existentes entre clave y valor asociado. La función de comparación debe devolver un entero menor, igual o mayor que cero, si el primer argumento es, respectivamente, menor que, igual que o mayor que el segundo. Si los dos argumentos resultan ser iguales, su orden en la matriz ordenada será cualquiera.

El siguiente ejemplo muestra la utilización de estas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays funciones <I>usort y uksort</I></H2>
     <?php
       $matriz1[]="Madrid";
       $matriz1[]="Zaragoza";
       $matriz1[]="Bilbao";
       $matriz1[]="Valencia";
       $matriz1[]="Lerida";
       $matriz1[]="Alicante";
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center"><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>Array
Original</TH>
```

```
        <TR ALIGN="center" BGCOLOR="yellow">
        <TD>Posición</TD><TD>Valor</TD></TR>
        <?php
            while(list($pos,$valor)=each($matriz1)){
                echo "<TR ALIGN='center'><TD>".$pos."</TD>";
                echo "<TD>".$valor."</TD></TR>";
            }
        ?>
      </TABLE></TD>
      <TD>
      <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
        <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>usort()</TH>
        <TR ALIGN="center" BGCOLOR="yellow">
        <TD>Posición</TD><TD>Valor</TD></TR>
        <?php
            // ordena alfabéticamente en función de la última letra
            function ordenarValores($a,$b){
                $fina=$a[strlen($a)-1];
                $finb=$b[strlen($b)-1];
                if ($fina == $finb) return 0;
                return ($fina > $finb) ? 1 : -1;
            }

            usort($matriz1,"ordenarValores");
            while(list($pos,$valor)=each($matriz1)){
                echo "<TR ALIGN='center'><TD>".$pos."</TD>";
                echo "<TD>".$valor."</TD></TR>";
            }
        ?>
      </TABLE></TD><TD>
      <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
        <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>uksort()</TH>
        <TR ALIGN="center" BGCOLOR="yellow">
        <TD>Posición</TD><TD>Valor</TD></TR>
        <?php
            // ordena primero índices impares y después pares
            function ordenarClaves($a,$b){
                $a=$a%2;
                $b=$b%2;
                if ($a == $b) return 0;
                return ($a > $b) ? -1 : 1;
            }

            uksort($matriz1,"ordenarClaves");
            while(list($pos,$valor)=each($matriz1)){
                echo "<TR ALIGN='center'><TD>".$pos."</TD>";
                echo "<TD>".$valor."</TD></TR>";
            }
        ?>
      </TABLE></TD></TR>
    </TABLE>
  </CENTER>
</BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:

Arrays funciones *usort y uksort*

Array Original		usort()		uksort()	
Posición	Valor	Posición	Valor	Posición	Valor
0	Madrid	0	Lerida	3	Madrid
1	Zaragoza	1	Zaragoza	5	Bilbao
2	Bilbao	2	Valencia	1	Zaragoza
3	Valencia	3	Madrid	4	Alicante
4	Lerida	4	Alicante	2	Valencia
5	Alicante	5	Bilbao	0	Lerida

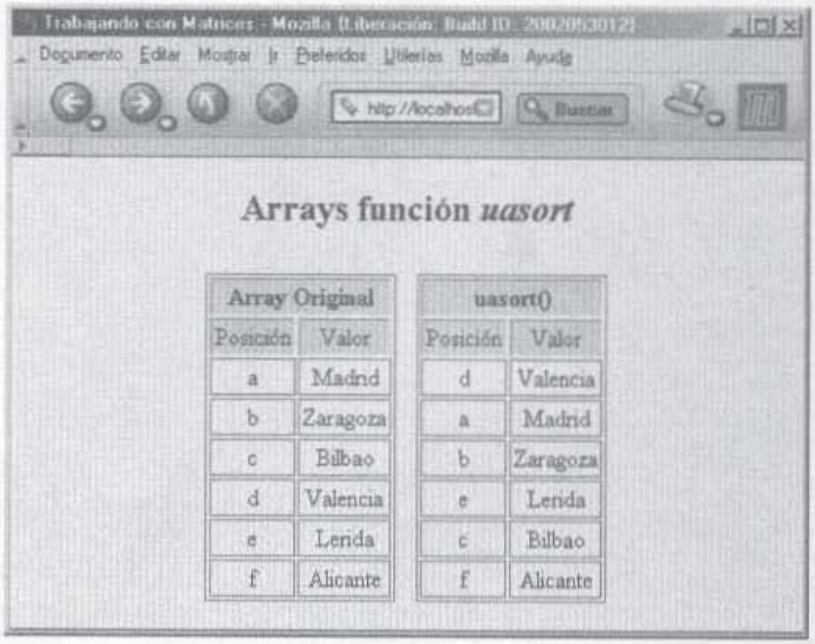
- `uasort(matriz, func_comparar)`: Ordena los valores de un *array* según el criterio de la función pasada por el usuario como argumento, manteniendo las correlaciones entre clave y valor asociado.

El siguiente ejemplo muestra la utilización de estas funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays función <I>uasort</I></H2>
     <?php
       $matriz1['a']="Madrid";
       $matriz1['b']="Zaragoza";
       $matriz1['c']="Bilbao";
       $matriz1['d']="Valencia";
       $matriz1['e']="Lerida";
       $matriz1['f']="Alicante";
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center"><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>Array
Original</TH>
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Posición</TD><TD>Valor</TD></TR>
       <?php
           while(list($pos,$valor)=each($matriz1)){
               echo "<TR ALIGN='center'><TD>".$pos."</TD>";
               echo "<TD>".$valor."</TD></TR>";
           }
       ?>
     </TABLE></TD><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>uasort()</TH>
       <TR ALIGN="center" BGCOLOR="yellow">
```

```
        <TD>Posición</TD><TD>Valor</TD></TR>
        <?php
            // ordena alfabéticamente en función de a la segunda letra
            function ordenarValores($a,$b){
                $fina=$a[1];
                $finb=$b[1];
                if ($fina == $finb) return 0;
                return ($fina > $finb) ? 1 : -1;
            }

            uasort($matriz1,"ordenarValores");
            while(list($pos,$valor)=each($matriz1)){
                echo "<TR ALIGN='center'><TD>".$pos."</TD>";
                echo "<TD>".$valor."</TD></TR>";
            }
        ?>
      </TABLE></TD></TR>
    </TABLE>
  </CENTER>
</BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



5.6 OTRAS OPERACIONES

Veremos a continuación un grupo de funciones que nos serán de gran utilidad a la hora de trabajar con un *array* ya existente.

5.6.1 Modificar un *array*

Las siguientes funciones nos permiten modificar el tamaño de un *array* añadiendo elementos o uniendo diferentes *arrays* en uno solo.

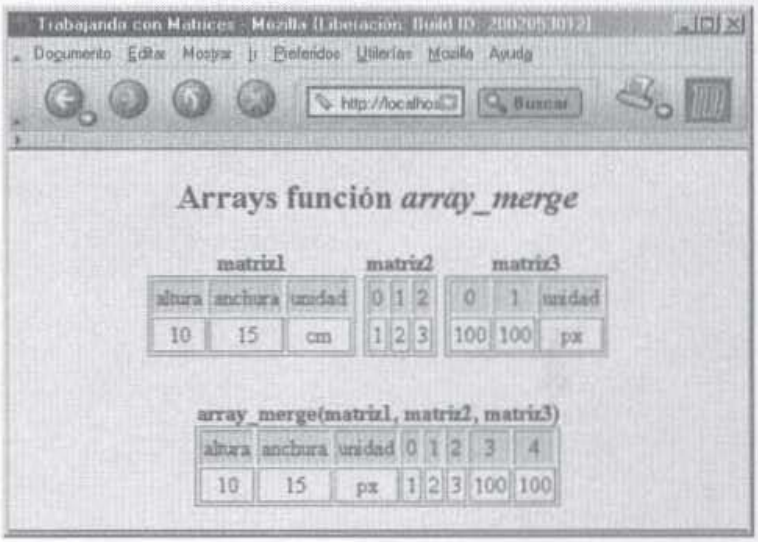
- `array_merge(matriz1, matriz2)`: Combina en un solo *array* los valores de los *arrays* que recibe como argumentos. Los elementos van siendo añadidos al final del *array* precedente. Si estamos juntando *arrays* asociativos, hay que tener en cuenta que las claves con igual nombre no se añaden al *array*, sino que se actualizan con el último valor suministrado.

El siguiente ejemplo muestra la utilización de esta función:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays función <I>array_merge</I></H2>
     <?php
       $matriz1=array("altura"=>"10","anchura"=>"15","unidad"=>"cm");
       $matriz2=array("1","2","3");
       $matriz3=array("100","100","unidad"=>"px");
       $matrizM=array_merge($matriz1,$matriz2,$matriz3);

       echo "<TABLE BORDER='0' CELLPADDING='2' CELLSPACING='2'>\n";
       echo " <TR ALIGN='center'>\n";
       for($j=1;$j<4;$j++){
           if ($j==1)$matriz=$matriz1;
           if ($j==2)$matriz=$matriz2;
           if ($j==3)$matriz=$matriz3;
           echo "    <TD><b>matriz$j</b>\n";
           echo "      <TABLE BORDER='1' CELLPADDING='2' CELLSPACING='2'>\n";
           echo "        <TR ALIGN='center' BGCOLOR='yellow'>\n";
           do{
              echo "<TD>".key($matriz)."</TD>";
           } while(next($matriz));
           echo "        </TR>\n";
           echo "        <TR ALIGN='center'>\n";
           reset($matriz);
           do{
              echo "<TD>".current($matriz)."</TD>";
           } while(next($matriz));
           echo "        </TR>\n";
           echo "      </TABLE>\n";
           echo "    </TD>\n";
       }
       echo "  </TR>\n";
       echo " <TR><TD COLSPAN='3'><BR></TD></TR>";
       echo " <TR ALIGN='center'>\n";
     echo " <TD COLSPAN='3'><b>array_merge(matriz1, matriz2, matriz3)</b>\n";
       echo "      <TABLE BORDER='1' CELLPADDING='2' CELLSPACING='2'>\n";
       echo "        <TR ALIGN='center' BGCOLOR='yellow'>\n";
       do{
```

```
            echo "<TD>".key($matrizM)."</TD>";
        }while(next($matrizM));
        echo "          </TR>\n";
        echo "          <TR ALIGN='center'>\n";
        reset($matrizM);
        do{
            echo "<TD>".current($matrizM)."</TD>";
        } while(next($matrizM));
        echo "          </TR>\n";
        echo "        </TABLE>\n";
        echo "    </TD>\n";
        echo "  </TR>\n";
        echo "</TABLE>\n";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra el resultado:



- `array_pad(matriz, tamaño, relleno)`: Devuelve un *array* igual a `matriz`, pero modificando su número de elementos hasta alcanzar la longitud deseada (`tamaño`) añadiendo nuevos elementos por la derecha (`tamaño` es positivo) o por la izquierda (`tamaño` es negativo).

El siguiente ejemplo muestra la utilización de esta función:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays función <I>array_pad</I></H2>
     <?php
```

```
        $matriz1=array("1","2","3");
        $matrizP1=array_pad($matriz1,-5,"-?-");
        $matrizP2=array_pad($matriz1,5,"-?-");
      ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center" VALIGN='top'><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow"><TH colspan='2'>Array
Original</TH>
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Posición</TD><TD>Valor</TD>
       <?php
           while(list($pos,$valor)=each($matriz1)){
               echo "<TR ALIGN='center'><TD>".$pos."</TD>";
               echo "<TD>".$valor."</TD></TR>";
           }
       ?>
     </TABLE></TD>
     <TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow"><TH
colspan='2'>array_pad($matriz1,-5,"-?-");</TH>
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Posición</TD><TD>Valor</TD></TR>
       <?php
           while(list($pos,$valor)=each($matrizP1)){
               echo "<TR ALIGN='center'";
               echo ($pos>1)?">":" BGCOLOR='#FFBBAA'>";
               echo "<TD>".$pos."</TD>";
               echo "<TD>".$valor."</TD></TR>";
           }
       ?>
     </TABLE></TD><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow"><TH
colspan='2'>array_pad($matriz1,5,"-?-");</TH>
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Posición</TD><TD>Valor</TD></TR>
       <?php
           while(list($pos,$valor)=each($matrizP2)){
               echo "<TR ALIGN='center'";
               echo ($pos<3)?">":" BGCOLOR='#FFBBAA'>";
               echo "<TD>".$pos."</TD>";
               echo "<TD>".$valor."</TD></TR>";
           }
       ?>
     </TABLE></TD></TR>
    </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra el resultado:

- `array_walk(matriz, func_usuario [, parametro])`: Nos permite aplicar una función definida por el usuario a cada uno de los elementos de un *array*. La función `func_usuario()` recibe, al menos, dos parámetros (el valor del elemento y su clave asociada); si `func_usuario()` necesitara más parámetros, se pasan como parámetros a la función `array_walk()`. Una vez aplicada la función, el puntero interno del *array* se encontrará al final de él.

El siguiente ejemplo muestra la utilización de `array_walk()`:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays función <I>array_walk</I></H2>
     <?php
       $precios['prod1']=1500;
       $precios['prod2']=1000;
       $precios['prod3']=800;
       $precios['prod6']=100;
       $precios['prod7']=500;
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center"><TD>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
       <TR ALIGN="center" BGCOLOR="yellow">
       <TD>Producto</TD><TD>Precio</TD></TR>
       <?php
           while(list($pos,$valor)=each($precios)){
               echo "<TR ALIGN='center'><TD>".$pos."</TD><TD>";
               printf("%4d Ptas.",$valor);
               echo "</TD></TR>";
```

```
        }
      ?>
    </TABLE></TD><TD>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
      <TR ALIGN="center" BGCOLOR="yellow">
      <TD>Producto</TD><TD>Precio</TD></TR>
      <?php
          function aEuros(&$valor,$clave){
              $valor=$valor/166.386;
          }

          array_walk($precios,'aEuros');
          reset($precios);
          while(list($pos,$valor)=each($precios)){
              echo "<TR ALIGN='center'><TD>".$pos."</TD><TD>";
              printf("%02.2f €",$valor);
              echo "</TD></TR>";
          }
      ?>
    </TABLE></TD></TR>
  </TABLE>
 </CENTER>
</BODY>
</HTML>
```

La siguiente imagen muestra el resultado:

5.6.2 Trabajando con porciones del *array*

De igual forma que es habitual trabajar con porciones de cadenas de caracteres (modificar, extraer, etc.), también pueden realizarse este tipo de operaciones sobre *arrays*. PHP nos proporciona, entre otras, las siguientes funciones:

- `array_slice(matriz, desplazamiento [, int tamaño])`: Devuelve los elementos del *array* que están situados a partir de una posición determinada por `desplazamiento`. Opcionalmente, podemos indicar el total de elementos que queremos.

Los parámetros `desplazamiento` y `tamaño` pueden ser valores positivos o negativos. Esto da lugar a las siguientes interpretaciones reflejadas en la tabla de abajo:

desplazamiento	
Valor	**Significado**
positivo	Posición de comienzo
negativo	Posición de comienzo desde el final
tamaño	
Valor	**Significado**
positivo	Número de elementos a considerar
negativo	Último elemento a considerar desde el final
nulo	Se consideran todos los elementos hasta el final del *array*

Con un ejemplo vemos la utilización de esta función:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays función <I>array_slice</I></H2>
     <?php
       $matriz=array("a","b","c","d","e");
       $matriz1=array_slice($matriz,2);
       $matriz2=array_slice($matriz,2,-1);
       $matriz3=array_slice($matriz,-2,1);
       $matriz4=array_slice($matriz,0,3);

       echo "<BR><TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>\$matriz</TD>";
       echo "<TD>".implode('-',$matriz)."</TD>\n";
       echo "<TR ALIGN='center'>";
       echo '<TD BGCOLOR="yellow">array_slice($matriz,2)</TD>';
       echo "<TD>".implode('-',$matriz1)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo '<TD BGCOLOR="yellow">array_slice($matriz,2,-1)</TD>';
       echo "<TD>".implode('-',$matriz2)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo '<TD BGCOLOR="yellow">array_slice($matriz,-2,1)</TD>';
       echo "<TD>".implode('-',$matriz3)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo '<TD BGCOLOR="yellow">array_slice($matriz,0,3)</TD>';
       echo "<TD>".implode('-',$matriz4)."</TD></TR>\n";
```

```
        echo "</TABLE>\n";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra el resultado:

❑ `array_splice(matriz, despl [,tam [array substituto]])`: Elimina elementos de un *array*, sustituyéndolos opcionalmente por los elementos de otro *array* (`substituto`). Elimina y, si está indicado, sustituye los elementos que están situados a partir de la posición indicada por `despl`. Devuelve los elementos eliminados.

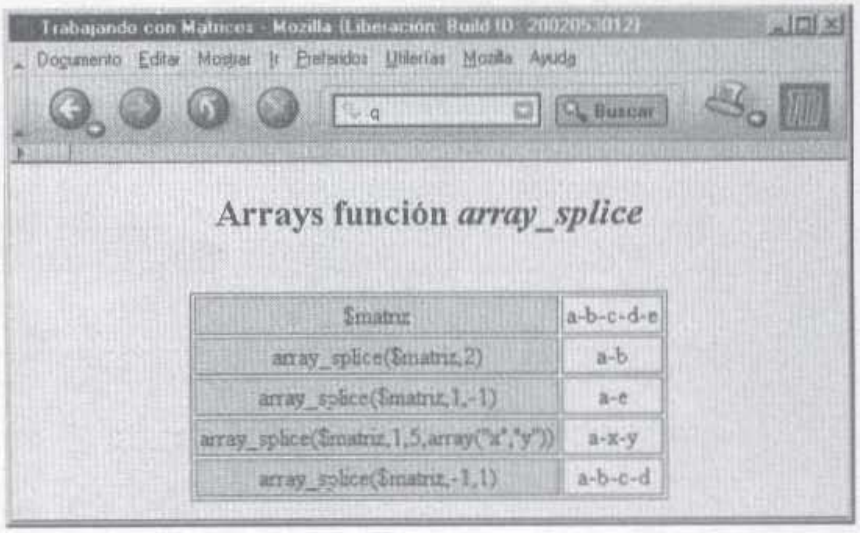
Los significados de los diferentes valores que pueden tomar `despl` y `tam` se especifican en la siguiente tabla:

despl	
Valor	**Significado**
positivo	Posición de comienzo de la sustitución/eliminación
negativo	Posición de comienzo de la sustitución/eliminación desde el final
tam	
Valor	**Significado**
positivo	Número de elementos a eliminar/sustituir
negativo	Último elemento a eliminar/sustituir desde el final
nulo	Se eliminan/sustituyen todos los elementos hasta el final del *array*

El siguiente ejemplo muestra la utilización de esta función:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays función <I>array_splice</I></H2>
     <?php
       $matriz=array("a","b","c","d","e");
       $matriz1=$matriz2=$matriz3=$matriz4=$matriz;
       array_splice($matriz1,2);
       array_splice($matriz2,1,-1);
       array_splice($matriz3,1,count($matriz),array("x","y"));
       array_splice($matriz4,-1,1);

       echo "<BR><TABLE BORDER='1'CELLPADDING='2' CELLSPACING='2'>\n";
       echo "<TR ALIGN='center'>";
       echo "<TD BGCOLOR='yellow'>\$matriz</TD>\n";
       echo "<TD>".implode('-',$matriz)."</TD>\n";
       echo "<TR ALIGN='center'>";
       echo '<TD BGCOLOR="yellow">array_splice($matriz,2)</TD>';
       echo "<TD>".implode('-',$matriz1)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo '<TD BGCOLOR="yellow">array_splice($matriz,1,-1)</TD>';
       echo "<TD>".implode('-',$matriz2)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo '<TD BGCOLOR="yellow">';
       echo "array_splice($matriz,1,5,array("x","y"))</TD>";
       echo "<TD>".implode('-',$matriz3)."</TD></TR>\n";
       echo "<TR ALIGN='center'>";
       echo '<TD BGCOLOR="yellow">array_splice($matriz,-1,1)</TD>';
       echo "<TD>".implode('-',$matriz4)."</TD></TR>\n";
       echo "</TABLE>\n";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

La siguiente imagen muestra el resultado:



5.6.3 Usando *arrays* como pilas

Las pilas son estructuras de datos en las que la inserción y la recuperación de los datos que almacenan se realiza bajo un orden prefijado. Son estructuras con una gestión de tipo LIFO (*Last In First Out*), es decir, el último en llegar es el primero en salir. PHP proporciona las siguientes funciones que nos permiten trabajar con los *arrays* como si fuesen pilas:

- `array_push(matriz, valor1 [, valor2, ...])`: Inserta uno o más elementos al final de `matriz`. Devuelve el nuevo número de elementos.

- `array_pop(matriz)`: Devuelve y elimina el último elemento del *array* (el que se corresponde con la cima de la pila).

- `array_shift(matriz)`: Devuelve y elimina el primer elemento del *array* (el que se corresponde con la base de la pila).

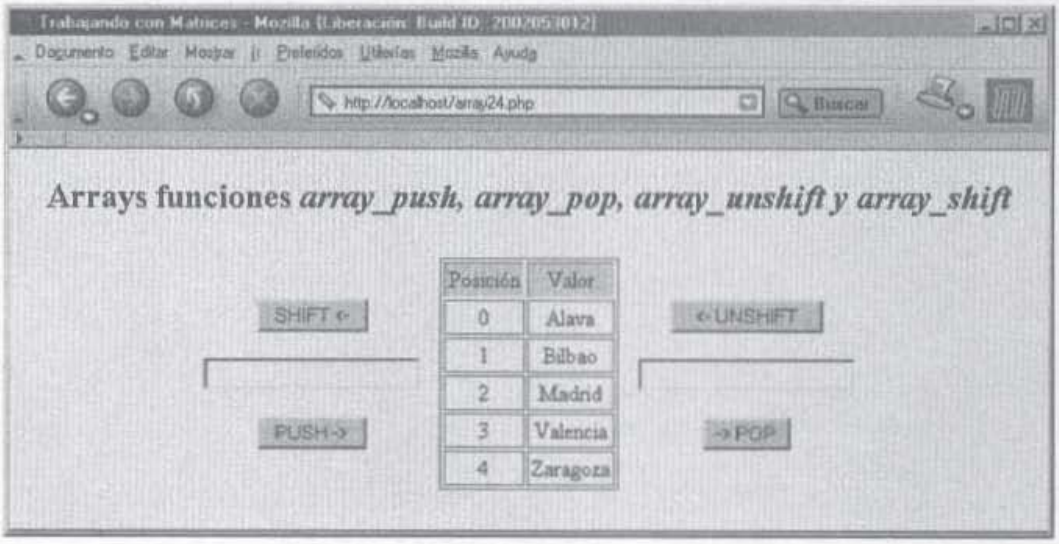
- `array_unshift(matriz, valor1 [, valor2, ...])`: Inserta uno o más elementos al principio del *array*. Devuelve el número total de elementos contenidos en él.

El siguiente ejemplo muestra la utilización de estas funciones: nos presenta una tabla con un *array* y cuatro botones para realizar las operaciones anteriores sobre dicho *array*:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Matrices</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Arrays funciones
        <I>array_push, array_pop, array_unshift y array_shift</I>
     </H2>
     <?php
       if (empty($_POST['PILA']))
          {
          $pila=array("Alava","Bilbao","Madrid","Valencia","Zaragoza");
          $dato1="";
          $dato2="";
          $BOTON="";
          }
       else {
          $PILA=$_POST['PILA'];
          $dato1=$_POST['dato1'];
          $dato2=$_POST['dato2'];
          $BOTON=$_POST['BOTON'];
          $pila=explode("-",$PILA);

          switch($BOTON){
             case "SHIFT <-"  : $dato1=array_shift($pila);
                                break;
             case "PUSH ->"   : array_push($pila,$dato1);
```

```
                                break;
            case "-> POP"     : $dato2=array_pop($pila);
                                break;
            case "<- UNSHIFT": array_unshift($pila,$dato2);
                                break;
        }
    }
  ?>
<FORM ACTION="<?php echo $_SERVER['PHP_SELF'] ?>" METHOD="POST">
 <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
  <TR ALIGN="center">
  <TD>
        <INPUT TYPE="SUBMIT" NAME="BOTON" VALUE="SHIFT <-"><BR><BR>
        <INPUT TYPE="TEXT" NAME="dato1"
               VALUE="<?php echo '$dato1' ?>"><BR><BR>
        <INPUT TYPE="SUBMIT" NAME="BOTON" VALUE="PUSH ->">
  </TD>
  <TD>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
    <TR ALIGN="center" BGCOLOR="yellow">
    <TD>Posición</TD><TD>Valor</TD></TR>
    <?php
        while(list($pos,$valor)=each($pila)){
            echo "<TR ALIGN='center'><TD>".$pos."</TD>";
            echo "<TD>".$valor."</TD></TR>";
        }
        $num=sizeof($pila);
        reset($pila);
    ?>
  </TABLE></TD>
  <TD>
        <INPUT TYPE="SUBMIT" NAME="BOTON" VALUE="<- UNSHIFT"><BR><BR>
        <INPUT TYPE="TEXT" NAME="dato2"
               VALUE="<?php echo '$dato2' ?>"><BR><BR>
        <INPUT TYPE="SUBMIT" NAME="BOTON" VALUE="-> POP">
  </TD>
 </TR>
 </TABLE>
 <INPUT TYPE="HIDDEN" NAME="PILA"
        VALUE="<?php echo implode('-',$pila) ?>">
</FORM>
</CENTER>
</BODY>
</HTML>
```

El ejemplo se muestra en la siguiente imagen:



NOTA: En ejemplo se hace uso de formularios que serán explicados en profundidad en capítulos posteriores.

- `array_reverse(matriz)`: Devuelve el *array* pasado como parámetro, pero con sus componentes en orden inverso.

- `range(limite_inf, limite_sup)`: Devuelve un *array* con los valores enteros comprendidos entre el primer argumento y el segundo que recibe la función, ambos inclusive.

- `array_count_values(matriz)`: Recibe como argumento un *array* y devuelve otro cuyos índices son los mismos que los del *array* original y sus valores asociados son la frecuencia con la que se repiten dichos valores en el *array* original.

- `in_array(elemento_busqueda, matriz)`: Con esta función podremos saber si un elemento está contenido dentro de un *array*.

- `compact()`: Esta función recibe como argumento una lista de variables que han sido previamente definidas, que pueden aparecer como cadenas de caracteres o como *arrays* y devuelve un nuevo *array* en el que los índices son los nombres de las variables y el contenido de los elementos del *array* son sus correspondientes valores.

CAPÍTULO 6

FUNCIONES

6.1 TRABAJANDO CON FUNCIONES

La mayor parte de las secciones del código de un *script* PHP deben ser ejecutadas tan pronto como dicho *script* es interpretado, pero, otras veces, es preferible que el código actúe después de que se haya producido alguna acción específica o sólo si se dispara un evento. También es habitual que partes del código se repitan un número indeterminado de veces durante la ejecución del *script* PHP. Estas necesidades hacen que nazca la idea de dividir el código de un *script* en partes menores, para que cada una de las cuales sirva a un propósito específico e individual.

Una función PHP es simplemente una sección separada de código a la que se le ha dado un nombre (cualquier instrucción PHP válida puede aparecer en el cuerpo de la función, incluso la llamada a otra función o la definición de clases). Utilizando este nombre, en un *script* se puede llamar a esta sección de código tantas veces como se quiera y en los momentos en que se necesite. Por tanto, las funciones dividen las tareas que debe hacer un *script*, agrupando instrucciones relacionadas para la ejecución de una tarea. Esta estructuración del código nos permite escribir *scripts* más sencillos, legibles y fáciles de entender.

Las funciones pueden recibir valores desde las sentencias que las llaman. Estos valores se denominan **parámetros** o **argumentos** y pueden devolver valores. Los parámetros, como veremos más adelante, se usarán como variables locales dentro del bloque de sentencias que conforman la función.

6.1.1 Declaración de una función

La sintaxis de la declaración de una función es la siguiente:

```
function nombreFunción ([parametro1 [,...]])
  {
      [sentencias]
  }
```

La palabra reservada `function` se utiliza para especificar un nombre, `nombreFunción`, el cual sirve como identificador para el conjunto de sentencias comprendidas entre las llaves. Encerrados entre paréntesis y separados por comas, se encuentran los nombres de los parámetros, que son los que recibirán los valores con los que es llamada la función. Técnicamente, los argumentos son variables que contienen informaciones necesarias para que la función realice correctamente su labor y que son pasados a ella en la sentencia de llamada. Aunque no se incluyan parámetros, en la declaración de la función deben escribirse los paréntesis. Las sentencias, que conforman el núcleo de la función, son ejecutadas cada vez que se llama a la función.

> **NOTA:** En PHP3 las funciones deben definirse antes de ser llamadas. En PHP4 no existe esta restricción.

6.1.2 Llamada a una función

Una vez declarada una función, ésta no se ejecuta hasta que se le llama desde cualquier parte del *script*. Para llamar a una función, sólo hay que escribir una sentencia que contenga el nombre de la función y, entre paréntesis, los valores de los argumentos de llamada de ella. En el momento en que ocurre esto, la ejecución del programa salta inmediatamente a la primera línea de la función llamada. Después de ejecutar las sentencias que componen el cuerpo de la función, el programa salta de nuevo a la posición en que fue llamada la función y continúa su ejecución. El resultado es el mismo que se hubiese obtenido al insertar todas las sentencias de la función en la posición de llamada a ella. Para volver a hacer uso de la función, simplemente hay que volver a llamarla: una función puede ser llamada tantas veces como se necesite.

El siguiente ejemplo nos muestra de una forma sencilla cómo definir y llamar a una función. Como podemos observar, se trata de una función declarada sin parámetros:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Funciones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones de Usuario<I></I></H2>
      <?php

      function cuentaAtras(){
          for($i=10;$i>0;$i--)
              echo $i,"...<BR>";
          echo "; Booooom !";
      }

     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center">
      <TD BGCOLOR="#FFBBAA">
       <?php
          cuentaAtras();
       ?>
      </TD>
      <TD BGCOLOR="#FFFBAD">
       <?php
          cuentaAtras();
       ?>
       </TD>
      </TR>
    </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:

6.1.3 Paso de parámetros

En ocasiones, necesitaremos definir funciones que aceptan parámetros o argumentos. De esta forma conseguimos que la función sea más utilizable dentro del propósito general para el que está definida. Por ejemplo, podemos querer utilizar una función múltiples veces a lo largo del programa, pero algunos de sus valores internos pueden variar con cada llamada. La mejor solución a este problema es definir una función que pueda aceptar argumentos en su llamada para cada valor con el que queramos trabajar.

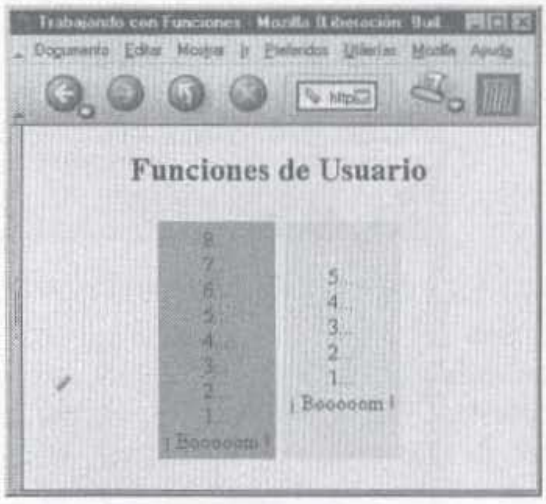
Otra forma de solucionar este problema es utilizar variables globales (las veremos en una próxima sección del capítulo) que puedan ser modificadas tanto dentro, como fuera de la función. Pero esto incrementa la confusión del código y su mantenimiento, pues se hace más difícil seguir su comportamiento.

El siguiente ejemplo muestra una modificación de la función presente en el código anterior para que el inicio de la cuenta pueda ser configurable por el usuario. Como podemos observar, los parámetros se utilizan como variables dentro del cuerpo de la función:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Funciones</TITLE>
 </HEAD>
```

```
<BODY>
  <CENTER>
    <H2>Funciones de Usuario<I></I></H2>
    <?php

      function cuentaAtras($inicio){
          for($i=$inicio;$i>0;$i--)
              echo $i,"...<BR>";
          echo "; Booooom !";
      }

    ?>
   <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
    <TR ALIGN="center">
     <TD BGCOLOR="#FFBBAA">
      <?php
         cuentaAtras(8);
      ?>
     </TD>
     <TD BGCOLOR="#FFFBAD">
      <?php
         cuentaAtras(5);
      ?>
      </TD>
     </TR>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

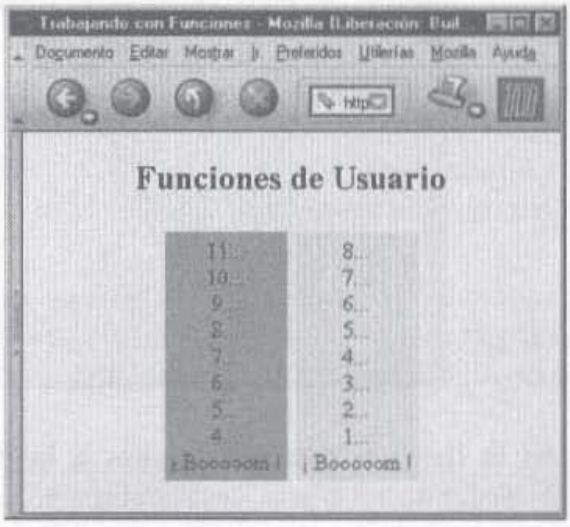
El resultado se muestra en la siguiente imagen:



Dado que en la llamada pasamos valores a la función, esta información puede suministrarse mediante una lista de variables y/o constantes separadas por comas. El siguiente ejemplo nos muestra una combinación de estos elementos pasados como argumentos a una función:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Funciones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones de Usuario<I></I></H2>
     <?php

       define("Fin_Cuenta",0);
       $mifinal=3;

       function cuentaAtras($inicio, $fin){
           for(;$inicio>$fin;$inicio--)
               echo $inicio,"...<BR>";
           echo "¡ Booooom !";
       }

     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center">
      <TD BGCOLOR="#FFBBAA">
       <?php
          cuentaAtras(11,$mifinal);
       ?>
      </TD>
      <TD BGCOLOR="#FFFBAD">
       <?php
          cuentaAtras(8,Fin_Cuenta);
       ?>
      </TD>
     </TR>
    </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



PHP permite pasar los parámetros de tres formas distintas: por *valor* (el comportamiento por defecto que hemos visto en los ejemplos anteriores), por *referencia* y con parámetros por *defecto*.

6.1.3.1 Parámetros por valor

Cuando pasamos una variable como argumento en la llamada a una función, podríamos pensar que las modificaciones que se realicen con dicho argumento dentro del cuerpo de la función afectan a la variable. En el caso del paso de parámetros por valor, que es la opción por defecto en PHP, lo que recibe la función es una copia del valor de la variable pasada como parámetro; de esta forma, las modificaciones que puedan hacerse dentro del cuerpo de la función a la variable *parámetro* no afectan al valor final de la variable pasada como argumento.

6.1.3.2 Parámetros por referencia

En el caso de que queramos que los cambios que se producen en el cuerpo de la función afecten a la variable que se pasó como argumento en la llamada a la función deberemos pasar el parámetro por referencia. Como su propio nombre indica, en este caso, a la función le llega una referencia a la variable y, por tanto, los cambios que realice sobre el parámetro se realizan sobre la variable.

Para indicar qué parámetros se pasan por referencia, hay que marcarlos en la definición de la función, anteponiendo el símbolo *ampersand* (**&**) al nombre del parámetro.

6.1.3.3 Parámetros por defecto

Los parámetros por defecto son la forma en que PHP implementa los parámetros opcionales en la llamada a las funciones. De este modo, este tipo de parámetros toma un valor predefinido cuando, desde la llamada a la función, no se les ha proporcionado ningún argumento. Para definir un parámetro por omisión, hay que, además de nombrar el parámetro, escribir el operador de asignación ("**=**") y, a continuación, el valor que vaya a recibir el parámetro en caso de no especificarse en la llamada.

Cuando se usan parámetros por defecto, éstos tienen que situarse los últimos en la declaración, es decir, a la derecha de cualquier parámetro *normal*; de otra

manera, las cosas no funcionarán de la forma esperada. Cuando se utiliza el valor por defecto de un parámetro, obligatoriamente han de utilizarse todos los valores por defecto de todos aquellos parámetros que se encuentren a su derecha.

> **NOTA:** En PHP 4.0 también es posible especificar `unset` como parámetro por defecto. Esto significa que el argumento no tomará ningún valor en absoluto si el valor no es suministrado.

El siguiente ejemplo nos muestra el uso de estos tres tipos de parámetros:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Funciones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones de Usuario</H2>
     <?php

       $mifinal=0;

       function cuentaAtras($inicio, &$fin, $mensaje="¡ Booooom !"){
           for(;$inicio>$fin;$inicio--)
               echo $inicio,"...<BR>";
           $fin=$fin+2;
           echo $mensaje;
       }
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center">
      <TD BGCOLOR="#FFBBAA">
       <?php
          // $mifinal vale 0
          cuentaAtras(6, $mifinal);
          // $mifinal vale 2
       ?>
      </TD>
      <TD BGCOLOR="#FFFBAD">
       <?php
          // $mifinal vale 2
          cuentaAtras(8, $mifinal,"¡ Despierta !");
          // $mifinal vale 4
       ?>
      </TD>
     </TR>
    </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:

6.1.4 Ámbito de las variables

Una vez vistas las funciones, podemos abordar el problema del *ámbito* o *alcance* de las variables, es decir, del contexto dentro del cual existen las variables. De esta forma podremos determinar desde qué partes del código del programa o *script* son accesibles las variables. Básicamente, podemos definir dos tipos de variables respecto a su ámbito:

- ***Variables globales*** son todas aquéllas que se definen fuera del cuerpo de una función y son accesibles, en general, desde cualquier punto del código.

  En PHP, las variables globales deben ser declaradas globales dentro de la función si van a ser utilizadas dentro de dicha función, anteponiendo a su definición la palabra reservada `global`. Un segundo método para acceder a las variables globales desde un ámbito local es usando el *array* `$GLOBALS` (es un *array* asociativo con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento del *array*).

- ***Variables locales*** aparecen definidas dentro del cuerpo de una función y sólo pueden ser accedidas desde dentro de esta función. Cualquier variable que se use dentro de una función está, por defecto, limitada al ámbito local de la función. Esto quiere decir que, si declaramos una variable local con el mismo nombre que una variable global, dentro de la función trabajaremos con la versión local de la variable.

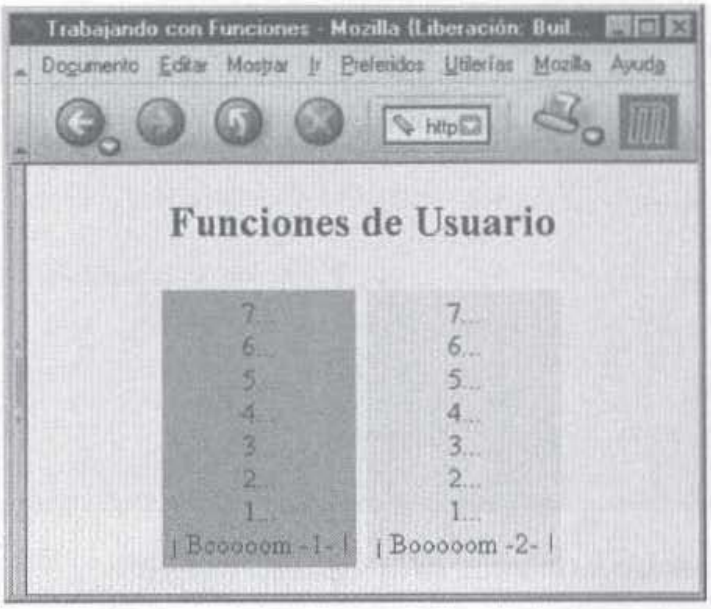
- ***Variables locales estáticas*** son variables locales que tienen ámbito local; por defecto, se crean cada vez que se comienza a ejecutar la función que las define, y se destruyen cuando se acaba la función. Para

evitar que una variable local pierda su valor entre diferentes llamadas a la función que la define, hay que definirla como variable estática, anteponiendo a su nombre la palabra reservada `static`. Una variable estática existe sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito.

El siguiente ejemplo nos muestra estos tres tipos de variables y sus diferentes comportamientos:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Funciones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones de Usuario</H2>
     <?php

       $inicio=9;
       $final=0;

       function cuentaAtras(){
           // variable global
           global $final;

           // variable local
           $inicio=7;
           // variable estática
           static $num=0;

           for(;$inicio>$final;$inicio--)
               echo $inicio,"...<BR>";
           $num++;
           echo "¡ Booooom -$num- !";
       }
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center">
      <TD BGCOLOR="#FFBBAA">
       <?php
          cuentaAtras();
          // $num vale 1
       ?>
      </TD>
      <TD BGCOLOR="#FFFBAD">
       <?php
          cuentaAtras(8,$mifinal,"¡ Despierta !");
          // $num vale 2
       ?>
       </TD>
      </TR>
    </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



6.1.5 Devolución de valores

Del mismo modo que las funciones pueden recibir valores, también pueden devolverlos. La devolución de un valor desde una función trabaja de igual forma que la devolución de un valor en una expresión, de manera que el valor devuelto desde una función puede ser asignado a una variable o utilizado dentro de una expresión. Se puede devolver cualquier tipo de valores incluyendo listas y objetos, pero sólo un único valor; para devolver múltiples valores, deberemos utilizar un *array*.

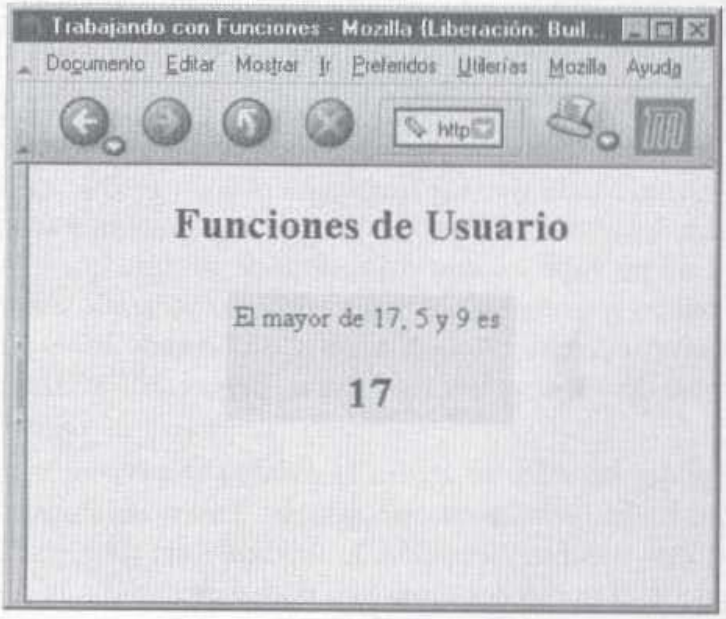
Para poder hacerlo, se utiliza la palabra reservada `return` acompañada de una expresión. En el instante en que aparece dentro del cuerpo de una función una sentencia con esta palabra reservada, la función deja de ejecutarse para devolver el flujo de ejecución al punto del programa donde se llamó a la función. Si después de `return` hay más líneas de código, dichas líneas no se ejecutarán nunca; por eso, es habitual que aparezca como última instrucción del cuerpo de la función.

El siguiente ejemplo muestra la devolución de valores desde una función:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Funciones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones de Usuario<I></I></H2>
     <?php
       function elMayor($dato1,$dato2,$dato3){
          $elmayor=($dato1>$dato2)?$dato1:$dato2;
```

```
            $elmayor=($elmayor>$dato3)?$elmayor:$dato3;
            return $elmayor;
        }
    ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center">
      <TD BGCOLOR="#FFFBAD">
       <?php
          echo "El mayor de 17, 5 y 9 es <BR><H2>".elMayor(17,5,9)."</H2>";
       ?>
      </TD>
     </TR>
    </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



6.1.6 Funciones con número variable de parámetros

PHP nos permite definir funciones en las que el número de parámetros no está fijado a priori, es decir, en PHP se consiente que una función reciba como parámetro una lista de valores de longitud variable. No se necesita de una sintaxis específica, pero su funcionamiento se basa en el siguiente conjunto de funciones definidas en PHP:

- `func_num_args()`: Devuelve el número de argumentos pasados a la función.

- `func_get_args()`: Devuelve un *array* con los argumentos pasados a la función.

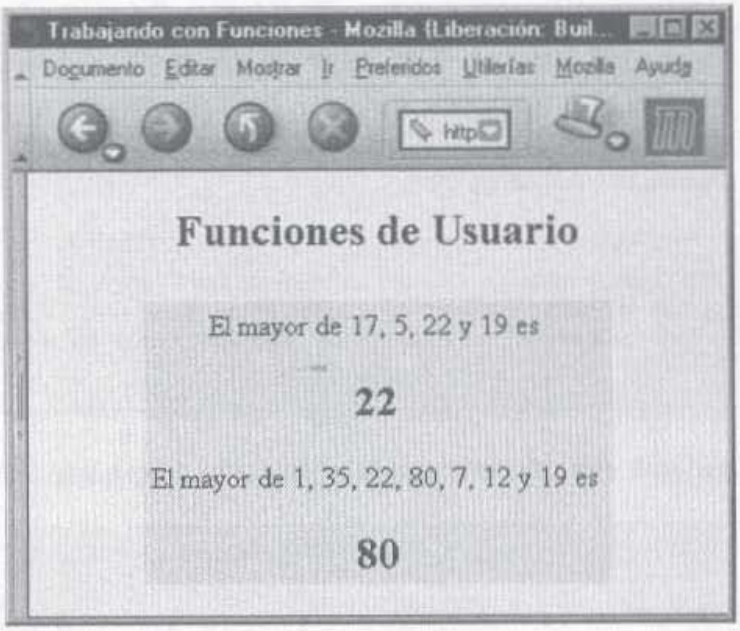
- `func_get_arg()`: Devuelve un elemento de la lista de argumentos pasados a la función. Los argumentos comienzan en la posición 0, al igual que los *arrays*. Si se solicita un argumento de una posición que no existe, devuelve `false`.

> **NOTA:** PHP3 no soporta un número variable de parámetros, si bien el problema se puede solventar pasando un *array* como parámetro. Todas las funciones anteriores generan un aviso si son llamadas desde fuera del cuerpo de una función.

El siguiente ejemplo muestra la utilización de estas tres funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Funciones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones de Usuario<I></I></H2>
     <?php
       function elMayor(){
          $num_args = func_num_args();
          $args = func_get_args();

          $elmayor=($dato1>$dato2)?$dato1:$dato2;
          for($i=2;$i<$num_args;$i++)
             $elmayor=($elmayor>$args[$i])?$elmayor:func_get_arg($i);
          return $elmayor;
       }
     ?>
    <TABLE BORDER="0" CELLPADDING="4" CELLSPACING="6">
     <TR ALIGN="center">
      <TD BGCOLOR="#FFFBAD">
       <?php
          echo "El mayor de 17, 5, 22 y 19 es
                <BR><H2>".elMayor(17,5,22,19)."</H2>";
          echo "El mayor de 1, 35, 22, 80, 7, 12 y 19 es
                <BR><H2>".elMayor(1, 35, 22, 80, 7, 12, 19)."</H2>";

       ?>
      </TD>
      </TR>
    </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



6.1.7 Funciones variables

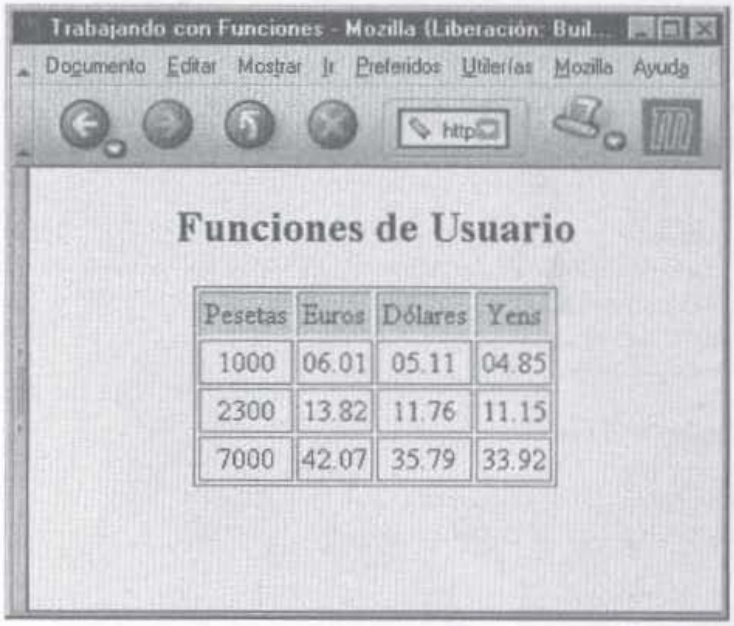
Son una herramienta muy útil de PHP que permite implementar, entre otras cosas, retrollamadas (*callbacks*) y tablas de llamadas. Su funcionamiento es el siguiente: si una variable tiene unos paréntesis añadidos al final, PHP buscará una función con el mismo nombre que el contenido de la variable e intentará ejecutarla.

El siguiente ejemplo muestra la utilización de estas tres funciones:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Funciones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones de Usuario<I></I></H2>
     <?php
       $array_func=array("aEuros","aDolares","aYens");
       $precios=array(1000,2300,7000);

       function aEuros($dato){
          return sprintf("%02.2f",$dato/166.386);
       }
       function aDolares($dato){
          return sprintf("%02.2f",$dato/195.6);
       }
       function aYens($dato){
          return sprintf("%02.2f",$dato/206.36);
       }
```

```
    ?>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
      <TR ALIGN="center" BGCOLOR="YELLOW">
        <TD>Pesetas</TD><TD>Euros</TD><TD>Dólares</TD><TD>Yens</TD>
      </TR>
        <?php
          for($i=0;$i<sizeof($precios);$i++){
            echo "<TR ALIGN='center'>";
            echo "<TD>$precios[$i]</TD>";
            for($j=0;$j<sizeof($array_func);$j++){
              $funcion=$array_func[$j];
              echo "<TD>".$funcion($precios[$i])."</TD>";
            }
            echo "</TR>";
          }
        ?>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



6.1.8 Funciones recursivas

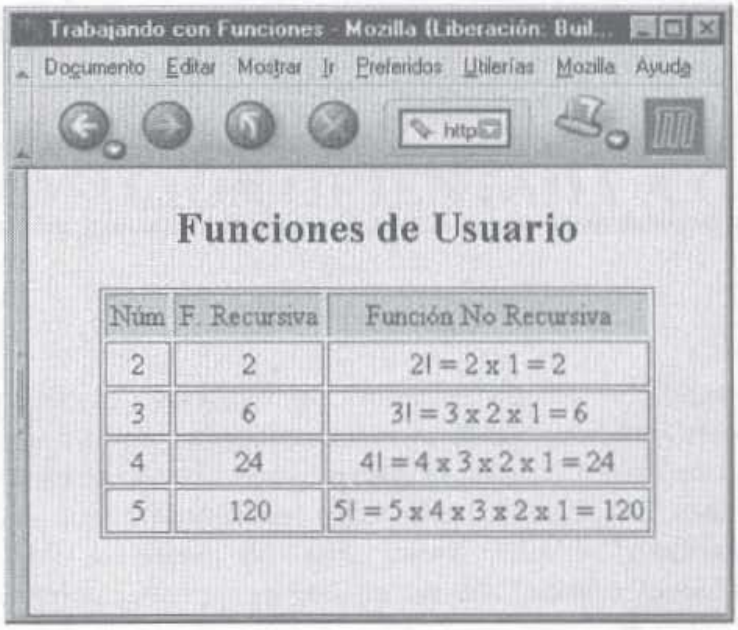
Se dice que una función es recursiva cuando en algún punto de su cuerpo se llama a sí misma. Hay que tener cuidado al escribir una función recursiva, ya que puede ocurrir que se llame a sí misma indefinidamente. Por tanto, es esencial asegurarse de implementar una forma adecuada de terminar la recursión: es lo que se denomina como condición de parada.

La recursión nos vale para solucionar algunos problemas complejos; un ejemplo típico de recursión es hallar el factorial de un número. El siguiente ejemplo nos muestra cómo implementarlo en PHP:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Funciones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones de Usuario<I></I></H2>
     <?php
       function factorial($numero){
          if ($numero==0) return 1;
          return $numero * factorial($numero-1);
       }

       function factor_iterativo($numero){
          echo "$numero! = ";
          for($factorial=1;$numero>1;$numero--){
              $factorial*=$numero;
              echo "$numero x ";
          }
          echo "1 = $factorial";
          return $factorial;
       }
     ?>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="2">
      <TR ALIGN="center" BGCOLOR="YELLOW">
        <td>Núm</td>
        <TD>F. Recursiva</TD>
        <TD>Función No Recursiva</TD>
      </TR>
      <TR ALIGN="center">
        <td>2</td>
        <TD><?php echo factorial(2); ?></TD>
        <TD><?php factor_iterativo(2); ?></TD>
      </TR>
      <TR ALIGN="center">
        <td>3</td>
        <TD><?php echo factorial(3); ?></TD>
        <TD><?php factor_iterativo(3); ?></TD>
      </TR>
      <TR ALIGN="center">
        <td>4</td>
        <TD><?php echo factorial(4); ?></TD>
        <TD><?php factor_iterativo(4); ?></TD>
      </TR>
      <TR ALIGN="center">
        <td>5</td>
        <TD><?php echo factorial(5); ?></TD>
        <TD><?php factor_iterativo(5); ?></TD>
```

```
      </TR>
    </TABLE>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



CAPÍTULO 7

PROGRAMACIÓN ORIENTADA A OBJETOS

Por fin, PHP 5, cubriendo las carencias de las versiones anteriores, nos provee con un soporte muy completo para la Programación Orientada a Objetos (POO). El nuevo motor Zend (*Zend Engine 2*) ha sido rediseñado por completo y proporciona todos los mecanismos necesarios asociados a este paradigma de programación. Así, a partir de esta nueva versión, podremos usar PHP indistintamente, según el gusto personal, en sus dos facetas: Procedural y Orientada a Objetos (OO).

A diferencia de los lenguajes OO nativos tipo Java, C++, Eiffel, etc., PHP es un lenguaje de *script* en el que no es indispensable dominar y conocer toda la parafernalia de clases que incorpora el lenguaje. De hecho, valga como ejemplo el programa más simple, escrito en Java y en PHP, que nos imprime un mensaje:

Java	PHP

`public class Saludo {`
`  public static void main(String[] args) {`
`  System.out.print("Hola");`
`  }`
`}`
	`echo "Hola";`

Por otra parte, la literatura que podemos encontrar sobre la POO es mucha y muy variada. Sin embargo, en general, en toda ella nos encontramos con ciertas palabras un tanto grandilocuentes al estilo de '*encapsulación*', '*polimorfismo*', '*herencia*', '*abstración*', '*modelización*', '*comportamiento*', etc., que suelen generar una cierta impaciencia en un programador clásico (procedural) y provocar que la POO se vaya dejando "*para más tarde*".

Por supuesto, no es el propósito de este capítulo, ni del libro, explicar teoría de la POO, sino mostrar cómo esta metodología puede ayudarnos a hacer nuestros programas y aplicaciones más fáciles, útiles y reaprovechables. Por tanto, en este capítulo iremos entremezclando los conceptos de la OO con las herramientas que PHP5 nos ofrece para su implementación.

7.1 CLASES Y OBJETOS

En la POO vemos que aparecen constantemente dos entes principales: las **clases** y los **objetos**. La relación que hay entre éstos, dicho de una manera sencilla e intuitiva, es la misma que hay entre un plano dibujado por un arquitecto y la casa o casas construidas a partir de ese plano. En la POO, la clase sería el plano y el objeto la casa. Otra equivalencia válida sería, por un lado, el fichero ejecutable que contiene código máquina y, por otro, el proceso que está ejecutándose con el contenido de ese fichero: podremos crear o ejecutar tantos procesos (objetos) como queramos a partir de un mismo fichero (clase).

Por lo tanto, en la práctica, siempre trabajaremos con objetos y, para poder utilizarlos, primero hay que declarar una clase y luego crear un ejemplar o *instanciar*[1] un objeto de esa clase.

Los objetos, por otra parte, no son más que una colección de variables y funciones que actúan sobre esas variables. No obstante, en la POO, la nomenclatura cambia, de manera que las variables que contiene un objeto reciben el nombre de **propiedades** y las funciones se llaman **métodos**. Esta redenominación se justifica porque en el enfoque OO, las variables contienen **información de estado** del objeto, y las funciones realizan modificaciones sobre dicho estado, definiendo el comportamiento del objeto.

7.1.1 Declaración de una clase y creación de un objeto

A continuación definimos una clase que contiene una propiedad (una variable) y un método (una función), creamos un objeto a partir de la clase declarada, asignamos un valor a la propiedad del objeto y, por último, ejecutamos el método contenido en el objeto:

[1] La traducción de la palabra inglesa '*instance*' al castellano es 'ejemplo', 'muestra', y el significado con el que se utiliza en la POO es el de 'creación de un nuevo ejemplar'. La palabra '*instancia*' es una mala traducción porque, de hecho, no existe con esa acepción en castellano. Desgraciadamente, está tan extendido su uso que seguiremos utilizándolo.

```
<?
class mi_Primera_Clase
{
  var $mi_primera_propiedad;
  function saludame() {
    echo "Hola Mundo :)";
  }
}

$mi_primer_objeto = new Mi_Primera_Clase();

$mi_primer_objeto->mi_primera_propiedad ='hola que tal';

$mi_primer_objeto->saludame();
?>
```

Vemos que en el código aparecen dos palabras reservadas: `class` y `new`, las que utilizamos, respectivamente, para declarar la clase y para crear un objeto de la misma. Además, observamos que para hacer referencia a cualquier miembro del objeto (propiedad o método) usamos el operador '`->`'.

7.2 PRIMER CONTACTO CON LA POO

En este apartado queremos mostrar que sin tener conocimientos previos sobre los conceptos asociados a la POO es posible programar utilizando este paradigma. Por supuesto, no haremos uso de ningún concepto asociado (abstracción, herencia, polimorfismo, etc.)[2], dado que lo que pretendemos es hacer ver que trabajar con clases y objetos está al alcance de cualquiera. De hecho, intentaremos hacer ver la conveniencia o ventaja, desde un punto de vista práctico, de usar objetos en lugar de funciones y variables.

Para ello, vamos a resolver de las dos formas distintas el siguiente problema: queremos generar tablas HTML cuyas filas tengan un aspecto (colores, tipo de letra, tamaño, etc.) que podamos establecerlo a nuestro gusto. Es bastante fácil, tenemos que ser capaces de poder definir las características de las celdas de una fila: su continente (alineación vertical, alineación horizontal, color de fondo) y su contenido (tipo de letra, tamaño, color)[3].

---

[2] La verdadera potencialidad de la POO, y cómo implementarla en PHP, la mostraremos en los siguientes apartados. Por tanto, aquellos que ya estén familiarizados con este paradigma pueden ir directamente a ellos.

[3] Evidentemente, para resolver este ejemplo de manera más elegante, y mucho más eficaz, habría que usar hojas de estilo (CSS).

7.2.1 Aproximación Procedural

En este caso, la solución es bien sencilla: a tantas características diferentes, tantas variables donde almacenar los valores correspondientes. Además, necesitamos una función que genere las etiquetas HTML necesarias para la celda y para el contenido de la misma. En el código siguiente generamos una fila con dos celdas:

```
<?
function pinta_celda($contenido,
                     $alin_h, $alin_v, $color_fondo,
                     $tipo, $tamano, $color_letra) {

    echo "<td align='$alin_h' valign='$alin_v' bgcolor='$color_fondo'>
          <font face='$tipo'size='$tamano' color='$color_letra'>
          $contenido
          </font>";
    echo "</td>";
}

$celda_alin_horiz = 'center';
$celda_alin_vert  = 'middle';
$color_celda      = 'orange';

$letra_tipo       = 'Arial';
$letra_tamano     = '10';
$letra_color      = 'white';

echo "<table border='0'><tr>";
pinta_celda('PHP',
            $celda_alin_horiz, $celda_alin_vert, $color_celda,
            $letra_tipo, $letra_tamano, $letra_color);

pinta_celda('5',
            $celda_alin_horiz, $celda_alin_vert, $color_celda,
            $letra_tipo, $letra_tamano, $letra_color);
echo "</tr></table>";
?>
```

Simple: en total son seis las variables que necesitamos. Sin embargo, esta solución tan rápida se nos podría complicar (o, más bien, embrollar) si, por cualquier razón, necesitáramos tener dos filas con aspectos diferenciados: el total de variables a manejar se vería duplicado por dos... Y como las cosas siempre son susceptibles de empeorar, pensemos en el enredo que tendríamos si el total de filas con aspectos diferentes fuese mayor.

Obviamente, el problema no estaría relacionado con la lógica del programa sino más bien con su codificación, depuración y mantenimiento: hay que controlar que las variables que pasamos como parámetro son las que realmente queremos; tenemos que acordarnos de declarar, inicializar y asignar todas y cada una de las variables; no confundir el nombre de los argumentos (los usados en la declaración de la función) con el de los parámetros (los pasados a la función). En definitiva, el código queda más difícil de mantener por su mayor *'complejidad'*.

7.2.2 Aproximación con Objetos

Si el eje fundamental de la programación procedural son las funciones, y las variables se usan simplemente para almacenar datos intermedios, en la POO, es al contrario: se pone el énfasis en las variables (propiedades), y las funciones (métodos) se usan para acceder a ellas y modificarlas.

Dado que un objeto es una especie de *caja negra* en la que está todo **autocontenido** (propiedades y métodos), veremos que el ejemplo propuesto, sin cambiar la lógica de la programación, se nos puede hacer más *'cómodo'* de implementar.

En primer lugar, definimos una clase que contendrá tanto las propiedades que almacenarán las características de las celdas, como el método que imprimirá el código HTML oportuno. Lógicamente, el método operará directamente con las propiedades que estén declaradas en la clase.

```
<?
class Clase_Celda {
  var $celda_alin_vertical;
  var $celda_alin_horizontal;
  var $celda_color_fondo;

  var $letra_tipo;
  var $letra_tamano;
  var $letra_color;

  function pinta_celda($contenido) {
    echo "<td   align='$this->celda_alin_horizontal'
               valign='$this->celda_alin_vertical'
              bgcolor='$this->celda_color_fondo'>";
    echo "<font  face='$this->letra_tipo'
                 size='$this->letra_tamano'
                color='$this->letra_color'>
          $contenido
          </font>";
    echo "</td>";
  }
}

$fila1 = new Clase_Celda();
$fila1->celda_alin_vertical='middle';
$fila1->celda_alin_horizontal='center';
$fila1->celda_color_fondo='orange';

$fila1->letra_tipo='Arial';
$fila1->letra_tamano='10';
$fila1->letra_color='white';

echo "<table border='0'><tr>";
$fila1->pinta_celda('PHP');
$fila1->pinta_celda('5');
echo "</tr></table>";
?>
```

A priori, el número de líneas no sería menor con lo que podría argumentarse que no parece muy útil. Sin embargo, dado que el pensamiento humano tiende a pensar en estructuras jerárquicas, conceptualmente es más lógico o natural pensar en entidades con una serie de características comunes, que en elementos dispersos como serían las variables *sueltas* (variante procedural).

Viendo el código, podemos comprobar que queda más *'limpio'*, se simplifica enormemente y es más fácil de mantener ya que, en este caso, el método (la función) sólo recibe el parámetro realmente importante para el programador: el del contenido de la celda (las características de la misma las toma del propio objeto). Además, no habrá problema al equivocarnos al escribir los parámetros en el código correspondiente que, incluso, quedaría menos farragoso pues no habría ni tan siquiera que poner las propiedades como parámetros.

Si necesitáramos más filas, sólo tendríamos que crear más objetos, asignarles los valores requeridos y mandarles imprimir a través del método:

```
$fila1 = new Clase_Celda();
// codigo de inicializacion con las características de $fila1

$fila2 = new Clase_Celda();
// codigo de inicializacion con las características de $fila2

echo "<table border='0'><tr>";
$fila1->pinta_celda('PHP');
$fila1->pinta_celda('5');
echo "</tr>";
echo "<tr>";
$fila2->pinta_celda('Segunda');
$fila2->pinta_celda('Fila');
echo "</tr></table>";
```

En la versión procedural, podríamos haber eliminado los parámetros y usar las variables globales directamente dentro de la función, simplemente, declarándolas como `'global'`. Sin embargo, esta solución tiene el gran inconveniente de estar expuesta a efectos laterales difíciles de detectar y depurar: se corre el riesgo de que se modifique la variable global en cualquier punto del programa y no lo detectemos. Además, y lo que es peor, nos imposibilitaría usar la función para imprimir dos tipos de filas distintas. Afortunadamente, en la versión orientada a objetos no existe este problema dado que cada objeto es dueño de sus propias variables (propiedades).

7.2.2.1 Palabra reservada $this

Como podemos observar, desde dentro del método `pinta_celda()` para acceder a las propiedades del objeto se usa la palabra reservada **`$this`**: ésta es una referencia al objeto concreto en que el método se está ejecutando, la cual le permite acceder a otras propiedades y métodos que pertenecen a ese objeto en particular.

Así, el operador flecha ('->') que sigue a `$this` se utiliza para discriminar cualquier propiedad o método que esté declarado dentro del objeto[4].

Si se hace referencia a una variable o función sin que vaya prefijada de `$this` u otro nombre, se estará haciendo referencia a una variable local o función primitiva de PHP:

```
<?
class unaClase {
  var $a = 'soy una propiedad';

  function pp() {
    $a = 'soy una variable local';
    echo $a . "<br />";
    echo $this->a . "<br />";
  }
}

$o = new unaClase();
$o->pp();
?>
```

Imprimiría:

```
soy una variable local
soy una propiedad
```

7.2.3 Reusabilidad y mantenibilidad del código

Por propia experiencia, sabemos que las especificaciones de una aplicación tienden a cambiar con más frecuencia de la deseada. Así, nos vemos obligados a modificar el programa que genera tablas HTML con las filas diferenciadas entre sí, puesto que ahora necesitamos, además, poder variar el ancho y el alto de la fila.

Con la aproximación procedural tendríamos que añadir más variables ('*sueltas*'), modificar la definición de la función para que admitiera como parámetros las nuevas variables y, además, modificar todas y cada una de las llamadas a esa función para que refleje la nueva definición.

Sin embargo, si hubiéramos optado por definir una clase, la tarea habría sido muchísimo más sencilla y, sobre todo, reusable: bastaría con añadir nuevas propiedades a la clase y modificar ligeramente el método usado; no haría falta modificar las llamadas al método y, el código que usa al objeto, quedaría

[4] No confundir la expresión `$this->propiedad` con `$this->$propiedad`, pues con esta última se intentará acceder a la propiedad del objeto cuyo nombre es el valor que está almacenado en la variable `$propiedad`, que estará vacía.

prácticamente invariable: sólo asignar valores a las nuevas propiedades. Como vemos, una ventaja de usar objetos es que podemos modificar su funcionalidad, añadir mejoras o corregir errores sin necesidad de cambiar su interfaz, o lo que es lo mismo, la forma de usarlo.

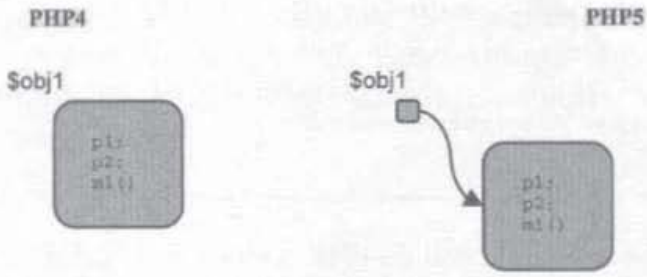
En definitiva, y como hemos querido mostrar a lo largo del ejemplo, el código OO es más fácil de mantener, más fácil de leer y más fácil de reusar (elementos todos ellos que constituyen los principios de la ingeniería del software).

7.2.3.1 Creación de espacios de nombres

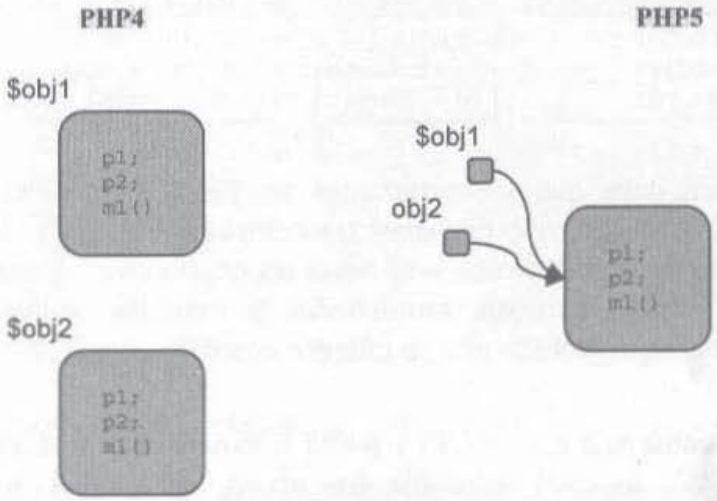
Una última consideración de orden práctico: implícitamente, el uso de clases y objetos nos hace crear **espacios de nombres** diferentes. Esto quiere decir que podremos usar el mismo nombre para tantos métodos o propiedades como queramos, siempre y cuando estén en objetos diferentes. Por ejemplo, podríamos usar el mismo identificador '`cierra`' para métodos que cierren un fichero (`$obj_fichero->cierra()`), una conexión con una base de datos (`$obj_bd->cierra()`), un fichero gráfico (`$obj_imagen->cierra()`), una conexión HTTP (`$obj_http->cierra()`), etc. Por otro lado, si utilizáramos funciones, para evitar colisiones de nombres, tendríamos que usar identificadores diferentes en cada uno de los casos anteriores: `bd_cierra(...)` o `fichero_cierra(...)` o `imagen_cierra(...)`. Como vemos, otro de los beneficios añadidos de la POO es que no es necesitamos definir y recordar tantos nombres.

7.3 MODELO DE OBJETOS DE PHP 5

Una de las principales carencias que tenía PHP 4 para ser considerado por los puristas de la POO como un verdadero lenguaje orientado a objetos (tipo Java o C++), era que la operación de creación de un objeto (sentencia `new`) lo que devolvía era el propio objeto y no una referencia o puntero (*handler*) al mismo. Esta característica era en ocasiones fuente de errores que en muchos casos eran difíciles de detectar. Afortunadamente en esta nueva versión del lenguaje esto ya no es así y lo que se obtiene en la creación de un objeto es siempre el manejador del mismo. Vemos gráficamente el resultado de la creación de objeto (`$obj1=new Una_clase();`):



Evidentemente, esta característica provoca diferencias de comportamiento, por ejemplo, al hacer una asignación de variables (`$obj1=$obj2;`): mientras que en PHP4 tenemos dos variables que contienen dos objetos distintos, en PHP 5 tenemos dos variables que referencian el mismo objeto:



El siguiente código de ejemplo nos ayudará a comprender mejor las diferencias existentes entre ambas versiones del intérprete:

```
<?

function saludo_matinal($obj_param) {
  $obj_param->saludo='buen dia';
  echo "[saludo_matinal]: ($obj_param->saludo)<br />";
}

class Clase_Saludo {
  var $saludo;
}

$o1 = new Clase_Saludo();
$o1->saludo='soy o1';

$o2 = $o1;
$o2->saludo='y yo o2';
```

```
echo "o1: ($o1->saludo) ; o2: ($o2->saludo)<br />";
saludo_matinal($o1);
echo "o1: ($o1->saludo) ; o2: ($o2->saludo)<br />";
?>
```

Según se trate de una versión u otra de PHP, vemos en la tabla de abajo que la salida es diferente, pues, como hemos comentado, PHP 4 crea un nuevo objeto[5] en la asignación y, sin embargo, PHP 5 lo único que asigna es la referencia al mismo, por lo tanto, cualquier modificación que se haga sobre una de las variables, se estará *repercutiendo* sobre la otra.

PHP 4	PHP 5
o1: (soy o1) ; o2: (y yo o2) [saludo_matinal]: (buen dia) o1: (soy o1) ; o2: (y yo o2)	o1: (y yo o2) ; o2: (y yo o2) [saludo_matinal]: (buen dia) o1: (buen dia) ; o2: (buen dia)

Esta característica hace que los programas en PHP 5 consuman menos memoria y sean más rápidos dado que no habrá tanto trasiego de datos: al asignar una variable que contiene un objeto a otra, o al pasar un objeto como parámetro no se hará una copia del objeto con sus propiedades y métodos, solamente del manejador (que suele estar representado por un número entero).

En PHP 4, para conseguir este efecto y poder trabajar con referencias había que usar el operador '`&`', lo cual resultaba engorroso, proclive a errores de codificación y muy difícil de depurar en caso de errores:

Creación de objetos:	`$obj = & new Clase_Objeto();`
Asignación de objetos:	`$obj2 = & obj;`
Paso de parámetros:	`function una_f (& $parametro) {}`

7.3.1 Clonación de objetos

Dado que ahora todas las operaciones con objetos (creación, asignaciones, paso de parámetros, etc.) se realizan mediante sus manejadores, PHP 5 nos ofrece la sentencia `clone` para hacer una copia exacta de un objeto.

No obstante, en determinadas ocasiones, necesitaremos que la copia realizada tenga alguna característica propia del nuevo objeto creado, es decir, que no

[5] Cuando veamos los destructores se comprenderá mejor.

sea idéntica. Para conseguir esto, si PHP 5 encuentra declarado un método con el nombre `__clone()`, lo ejecutará nada más realizarse la copia del objeto, ocasión que podemos aprovechar para hacer las modificaciones a las propiedades que necesitemos.

```
<?
class una_clase {
  var $un_contador=0;
  var $una_propiedad='nada';

  function __clone() {
     $this->un_contador++;
  }
}

$ob1 = new una_clase();
$ob2 = clone $ob1;

echo "<pre>";
print_r($ob1);
print_r($ob2);
echo "</pre>";
?>
```

Así, la salida del ejemplo anterior sería:

```
una_clase Object
(
    [un_contador] => 0
    [una_propiedad] => nada
)
una_clase Object
(
    [un_contador] => 1
    [una_propiedad] => nada
)
```

7.4 ACCESO A LOS MIEMBROS DE UN OBJETO

Uno de los principales pilares de la POO es el concepto de **encapsulación**, lo que significa que no se pueden acceder a las propiedades de un objeto si no es accediendo a ellas a través de sus métodos. Encapsular las propiedades de un objeto comporta que éstas están ocultas o inaccesibles desde fuera del propio objeto, sólo los métodos de éste son capaces de manipularlas. De esta manera, los métodos se

convierten en la interfaz de comunicación con el objeto o, dicho en otros términos, definen el comportamiento del objeto.

7.4.1 Propiedades privadas

Hasta ahora, todas las propiedades que hemos usado se dice que son **públicas** puesto que podemos trabajar con ellas desde fuera del objeto simplemente escribiendo: `$obj->propiedad`. Esta posibilidad, sin embargo, no es recomendable porque si desde fuera del objeto se pueden modificar sus propiedades, se puede llegar a alterar el propósito o la funcionalidad del mismo.

Lo vemos mejor con un caso concreto: imaginemos que en el ejemplo inicial, necesitamos que el color de fondo de la celda sólo pueda ser elegible entre tres posibles, los que se corresponden con los colores corporativos de la empresa: naranja, rojo y amarillo. Con lo que sabemos hasta ahora, no tenemos forma de evitar que quien use el objeto pueda escribir algo como:

```
class Clase_Celda {
  var $celda_color_fondo;
....
}
$fila = new Clase_Celda();
$fila->celda_color_fondo='fucsia';
```

Por esto, para tener el control del uso de las propiedades del objeto, tenemos que ser capaces de indicar que una determinada propiedad es *interna* al objeto, de su uso exclusivo, en definitiva, **privada** del objeto (y, por tanto, sólo accesible y/o modificable desde los métodos del objeto). En PHP 4 no existía esta posibilidad y por ello era ampliamente criticado por los puristas de la POO. Sin embargo, en PHP 5, declarando una propiedad con la palabra reservada **`private`** cualquier referencia del tipo `$obj->prop_privada` provocará un error del intérprete de PHP.

```
class Clase_Celda {
  ....
  private $celda_color_fondo;

  function pon_color_fondo($color) {
    if ($color == 'orange' || $color == 'red' || $color == 'yellow') {
      $this->celda_color_fondo=$color;
    } else {
      die("ERROR: no está permitido '$color' como color de fondo");
    }
  }

  ....
}
$fila = new Clase_Celda();
// $fila->celda_color_fondo='fucsia';  // provocara un error del interprete
$fila->pon_color_fondo('fucsia');
```

Como vemos, los intentos de actualización de la propiedad `$celda_color_fondo` han de efectuarse obligatoriamente a través del método `pon_color_fondo()`. Así, podremos introducir comprobaciones en los accesos a una propiedad declarándola como `private`.

> **NOTA:** En la POO es práctica habitual declarar todas las propiedades como privadas y acceder a ellas de manera controlada mediante métodos denominados '***setters* & *getters***', los cuales, respectivamente, modifican y devuelven sus contenidos.

7.4.2 Métodos `__set()` y `__get()`

PHP 5 incorpora dos nuevos métodos, `__set()` y `__get()`, que sirven para 'atrapar' todos aquellos accesos (de modificación de valor o de consulta, respectivamente) a propiedades que no están declaradas en el objeto. Los parámetros definidos en estos métodos son el nombre de la propiedad no definida que se ha querido asignar más el valor, y el nombre de la propiedad que se ha querido consultar:

```
function __set($nombre_prop, $valor) {
 // codigo del usuario
}

function __get($nombre_prop) {
// codigo del usuario
}
```

En un lenguaje interpretado como PHP, en el que no se necesita declarar las variables que se van a usar, estas funciones nos pueden ser de mucha ayuda, sobre todo, en la fase de depuración. Por ejemplo, el *script* siguiente, siendo perfectamente 'legal', contiene una errata que nos podría dar más de un quebradero de cabeza:

```
<?
class Despiste {
  var $identif_muy_largo;
}
$obj = new Despiste ();
$obj->identi_muy_largo='hola'; // aqui esta la errata
echo $obj->identif_muy_largo;
?>
```

Afortunadamente para nosotros, si definimos cualquiera de estos dos métodos, PHP nos avisará de que estamos accediendo a una propiedad que, por un descuido o errata al codificar, no existe:

```
<?
class Despiste2 {
  function __set($prop, $valor) {
    echo "OJO!!!, propiedad '$prop' NO declarada ($valor)<br />";
```

```
  }
  function __get($prop) {
    echo "OJO!!!, propiedad '$prop' NO declarada<br />";
  }
}
$obj=new Despiste2();
$obj->a='hola';
echo $obj->b;
?>
```

La salida de este *script* sería:

```
OJO!!!, propiedad 'a' NO declarada (hola)
OJO!!!, propiedad 'b' NO declarada
```

Estas funciones, además de usarlas para la fase de desarrollo y depuración, podemos aprovecharlas, a modo de divertimiento, para añadir propiedades de forma dinámica a la clase (idea nada recomendable, por cierto):

```
class Clase_Cepo_Prop_No_Declaradas1 {
  function __set($nombre_var, $valor) {
    eval('$this->' . "$nombre_var='$valor';");
  }

  function __get($nombre_var) {
    eval('$this->' . "$nombre_var='';"); // este eval ejecutara el __set()
  }
}
$o = new Clase_Cepo_Prop_No_Declaradas1();
$o->var_inexistente='fulano';
$o->var_inexistente='mengano';          // esta ya no es atrapada por __set()
echo "Contenido de var_inexistente: ($o->var_inexistente)<br />";

echo "(1): ($o->otra_inexistente)<br />";
$o->otra_inexistente='soy la otra...';  // esta ya no es atrapada por __set()
echo "(2): ($o->otra_inexistente)<br />";  // tampoco pasa por __set()
```

La salida generada del último:

```
Contenido de var_inexistente: (mengano)
(1): ()
(2): (soy la otra...)
```

7.4.3 Métodos privados

Otra de las '*acepciones*' de la encapsulación es que al usuario hay que ocultarle todo aquello (propiedades y/o métodos) que no es importante para él. Los detalles internos de implementación no tienen por qué conocerlos (ni usarlos). Esto significa que aquellos métodos que han sido diseñados para consumo interno del objeto deben de encubrirse de manera que el usuario no pueda ejecutarlos desde

fuera del objeto, pues su uso directo puede falsear, potencialmente, la intención original del objeto.

Para ilustrar lo que queremos decir, modificamos la implementación del objeto `Clase_Celda` para que la especificación del tipo de letra sea mediante estilos en línea (esto es, `<td style='font-family:arial;...'>`) en lugar de las obsoletas etiquetas `<font>`.

```
<?
class Clase_Celda {
  // ...

  private $letra_tipo;
  private $letra_tamano;
  private $letra_color;

  private function alin_horizontal() {
    return "text-align:$this->celda_alin_horizontal;";
  }
  private function alin_vertical() {
    return "vertical-align:$this->celda_alin_vertical;";
  }
  private function color_fondo() {
    return "background-color:$this->celda_color_fondo;";
  }
  private function tipo_letra() {
    return "font-family:$this->letra_tipo;";
  }
  private function tamano_letra() {
    return "font-size:$this->letra_tamano;";
  }
  private function color_letra() {
    return "color:$this->letra_color;";
  }

  private function genera_estilo() {
    return $this->alin_horizontal() .
           $this->alin_vertical()   .
           $this->color_fondo()     .
           $this->tipo_letra()      .
           $this->tamano_letra()    .
           $this->color_letra();
  }

  function pon_color_fondo($color) {
    if ($color == 'orange' ||$color == 'red' ||$color == 'yellow') {
      $this->celda_color_fondo=$color;
    } else {
      die("ERROR: no está permitido '$color' como color de fondo");
    }
  }

  function pinta_TD($contenido) {
    echo "<td style='". $this->genera_estilo() . "'>$contenido</td>\n";
  }
}

$fila = new Clase_Celda();
```

```
$fila->pon_color_fondo('orange');
echo "<table border='0'><tr>";
$fila->pinta_TD('PHP');
$fila->pinta_TD('5');
echo "</tr></table>";

?>
```

Si los métodos 'auxiliares' `color_fondo()`, `tipo_letra()`, `tamano_letra()`, etc., pudieran ser llamados por el usuario, el resultado no sería el adecuado. Por ello, los declaramos como `private` para, de alguna manera, proteger al usuario contra sí mismo.

7.4.4 Método `__call()`

Hemos visto que para atrapar accesos a propiedades no declaradas en la clase tenemos a nuestra disposición las funciones `__set()` y `__get()`. Para hacer lo propio con llamadas a métodos no declarados, PHP 5 nos proporciona el método `__call()`, cuya definición es la siguiente:

```
function __call($metodo, $array_parametros_llamada) {
  // codigo del usuario
}
```

Al igual que con aquéllas, esta función nos puede ser muy útil mientras se está desarrollando pues, en caso de despiste, evita la generación de un error y la parada del *script*.

Además, `__call()` se suele usar para implementar la **sobrecarga** de métodos, o lo que es lo mismo, hacer llamadas a un método y, en función de los parámetros pasados a éste (podemos examinar el *array* `$array_parametros_llamada`), ejecutar una acción u otra. En el ejemplo siguiente simulamos el caso del operador '+' de JavaScript, que ejecuta una concatenación o una suma dependiendo de si los operandos son cadenas de caracteres o números, respectivamente.

```
class Sobrecargada {
  function __call($metodo, $atributos) {
    if ($metodo=='operador_mas') {
      if (is_integer($atributos[0]) &&
          is_integer($atributos[1]))
        return $atributos[0] + $atributos[1];
      else {
        for ($str="", $i=0; $i<count($atributos); $i++)
          $str.=$atributos[$i];
        return $str;
```

```
        }
    } else
        echo "*** OJO: metodo no declarado: '$metodo'<br />";
  }
}

$obj = new Sobrecargada();
$res = $obj->operador_mas(1,3);
echo "($res)\n";
$res = $obj->operador_mas("hola", " que", " tal");
echo "($res)\n";
```

7.5 CONSTRUCTORES

Una costumbre que debemos adquirir al programar es inicializar siempre todos aquellos elementos que intervienen en el programa (variables, objetos, conexiones, etc.), para partir así de una posición conocida. Por esto, es muy común que al crear un objeto queramos que tenga una serie de características por defecto o, incluso, que se ejecuten una serie de acciones previas. Así, en el ejemplo propuesto en el inicio del capítulo, al crear un nuevo objeto `Clase_Celda`, querremos que éste tenga ya un valor por defecto para todas y cada una de las características de la celda (tipo de letra, color, alineación, etc.).

Para conseguir el comportamiento descrito, se usan lo que en terminología de POO se denominan **constructores** o, lo que es lo mismo, métodos que se ejecutan automáticamente nada más crearse un ejemplar de un objeto. En PHP 5, logramos este efecto cuando el objeto tiene definido un método con el nombre `__construct()`. Así, si en la declaración de `Clase_Celda` incluimos el siguiente código, cualquier objeto que creemos tendrá las propiedades con los valores asignados en el constructor.

```
class Clase_Celda {

  function __construct() {
    $this->celda_alin_vertical='middle';
    $this->celda_alin_horizontal='center';
    $this->celda_color_fondo='orange';

    $this->letra_tipo='Arial';
    $this->letra_tamano='10';
    $this->letra_color='white';
  }

....
}
```

Podemos, también, en el momento de creación del objeto, pasar como parámetros los valores con los que queramos que se inicialice. Para ello, previamente, declaramos estos parámetros en el constructor. Esta característica la aprovechamos, por ejemplo, para hacer alguna comprobación previa. Imaginemos que creamos una clase para tratar cadenas de caracteres: comprobamos que el valor con el que se inicializa el objeto es, efectivamente, de tipo 'string':

```
<?
class Cadenas {
  private $str;
  function __construct($param) {
    if (gettype($param) != 'string')
      die("ERROR: parámetro NO válido ");
    else
      $this->str=$param;
  }
}
$obj=new Cadenas('hola');
$obj=new Cadenas(7);                    // dará un error
?>
```

ATENCIÓN: el parámetro pasado al nombre de la clase en el momento de crear el objeto se corresponde con el declarado en el constructor, no tiene nada que ver con la declaración de la clase que no tiene parámetros algunos.

En PHP 4 el constructor era aquel método que tenía por nombre el mismo que la clase[6]. Sin embargo, ésta no es una buena práctica, dado que si en algún momento hubiéramos de renombrar el nombre de la clase, también habría que hacerlo con el del constructor[7].

7.6 DESTRUCTORES

Como puede deducirse de su nombre, y como complemento al método constructor, un destructor es aquel método de un objeto que se ejecuta automáticamente cuando se libera la última referencia a éste, es decir, el destructor se ejecuta cuando desaparece por completo el objeto de memoria. En PHP 5, este método recibe por nombre `__destruct()`.

---

[6] Por compatibilidad, PHP 5 sigue manteniendo esta característica. No obstante, si en la clase existen simultáneamente la función `__construct()` y una función con el nombre de la clase, se ejecuta la última que ha sido declarada.

[7] En el apartado de la herencia veremos las implicaciones negativas que esto tiene.

```
<?php
class Ajedrez {
  function __destruct() {
    echo "me muero... :((";
  }
}

$rey       = new Ajedrez();
$principe = $rey;
echo 'jaque al rey...<br />';
unset ($rey);
echo 'mate al rey...<br />';
unset ($principe);
?>
```

Al copiar el manejador del objeto creado (almacenado en `$rey`) en la variable `$principe` tenemos dos referencias al mismo objeto. Por esto, aunque eliminemos la variable `$rey`, sigue existiendo otra referencia al objeto creado y, por tanto, el destructor no se ejecuta hasta que no desaparece la última referencia al objeto.

```
jaque al rey...
mate al rey...
me muero... :((
```

Si no se hubiera hecho la asignación `$principe=$rey`, entonces la salida hubiese sido:

```
jaque al rey...
me muero... :((
mate al rey...
```

NOTA: un matiz a tener en cuenta es que es el recolector de basura de PHP 5 quien, antes de eliminar al objeto, lanza la ejecución de `__destruct()`, y no este método quien elimina al objeto de la memoria.

La existencia de un método destructor es muy útil dado que es buena costumbre liberar aquellos recursos que no vayan a usarse más. Esto puede ser cerrar descriptores de ficheros, *sockets*, conexiones con un gestor de base de datos, destruir otros objetos, etc. También para realizar tareas de última hora como generar información de depuración, hacer algún tipo de comprobación última o, en el caso de un objeto que genera una página HTML, escribir las últimas etiquetas '`</body></html>`' automáticamente.

Vemos en un ejemplo cómo combinar el uso de constructor y destructor para saber el tiempo que tarda el servidor Web en servir una página al cliente:

```
<?
class Cronometro {
  private $nacimiento;
  private $muerte;

  function __construct() {
    $this->nacimiento = $this->dame_instante();
  }

  function __destruct() {
    $muerte = $this->dame_instante();
    $tiempo_total = round(((float)$muerte - (float)$this->nacimiento), 3);
    echo "La página ha tardado en cargarse: $tiempo_total segundos";
  }

  private function dame_instante(){
    $instante_actual=microtime();
    list($micro_seg, $segs) = explode(" ",$instante_actual);
    return ((float)$micro_seg + (float)$segs);
  }

}

$reloj_en_marcha = new Cronometro();
// resto de la pagina…
// al finalizar el script desaparecerá el objeto y se ejecutará el destructor
?>
```

Gracias a que la primera instrucción del programa en ejecutarse es la creación de un objeto de la clase `Cronometro`, y que esta clase tiene definido un constructor que guarda el instante (`microtime()`) en que se crea el objeto, sabemos en qué momento se empieza a ejecutar el *script*. La finalización de éste dará lugar también a la desaparición del objeto y, por tanto, a la ejecución del destructor, el cual averigua el instante en que esto ocurre, halla la diferencia de tiempos y la muestra en el navegador del cliente.

En PHP 4 no existía este método destructor y para simularlo se solía usar la función `register_shutdown_function()`, cuyo cometido es lanzar la ejecución de la función pasada como parámetro en el momento de la finalización del *script*.

7.7 ATRIBUTOS Y MÉTODOS DE CLASE (MIEMBROS ESTÁTICOS)

Los atributos y métodos de clase (también llamados **estáticos**) son aquellos que pueden usarse directamente desde la clase sin necesidad de crear un objeto. Este tipo de miembros de clase suelen tener cabida en una clase genérica que se usa a modo de librería para toda la aplicación.

```
<?
class miscelanea {
  static function autores() {
    return "Abraham Gutiérrez & Ginés Bravo";
  }
}
echo miscelanea::autores();
?>
```

Como vemos en el ejemplo anterior, la palabra reservada **static** se usa para calificar al miembro correspondiente como estático o de clase, y con el operador '**::**' hacemos una llamada al mismo.

Dado que se usan desde la propia clase, estos miembros de clase no pertenecen a ningún objeto en concreto (no ha habido ninguna instanciación), por esto es por lo que no se puede usar la palabra reservada `$this` desde dentro de un método estático.

```
class unaEstatica {
  static $uno = '1';
  var    $dos = '2';
}

echo "unaEstatica::\$uno: (" . unaEstatica::$uno . ")<br />";

// la línea de abajo daría un error
// echo "(" . unaEstatica::$dos . ")<br />";

$o = new unaEstatica();
echo "\$o->uno: (" . $o->uno . ")<br />"; // no existe en el objeto
echo "\$o->dos: (" . $o->dos . ")<br />";
```

Los miembros de clase pueden ser también privados a la clase, con lo cual, sólo estarán accesibles desde dentro de la clase. De hecho, un truco muy habitual para evitar que una clase sea instanciada, es decir, si no queremos que se puedan crear objetos de una clase, habría que declarar el constructor como privado:

```
private function __construct() {}
```

A continuación, vemos un ejemplo de clase con miembros estáticos privados que nos permitirá llevar la cuenta de cuántos objetos de esa clase hay activos. Para ello contamos con una propiedad estática privada `$contador` que se incrementará en el constructor y se decrementará en el destructor: al ser privada y estática sólo desde dentro de la clase se puede manipular dicha propiedad.

```
class cuenta_objs {

  static private $contador = 0;

  static function cuantosObjetos() {
    return cuenta_objs::$contador;
  }

  function __construct() {
    cuenta_objs::$contador++;
  }

  function __destruct() {
    cuenta_objs::$contador--;
  }
}

$o1 = new cuenta_objs();
$o2 = new cuenta_objs();
$o3 = new cuenta_objs();
echo "Objetos 'vivos': " . cuenta_objs::cuantosObjetos() . "<br />";
unset ($o2);
echo "Objetos 'vivos': " . cuenta_objs::cuantosObjetos() . "<br />"
```

El resultado que obtenemos con la ejecución de este *script* es:

```
Objetos 'vivos': 3
Objetos 'vivos': 2
```

7.8 HERENCIA

Siguiendo con la premisa de que las personas concebimos las estructuras de manera jerárquica, es natural pensar que un objeto siempre puede verse como parte de otro. Tomemos como ejemplo el 'objeto fichero'. Para nosotros, un fichero es un conjunto de un determinado número de octetos, almacenado de alguna forma en el ordenador y que tiene una serie de características[8] como el nombre, el propietario, los permisos de acceso, las fechas de creación y modificación, etc. Las acciones que podemos hacer con un fichero son también conocidas: borrarlo, renombrarlo, copiarlo, etc. Pero, por otra parte, su contenido puede ser de lo más variado: un texto en caracteres ASCII, un gráfico en formato GIF, una canción en mp3, una película en formato DivX, etc.

[8] Lógicamente, dependerán del sistema operativo en el que esté almacenado.

Tomemos ahora, por ejemplo, el caso de un fichero gráfico y otro de sonido: ambos comparten las mismas características y acciones descritas. Sin embargo, ambos tienen otra serie de características exclusivas, inherentes a su propia naturaleza, que no comparten entre sí como es el número de píxeles, número de bits por color, ancho, alto, etc., para un fichero gráfico, y la duración, muestreo de bits por segundo, si es estéreo o no, etc., para uno de sonido. Deducimos con facilidad que, para representar los distintos tipos de ficheros como objetos, necesitaremos una clase genérica que describa las características comunes a todos los ficheros, **más** una clase específica por cada tipo concreto que recoja las posibilidades especiales de cada uno de ellos. Además, la clase genérica deberá ser compartida por todas las específicas.

Trasladando esto a un terreno más formal: el concepto de herencia en la POO es la relación que se da entre dos clases por la cual una de ellas (a la que llamaremos **hija**, subclase o derivada), además de tener sus propias propiedades y métodos, tiene a su disposición (hereda) los miembros definidos en la otra (a la que llamaremos **padre** o superclase).

El propósito de la herencia es la **reutilización de código**: si quisiéramos implementar un nuevo tipo de fichero (por ej., un documento PDF), podríamos aprovechar el código de la clase que implementa las características comunes (de esta manera nos evitamos 'reinventar la rueda').

En PHP 5, para que una clase pueda heredar las características de otra (convirtiéndose en hija suya) tiene que usar, en su declaración, la palabra reservada **`extends`** seguida del identificador de la clase que quiere obtener sus características:

```
<?
class Clase_Padre {
...
}

class Clase_Hija extends Clase_Padre {
...
}
```

Para ilustrar lo que acabamos de explicar, vamos a hacer una pequeña implementación de la clase `Fichero` que, por ser la que refleja las características comunes a todo tipo de fichero, se convertirá en la clase padre de todas ellas:

```
<?
class Fichero {
  private $nombre_fich;
  private $octetos;
  private $fecha_modificacion;

  function __construct($f) {
    if (is_file($f)) {
      $this->nombre_fich=$f;
      $this->octetos=(filesize($this->nombre_fich));
      $this->fecha_modificacion=(filemtime($this->nombre_fich));
    } else {
      die("*** ERROR: No se encuentra el fichero '$f'");
    }
  }

  function octetos() {
    return $this->octetos;
  }

  function fecha_modificacion() {
    return $this->fecha_modificacion;
  }

  function renombra($nombre_nuevo){
    if(rename($this->nombre_fich, $nombre_nuevo)) {
      $this->nombre_fich=$nombre_nuevo;
      return 1;
    } else {
      echo "*** ERROR: no se ha podido renombrar el fichero";
    }
  }

}
?>
```

Ahora, tal y como hemos dicho, creamos una clase que contemple las 'particularidades' de los ficheros gráficos (en concreto, los que tienen formato PNG), tales como las dimensiones de la imagen, los bits empleados para representar el color. Por supuesto, esta clase 'heredará' las características de la clase `Fichero` definida anteriormente.

```
<?
class Fichero_PNG extends Fichero {
  private $alto;
  private $ancho;
  private $bits_por_color;

  function __construct($f) {

    parent::__construct($f);
```

```
    $props = getimagesize($f);
    $ind_tipo_img=2;
    if ($props[$ind_tipo_img] != 3) { // es un fichero PNG?
      die("**** ERROR: '$f' no tiene formato gráfico PNG");
    } else {
      $ind_alto_img=0;
      $ind_ancho_img=1;
      $this->alto=$props[$ind_alto_img];
      $this->ancho=$props[$ind_ancho_img];
      $this->bits_por_color=$props['bits'];
    }
  }

  function bits_por_color() {
    return $this->bits_por_color;
  }

  function alto() {
    return $this->alto;
  }

  function ancho() {
    return $this->ancho;
  }

}

$obj_png = new Fichero_PNG('graf.png');
echo "El tamaño del fichero es: " . $obj_png->octetos() . "octetos<br />";
echo "Las dimensiones del gráfico son: " .
     $obj_png->alto() . "x" . $obj_png->ancho() . "<br />";
?>
```

En la salida comprobamos cómo podemos aprovechar el método `octetos()` que está declarado en la clase padre y que, vía herencia, tenemos disponible en la hija:

```
El tamaño del fichero es: 96731 octetos
Las dimensiones del gráfico son: 384x320
```

Si nos fijamos bien en el constructor de la clase `Fichero_PNG`, veremos que la primera sentencia es la expresión **`parent::__construct($f)`**. De lo que se deduce con facilidad que, en PHP 5, el constructor de una clase hija no ejecuta el de la clase padre de forma automática: hay que hacerlo explícitamente.

Por otra parte, en la herencia es donde vemos la importancia de que el nombre del constructor esté unificado: si el constructor es aquel método que tiene el nombre de la clase (como ocurre en PHP 4) y, en un momento dado, hay que modificar el nombre de una clase, si ésta es heredada, también habrá que modificar el nombre del constructor en todas sus hijas.

7.8.1 Miembros `protected`

Vamos a añadir ahora a la clase `Fichero_PNG` una nueva funcionalidad: crear un fichero cuyo contenido será una muestra (*thumbnail*) del fichero gráfico representado en el objeto[9]. Para ello, añadimos a la declaración de la clase este nuevo método al que hay que indicarle el nombre del fichero a crear y las dimensiones de la muestra:

```
function crea_muestra($fich_muestra, $ancho_muestra, $alto_muestra) {
  $imag_muestra  = ImageCreate($ancho_muestra, $alto_muestra);
  $imag_original = ImageCreateFromPNG($this->nombre_fich);
  ImageCopyResized($imag_muestra, $imag_original,
                   0, 0, 0, 0,
                   $ancho_muestra, $alto_muestra,
                   ImageSX($imag_original), ImageSY($imag_original));
  ImagePNG($imag_muestra, $fich_muestra);
}
```

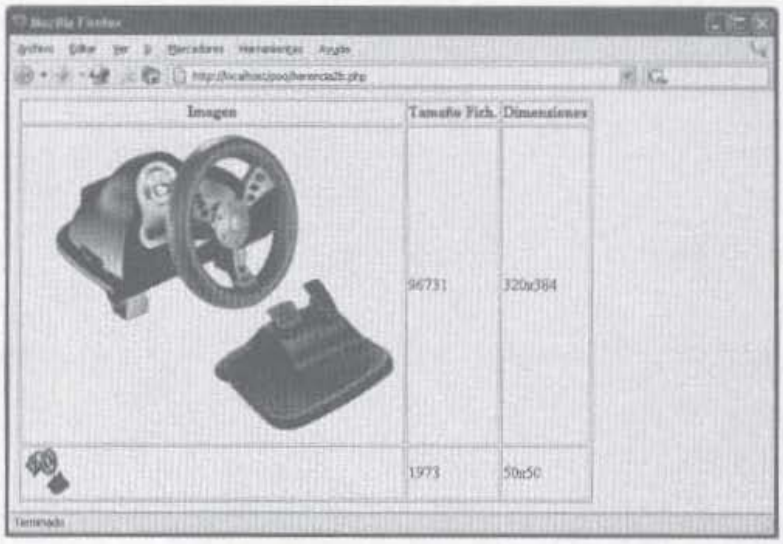
Aquí nos encontramos con un problema: necesitamos hacer referencia a la propiedad `nombre_fich` que está declarada como `private` en la clase padre `Fichero` y, por tanto, no podemos usarla en la clase hija pues los miembros privados no se heredan. Para resolver este conflicto, existe otro tipo de modificador del ámbito de acceso de un miembro de una clase que es **`protected`**. Una propiedad o método `protected` es accesible tanto desde la propia clase como desde las clases derivadas, pero no desde cualquier otra clase que no tenga ninguna relación de parentesco con ellas.

Así, en la clase `Fichero`, modificamos la declaración de la propiedad `nombre_fich` para que sea `protected` en lugar de `private`, y añadimos el código siguiente para comprobar el funcionamiento de ambas clases:

```
$nombre_fich         = 'graf.png';
$nombre_fich_muestra = 'graf_muestra.png';

$obj_png     = new Fichero_PNG($nombre_fich);
$obj_png->crea_muestra($nombre_fich_muestra, 50, 50);

$obj_muestra = new Fichero_PNG($nombre_fich_muestra);

echo "<table border='1'>
<tr><th>Imagen<th>Tamaño Fich.<th>Dimensiones
<tr><td> <img src='$nombre_fich'>
    <td>"  . $obj_png->octetos() .
   "<td>"  . $obj_png->ancho() . "x" . $obj_png->alto() .
"<tr><td><img src='$nombre_fich_muestra'>
    <td>" . $obj_muestra->octetos() .
    "<td>" . $obj_muestra->ancho()."x".$obj_muestra->alto() .
"</table>";
```

[9] Las funciones `ImageCreate()`, `ImageCreateFromPNG()`, etc., son de la biblioteca GD2.

El resultado lo vemos en la siguiente figura:



7.8.2 Redefinición

La verdadera potencia de la herencia está en la redefinición (*overriding*), en la clase hija, de los métodos declarados en la clase padre. La redefinición, en muchas ocasiones, la usamos para reflejar las lógicas particularidades de la clase hija, con respecto a la clase padre.

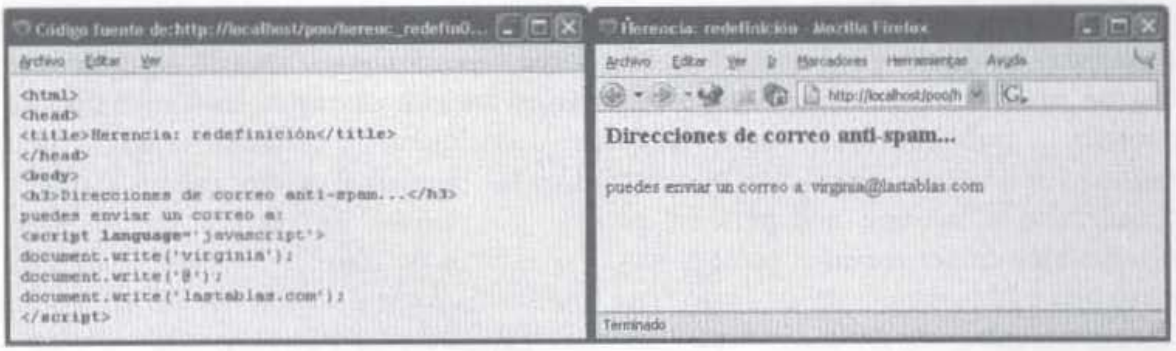
Veámoslo con un ejemplo. De igual manera que los buscadores rastrean y almacenan cualquier página que haya en la red, los *spammers*[10] hacen algo similar: una de sus fuentes de direcciones de correo, a las que enviarles cualquier tipo de *consejo publicitario*, es guardar cualquier cadena del tipo 'usuario@dominio.tld' que encuentren en las páginas. Por esta razón, vamos a definir una clase que nos permita generar direcciones de correo que no sean susceptibles de ser *robadas* por los robots o agentes de estos individuos: cualquier texto que esté generado desde JavaScript[11] no forma parte de la página HTML como tal, por lo tanto, con el siguiente método conseguiríamos nuestro propósito:

```
<?
class Dir_Correo_Anti_Spam {
  private $usuario;
  private $dominio;

  function __construct($u, $d) {
    $this->usuario=$u;
    $this->dominio=$d;
  }
```

[10] Personas que se dedican a enviar correo no solicitado (comúnmente llamado basura).
[11] ¡¡Ojo con los literales!!

```
  function pinta_dir_correo() {
    echo "<script language='javascript'>\n";
    echo "document.write('" . $this->usuario . "');\n";
    echo "document.write('@');\n";
    echo "document.write('" . $this->dominio . "');\n";
    echo "</script>\n";
  }
}
?>
<html>
<head>
<title>Herencia: redefinición</title>
</head>
<body>
<h3>Direcciones de correo anti-spam...</h3>
puedes enviar un correo a:
<?
$o = new Dir_Correo_Anti_Spam('virginia', 'lastablas.com');
$o->pinta_dir_correo();
?>
```

En las dos siguientes figuras podemos comprobar, viendo el código HTML de la página y su visualización en el navegador, que, efectivamente, no existe ninguna cadena '*expuesta*' a los robots:



Ahora creamos otra clase que, teniendo las mismas funcionalidades que la anterior, lo que hará será generar la dirección de correo en otro formato no '*comprometido*' como es una imagen, concretamente, la dibujaremos en un gráfico GIF.

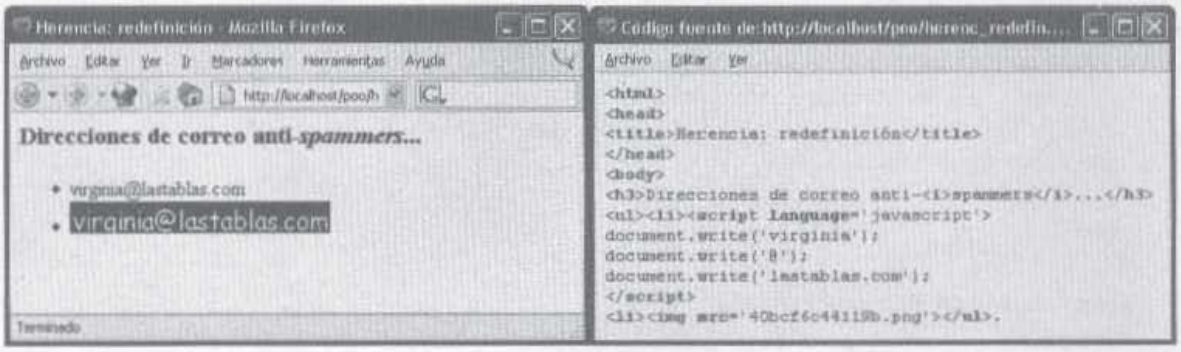
```
class Dir_Correo_Anti_Spam_GIF extends Dir_Correo_Anti_Spam {
  private $tam_x=205;
  private $tam_y=25;
  private $despl_x=2;
```

```
  private $despl_y=22;
  private $tamanyo_letra=12;
  private $fuente="/WINDOWS/Fonts/comic.ttf";
  private $fich_temp;

  function __construct($u, $d) {
    parent::__construct($u, $d);
    $this->fich_temp= uniqid() . '.png';
  }

  function __destruct() {
    $this->haz_tiempo();
    $dir_script=dirname($_SERVER['SCRIPT_FILENAME']);
    unlink("$dir_script/$this->fich_temp");
  }

  private function haz_tiempo() {
    // para que el fichero temporal con la imagen llegue al
    // navegador antes de que se borre en el servidor
    system("echo .");
    sleep(1);          // necesaria para IE
  }

  function pinta_dir_correo() {
    $angulo      = 0;
    $img         = imagecreate ($this->tam_x, $this->tam_y);
    $color_fondo = imagecolorallocate ($img, 0, 0, 0);
    $color_texto = imagecolorallocate ($img, 255, 255, 255);
    imagettftext ($img,
                  $this->tamanyo_letra, $angulo,
                     $this->despl_x, $this->despl_y,
                  $color_texto, $this->fuente,
                     $this->usuario . '@' . $this->dominio);
    imagePNG ($img, $this->fich_temp);
    echo "<img src='$this->fich_temp'>";
    imagedestroy ($img);
  }

}
?>
<html>
<head>
<title>Herencia: redefinición</title>
</head>
<body>
<h3>Direcciones de correo anti-spam...</h3>
<?
echo "<ul><li>";
$dir_txt = new Dir_Correo_Anti_Spam('virginia', 'lastablas.com');
$dir_txt->pinta_dir_correo();
```

```
echo "<li>";
$dir_gif = new Dir_Correo_Anti_Spam_GIF('virginia', 'lastablas.com');
$dir_gif->pinta_dir_correo();
echo "</ul>";
?>
```

Dado que las propiedades `$usuario` y `$dominio`, declaradas en la clase `Dir_Correo_Anti_Spam` como privadas, se usan en la clase hija, tendrán que ser redeclaradas como `protected`.

Comprobamos que la dirección de correo generada es inteligible para las personas, aunque ya no tanto para los programas de rastreo automático:



Aun redefiniendo un método, la clase hija puede seguir accediendo o ejecutando el método correspondiente de la clase padre invocándolo de la forma: **`parent::metodo()`**, es decir, se puede redefinir en la clase hija un método de la clase padre, a la vez que se puede usar éste.

Como sugerencia sobre la redefinición de métodos, a la hora de diseñar una jerarquía de clases, una regla de diseño básica a seguir es hacer el código de los métodos lo más pequeño y específico posible ya que así se podrán reusar con mayor facilidad. Por el contrario, si hacemos que un método sirva para varias tareas a la vez, lo más probable es que tengamos que duplicar parte de ese código en los métodos de las clases hijas.

Por último, las propiedades también pueden verse afectadas por la redefinición: una propiedad declarada en la clase padre y en la clase hija con el mismo nombre podrá ser tratada como una sola propiedad o como dos distintas, dependiendo de cómo esté declarada en ambas.

```
<?
class clase1 {
  protected $a='111';
  function dame_a() {
```

```
    return "($this->a)";
  }
}

class clase2 extends clase1 {
  protected $a='2222';
  function dame_a() {
    echo parent::dame_a();
    echo "[[$this->a]]";
  }
}

$o = new clase2();
$o->dame_a();
?>
```

Vemos, a partir del código de ejemplo anterior, un resumen en forma tabular donde están reflejadas las distintas posibilidades de protección de las propiedades y la salida generada:

Clase1::a	Clase2::a	Salida
public	public	(2222)[[2222]]
protected	public	(2222)[[2222]]
private	Public	(111)[[2222]]
private	private	(111)[[2222]]

En concreto, observamos que si la propiedad de la clase padre está declarada como privada, entonces tendremos dos propiedades dado que la hija no la hereda.

7.8.3 Métodos y Clases `final`

Un método declarado como **`final`** no podrá ser redefinido en ninguna clase derivada. Con esto podremos evitar que determinados métodos no puedan subvertir la intención inicial para la que estaban destinados. El siguiente código provocará un error:

```
class MiClase {
  final function saluda() {
    echo "hola mundo!";
  }
}

class MiClase2 extends MiClase {
  function saluda() {
```

```
    echo "passa, figuras.. ;)";
  }
}
```

Por último, las clases que se declaren como **final** no pueden ser heredadas. El siguiente ejemplo daría un error del intérprete de PHP:

```
<?
final class clase_final { }

class otra_clase extends clase_final { }
?>
```

7.9 CLASES ABSTRACTAS

La característica fundamental de una clase **abstracta** es que **no** puede instanciarse, es decir, no se puede usar directamente sino a través de una subclase (la cual, recibe el nombre de **concreta**). Una clase abstracta, como cualquier clase padre, además de declarar e implementar las propiedades y métodos que desee, puede definir **métodos abstractos**, los cuales, no tienen código alguno porque, obligatoriamente, tienen que ser implementados en las clases hijas. Además, cabe la posibilidad de que en una clase hija, el método se vuelva a declarar abstracto: si ocurre esto, la responsabilidad de la implementación recaerá en el siguiente nivel jerárquico.

Estas peculiaridades hacen que estas clases sean muy útiles para los diseñadores o analistas ya que, definiendo una clase como abstracta, obligan a los programadores a codificar una subclase que implemente la especificación dada en la clase padre. De hecho, las clases abstractas se encuentran en lo alto de la jerarquía de objetos.

Por ejemplo, queremos disponer de una clase que analice las bitácoras (ficheros *log*) de un servidor web y genere los siguientes datos estadísticos de accesos:

- Total de octetos enviados por el servidor.
- Número total de visitas recibidas.
- Cuántos clientes distintos nos han visitado.
- Cuántas páginas diferentes han sido descargadas.
- Para una determinada página, cuáles han sido los clientes que se han conectado y cuántas veces.

- Para una determinada dirección IP, cuáles han sido las páginas que ha visitado, y cuántas veces lo ha hecho.
- Definir la franja de tiempo a la que se limitará el análisis (por defecto, los datos que se tomarán en cuenta serán los del día anterior).

Aquí las clases abstractas son muy útiles ya que los servidores web con que nos podemos encontrar son muchos (Apache, IIS, AOL Server, Xitami, etc.) y, además, cada uno de ellos suele tener un formato particular para guardar los '*logs*' de accesos. Por esto, lo que hacemos es diseñar una clase abstracta con las funcionalidades expuestas y, con posterioridad, encargar a distintas personas la implementación de las clases correspondientes a cada uno de los servidores que tengamos y queramos analizar.

```
<?
abstract class Bitacora {
  // total de octetos enviados por el servidor
  abstract function total_octetos();

  // numero de visitas recibidas en el servidor
  abstract function total_visitas();

  // numero de paginas distintas servidas
  abstract function paginas_diferentes();

  // numero de visitantes distintos
  abstract function clientes_diferentes();

  // devolvera el array asociativo: array['ip_cliente']=num_visitas
  abstract function quien_visita($que_pag);

  // devolvera el array asociativo: array['pagina_visitada']=num_visitas
  abstract function que_pags_visita($que_ip);

  // para definir el intervalo de tiempo en que se calculan las estadisticas
  // las fechas deben ser timestamps de Unix
  // por defecto, es el dia anterior
  abstract function intervalo($timestamp_desde, $timestamp_hasta);
}
?>
```

Aquí, haremos una implementación para leer las bitácoras de Apache. Lógicamente, la parte más importante de la implementación es la relativa a la lectura y análisis de las líneas de la bitácora. Por ello, hemos escrito una función (`lee_y_analiza()`), la cual, para cada registro, separa los campos que contiene (*parsing*), comprueba que ese acceso ha ocurrido dentro de la franja de tiempo deseada y almacena en una serie de variables los datos relativos al acceso en curso.

```
<?
class Bitacora_Apache extends Bitacora {

  public function total_octetos() {
```

```
    return $this->octetos_enviados;
  }

  public function total_visitas() {
    $tot=0;
    foreach ($this->ip_cliente as $ip=>$paginas) {
      foreach ($paginas as $pagina=>$cont_pagina) {
         $tot += $cont_pagina;
      }
    }
    return $tot;
  }

  public function paginas_diferentes() {
    return count($this->pag_visitada);
  }

  public function clientes_diferentes() {
    return count($this->ip_cliente);
  }

  public function quien_visita($que_pag) {
    return $this->pag_visitada[$que_pag];
  }

  public function que_pags_visita($que_ip) {
    return $this->ip_cliente[$que_ip];
  }

  public function intervalo($timestamp_desde, $timestamp_hasta) {
    $this->ini_intervalo=$timestamp_desde;
    $this->fin_intervalo=$timestamp_hasta;
  }

  /* ------------------------------------------------------ */

  private $pag_visitada = array();    // array['ip']=num_visitas
  private $ip_cliente=array();        // visitas por ips
  private $octetos_enviados=0;        // visitas a cada pagina
  private $formato_log;    // posibilidades: combined, common, referer, agent
  private $fich_log;
  private $pos_meses_en_anyo;
  private $ini_intervalo;
  private $fin_intervalo;

  function __construct($formato_log, $fich_log) {
    // posibilidades del formato tipico: combined, common referer agent
    $this->formato_log=$formato_log;
    $this->ini_pos_meses_en_anyo();

    $this->ini_intervalo=mktime(0,0,0,date("m"),date("d")-1,date("Y"));//ayer
    $this->fin_intervalo=$this->ini_intervalo + 86400; // sumo un dia

    if (is_file($fich_log)) {
      $this->fich_log=$fich_log;
      $this->lee_y_analiza();
    } else
      die("**** ERROR: NO existe la bitácora '$fich_log'");
  }

  private function dame_pagina($str) {
    // poner en urlencode y eliminar el posible query_string
```

```
    $pagina_y_query_string=explode("?", urldecode($str));
    return $pagina_y_query_string[0];
  }

  private function ini_pos_meses_en_anyo() {
    $this->pos_meses_en_anyo=array('Jan'=>1, 'Feb'=>2, 'Mar'=>3, 'Apr'=>4,
                                   'May'=>5, 'Jun'=>6, 'Jul'=>7, 'Aug'=>8,
                                'Sep'=>9, 'Oct'=>10, 'Nov'=>11, 'Dec'=>12);
  }

  private function esta_en_fecha($s) {
    sscanf($s, "[%2d/%3c/%4d:%2d:%2d:%2d",$dia, $mes,$anyo,$hora,$min, $seg);
    $ahora=mktime($hora,$min,$seg,$this->pos_meses_en_anyo[$mes],$dia,$anyo);
  if ((int)$ahora < (int)$this->ini_intervalo)
    return 0;
  elseif ((int)$ahora < (int)$this->fin_intervalo)
    return 1;
  else
    return 2;
  }

  private function lee_y_analiza() {
    if ($this->formato_log != 'common') {
      die ("ERROR: de momento, sólo está implementado 'common'<br>");
    }
    // ejemplo de linea de log tipo 'common'
   //127.0.0.1 - - [13/Apr/2004:14:25:13 +0200] "GET /k.php HTTP/1.1" 200 35074
    $id_campos = array ('pos_ip', 'pos_guion1', 'pos_guion2', 'pos_fecha',
                        'pos_gmt','pos_metodo','pos_pagina', 'pos_protocolo',
                        'pos_msg_http', 'pos_octetos');
    for ($i=0; $i<count($id_campos); $i++) {
      eval('$' . $id_campos[$i] . "=$i;");
    }
   /* $pos_ip=0; $pos_guion1=1; $pos_guion2=2; $pos_fecha=3; $pos_gmt=4;
 $pos_metodo=5;$pos_pagina=6;$pos_protocolo=7;$pos_msg_http=8;$pos_octetos=9;
   */

    $esta_en_fecha=true;
    $df=fopen($this->fich_log, "r");
    while (!feof($df) && $esta_en_fecha) {
      $linea=fgets($df, 4096);
      $campos=explode (" ", $linea);
      switch ($this->esta_en_fecha($campos[$pos_fecha])) {
        case 0: // nada, fecha anterior
                break;
        case 1:
               $this->octetos_enviados += $campos[$pos_octetos];
               $la_pag=$this->dame_pagina($campos[$pos_pagina]);
               $this->ip_cliente[$campos[$pos_ip]][$la_pag]++;
               $this->pag_visitada[$la_pag][$campos[$pos_ip]]++;
               break;
        case 2: $esta_en_fecha=false;
      }
    }
    fclose($df);
  }

} // fin de la clase 'Bitacora_Apache'

$bitacora = new Bitacora_Apache('common', 'logs_mas');

$gigas = round($bitacora->total_octetos() / 1048576);
```

```
echo "Total Octetos devueltos: $gigas GB<br>";

echo "Paginas visitadas: (" . $bitacora->total_visitas() . ")<br>";
echo "Paginas Diferentes (" . $bitacora->paginas_diferentes() . ")<br>";
echo "Clientes Diferentes (" . $bitacora->clientes_diferentes() . ")<br>";

$ip="11.22.33.44";
imprime_tabla($bitacora->que_pags_visita($ip), "páginas visitadas por $ip");

$pag="/index.htm";
imprime_tabla($bitacora->quien_visita($pag), "ips que visitan '$pag'");

function imprime_tabla($el_array, $encabezado) {
  if ($el_array) {
    echo "<table border='1'><tr><td colspan='2'>$encabezado</td></tr>";
    foreach ($el_array as $clave=>$valor)
      echo "<tr><td>$clave</td><td align='right'>$valor</td></tr>";
    echo "</table>";
  } else
    echo "$encabezado: 0<br>";
}

?>
```

PHP 4 no reconocía las clases abstractas: un truco que se solía usar para evitar que la clase se pudiera instanciar era poner una sentencia `die()` en el constructor de la clase. Además, para evitar que los métodos abstractos se usaran de forma estática (`Clase::metodo()`), o bien se hacía como con el constructor y se ponía una sentencia `die()` como única sentencia de un método abstracto, o bien, se empleaba la función `is_sub_class_of()`:

```
class abstract_php4 {
 function abstract_php4() {
   die ("ERROR: no me puedes instanciar :(");
 }
 function saluda() {
   die("que no, que no...");
   if (!is_subclass_of($this, "segunda"))
     die ("ERROR: la llamada no desde una subclase");
 }
}
```

7.10 INTERFACES

Otro tipo de clase, muy similar funcionalmente a la abstracta, es el `interface`. Al igual que aquélla, no pueden instanciarse. Se diferencian en que, mientras que la abstracta puede ofrecer una funcionalidad (porque puede implementar el código que necesite), en el `interface` sólo se declaran los métodos (no se pueden declarar propiedades), es una especie de catálogo de métodos a

implementar (de hecho, no tiene sentido declarar un método privado dentro de una interfaz).

Sintácticamente, se diferencian en que para declarar una interfaz no se usa la palabra reservada `class` sino `interface`. Además, si para derivar una clase hija de una clase abstracta se usa la palabra reservada `extends`, para hacer lo propio con una interfaz se usa `implements`.

```
interface interfaz {
  function met_interfaz();
}

class hijo_interfaz implements interfaz {
  function met_interfaz() {
    echo "soy el hijo del interfaz...";
  }
}
$p = new hijo_interfaz();
$p->met_interfaz();
```

Por último, una clase puede heredar de una clase solamente (PHP 5 no permite la herencia múltiple) pero puede implementar tantas interfaces como quiera:

```
class clase_impl implements interfaz1, interfaz2 { ... }
```

7.11 POLIMORFISMO

Para los programadores de lenguajes interpretados como PHP, Perl o la Shell de U*ix, la programación de código genérico con variables de función se usa con bastante frecuencia y de manera natural. Por esto, el polimorfismo no les resultará un concepto difícil de comprender puesto que, en cierta forma, es equivalente a este tipo de programación. El siguiente trozo de código es un ejemplo de lo que acabamos de mencionar:

```
function func1() { echo "soy f_UNO\n"; }
function la_f2() { echo "soy f_DOS\n"; }
function f3()    { echo "soy f_TRES\n"; }
$lista_f     = array ("func1", "la_f2", "f3");

foreach ($lista_f as $una_f)
  $una_f();
```

Lo que hacemos es especificar una variable que irá recibiendo distintos identificadores de función y, luego (poniéndole los paréntesis abierto y cerrado), mandar ejecutarla: es PHP quien, en tiempo de ejecución, decide, de entre todas las funciones definidas, a cuál llamar.

Consideremos ahora el siguiente ejemplo paralelo pero, esta vez, con objetos:

```
$objetos = array (new clase1(), new clase2(), new clase3());
foreach ($objetos as $un_obj) {
  $un_obj->metodo_comun(); // redefinido en cada clase
}
```

Vemos, cómo a la variable `$un_obj`, sucesivamente, se le asignan todos los objetos (que son de clases diferentes) que hay en el *array* `$objetos`. Vemos, también, que dentro del bucle se invoca al método `metodo_comun()` que está implementado y redefinido en cada una de las clases de las que se han creado los objetos. En esto consiste el polimorfismo en que, **en tiempo de ejecución**, es el intérprete quien, basándose en la clase a la que pertenezca en cada momento el objeto, decide cuál de las implementaciones del método es la que va a ejecutar.

Dicho al contrario, el polimorfismo permite que un mismo método pueda ser ejecutado, de manera distinta, por un número indeterminado de objetos sin importarnos cómo lo implementa cada uno. El polimorfismo es una herramienta muy potente ya que permite a clases diferentes compartir una misma interfaz, pudiéndose, por tanto, crear código genérico.

El ejemplo propuesto define una clase abstracta, `Campo_Formulario`, que declara el método abstracto `pinta_campo()`, el cual debe generar las etiquetas HTML necesarias para mostrar un elemento de formulario. Esta clase ofrece dos métodos internos que generan el código HTML y *JavaScript* preciso para mostrar (u ocultar) una capa flotante, con un texto a modo de ayuda, cada vez que el usuario se sitúa encima (o sale) del elemento con el ratón.

```
<?
abstract class Campo_Formulario {
  protected $etiqueta;
  protected $id_campo;
  protected $capa_ayuda;

  function __construct($id, $etiq, $ayuda) {
    $this->id_campo=$id;
    $this->etiqueta=$etiq;
    $this->capa_ayuda=$ayuda;
  }

  abstract function pinta_campo();

  protected function pon_eventos_js() {
    $cmd_js='document.getElementById("c_"+this.name).style.visibility';
    return "onfocus='$cmd_js=\"visible\"' onblur='$cmd_js=\"hidden\"'\n";
  }

  protected function pon_capa_ayuda() {
    // el identif de la capa es el prefijo 'c_' + el nombre del elemento
    $s ='position:relative;top:-10;';
    $s.="font-family:Arial;";
```

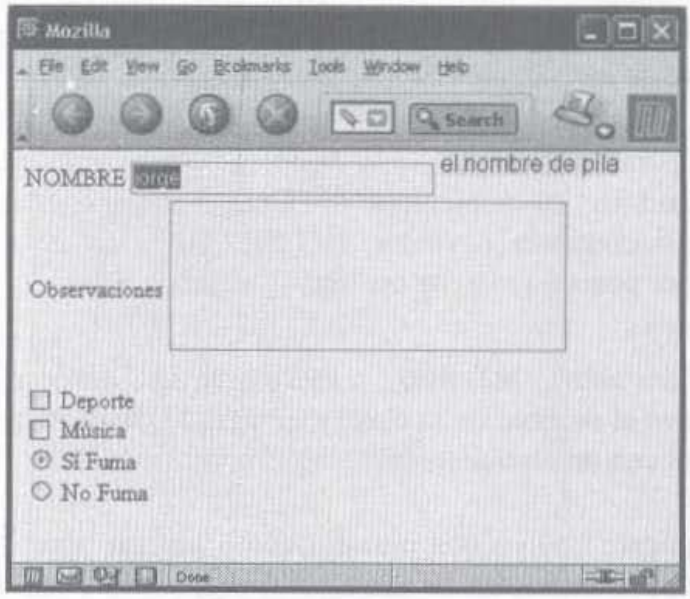
```
    $s.="background-color:#fffbad;";
    $s.='visibility:hidden';
    return "<span id='c_$this->id_campo' style='$s'>
          $this->capa_ayuda</span>\n";
  }

}
```

Declaramos tres clases derivadas (`Campo_Text`, `Campo_TextArea` y `Campo_Casilla`) que tienen la misma interfaz (el método `pinta_campo()`), pero, lógicamente, redefinida cada una de ellas para generar las etiquetas HTML apropiadas. Por ejemplo, los controles `textarea` se dibujan en una tabla de manera que la etiqueta esté en una celda y la caja de texto en otra, estando la etiqueta alineada verticalmente dentro de la celda; en el caso de los `radio` o `checkbox`, la etiqueta y el elemento de formulario se imprimen en orden inverso al de los `text` y `textarea`: primero el elemento y luego la etiqueta.

```
class Campo_Text extends Campo_Formulario {

  private $tamanyo;
  function __construct($id, $etiq, $tamanyo, $ayuda) {
    parent::__construct($id, $etiq, $ayuda);
    $this->tamanyo=$tamanyo;
  }

  function pinta_campo() {
    echo "$this->etiqueta\n";
    echo "<input type='text' name='$this->id_campo' size='$this->tamanyo' ";
    echo $this->pon_eventos_js() . ">\n";
    echo $this->pon_capa_ayuda();
    echo "<br />\n";
  }
}

class Campo_TextArea extends Campo_Formulario {
  private $lineas;

  function __construct($id, $etiq, $lineas, $cols, $ayuda) {
    parent::__construct($id, $etiq, $ayuda);
    $this->lineas=$lineas;
    $this->cols=$cols;
  }

  function pinta_campo() {
  echo "<table border='0'><tr><td valign='middle'>
        $this->etiqueta</td><td>
        <textarea type='text' name='$this->id_campo'
          rows='$this->lineas' cols='$this->cols' " .
          $this->pon_eventos_js()                  . ">" .
       "</textarea> "                                    .
       $this->pon_capa_ayuda()                           .
       "</td></tr></table>";
  }
}
```

```
class Campo_Casilla extends Campo_Formulario {

  private $valor;
  private $preselecc;
  private $tipo_casilla;

  function __construct($tipo_casilla, $id, $etiq, $valor, $preselecc,$ayuda){
    parent::__construct($id, $etiq, $ayuda);
    $this->valor=$valor;
    if ($preselecc == 1)
      $this->preselecc=' checked';
    else
      $this->preselecc='';
    if ($tipo_casilla == 'radio' || $tipo_casilla == 'checkbox')
      $this->tipo_casilla=$tipo_casilla;
    else
      die("ERROR: la casilla debe ser 'radio' o 'checkbox'");
  }

  function pinta_campo() {
    echo "<input type='$this->tipo_casilla' name='$this->id_campo'
          value='$this->valor' " .
         $this->pon_eventos_js() . "
         $this->preselecc>\n
          $this->etiqueta\n" .
          $this->pon_capa_ayuda() . "
          <br />\n";
  }
}

echo "<form onsubmit='return false'>";
$campos = array (
   new Campo_Text    ("nombre", "NOMBRE", 40, "el nombre de pila"),
   new Campo_TextArea("observ", "Observaciones", 40, 40, "Lo que quiera"),
   new Campo_Casilla ("checkbox", "af1", "Deporte", 'si', 0, "Deportista?"),
   new Campo_Casilla ("checkbox", "af2", "Música", 'si', 0, "Melómano?"),
   new Campo_Casilla ("radio", "fumador", "Sí Fuma", 'si', 0, "Fumar Mata!"),
   new Campo_Casilla ("radio", "fumador", "No Fuma", 'no', 1, "Fumar Mata!")
);

foreach ($campos as $un_obj)
  $un_obj->pinta_campo();

echo "</form>";
?>
```

Por último, el uso que hacemos de las clases consiste, primero, en crear un *array* de objetos de distintas clases distintos y, segundo, con un bucle `foreach` ejecutar la interfaz común (se trate del objeto que se trate). La salida de este *script* es:



7.12 FUNCIONES RELACIONADAS

Para finalizar el capítulo dedicado a la POO, hacemos una somera descripción de las funciones que guardan relación con las clases y los objetos, el operador `instanceof` y la constante `__METHOD__`.

- `class_exists(clase)`: Si `clase` está definida devuelve TRUE, en caso contrario devuelve FALSE.

- `get_declared_clases()`: Devuelve un *array* numérico con las clases que están disponibles en el *script*: las nativas de PHP y las definidas por el usuario.

- `get_class(obj)`: Para el objeto `$obj`, devuelve el nombre de la clase de la que es una instancia.

- `get_parent_class(obj_o_clase)`: Devuelve el nombre de la superclase de la que está derivado el objeto o la clase pasada como parámetro.

- `get_class_vars(clase)`, `get_object_vars(obj)`: Devuelve un *array* asociativo con las propiedades (públicas) de que dispone la clase (o el objeto), respectivamente.

- `get_class_methods(clase)`: Devuelve un *array* numérico con el nombre de los métodos de que dispone la clase.

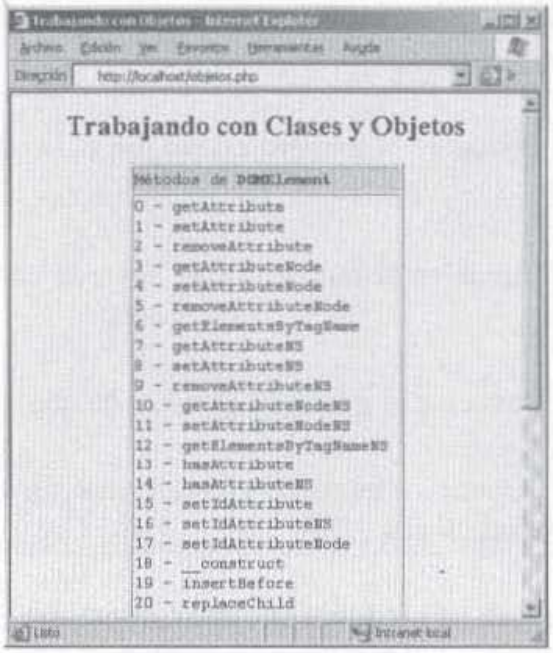
- Operador **`instanceof`**: `$obj instanceof nombre_clase`. Si la variable `$obj` es un ejemplar de la clase `nombre_clase`, devuelve TRUE, en caso contrario devuelve FALSE. OJO, no es una función sino un operador puesto que tiene operandos a ambos lados.

- Constante **`__METHOD__`**: cuando se usa desde dentro de un método, devuelve el nombre de la clase y el método desde el que se está ejecutando. Si se usa en una función, devuelve el nombre de ésta.

El siguiente código nos muestra cómo utilizar algunas de estas funciones para recuperar información sobre una clase u objeto en particular:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con Objetos</TITLE>
   <STYLE>
      TABLE, P, A {font: 14px "monospace"}
      P {font: 12px "Verdana"; color: red;}
   </STYLE>
 </HEAD>
 <BODY>
   <CENTER>
   <H2>Trabajando con Clases y Objetos</H2>
<?php

   $docXML = new domDocument();
   $elemento = $docXML->createElement("Elemento");

   $obj_explorar = $elemento;

   echo "<table border='1'>";
   echo "<tr><td colspan='2' bgcolor='yellow'>";
   echo " Métodos de <b>".get_class($obj_explorar)."</b>";
   echo "</td></tr><tr><td>";

   $metodos = get_class_methods(get_class($obj_explorar));
   foreach ($metodos as $pos => $nombre) {
       echo "$pos - $nombre<BR>";
   }
   echo "</td></tr></table>";
?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



7.13 EXCEPCIONES

PHP 5 incorpora, al igual que lenguajes como Java, Python, C#, etc., mecanismos para la gestión de excepciones. Para ello, provee la estructura de control `try/catch`, que tiene el aspecto siguiente:

```
try {
// codigo a proteger
} catch {
// codigo para tratar el error producido
}
```

La idea es que si, dentro del bloque `try`, detectamos cualquier anomalía, generamos una excepción (con la palabra reservada `throw`), gracias a la cual, el flujo del programa saldrá de este bloque para saltar a ejecutar las sentencias del bloque `catch`. Se pretende que el código sea más legible: en lugar de controlar los errores en cada línea donde pueda producirse un error, los juntamos y separamos del flujo normal de la aplicación.

La gestión de las excepciones se suele ver dentro de la POO puesto que se basa en una clase nativa de PHP 5: `Exception`. De hecho, la sentencia `throw` sólo permite objetos de esta clase, o derivados de ella. El constructor de `Exception` admite hasta dos parámetros: una cadena de caracteres (que describiría el error) y un número entero (asignado por nosotros que se correspondería con un código de error). Los métodos que podemos usar son:

- `getMessage()`: Devuelve la cadena de caracteres pasada como primer parámetro al constructor.
- `getCode()`: Devuelve el número entero pasado como segundo parámetro al constructor.
- `getLine()`: Devuelve el número de la línea de código en que ocurrió la excepción.
- `getFile()`: Devuelve el nombre del *script* en que ocurrió la excepción.
- `getTrace()`: Devuelve un *array* con información sobre los puntos donde han ocurrido excepciones.
- `getTraceAsString()`: Igual que el anterior pero en forma de cadena de caracteres.

A continuación, el código de un ejemplo anterior que generaba una muestra de un fichero gráfico lo reescribimos para tratar con excepciones los posibles errores que se puedan producir. En concreto, si existe el fichero del cual se quiere sacar la muestra y si el contenido de este fichero tiene formato gráfico PNG:

```
<?
try {
  $nombre_fich='fich.txt';
  if (! file_exists($nombre_fich))
    throw new Exception("No existe $nombre_fich", 99);


  $ancho_muestra=100;
  $alto_muestra=100;
  $imag_original = @ImageCreateFromPNG($nombre_fich);
  if (! $imag_original)
    throw new Exception("$nombre_fich no tiene formato PNG",88);

  $imag_muestra  = ImageCreate($ancho_muestra, $alto_muestra);
  ImageCopyResized($imag_muestra, $imag_original,
                   0, 0, 0, 0,
                   $ancho_muestra, $alto_muestra,
                   ImageSX($imag_original),
                   ImageSY($imag_original));
```

```
  ob_start();
  ImagePNG($imag_muestra);
  $en_b64=chunk_split(base64_encode(ob_get_contents()));
  ob_end_clean();
  echo "<img src='$en_b64'>";

} catch (Exception $e) {
  echo "ERROR "          . $e->getCode() .
       " en la línea " . $e->getLine() .
       ": '"             . $e->getMessage() . "'<br>";
}

?>
```

Por supuesto, podemos crear subclases de `Exception`, simplemente extendiéndola, con las funcionalidades que consideremos oportunas.

CAPÍTULO 8

FUNCIONES DE FECHA Y HORA

8.1 INTRODUCCIÓN

Es muy importante en muchos problemas y aplicaciones llevar un control con la fecha y la hora en un determinado momento, o bien, conocer la fecha para saber si tenemos que ejecutar un programa u otro...; existe un montón de circunstancias donde es necesario conocer estos datos. PHP nos ofrece una gran variedad de funciones para abordar con mayor rapidez y de una forma más sencilla los distintos problemas relacionados con el manejo de fechas y tiempos que nos puedan ir saliendo a la hora de realizar nuestros programas.

En casi todos los sistemas informáticos hay una fecha de inicio común, a partir de la cual se empieza a contar el tiempo. En el caso de los sistemas UNIX la fecha elegida como comienzo es el día 1 de enero de 1970 a las 00:00:00 GMT, fecha que se conoce como el principio de la era UNIX. El contador de tiempo se conoce como marca de tiempo (*timestamp*) y representa el número de segundos transcurridos desde una fecha dada. En PHP todas las funciones de fecha/hora que trabajan con marcas de tiempo hacen referencia a esta fecha.

8.2 FUNCIONES DE FECHA Y HORA

La siguiente tabla nos muestra el resumen de las funciones de fecha/hora proporcionadas por PHP, al igual que una breve descripción de ellas:

función	descripción
`time()`	Obtiene la marca de tiempo UNIX actual
`checkdate()`	Valida una fecha en formato gregoriano
`date()`	Da formato a la hora y la fecha locales

función	descripción
`getdate()`	Obtiene información sobre la fecha y la hora locales
`gettimeofday()`	Obtiene la hora actual
`gmdate()`	Formatea la fecha y la hora a formato GMT
`gmmktime()`	Obtiene la marca de tiempo UNIX de una fecha con formato GMT
`gmstrftime()`	Formatea la fecha y la hora con formato GMT a las características locales
`microtime()`	Obtiene la marca de tiempo UNIX actual en microsegundos
`mktime()`	Obtiene la marca de tiempo UNIX para una fecha dada
`strftime()`	Formatea la hora actual de acuerdo con las características locales
`strtotime()`	Traduce una fecha u hora descrita en inglés a su correspondiente marca de tiempo UNIX

Pasamos ahora a ver cada una de ellas en mayor profundidad:

- `time()`: Devuelve la marca de tiempo correspondiente al instante en que se ejecuta.

- `checkdate (mes, dia, anio)`: Verifica si la fecha que se le pasa como parámetro es una fecha correcta. Esta función es bastante útil en los formularios en los cuales hay que rellenar campos de tipo fecha.

Si pasamos una fecha correcta, es decir, todos los parámetros están dentro de los rangos establecidos: `anio` es un número entero positivo, `mes` es un número entre 1 y 12 y `dia` entre 1 y 31 (teniendo en cuenta el total de días que hay en cada mes, incluyendo los años bisiestos); la función devuelve un valor verdadero; en caso contrario, la función devuelve un valor falso.

- `date (formato [,timestamp])`: Esta función nos permite darle un formato específico a una cadena que contendrá una fecha y una hora. Acepta como parámetros una cadena de `formato` y un parámetro `timestamp`; si éste se omite, se tomará el instante de ejecución de la orden.

Hay una serie de parámetros que tienen un significado propio y son reconocidos dentro de la cadena de `formato`. Estos caracteres son:

valores	descripción
a	"a.m." o "p.m."
A	"A.M." o "P.M."
d	Día del mes con dos dígitos (de "01" a 31")
D	Día de la semana con tres caracteres
F	Nombre del mes
h	Hora en formato "01" a "12"
H	Hora en formato "00" a "23"
g	Hora en formato "1" a "12" (sin cero)
G	Hora en formato "0" a "23" (sin cero)
i	Minutos de "00" a "59"
j	Día del mes en formato "1" a "31"
l	Día de la semana, en texto completo
L	1: si es un año bisiesto 0: si no es un año bisiesto
m	Mes de "01" a "12"
M	Mes con tres caracteres
n	Mes de "1" a "12" (sin cero inicial)
s	Segundos de "00" a "59"
S	Sufijo ordinal en inglés ("th", "nd")
t	Número de días del mes dado, de "28" a "31"
U	Segundos transcurridos desde el valor de inicio (01-01-1970)
w	Día de la semana de "0" (domingo) a "6" (sábado)
Y	Año con cuatro dígitos
y	Año con dos dígitos
z	Día del año de "0" a "365"
Z	Diferencia horaria en segundos de "43200" a "43200"

Los caracteres distintos a los que aparecen en la tabla, que estén dentro de la cadena `formato`, se imprimirán tal cual aparecen. Para que los caracteres utilizados en la cadena `formato` se puedan imprimir, es necesario enmascararlos, es decir, deben ir precedidos del carácter "\".

❑ `getdate ([timestamp])`: Esta función devuelve un *array* asociativo que contiene información sobre la fecha y hora asociadas a la marca de tiempo, `timestamp`, pasada como parámetro. En caso de no pasar ningún parámetro a la función, ésta obtendrá la marca de tiempo del instante en que se ejecuta.

La estructura del *array* asociativo devuelto es la siguiente:

clave	descripción
`seconds`	Identifica los segundos.
`minutes`	Identifica los minutos.
`hours`	Identifica las horas.
`mday`	Identifica el día del mes.
`wday`	Identifica, en número, el día de la semana.
`mon`	Identifica, en número, el mes.
`year`	Identifica, en número, el año.
`yday`	Identifica, en número, el día del año.
`weekday`	Identifica, en texto, el día de la semana.
`month`	Identifica, en texto, el mes.

- `gettimeofday()`: Esta función obtiene la hora actual en un *array* asociativo, cuya estructura contiene los siguientes campos:

campo	descripción
`sec`	Segundos.
`usec`	Microsegundos.
`minuteswest`	Minutos al oeste del meridiano de Greenwich.
`dsttime`	Corrección horaria entre los horarios de verano e invierno.

- `gmdate (format [, timestamp])`: Esta función es muy parecida a la función `date()` anteriormente vista, con una salvedad: la hora devuelta por esta función tiene formato GMT (*Greenwich Mean Time*).

> **NOTA:** Este formato se toma como referencia para las diferencias horarias. Como curiosidad, la diferencia horaria de España respecto a la de Greenwich en verano es de +2 horas y en invierno es de +1 hora.

- `gmmktime(hora, min, seg, mes, dia, anio [,is_dst])`: Es muy parecida a la función `mktime()`, a excepción de que los parámetros que se pasan en la llamada a la función representan la fecha en formato GMT.

- `gmstrftime (format, timestamp)`: Da formato a una fecha/hora GMT según las convenciones locales. Al igual que en las funciones anteriores, la fecha devuelta es la de GMT; por lo demás, es muy parecida a la función `strftime()`.

❑ `microtime (void)`: Devuelve una cadena compuesta de dos elementos "`msec sec`". La segunda parte, `sec`, representa los segundos transcurridos desde la fecha inicial de referencia, es decir, el 1 de enero de 1970 a las 00:00:00, mientras que la primera parte, `msec`, representa los microsegundos restantes. Ambas porciones se devuelven en unidades de segundo.

NOTA: Este función sólo está disponible en los sistemas operativos que soporten la llamada al sistema `gettimeofday()`.

❑ `mktime (hora, min, seg, mes, dia, anio [,is_dst])`: Esta función devuelve la marca de tiempo (el número de segundos transcurridos desde el 1 de enero de 1970 a las 00:00:00), correspondiente a la fecha y hora pasadas a la función como parámetros. Esta función es especialmente útil para realizar cálculos matemáticos con las fechas o validaciones de ellas.

NOTA: Los parámetros se toman de izquierda a derecha de forma que, si alguno de ellos se omite, se sustituye por el valor de la fecha y hora actual correspondiente. Ninguno de los valores de `dia`, `mes` y `anio` pueden tomar el valor 0. El parámetro `is_dst` se utiliza para indicar si se tiene en cuenta el horario de verano o no; por defecto tiene el valor -1, que indica a PHP que decida cuál es el uso correcto del parámetro.

Es muy importante tener en cuenta que esta función posee la característica avanzada de calcular la marca de tiempo para parámetros que estén fuera de rango: aproxima los datos introducidos a la fecha más cercana correcta, es decir, cuando ejecutamos la función sobre una fecha incorrecta, por ejemplo, "32 de marzo de 2001", obtendremos el valor de *timestamp* correspondiente al "1 de abril de 2002".

También hay que tener en cuenta que la función admite que el parámetro `anio` sea codificado como un valor de dos dígitos. En este caso, la función realiza la siguiente equiparación automática de fechas: los valores comprendidos entre 0 y 69 se ajustan a los años 2000 a 2069 y los valores entre 70 y 99, a los años 1970 a 1999.

NOTA: En los sistemas que codifican el tiempo como un entero con signo de 32 bits, los valores válidos para el año están entre 1902 y 2037.

Existe una última aclaración de esta función consistente en que el último día de cada mes puede indicarse como el día "0" del mes anterior.

❑ `strftime (format [,timestamp])`: Esta función nos permite dar un formato específico de hora y fecha a la marca de tiempo que se le pasa como parámetro. En caso de no proporcionar este parámetro, se tomará por defecto la marca de tiempo correspondiente al instante en que se ejecuta la

función. Los formatos posibles a tener en cuenta se especifican en la siguiente tabla:

valores	descripción
%a	Nombre del día de la semana (abreviado)
%A	Nombre del día de la semana
%b	Nombre del mes (abreviado)
%B	Nombre del mes
%c	Fecha y hora en el idioma actual
%d	Día del mes de 00 a 31
%H	Hora de 00 a 23
%I	Hora de 01 a 12
%j	Día del año de 001 a 366
%m	Mes de 01 a 12
%M	Minutos de 00 a 59
%p	"am" o "pm", según corresponda a la hora dada
%S	Segundos de 00 a 59
%w	Día de la semana de 0 a 6 (0 se corresponde con domingo)
%W	Número de semana en el año (el primer lunes del año es el primer día de la primera semana)
%x	Fecha sin hora
%X	Hora sin fecha
%y	Año de 00 a 99
%Y	Año en cuatro dígitos
%Z	Zona horaria
%%	%

NOTA: No se permiten todas las combinaciones de formatos posibles.

- `strtotime (cad_fecha [,timestamp])`: Esta función traduce una cadena que contiene un texto en inglés que hace referencia a una fecha en su correspondiente marca de tiempo relativa a la marca de tiempo dada en el parámetro opcional `timestamp`, o bien, a la marca de tiempo actual, si este parámetro no se proporciona en la llamada a la función.

Esta función se comporta de acuerdo con la sintaxis de fechas de GNU; por tanto, se le pueden pasar como argumento cadenas del tipo `now`, `next Friday`, `+1 day`, etc.

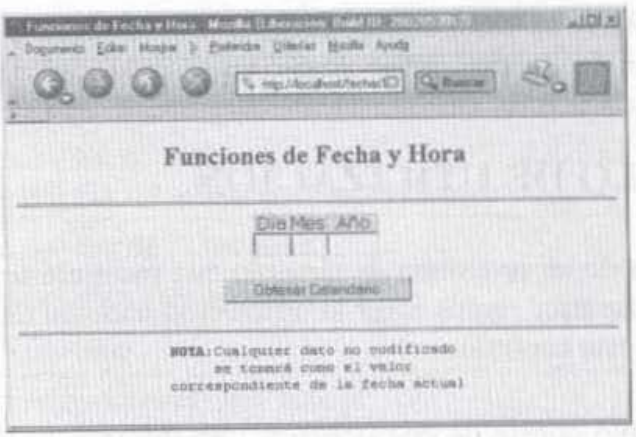
8.3 EJEMPLO DE UTILIZACIÓN

A través de un programa de ejemplo que mostrará un calendario con el mes deseado por el usuario, vamos a ver la utilización habitual de las funciones de fecha y hora vistas en este capítulo.

El ejemplo consta de dos ficheros, uno primero de HTML que contiene un formulario en el que el usuario podrá introducir el día, mes y año de una fecha en particular. El código de esta primera página es el siguiente:

```
<HTML>
 <HEAD>
   <TITLE>Funciones de Fecha y Hora</TITLE>
   <STYLE>
     TABLE {font-family:Verdana;font:12px;}
     INPUT {text-align:right;}
   </STYLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Funciones de Fecha y Hora</H2><HR>
     <FORM METHOD="GET" ACTION="fechas3.php">
         <TABLE CELLPADDING="0" CELLSPACING="0">
           <TR BGCOLOR="YELLOW" ALIGN="CENTER">
              <TD>Dia</TD><TD>Mes</TD><TD>Año</TD></TR>
           <TR>
              <TD><INPUT TYPE="TEXT" NAME="dia" MAXLENGTH="2" SIZE="3"></TD>
              <TD><INPUT TYPE="TEXT" NAME="mes" MAXLENGTH="2" SIZE="3"></TD>
              <TD><INPUT TYPE="TEXT" NAME="anio" MAXLENGTH="4" SIZE="5"></TD>
           </TR>
         </TABLE><BR>
         <INPUT TYPE="SUBMIT" VALUE="Obtener Calendario">
     </FORM><HR>
     <CODE>
        <B>NOTA:</B>Cualquier dato no codificado
        <BR>se tomará como el valor
        <BR>correspondiente de la fecha actual
     </CODE>
   </CENTER>
 </BODY>
</HTML>
```

El código se visualizaría del siguiente modo:



El segundo fichero del ejemplo es el *script*, que se encarga de generar, a partir de la fecha dada en el formulario, el calendario del mes que contiene dicha fecha (si el usuario no proporciona ninguna fecha, el programa asume la del día en curso).

> **NOTA:** Se ha optado por dividir el ejemplo en dos ficheros para mantener el código principal del ejemplo lo más legible posible, si bien ambos podrían haberse combinado en uno solo.

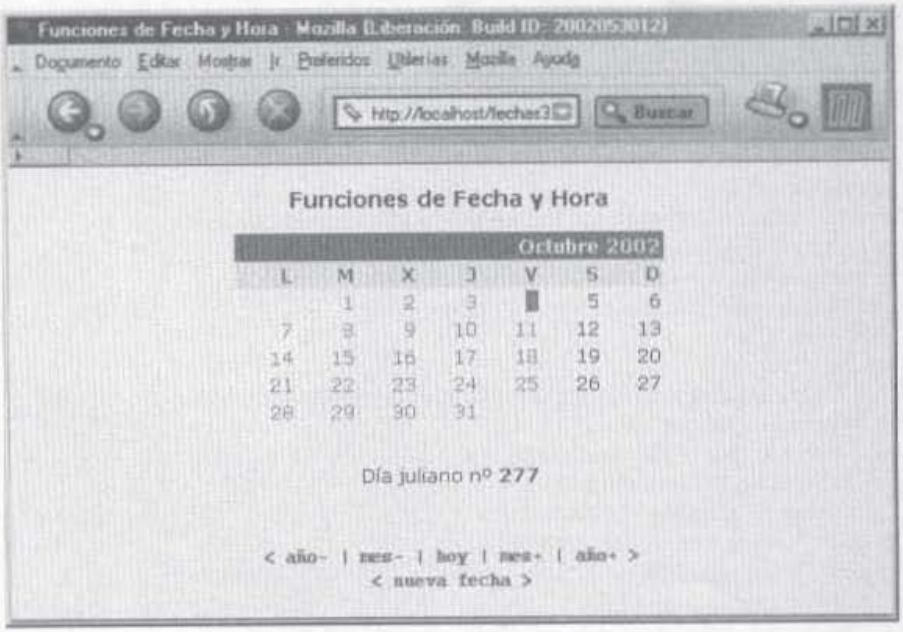
El código completo del segundo *script* es el siguiente:

```
<HTML>
 <HEAD>
   <TITLE>Funciones de Fecha y Hora</TITLE>
   <style>
    body {font: 12px Verdana;}
    table {font: 12px Verdana;color:orange;text-align:right;}
    tr.cabecera {background-color:#808080;color:#F8F8F8;font-weight:bold;}
    tr.semana {background-color:#FFFBAD;color:#808080;font-weight:bold;}
    a {text-decoration:none;color:orange;}
    a.marcado {background-color:green;}
    a.festivo {color:#B00000;}
    a.opc {color:gray;font-weight:bold;}
    p.error {font:14px;color:red;font-weight:bold;}
  </style>
 </HEAD>

 <?php

$meses_txt=array("","Enero","Febrero","Marzo","Abril","Mayo","Junio","Julio",
"Agosto","Septiembre","Octubre","Noviembre","Diciembre");
   $dias_txt=array("L","M","X","J","V","S","D");

   // Función que transforma el dia de la semana para que el 0 sea el lunes y
   // el 6 el domingo
   function actualiza_dia_semana($dia){
     return ($dia>0)?$dia-1:6;
   }
   // Función que informa si un día pertenece al fin de semana
   function festivo($dia){
     return ($dia>4)?true:false;
```

```
    }

 ?>
 <BODY>
   <CENTER>
     <H3>Funciones de Fecha y Hora</H3>
     <?php
       $hoy=getdate();
       $dia=!empty($_GET['dia'])?$_GET['dia']:$hoy['mday'];
       $mes=!empty($_GET['mes'])?$_GET['mes']:$hoy['mon'];
       $anio=!empty($_GET['anio'])?$_GET['anio']:$hoy['year'];
       if($anio<=99) $anio+=2000;
       if(!checkdate($mes,$dia,$anio)||$anio<1971){
         echo "<HR>";
         echo "<P CLASS='error'>ERROR: La fecha introducida no es
válida...</P>";
         echo "<BR>< <A HREF='fechas3.html'>volver</A> ><HR>";
       } else {
         // obtenemos el día de la semana del primer día del mes

$primer_dia=actualiza_dia_semana(date("w",mktime(0,0,0,$mes,1,$anio)));

         // obtenemos el último día del mes
         $ultimo_dia=date("t",mktime(0,0,0,$mes,1,$anio));

       // escritura de la tabla que representa el calendario de un MES
       echo "<TABLE BORDER='0' CELLPADDING='2' CELLSPACING='0'
WIDTH='50%'>\n";
       // escribir la cabecera que incluye el mes y el año del calendario
       echo "<TR CLASS='cabecera'>";
       echo "<TD COLSPAN='7'>",$meses_txt[$mes]," $anio</TD></TR>\n";
       // escribir la cabecera que indica los días de la semana
       echo "<TR CLASS='semana'>";
       for ($i=0; $i<7; $i++)
          echo "<TD>$dias_txt[$i]</TD>";
       echo "</TR>\n<TR>";

       // escribir los días del mes
       $contador_de_dias=1;
       // huecos de la primera semana
       if($primer_dia>0)
          echo "<TD COLSPAN='$primer_dia'> </TD>";
       // resto de semanas
       $dia_semana=$primer_dia;
       while($contador_de_dias <= $ultimo_dia){
           echo "<TD>";
           echo "<A
HREF='fechas3.php?dia=$contador_de_dias&mes=$mes&anio=$anio'";
           if($contador_de_dias==$dia)
               echo " class='marcado'";
           if(festivo($dia_semana))
              echo " class='festivo'";
           echo ">$contador_de_dias</A></TD>";
           $contador_de_dias++;
           if($dia_semana==6){
              echo "</TR>\n<TR>";
              $dia_semana=0;
           }
           else $dia_semana++;
       }
```

```
      echo "</TR></TABLE><BR>\n";
      $fecha=getdate(mktime(0,0,0,$mes,$dia,$anio));
      echo "<P STYLE='color:red;'>Día juliano nº <B>";
      echo $fecha['yday']+1,"</B></P><BR>\n";
      $url = "fechas3.php?dia=$dia&mes=$mes&anio=".($anio-1);
      echo "<PRE>\n< <A CLASS='opc' HREF='$url'>año-</A> | ";
      $url = "fechas3.php?dia=$dia";
      if ($mes==1)
          $url .= "&anio=".($anio-1)."&mes=12";
      else
         $url .= "&anio=$anio&mes=".($mes-1);
      echo "<A CLASS='opc' HREF='$url'>mes-</A> | ";
      echo "<A CLASS='opc' HREF='fechas3.php?'>hoy</A> | ";
      $url = "fechas3.php?dia=$dia";
      if ($mes==12)
          $url .= "&anio=".($anio+1)."&mes=1";
      else
          $url .= "&anio=$anio&mes=".($mes+1);
      echo "<A CLASS='opc' HREF='$url'>mes+</A> | ";
      $url = "fechas3.php?dia=$dia&mes=$mes&anio=".($anio+1);
      echo "<A CLASS='opc' HREF='$url'>año+</A> >";
      echo "<BR>< <A CLASS='opc' HREF='fechas3.html'>nueva fecha</A>
></PRE>";
    }
  ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado de ejecutarlo pasándole como fecha el "30/9/2002" se visualiza en la siguiente imagen:



Vamos a ver pormenorizadamente cada una de las acciones que realiza el código. Lo primero que hacemos es recuperar la información enviada desde el

cliente para procesarla. Del código del primer fichero HTML podemos observar que hemos elegido por utilizar el método GET:

```
<FORM METHOD="GET" ACTION="fechas3.php">
    .
    .
    .
</FORM>
```

Como se ha comentado anteriormente, el usuario puede optar por no completar todos los datos referentes a la fecha con la que quiere trabajar, en cuyo caso se asumirá que desea utilizar los valores de la fecha actual del sistema.

```
$hoy=getdate();
$dia=!empty($_GET['dia'])?$_GET['dia']:$hoy['mday'];
$mes=!empty($_GET['mes'])?$_GET['mes']:$hoy['mon'];
$anio=!empty($_GET['anio'])?$_GET['anio']:$hoy['year'];
```

En la variable `$hoy` obtenemos, haciendo uso de la función `getdate()`, un *array* que contiene la información de la fecha actual del sistema. De este modo, si el usuario no proporciona cualquiera de los datos solicitados (día, mes y año que se almacenan en las variables `$dia`, `$mes` y `$anio`, respectivamente), los obtendremos de la fecha actual del sistema.

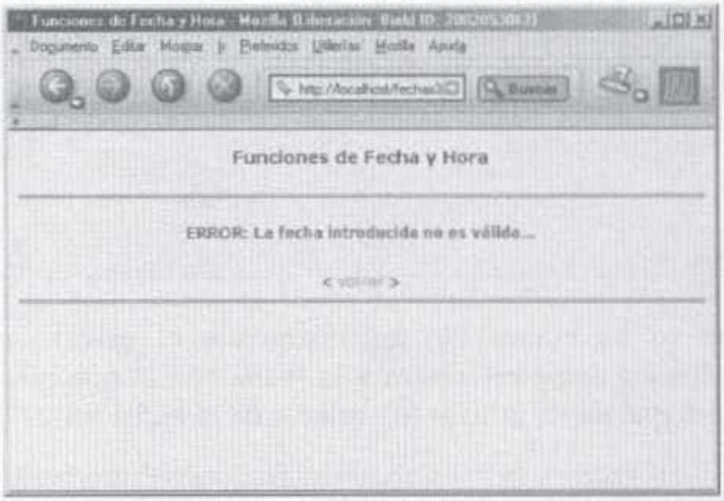
Antes de pasar a validar la fecha, se hace una última comprobación con el año mandado por el usuario; si éste se compone de dos dígitos solamente, se le complementa hasta cuatro sumándole 2000.

```
if($anio<=99) $anio+=2000;
```

Una vez que la fecha ha sido recuperada y completada, se valida a través de la función `checkdate()`. Además, se comprueba que el año proporcionado no sea inferior a 1970. Hay que tener en cuenta que, aunque la fecha esté correctamente formada, si el año es anterior a 1970 (año de comienzo de la era UNIX), todas las funciones que generan una marca de tiempo de tipo UNIX fallarían:

```
if(!checkdate($mes,$dia,$anio)||$anio<1971){
   echo "<HR><P CLASS='error'>ERROR: La fecha introducida no es
válida...</P>";
   echo "<BR>< <A HREF='fechas1.html'>volver</A> ><HR>";
} else {
  // tratamiento de fecha correcta
  .
  .
  .
}
```

En caso de proporcionar una fecha errónea, se mostraría la siguiente pantalla de error:



Si la comprobación ha validado la fecha, pasamos a generar el calendario del mes que contiene dicha fecha. Lo primero que hacemos es recuperar algunos datos esenciales para poder generar correctamente el calendario.

- Día de la semana del primer día del mes (almacenado en la variable `$primer_dia`): para ello, utilizamos las funciones `date()` y `mktime()`. Con esta última obtenemos la marca de tiempo del primer día del mes de la fecha dada. Este valor se pasa a su vez como argumento a la función `date()` indicándole en la llamada, a través de la opción de formato "w", que lo que deseamos obtener es el día de la semana. El valor devuelto varía entre 0 para el domingo y 6 para el sábado. Como dicho valor no coincide con la estructura de semana que deseamos utilizar (el lunes como primer día de la semana), llamamos a la función `actualiza_dia_semana()`, que se encarga de actualizarlo con nuestras preferencias.

```
// obtenemos el día de la semana del primer día del mes
function actualiza_dia_semana($dia){
  return ($dia>0)?$dia-1:6;
}
.
.
.
$primer_dia=actualiza_dia_semana(date("w",mktime(0,0,0,$mes,1,$anio)));
```

- Número de días del mes (almacenado en la variable `$ultimo_dia`): para ello, utilizamos también la misma combinación de funciones `date()` y `mktime()` que anteriormente, sólo que esta vez solicitamos a través de la opción de formato "t" una información diferente:

```
// número de días del mes
$ultimo_dia=date("t",mktime(0,0,0,$mes,1,$anio));
```

Una vez obtenidos estos valores, podemos generar el calendario del mes deseado. Para ello vamos a utilizar una tabla en la que la primera fila corresponde a una columna que contiene el nombre del mes y el año consignados en la fecha:

```
// escritura de la tabla que representa el calendario de un MES
echo "<TABLE BORDER='0' CELLPADDING='2' CELLSPACING='0' WIDTH='50%'>\n";
// escribir la cabecera que incluye el mes y el año del calendario
echo "<TR CLASS='cabecera'>";
echo "<TD COLSPAN='7'>",$meses_txt[$mes]," $anio</TD></TR>\n";
```

Cabe destacar que el *array* `$meses_txt`, del cual se obtiene el nombre del mes solicitado en castellano, pues de las funciones proporcionadas por PHP sólo podemos obtener información textual en inglés, tiene una primera posición vacía para hacer coincidir el número de mes proporcionado por el usuario con su posición dentro del *array*.

```
$meses_txt=array("","Enero","Febrero","Marzo",.....,"Noviembre","Diciembre");
```

Las siguientes filas están compuestas de siete columnas cada una: una por día. La primera de estas filas contiene una etiqueta por cada uno de los días de la semana:

```
// escribir la cabecera que indica los días de la semana
echo "<TR CLASS='semana'>";
for ($i=0; $i<7; $i++)
   echo "<TD>$dias_txt[$i]</TD>";
echo "</TR>\n<TR>";
```

A partir de aquí se completan las diferentes filas con cada una de las semanas que componen el mes. Hay que tener en cuenta que en la primera y última de estas filas se pueden producir columnas vacías, puesto que es habitual que un mes no comience en lunes, o bien, no acabe en domingo. Por esto, el código que genera el calendario debe considerar si se produce esta circunstancia. Una primera aproximación a dicho código sería el siguiente bucle:

```
// escribir los días del mes
$contador_de_dias=1;
// huecos de la primera semana
if($primer_dia>0)
   echo "<TD COLSPAN='$primer_dia'> </TD>";
// resto de semanas
$dia_semana=$primer_dia;
while($contador_de_dias <= $ultimo_dia){
   echo "<TD>$contador_de_dias</TD>";
   $contador_de_dias++;
   if($dia_semana==6){
      echo "</TR>\n<TR>";
      $dia_semana=0;
   }
   else $dia_semana++;
}
```

Vamos a desglosar los diferentes elementos de esta porción de código. La variable `$contador_de_dias` lleva la cuenta de los días del mes seleccionado, comenzando, como es lógico, por el primer día: el 1. La primera sentencia `if` soluciona la generación de espacios en blanco que se producen cuando el mes a generar no comienza por lunes (día 0). En el bucle `while` posterior, recorremos todos los días que pertenecen al mes dado (dentro de este bucle se incrementa la variable `$contador_de_dias`), generando una celda de la tabla por cada uno de los días. Gracias a la sentencia `if` que contiene, se genera una fila por cada semana contenida en el mes; para ello, hace uso de la variable `$dia_semana` que, en todo momento, indica el día de la semana que estamos generando (siempre contiene valores entre 0 y 6). En principio, no hace falta ningún tratamiento específico para los posibles huecos que se generan al final del mes.

Una segunda revisión del código nos debería permitir diferenciar entre los días festivos, en principio, sábados y domingos, del resto de días laborables de la semana. Para ello, podemos completar el código anterior modificando el bucle `while`, que escribía los días de cada semana de la siguiente forma:

```
$dia_semana=$primer_dia;
while($contador_de_dias <= $ultimo_dia){
   echo "<TD>";
   if(festivo($dia_semana)) echo " class='festivo'";
   echo ">$contador_de_dias</TD>";
   $contador_de_dias++;
   if($dia_semana==6){
     echo "</TR>\n<TR>";
     $dia_semana=0;
   }
   else $dia_semana++;
}
```

Así, antes de escribir la celda correspondiente a un día, se comprueba si éste es festivo (hacemos uso de la función `es_festivo()`, que devuelve `true` en caso de que el día pasado como parámetro sea sábado o domingo (valores 5 y 6, respectivamente), en cuyo caso se le modifica su aspecto haciendo que pertenezca a una clase diferenciada, la clase "`festivo`".

Otra mejora consiste en marcar de una forma diferenciada el día concreto para el que el usuario solicitó el calendario. Para ello, sólo tenemos que completar el código anterior de la siguiente manera:

```
$dia_semana=$primer_dia;
while($contador_de_dias <= $ultimo_dia){
   echo "<TD>";
   if($contador_de_dias==$dia) echo " class='marcado'";
   if(festivo($dia_semana))    echo " class='festivo'";
   echo ">$contador_de_dias</TD>";
   $contador_de_dias++;
   if($dia_semana==6){
```

```
        echo "</TR>\n<TR>";
        $dia_semana=0;
    }
    else $dia_semana++;
}
```

De este modo, la celda del día solicitado por el usuario tiene un aspecto diferenciado que se debe al estilo particular al que pertenece: "marcado".

Una funcionalidad más que tiene el *script* es aquélla por la que el usuario pueda pulsar y seleccionar cualquiera de los días que le muestra el calendario. Para ello, se convierte cada una de las celdas correspondientes a un día en un hiperenlace que llama al mismo *script* pasándole como argumento la nueva fecha seleccionada. El código quedaría del siguiente modo:

```
$dia_semana=$primer_dia;
while($contador_de_dias <= $ultimo_dia){
  echo "<TD>";
  echo "<A HREF='fechas1.php?dia=$contador_de_dias&mes=$mes&anio=$anio'";
  if($contador_de_dias==$dia) echo " class='marcado'";
  if(festivo($dia_semana))    echo " class='festivo'";
  echo ">$contador_de_dias</A></TD>";
  $contador_de_dias++;
  if($dia_semana==6){
     echo "</TR>\n<TR>";
     $dia_semana=0;
  }
  else $dia_semana++;
}
```

Como valor añadido al calendario, el código calcula la fecha juliana (los días del año se numeran desde el 1 –1 de enero– al 364 ó 365 –31 de diciembre– dependiendo de si el año es bisiesto o no) del día seleccionado por el usuario. Para obtener la fecha juliana, se realizan los siguientes cálculos: obtenemos la fecha del día seleccionado y la guardamos en un *array* asociativo, del que, posteriormente, podremos obtener el día juliano correspondiente. Es necesario sumar 1 al valor almacenado en el *array* porque el campo `yday` guarda valores entre el 0 y 364. El código que realiza estas operaciones y muestra el resultado por pantalla es el siguiente:

```
$fecha=getdate(mktime(0,0,0,$mes,$dia,$anio));
echo "<P STYLE='color:red;'>Día juliano nº <B>";
echo $fecha['yday']+1,"</B></P><BR>";
```

Finalmente, el juego de enlaces que aparecen al final de la página generada se obtienen del siguiente modo: para obtener un acceso al año anterior y posterior, simplemente se generan dos enlaces cuyas referencias hacen una llamada al *script* pasándole el día y el mes de la fecha con que se está trabajando y el valor del nuevo año deseado (sumando o restando 1 al año actual según corresponda).

```
$url = "fechas1.php?dia=$dia&mes=$mes&anio=".($anio-1);
echo "<PRE>< <A CLASS='opc' HREF='$url'>año-</A> | ";
$url = "fechas1.php?dia=$dia&mes=$mes&anio=".($anio+1);
echo "<A CLASS='opc' HREF='$url'>año+</A> >";
```

Para obtener un enlace al mes anterior al actual, la generación del URL de llamada al *script* se complica, puesto que hay que tener en cuenta que, si actualmente estamos en enero (mes 1), el mes anterior será diciembre (mes 12), pero del año anterior al actualmente visualizado. La situación contraria ocurre en el caso de querer obtener el mes posterior y encontrarnos en diciembre (mes 12), en cuyo caso deberíamos saltar a enero (mes 1) del año siguiente. El código que solventa esta situación es el siguiente:

```
//mes anterior
$url = "fechas1.php?dia=$dia";
if ($mes==1)
   $url .= "&anio=".($anio-1)."&mes=12";
else
   $url .= "&anio=$anio&mes=".($mes-1);
echo "<A CLASS='opc' HREF='$url'>mes-</A> | ";
// mes posterior
$url = "fechas1.php?dia=$dia";
if ($mes==12)
   $url .= "&anio=".($anio+1)."&mes=1";
else
   $url .= "&anio=$anio&mes=".($mes+1);
echo "<A CLASS='opc' HREF='$url'>mes+</A> | ";
```

Para obtener el calendario del día actual, simplemente hay que llamar al *script* sin pasarle ningún parámetro. De esta forma, obtendrá el calendario de la fecha actual del sistema:

```
      echo "<A CLASS='opc' HREF='fechas1.php?'>hoy</A> | ";
```

Para aclarar la estructura de la página HTML generada por el *script*, se muestra el siguiente listado obtenido al solicitar la fecha "30/9/2002":

```
<HTML>
 <HEAD>
   <TITLE>Funciones de Fecha y Hora</TITLE>
   <style>
    body {font: 12px Verdana;}
    table {font: 12px Verdana;color:orange;text-align:right;}
    tr.cabecera {background-color:#808080;color:#F8F8F8;font-weight:bold;}
    tr.semana {background-color:#FFFBAD;color:#808080;font-weight:bold;}
    a {text-decoration:none;color:orange;}
    a.marcado {background-color:green;}
    a.festivo {color:#B00000;}
    a.opc {color:gray;font-weight:bold;}
    p.error {font:14px;color:red;font-weight:bold;}
```

```
  </style>
 </HEAD>
  <BODY>
   <CENTER>
     <H3>Funciones de Fecha y Hora</H3>
     <TABLE BORDER='0' CELLPADDING='2' CELLSPACING='0' WIDTH='50%'>
       <TR CLASS='cabecera'>
          <TD COLSPAN='7'>Septiembre 2002</TD>
       </TR>
       <TR CLASS='semana'>

<TD>L</TD><TD>M</TD><TD>X</TD><TD>J</TD><TD>V</TD><TD>S</TD><TD>D</TD>
       </TR>
       <TR>
          <TD> </TD><TD> </TD><TD> </TD>
          <TD> </TD><TD> </TD><TD> </TD>
          <TD><A HREF='fechas1.php?dia=1&mes=9&anio=2002'
                 class='festivo'>1</A></TD>
       </TR>
       <TR>
          <TD><A HREF='fechas1.php?dia=2&mes=9&anio=2002'>2</A></TD>
          <TD><A HREF='fechas1.php?dia=3&mes=9&anio=2002'>3</A></TD>
          <TD><A HREF='fechas1.php?dia=4&mes=9&anio=2002'>4</A></TD>
          <TD><A HREF='fechas1.php?dia=5&mes=9&anio=2002'>5</A></TD>
          <TD><A HREF='fechas1.php?dia=6&mes=9&anio=2002'>6</A></TD>
          <TD><A HREF='fechas1.php?dia=7&mes=9&anio=2002'
                 class='festivo'>7</A></TD>
          <TD><A HREF='fechas1.php?dia=8&mes=9&anio=2002'
                 class='festivo'>8</A></TD>
       </TR>
       <TR>
          <TD><A HREF='fechas1.php?dia=9&mes=9&anio=2002'>9</A></TD>
          <TD><A HREF='fechas1.php?dia=10&mes=9&anio=2002'>10</A></TD>
          <TD><A HREF='fechas1.php?dia=11&mes=9&anio=2002'>11</A></TD>
          <TD><A HREF='fechas1.php?dia=12&mes=9&anio=2002'>12</A></TD>
          <TD><A HREF='fechas1.php?dia=13&mes=9&anio=2002'>13</A></TD>
          <TD><A HREF='fechas1.php?dia=14&mes=9&anio=2002'
                 class='festivo'>14</A></TD>
          <TD><A HREF='fechas1.php?dia=15&mes=9&anio=2002'
                 class='festivo'>15</A></TD>
       </TR>
       <TR>
          <TD><A HREF='fechas1.php?dia=16&mes=9&anio=2002'>16</A></TD>
          <TD><A HREF='fechas1.php?dia=17&mes=9&anio=2002'>17</A></TD>
          <TD><A HREF='fechas1.php?dia=18&mes=9&anio=2002'>18</A></TD>
          <TD><A HREF='fechas1.php?dia=19&mes=9&anio=2002'>19</A></TD>
          <TD><A HREF='fechas1.php?dia=20&mes=9&anio=2002'>20</A></TD>
          <TD><A HREF='fechas1.php?dia=21&mes=9&anio=2002'
                 class='festivo'>21</A></TD>
          <TD><A HREF='fechas1.php?dia=22&mes=9&anio=2002'
                 class='festivo'>22</A></TD>
       </TR>
       <TR>
          <TD><A HREF='fechas1.php?dia=23&mes=9&anio=2002'>23</A></TD>
          <TD><A HREF='fechas1.php?dia=24&mes=9&anio=2002'>24</A></TD>
          <TD><A HREF='fechas1.php?dia=25&mes=9&anio=2002'>25</A></TD>
          <TD><A HREF='fechas1.php?dia=26&mes=9&anio=2002'>26</A></TD>
          <TD><A HREF='fechas1.php?dia=27&mes=9&anio=2002'>27</A></TD>
          <TD><A HREF='fechas1.php?dia=28&mes=9&anio=2002'
                 class='festivo'>28</A></TD>
          <TD><A HREF='fechas1.php?dia=29&mes=9&anio=2002'
                 class='festivo'>29</A></TD>
```

```
        </TR>
        <TR>
           <TD><A HREF='fechas1.php?dia=30&mes=9&anio=2002'
               class='marcado'>30</A></TD>
           <TD> </TD><TD> </TD><TD> </TD>
           <TD> </TD><TD> </TD><TD> </TD>
        </TR>
     </TABLE>
     <BR>
     <P STYLE='color:red;'>Día juliano nº <B>273</B></P><BR>
     <PRE>
       < <A CLASS='opc' HREF='fechas1.php?dia=30&mes=9&anio=2001'>año-</A> |
       <A CLASS='opc' HREF='fechas1.php?dia=30&anio=2002&mes=7'>mes-</A> |
       <A CLASS='opc' HREF='fechas1.php?'>hoy</A> |
       <A CLASS='opc' HREF='fechas1.php?dia=30&anio=2002&mes=9'>mes+</A> |
       <A CLASS='opc' HREF='fechas1.php?dia=30&mes=8&anio=2003'>año+</A> >
       <BR>< <A CLASS='opc' HREF='fechas1.html'>nueva fecha</A> >
     </PRE>
  </CENTER>
 </BODY>
</HTML>
```

CAPÍTULO 9

FORMULARIOS, *COOKIES* Y SESIONES

En este capítulo vamos a ver cómo PHP procesa o gestiona la información que el cliente (el navegador) envía, a través del uso de formularios, al servidor y cómo éste genera una respuesta adecuada a la petición solicitada.

9.1 EL PROTOCOLO HTTP

Aunque sabemos que PHP puede funcionar desde la línea de comandos, su uso principal está relacionado con los sitios Web; de hecho, PHP puede ser definido como un servicio complementario a los proporcionados por los servidores Web. Estos servidores fundamentan su funcionamiento en el uso del protocolo HTTP, del *Protocolo de Transferencia de Hipertextos*. Por ello, para entender el funcionamiento de las técnicas que se proponen en este capítulo, se hace necesario un conocimiento previo del protocolo HTTP[1].

Éste es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los clientes los servidores web a través de operaciones simples de tipo solicitud/respuesta. Básicamente controla el modo en el que los clientes web solicitan recursos de los servidores web y el modo en que éstos les envían dichos recursos de vuelta. Todos los recursos proporcionados por un servidor web (documentos HTML, gráficos, videos, ficheros de sonido...) están asociados a un URL o *Localizador Uniforme de Recursos*.

[1] La especificación completa del protocolo HTTP/1.1 está recogida en el RFC 2616, disponible en la dirección http://www.ietf.org/rfc/rfc2616.txt.

9.1.1 Estructura de los mensajes HTTP

Las peticiones y respuestas se envían como mensajes de texto que se componen de dos partes, una cabecera y un cuerpo (ambos elementos se separan en el mensaje mediante una línea en blanco). Sólo existen dos tipos de mensajes, uno para realizar peticiones y otro para responderlas. Las peticiones siempre cuentan con cabecera y, en algunas ocasiones, con cuerpo; sin embargo, las respuestas en la mayoría de las ocasiones cuentan con ambos componentes. La estructura de estos mensajes se puede resumir en la siguiente tabla:

Tipo de mensaje	Componentes
`petición`	Comando HTTP
	Cabeceras de la petición
	Línea en blanco (separador)
	Información adicional
`respuesta`	Resultado de la petición
	Cabeceras de la respuesta
	Línea en blanco (separador)
	Información adicional

9.1.1.1 Comandos HTTP

La primera línea de un mensaje de petición, el comando HTTP, tiene la siguiente estructura:

```
método_HTTP URL_recurso versión_del_protocolo
```

Es decir, proporciona el comando HTTP utilizado (normalmente se denomina *método*), la URL del recurso del servidor al que deseamos acceder y la versión del HTTP utilizada. La siguiente tabla muestra los *métodos* HTTP disponibles:

método	utilización
`Get`	Obtención de recursos del servidor
`Post`	Envío de información al servidor
`Head`	Obtención de información sobre un recurso específico del servidor
`Put`	Actualización de un recurso en el servidor
`Delete`	Eliminación de un recurso en el servidor
`Trace`	Obtención de información de diagnóstico sobre la comunicación establecida con el servidor
`Connect`	Reservado para la comunicación a través de un *proxy*
`Options`	Obtención de información sobre las opciones de comunicación de un recurso
`Link`	Creación de un enlace entre recursos
`Unlink`	Eliminación de un enlace entre recursos

A pesar de contar con todos los métodos enumerados en la tabla anterior, los más utilizados son *get*, *post* y *head*, además de ser los más estándares. Algunos de los métodos restantes están en desuso o tienen una utilización experimental.

NOTA: En la sección de formularios veremos a fondo el funcionamiento de los métodos *get* y *post*.

9.1.1.2 Cabeceras de petición y respuesta

Las cabeceras de petición proporcionan información sobre el cliente Web utilizado, normalmente un navegador, los tipos de contenidos que es capaz de aceptar, la longitud de los contenidos, etc. Por su parte, las cabeceras de respuesta proporcionan información referente al *software* del servidor y a la naturaleza de la información incluida en el cuerpo de la respuesta. El formato general de una cabecera es el siguiente:

```
variable_de_cabecera: valor
```

Existen cabeceras comunes a peticiones y resultados y cabeceras específicas. La siguiente tabla muestra las cabeceras más habituales:

Cabeceras comunes	
cabecera	**utilización**
`Content-Type`	Tipo MIME de la información contenida en el mensaje
`Content-Length`	Tamaño en *bytes* de la información contenida en el mensaje
`Content-Encoding`	Formato de codificación de los datos enviados en el mensaje
`Transfer-Encoding`	Tipo de codificación empleada en el cuerpo del mensaje
`Date`	Fecha y hora local de la operación incluyendo zona horaria
`Pragma`	Opciones de comportamiento del protocolo http
`Cache-Control`	Directivas de control de caché
`Connection`	Información del estado de la conexión después de un envío
`Via`	Información de pasarelas y servidores *proxy*
`Warning`	Información adicional sobre el estado de un mensaje

Cabeceras de petición	
cabecera	**Utilización**
`Accept`	Lista de tipos MIME aceptados por el cliente
`Accept-Charset`	Lista de juegos de caracteres aceptados por el cliente
`Accept-Encoding`	Tipos de codificación soportados por el cliente
`Accept-Language`	Lista de lenguajes aceptados por el cliente
`Authorization`	Información de autenticación
`Expect`	Estado esperado
`From`	Dirección de correo electrónico del usuario
`Host`	Nombre y puerto del *host* solicitado
`If-Match`	Cabecera utilizada para solicitudes condicionales
`If_Modified-Since`	Cabecera utilizada para solicitudes condicionales
`If-None-Match`	Cabecera utilizada para solicitudes condicionales
`If-Range`	Cabecera utilizada para solicitudes condicionales
`If-Unmodified-Since`	Cabecera utilizada para solicitudes condicionales
`Max-Forwars`	Límite de saltos para el método TRACE
`Proxy-Authorization`	Información de autentificación frente a un servidor *proxy*
`Range`	Rango de *bytes* solicitados
`Referer`	URL del recurso desde el que se ha realizado la petición
`User-Agent`	Información sobre el navegador del que ha partido la petición

Cabeceras de respuesta	
cabecera	**Utilización**
`Allow`	Comandos opcionales que también se pueden aplicar para obtener el recurso
`Accept-Ranges`	Indica el rango de *bytes* manejados por el servidor
`Age`	Estimación de segundos que han pasado desde el instante en que se generó la respuesta
`Expires`	Fecha en la que el recurso deja de estar disponible
`Last-Modified`	Fecha local de última modificación del objeto
`Location`	Dirección del recurso al que se intenta acceder. Permite redirigir solicitudes
`Server`	Información del tipo y versión de servidor HTTP utilizado
`Proxy-Authenticate`	Información de autentificación frente a servidores *proxy*
`Retry-After`	Instante en el que el recurso solicitado estará disponible
`Server`	Información sobre el *software* utilizado en el servidor
`Vary`	Información asociada al sistema de caché
`WWW-Authenticate`	Información sobre el acceso a un recurso protegido o de acceso restringido

9.1.1.3 Resultado de la petición

Ante cada petición, el servidor HTTP genera un mensaje de respuesta en el que inserta una primera línea denominada *línea de estado* en la que se informa sobre el resultado de la operación. El formato de esta primera línea del mensaje es la siguiente:

```
versión_del_protocolo_HTTP código_de_estado descripción
```

Es decir, la versión empleada del protocolo HTTP, un código numérico de tres cifras que indica el estado de la solicitud y un texto que contiene una breve descripción del código de estado devuelto. Existen cinco categorías de mensajes de estado, codificadas en función del primer dígito del código; éstas son:

categoría	**utilización**
`1xx`	Mensajes informativos
`2xx`	Mensajes asociados a operaciones finalizadas de forma correcta
`3xx`	Mensajes de redirección
`4xx`	Mensajes asociados a errores de cliente
`5xx`	Mensajes asociados a errores de servidor

La siguiente tabla muestra algunos de los mensajes más habituales:

mensaje	descripción	comentario
200	OK	Operación realizada correctamente
201	Created	Operación realizada correctamente; además se ha creado un nuevo recurso que se encuentra disponible
202	Accepted	Operación realizada correctamente; además se ha creado un nuevo recurso que no se encuentra disponible
204	No Content	Operación realizada correctamente; no se ha generado ningún contenido de interés
301	Moved Permanently	El recurso al que se pretendía acceder ha sido movido a otra ubicación de forma permanente
302	Moved Temporarily	El recurso al que se pretendía acceder ha sido movido de forma temporal a otra ubicación
400	Bad Request	Petición no entendida por el servidor
401	Unauthorized	La petición requiere de una autentificación
403	Forbidden	Esta prohibido el acceso al recurso solicitado
404	Not Found	El recurso solicitado no se encuentra en el servidor
500	Internal Server Error	Se ha producido un error interno del servidor
501	Not Implemented	El servidor no tiene implementado alguno de los requerimientos solicitados por el cliente
503	Service Unavailable	El servidor no se encuentra disponible

9.1.2 Funciones PHP relacionadas

PHP proporciona las siguientes funciones para obtener la información contenida en las cabeceras HTTP: `apache_request_headers()` (denominada `getallheaders()` en versiones anteriores a la 4.3.0) para los mensajes de petición y `apache_response_headers()` para los mensajes de respuesta. Ambas funciones devuelven un *array* asociativo que contiene todas las cabeceras HTTP de la petición/respuesta de la operación actual.

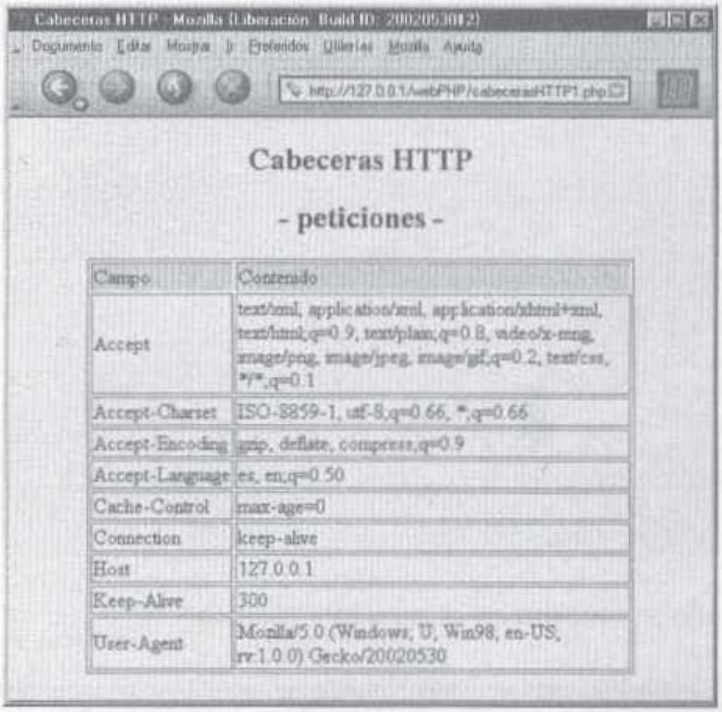
> **NOTA:** Estas funciones sólo están disponibles cuando se está ejecutando PHP como módulo de Apache.

El siguiente ejemplo nos muestra cómo recuperar las cabeceras de petición:

```
<HTML>
 <HEAD>
   <TITLE>Cabeceras HTTP</TITLE>
 </HEAD>
 <BODY>
```

```
    <CENTER>
      <H2>Cabeceras HTTP</H2>
      <H2>- peticiones -</H2>
      <?php
         $cabecera=apache_request_headers();
         //$cabecera=getallheaders();
         echo "<TABLE BORDER='1' WIDTH='80%'><TR BGCOLOR='yellow'>";
         echo "<TD>Campo</TD><TD>Contenido</TD></TR>";
         foreach($cabecera as $campo => $contenido)
             echo "<TR><TD>$campo</TD><TD>$contenido</TD></TR>";
         echo "</TABLE>";
      ?>
    </CENTER>
  </BODY>
</HTML>
```

El resultado se visualiza en la siguiente página:



PHP también nos proporciona funciones para la gestión de las cabeceras; éstas son: `header()`, para enviar cabeceras http, y `headers_sent()`, para confirmar que las cabeceras han sido enviadas correctamente. Su sintaxis es la siguiente:

```
header (cadena [, reemplazar[, http_reponse_code]])

headers_sent()
```

El primer parámetro opcional de la función `header()`, de carácter booleano, nos permite indicar si la cabecera debe reemplazar a una cabecera similar previamente fijada o, por el contrario, debe añadir una nueva cabecera del mismo

tipo (el valor por defecto es *true*). El segundo parámetro opcional, un valor entero, fuerza el código de respuesta que debe enviarse al procesar la solicitud.

El siguiente ejemplo nos muestra cómo utilizar la función `header()` para indicarle al navegador que la información que se le envía tiene que interpretarla como un formato gráfico:

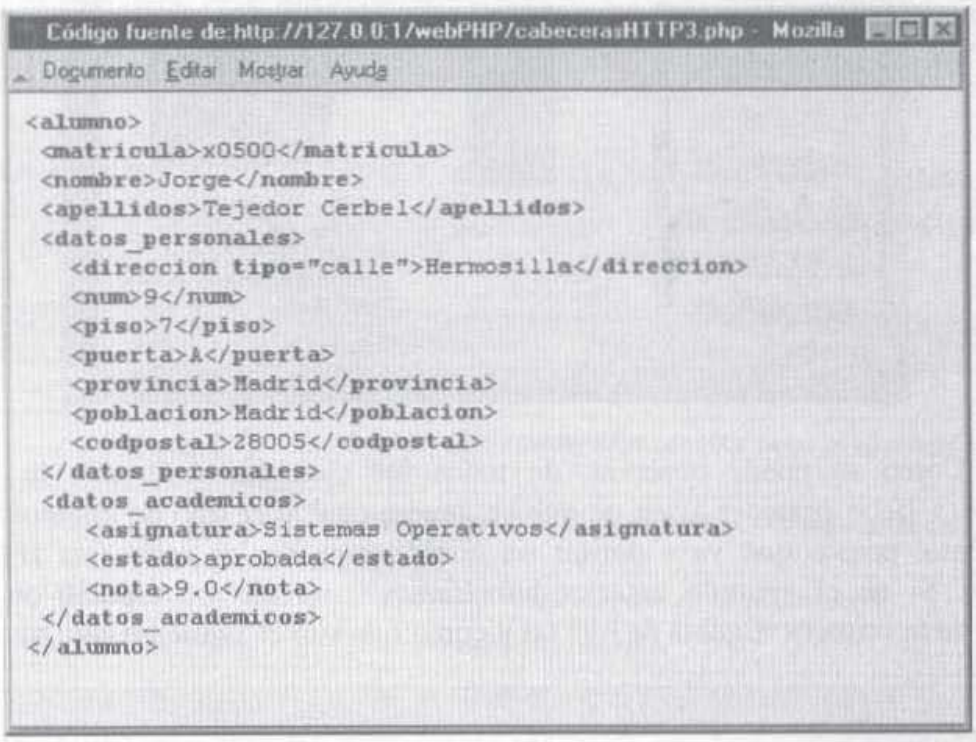
```
<?php
  header("Content-Type: image/gif");
  header("Content-Transfer-Encoding: base64");
?>
GIF89a_W f i 2 z B N W N + v v W 2 H 7 1 4 o T b S X y a B 1 s C 5 E + U 7 g
Q U N N e n k w g w b h y V C G j T 3 c E I Y u G q x q x w Y l B 1 m H Y C p
H X j j h S z y y A s 8 h q e J m g 2 G k o q k s c y c c W r d N c F P b t k
7 I 1 A N G B D b a 6 0 5 I K T G R U 5 Q E J J F T k y Q w g 0 n v Z b R y p
E m 5 W 3 L Z X y Q l h V t 7 N 3 V W G M G 8 s g i 1 z y q Z A F I Z 1 Q D G
y F w A A I p H u D B A B s M A N r b 8 J q F 0 g F x G + X A A H e q e 2 S +
R h / / h E F w E i A 8 E N J P N 8 R c 3 3 4 f P h A H F z e O U s d e f r x
Q d 5 I X N C b X m I s H A b u c d + 7 5 5 6 C H L v r o p J f O O c F h g 0
3 R g c i K W C x B r 5 s u O 0 S m 6 i 0 p o 2 h r l o B l A y N U 0 7 H / T
C q X s J s S Z d O v y S o F b F J W 8 X T g A S 9 V G J e m y m 9 l A P O z
Z 2 / Q A B o J C k A B B U C F W g U E m D R S S i z t f h j r w B o / q 0 y
S 4 i Q h / B V q l C m f z + 5 k P 7 A y c R S 7 Q K / z y V Z 8 o r 0 C k
R G t 6 q L k / y T m s 9 4 Q C k e M M z A a k J B A M J P V 9 e z n q S c t
5 M 5 D e t T S c k Q 7 P A T E u N d z 4 A o D F s B g G I A j 9 h M K b 6 R
A I y M 4 6 + 9 E A B c H l i J A i y D G N U l h D 0 M e V R X / l i Y Q h S
K j T 0 v 6 V Z v X g j D f M n Q L n 3 p V t w 0 4 8 A c N q B c e o P M B X
J S x C 5 W K m y 5 6 5 F f 8 j L D t t x r L j d 6 I h n b h r S 3 B B c m S
k M 6 i h S l 8 I l R G l 4 v A D i C n I B C / h R c A 2 B A A c Y 5 7 i U c
A B x G O i A I j v g N Y V Q b i W N J A g D J k n J S l q y k p G 0 y G T c
9 i 6 X n a Z F M G I i f m p k o / 3 g S C 0 x o 0 A B 5 F Y Y B L D N X B L
J z W 8 A m R C k U e A C u M y l L g m X E O f 4 Z Y 8 D k c 4 g h 0 1 M D m
i g O i n Z H E G s 5 g H I O T J M l g v Z A q Z J z W p a k z F 3 u 0 w c V
x a a t K j m N D l C p V 0 K A B c Z n V E m w O E N b 9 o m n t j A 5 m c I
I R J w B k K 0 3 a D O l r v M Z 5 Q 0 R v O l w 0 p S 5 p Z C A Y O i c i
s G f R q W x P P J S e Z S f W M z U Q r
```

El resultado se muestra en la siguiente imagen:

El siguiente ejemplo nos permite indicar al navegador que el contenido de la información que se le envía es de tipo XML:

```
<?php
  header("Content-type: application/xml");
?>
<alumno>
 <matricula>x0500</matricula>
 <nombre>Jorge</nombre>
 <apellidos>Tejedor Cerbel</apellidos>
 <datos_personales>
   <direccion tipo="calle">Hermosilla</direccion>
   <num>9</num>
   <piso>7</piso>
   <puerta>A</puerta>
   <provincia>Madrid</provincia>
   <poblacion>Madrid</poblacion>
   <codpostal>28005</codpostal>
 </datos_personales>
 <datos_academicos>
    <asignatura>Sistemas Operativos</asignatura>
    <estado>aprobada</estado>
    <nota>9.0</nota>
 </datos_academicos>
</alumno>
```

El resultado se visualiza en la siguiente imagen:



Uno de los usos más habituales de la función `header()` consiste en enviar la cabecera *Location* al navegador. De esta forma podemos redireccionar las peticiones enviadas a nuestro servidor. El siguiente ejemplo muestra su funcionamiento:

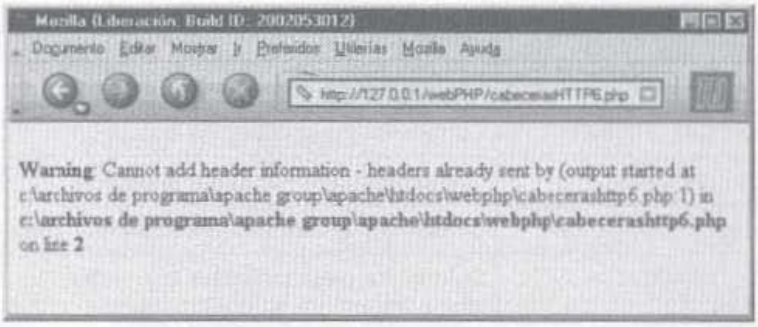
```
<?php
  header("Location: http://www.ra-ma.es");
?>
```

El resultado de ejecutar el *script* cargará en el navegador la página de la editorial Ra-Ma, tal y como muestra la imagen siguiente.

Como se puede observar, de todos los ejemplos anteriores la función `header()` debe aparecer antes de que se genere cualquier tipo de contenido en el documento, puesto que va a definir un componente de las cabeceras HTTP del mensaje. Si en el ejemplo anterior hubiéramos insertado un espacio en blanco delante de la primera etiqueta de *php* tal y como muestra el siguiente código:

```
  <?php
  header("Location: http://www.ra-ma.es");
?>
```

El resultado de ejecutar el *script* hubiera sido el siguiente:



Es decir, se genera un mensaje de error en el que se informa al cliente de que no se ha podido añadir información a las cabeceras puesto que éstas ya han sido enviadas. La inserción del espacio en blanco marca el inicio del cuerpo del mensaje, de modo que la llamada a la función `header()` se ejecuta fuera de la cabecera del mensaje.

9.1.3 Variables PHP relacionadas

Las variables PHP directamente relacionadas con la información manejada por el protocolo HTTP son `$_SERVER`, `$_GET`, `$_POST` y `$_REQUEST`. Estas tres últimas serán vistas en profundidad en el apartado dedicado a formularios.

La variable global `$_SERVER` es un *array* asociativo que contiene, entre otras, toda la información de las cabeceras tanto de petición como de respuesta. La siguiente tabla nos muestra los contenidos de esta variable:

campo	Descripción
`PHP_SELF`	Nombre del *script* PHP que se está ejecutando
`argv`	*Array* con los parámetros pasados como argumentos al *script*
`argc`	Número de parámetros pasados como argumentos al *script*
`GATEWAY_INTERFACE`	Especificación del CGI que utiliza el servidor
`SERVER_NAME`	Nombre del equipo servidor
`SERVER_SOFTWARE`	Identificación del *software* que está utilizando el servidor
`SERVER_PROTOCOL`	Nombre y versión del protocolo utilizado por el servidor
`REQUEST_METHOD`	Método utilizado por la solicitud para acceder a los recursos
`QUERY_STRING`	Cadena de petición
`DOCUMENT_ROOT`	Directorio raíz del documento en el que se ejecuta el *script*

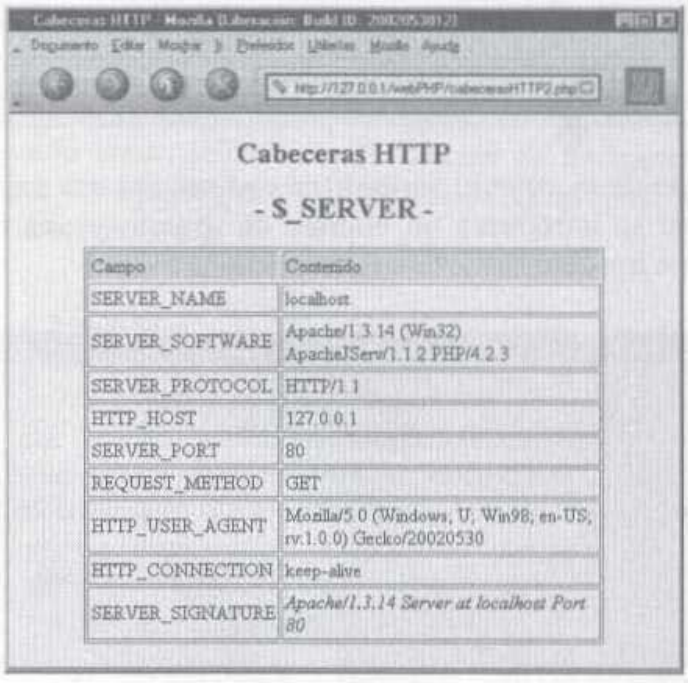
campo	Descripción
HTTP_ACCEPT	Contenidos de la cabecera *accept*
HTTP_ACCEPT_CHARSET	Contenidos de la cabecera *accept-charset*
HTTP_ENCODING	Contenidos de la cabecera *accept-encoding*
HTTP_ACCEPT_LANGUAGE	Contenidos de la cabecera *accept-language*
HTTP_CONNECTION	Contenidos de la cabecera *connection*
HTTP_HOST	Contenidos de la cabecera *host*
HTTP_REFERER	Dirección de la página de la que se procede
HTTP_USER_AGENT	Contenidos de la cabecera *user-agent*
REMOTE_ADDR	Dirección IP del usuario
REMOTE_PORT	Puerto utilizado por la máquina del usuario
SCRIPT_FILENAME	Camino de acceso al *script* que se está ejecutando
SERVER_ADMIN	Valor de la directiva SERVER_ADMIN de Apache
SERVER_PORT	Puerto utilizado por el equipo servidor
SERVER_SIGNATURE	Cadena que identifica al servidor
PATH_TRANSLATED	Camino de acceso al *script* teniendo en cuenta el sistema de ficheros
SCRIPT_NAME	Camino de acceso al *script* actual
REQUEST_URI	URI utilizada para acceder al recurso
PHP_AUTH_USER	Variable utilizada para realizar autentificación de usuarios con HTTP. Almacena el nombre de usuario. Sólo funciona cuando PHP se ejecuta como módulo de Apache.
PHP_AUTH_PW	Variable utilizada para realizar autentificación de usuarios con HTTP. Almacena la clave del usuario. Sólo funciona cuando PHP se ejecuta como módulo de Apache.
PHP_AUTH_TYPE	Variable utilizada para realizar autentificación de usuarios con HTTP. Almacena el tipo de autentificación. Sólo funciona cuando PHP se ejecuta como módulo de Apache.

El siguiente ejemplo muestra la utilización de la variable $_SERVER para recuperar información sobre las cabeceras HTTP:

```
<HTML>
 <HEAD>
   <TITLE>Cabeceras HTTP</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Cabeceras HTTP</H2>
     <H2>- $_SERVER -</H2>
     <?php
```

```
        echo "<TABLE BORDER='1' WIDTH='80%' CELLPADDING='2'>";
        echo "<TR BGCOLOR='yellow'>";
        echo "<TD>Campo</TD><TD>Contenido</TD></TR>";
        echo "<TR><TD>SERVER_NAME</TD>";
        echo "<TD>$_SERVER[SERVER_NAME]</TD></TR>";
        echo "<TR><TD>SERVER_SOFTWARE</TD>";
        echo "<TD>$_SERVER[SERVER_SOFTWARE]</TD></TR>";
        echo "<TR><TD>SERVER_PROTOCOL</TD>";
        echo "<TD>$_SERVER[SERVER_PROTOCOL]</TD></TR>";
        echo "<TR><TD>HTTP_HOST</TD>";
        echo "<TD>$_SERVER[HTTP_HOST]</TD></TR>";
        echo "<TR><TD>SERVER_PORT</TD>";
        echo "<TD>$_SERVER[SERVER_PORT]</TD></TR>";
        echo "<TR><TD>REQUEST_METHOD</TD>";
        echo "<TD>$_SERVER[REQUEST_METHOD]</TD></TR>";
        echo "<TR><TD>HTTP_USER_AGENT</TD>";
        echo "<TD>$_SERVER[HTTP_USER_AGENT]</TD></TR>";
        echo "<TR><TD>HTTP_CONNECTION</TD>";
        echo "<TD>$_SERVER[HTTP_CONNECTION]</TD></TR>";
        echo "<TR><TD>SERVER_SIGNATURE</TD>";
        echo "<TD>$_SERVER[SERVER_SIGNATURE]</TD></TR>";
        echo "</TABLE>";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



Cabeceras HTTP

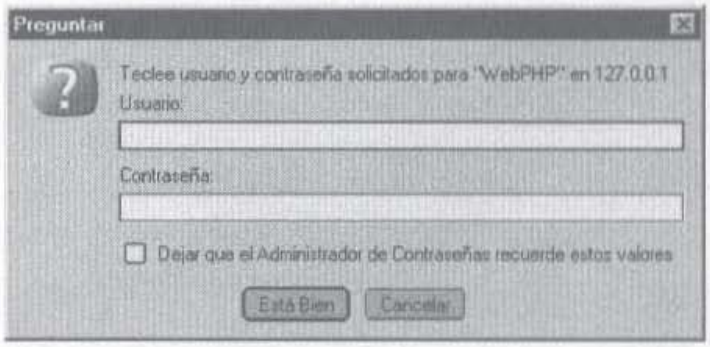
- $_SERVER -

Campo	Contenido
SERVER_NAME	localhost
SERVER_SOFTWARE	Apache/1.3.14 (Win32) ApacheJServ/1.1.2 PHP/4.2.3
SERVER_PROTOCOL	HTTP/1.1
HTTP_HOST	127.0.0.1
SERVER_PORT	80
REQUEST_METHOD	GET
HTTP_USER_AGENT	Mozilla/5.0 (Windows; U; Win98; en-US; rv:1.0.0) Gecko/20020530
HTTP_CONNECTION	keep-alive
SERVER_SIGNATURE	*Apache/1.3.14 Server at localhost Port 80*

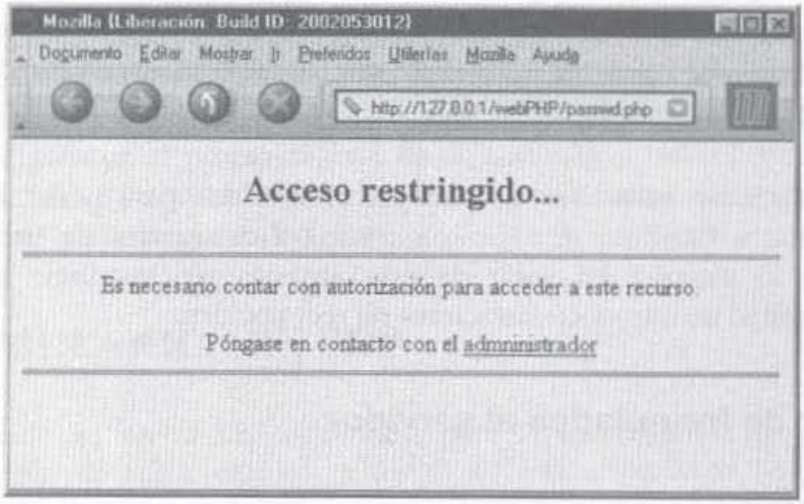
Un uso habitual de las cabeceras HTTP en conjunción con los campos `PHP_AUTH_USER` y `PHP_AUTH_PW` de la variable `$_SERVER` nos permite realizar una autentificación básica de usuarios para el acceso a recursos restringidos. El siguiente ejemplo muestra su funcionamiento:

```
<?php
  if (!isset($_SERVER[PHP_AUTH_USER])) {
    header('WWW-Authenticate: Basic realm="WebPHP"');
    header('HTTP/1.0 401 Unauthorized');
    echo '<CENTER><H2>Acceso restringido...</H2>';
    echo '<HR>Es necesario contar con autorización ';
    echo 'para acceder a este recurso.<BR>';
    echo '<BR>Póngase en contacto con el ';
    echo '<A HREF="mailto:'.$_SERVER[SERVER_ADMIN] ;
    echo '">admninistrador</A><BR><HR>';
    echo '</CENTER>';
    exit;
  } elseif
(($_SERVER[PHP_AUTH_USER]!='libroPHP')||($_SERVER[PHP_AUTH_PW]!='acceso')) {
    header('WWW-Authenticate: Basic realm="WebPHP"');
    header('HTTP/1.0 401 Unauthorized');
    echo '<CENTER><H2>Acceso restringido...</H2>';
    echo '<HR>Es necesario contar con autorización ';
    echo 'para acceder a este recurso.<BR>';
    echo '<BR>Póngase en contacto con el ';
    echo '<A HREF="mailto:'.$_SERVER[SERVER_ADMIN] ;
    echo '">admninistrador</A><BR><HR>';
    echo '</CENTER>';
    exit;
  } else // Acceso concedido...
    echo '<CENTER><H2>Acceso concedido...</H2></CENTER>';
?>
```

El resultado de ejecutar el *script* hace que se abra una caja de diálogo en la que se solicita que se introduzca un nombre de usuario y una clave, tal y como muestra la siguiente imagen:



Si los datos proporcionados son los esperados, se accedería al recurso solicitado; en cualquier otro caso, la respuesta sería la siguiente:



9.2 FORMULARIOS EN HTML

Esta primera sección pretende ser un resumen de los conceptos más importantes asociados a la definición de formularios en HTML. Como el lector conocerá, un formulario HTML es una sección de un documento que contiene texto normal, etiquetas HTML y elementos especiales llamados *controles* (casillas de verificación o *checkboxes*, radiobotones o *radiobuttons*, menús desplegables, etc.) y rótulos (*labels*) asociados a estos controles.

Los usuarios normalmente *completan* un formulario modificando sus controles (introduciendo texto, seleccionando objetos de un menú, etc.), antes de enviar el formulario a un agente para que lo procese (por ejemplo, a un servidor Web, a un servidor de correo, etc.).

9.2.1 El elemento FORM

Todos los controles presentes en un formulario, para que sean efectivos, deben aparecer incluidos dentro de un elemento `form` de HTML. La etiqueta `FORM` actúa, por tanto, como contenedor de controles. Pero, además, especifica entre otros:

- El programa que manejará los datos del formulario una vez haya sido completado y enviado (atributo `action`). El programa receptor debe ser capaz de interpretar las parejas nombre/valor para poder hacer uso de ellas.

- La forma en la que se enviarán los datos del usuario al servidor (atributo `method`).

- El tipo MIME usado para enviar los datos del formulario (atributo `enctype`). El valor por defecto de este atributo es `application/x-www-form-urlencoded`.

- Juegos de caracteres que acepta el servidor para poder manejar este formulario (atributo `accept-charset`). Los agentes de usuario pueden avisar al usuario del valor de este atributo y/o restringir al usuario la posibilidad de introducir caracteres no reconocidos.

9.2.2 Envío de formularios al servidor

El atributo `method` del elemento `form` especifica el método HTTP usado para enviar los datos del formulario al agente procesador. Este atributo puede tener dos valores:

- `get`: El conjunto de datos del formulario se agrega (con el carácter "`?`" como separador) al URI especificado en el atributo `action` (el programa que tratará los datos) y este nuevo URI se envía al agente procesador.

- `post`: Los datos del formulario se incluyen en el cuerpo del mensaje que se envía al agente procesador.

El conjunto de datos del formulario que se envía al agente servidor es una secuencia de parejas `nombre_de_control/valor` construida a partir de los elementos del formulario. Cada uno de los controles tiene asociado un nombre que viene dado por su atributo `name`. De igual forma, cada control tiene tanto un valor inicial, como un valor actual, que son ambos cadenas de caracteres. En general (excepto en el caso del elemento `textarea`), el valor inicial de un control puede especificarse con el atributo `value` del elemento de control. El valor actual del control se iguala en un primer momento al valor inicial y, a partir de ese momento, el valor actual del control puede ser modificado a través de la interacción con el usuario y/o mediante *scripts* que se ejecuten en el cliente. El valor inicial de un control no cambia. Así, cuando se reinicializa el formulario, el valor actual de cada control se iguala a su valor inicial. Si el elemento de control no tiene un valor inicial, se le asigna el valor nulo.

> **NOTA:** El método `get` restringe los valores del conjunto de datos del formulario a caracteres ASCII. Sólo el método *post* (con `enctype="multipart/form-data"`) cubre el conjunto de caracteres [ISO10646] completo.

El método `get` debería usarse cuando el formulario es idempotente (es decir, cuando no tiene efectos secundarios). Muchas búsquedas en bases de datos no tienen efectos secundarios visibles y constituyen aplicaciones ideales del método `get`. Si el servicio asociado con el procesamiento de un formulario causa efectos secundarios (por ejemplo, si el formulario modifica una base de datos o la suscripción a un servicio), debería usarse el método `post`.

9.3 FORMULARIOS EN PHP

Cuando se envía un formulario para su procesamiento, se produce el emparejamiento de sus controles, a través de sus respectivos nombres, con los valores actuales que tienen. Estas parejas `variable-valor` configuran el conjunto de datos del formulario que se envían al agente servidor. Como hemos visto, dependiendo del método HTTP usado (`get` o `post`), unas veces la información se enviará formando parte de la cadena de consulta (`query string`) que configura la solicitud y, otras veces, formando parte del cuerpo del mensaje.

PHP, a través de un conjunto de variables globales, es capaz de recuperar el conjunto de datos del formulario que han sido enviados desde el cliente (esto es, el navegador) para, después, poder trabajar con ellos. Las tres variables principales para realizar esta operación son:

Variable	Contenido
`_POST`	*Array* que contiene las variables pasadas a través del método `POST`. Su uso es análogo al array `HTTP_POST_VARS` de versiones anteriores a la 4.2.0 de PHP (aún disponible pero obsoleto).
`_GET`	*Array* que contiene las variables pasadas a través del método `GET`. Su uso es análogo al *array* `HTTP_GET_VARS` de versiones anteriores a la 4.2.0 de PHP (aún disponible pero obsoleto).
`_REQUEST`	*Array* que contiene las variables pasadas a través de cualquier mecanismo de entrada.

9.3.1 Formularios en PHP 4.2.x y versiones superiores

El siguiente ejemplo muestra el paso de información y su posterior procesamiento, haciendo uso del método `get`. Para ello, hemos escrito un documento HTML que contiene un formulario que hace uso de dicho método:

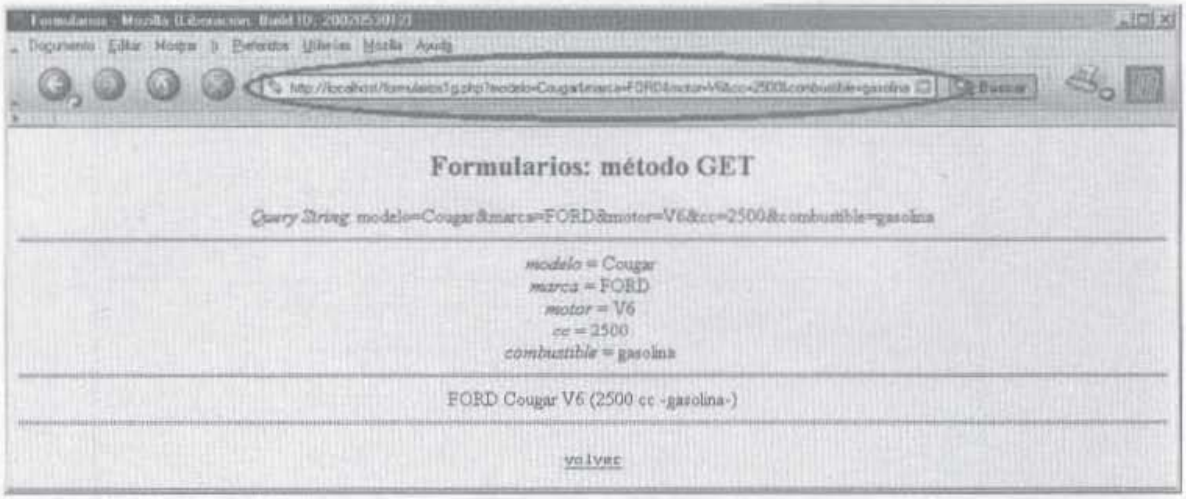
```
<HTML>
 <HEAD>
   <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Formularios: método GET</H2><HR>
     <FORM METHOD="GET" ACTION="formulariosig.php">
        <TABLE>
          <TR>
            <TD ALIGN="LEFT">Modelo:</TD>
            <TD ALIGN="RIGHT" COLSPAN="3">
               <INPUT TYPE="TEXT" NAME="modelo" SIZE=25">
            </TD>
          </TR>
          <TR>
            <TD ALIGN="LEFT">Marca:</TD>
            <TD ALIGN="RIGHT" COLSPAN="3">
                <INPUT TYPE="TEXT" NAME="marca" SIZE=25">
            </TD>
          </TR>
          <TR ALIGN="LEFT">
            <TD>Motor:</TD>
            <TD><INPUT TYPE="TEXT" NAME="motor" SIZE="5"></TD>
            <TD>Cilindrada:</TD>
            <TD><INPUT TYPE="TEXT" NAME="cc" SIZE="5"></TD>
          </TR>
          <TR>
            <TD ALIGN="LEFT">Combustible:</TD>
            <TD ALIGN="RIGHT" COLSPAN="3">
               <INPUT TYPE="RADIO" NAME="combustible"
                      VALUE="gasolina" CHECKED>Gasolina
               <INPUT TYPE="RADIO" NAME="combustible" VALUE="diesel">Diesel
            </TD>
          </TR>
        </TABLE><HR><BR>
        <INPUT TYPE="SUBMIT"> <INPUT TYPE="RESET">
     </FORM>
   </CENTER>
 </BODY>
</HTML>
```

El formulario se visualiza en la siguiente imagen:

El *script* encargado de procesar dicha información (valor del atributo `action` del formulario) es el siguiente:

```
<HTML>
 <HEAD>
   <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <?php
       $metodo=$_SERVER['REQUEST_METHOD'];
       $cad_consulta=$_SERVER['QUERY_STRING'];

       echo "<H2>Formularios: método $metodo</H2>";
       echo "<I>Query String</I>: $cad_consulta <HR>";
       foreach ($_GET as $clave => $valor)
            echo "<I>$clave</I> = $valor <BR>";
       echo "<HR>$_GET[marca] $_GET[modelo] $_GET[motor] " ;
       echo "($_GET[cc] cc -$_GET[combustible]-)<BR><HR>";
       echo "<PRE><A HREF='javascript:history.go(-1)'>volver</A></PRE>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

En la siguiente imagen podemos observar cómo aparecen en el URL tanto los nombres de los controles, como sus valores actuales:



En el caso de haber utilizado el método `post` para el envío de los datos, la única diferencia a nivel del formulario HTML se refleja en la siguiente línea:

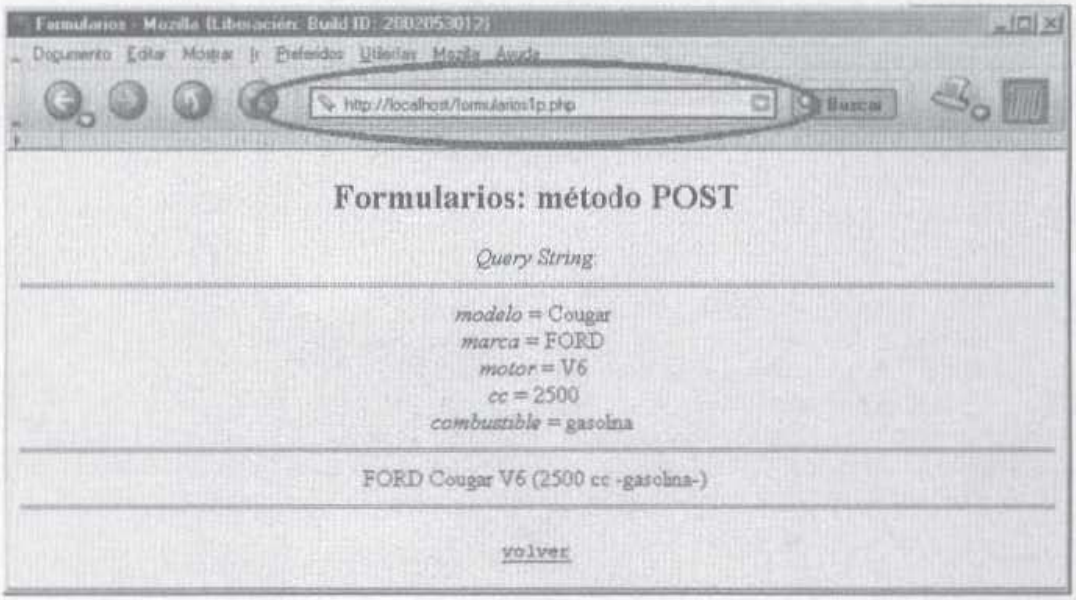
```
<FORM METHOD="POST" ACTION="formularios1p.php">
    .
    .
    .
</FORM>
```

El código del *script*, sin embargo, sí que muestra diferencias notables:

```
<HTML>
 <HEAD>
   <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <?php
       $metodo=$_SERVER['REQUEST_METHOD'];
       $cad_consulta=$_SERVER['QUERY_STRING'];

       echo "<H2>Formularios: método $metodo</H2>";
       echo "<I>Query String</I>: $cad_consulta <HR>";
       foreach ($_POST as $clave => $valor)
           echo "<I>$clave</I> = $valor <BR>";
       echo "<HR>$_POST[marca] ";
       echo "$_POST[modelo] ";
       echo "$_POST[motor] ";
       echo " ($_POST[cc] cc -$_POST[combustible]-)<BR><HR>";
       echo "<PRE><A HREF='javascript:history.go(-1)'>volver</A></PRE>";
```

```
    ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado es análogo al del ejemplo anterior, con la diferencia de que en este caso la cadena de consulta (`query string`) está vacía:



Existen dos formas para que, independientemente del método utilizado en el formulario, el *script* para procesarlo sea el mismo. El primero consiste en hacer uso de una variable intermedia que recoja los contenidos, tal y como muestra el siguiente código:

```
<HTML>
 <HEAD>
   <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <?php
       $metodo=$_SERVER['REQUEST_METHOD'];
       $vars_formulario=($metodo=="GET")?$_GET:$_POST;
       $cad_consulta=$_SERVER['QUERY_STRING'];

       echo "<H2>Formularios: método $metodo</H2>";
       echo "<I>Query String</I>: $cad_consulta <HR>";
       foreach ($vars_formulario as $clave => $valor)
           echo "<I>$clave</I> = $valor <BR>";
       echo "<HR>",$vars_formulario['marca']," ";
       echo $vars_formulario['modelo']," ";
       echo $vars_formulario['motor']," (";
       echo $vars_formulario['cc']," cc ";
       echo "-",$vars_formulario['combustible'],"-)<BR><HR>";
```

```
        echo "<PRE><A HREF='javascript:history.go(-1)'>volver</A></PRE>";
    ?>
  </CENTER>
 </BODY>
</HTML>
```

La otra posibilidad consiste en hacer uso de la variable _REQUEST, que contiene la información enviada desde el cliente al servidor independientemente del método utilizado:

```
<HTML>
 <HEAD>
  <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
  <CENTER>
   <?php
    $metodo=$_SERVER['REQUEST_METHOD'];
    $cad_consulta=$_SERVER['QUERY_STRING'];

    echo "<H2>Formularios: método $metodo</H2>";
    echo "<I>Query String</I>: $cad_consulta <HR>";
    foreach ($_REQUEST as $clave => $valor)
        echo "<I>$clave</I> = $valor <BR>";
    echo "<HR>$_REQUEST['marca'] $_REQUEST['modelo'] $_REQUEST['motor'] ";
    echo "($_REQUEST['cc'] cc -$_REQUEST['combustible']-)<BR><HR>";
    echo "<PRE><A HREF='javascript:history.go(-1)'>volver</A></PRE>";
   ?>
  </CENTER>
 </BODY>
</HTML>
```

En ambos casos el resultado es el mismo al obtenido en los ejemplos anteriores.

9.3.2 Formularios en versiones anteriores a PHP 4.2

En versiones anteriores a PHP 4.2.0, además de poder acceder a las variables pasadas en el formulario a través de los *arrays* HTTP_GET_VARS (equivalente al actual _GET) y HTTP_POST_VARS (ídem con _POST), todos los valores asociados a los controles de un formulario estaban accesibles a través de variables globales, es decir, las variables de un formulario pasaban a estar automáticamente disponibles en el *script* PHP, ya que éste generaba un conjunto de variables globales cuyos nombres coincidían con los dados a los controles del formulario a través de sus atributos name correspondientes.

> **NOTA:** Esta característica estaba disponible al activar en el fichero de configuración, (php.ini), la directiva register_globals (opción que aparecía activada por defecto). Actualmente, por cuestiones de seguridad, esta directiva aparece deshabilitada, si bien, por cuestiones de compatibilidad, aún se permite su utilización.

El siguiente ejemplo nos muestra el uso combinado de estas variables de tipo global utilizadas en versiones anteriores a PHP 4.2.0, con un resultado idéntico al de los ejemplos anteriores:

```
</BODY>
<HTML>
 <HEAD>
   <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <?php
       $metodo=$HTTP_SERVER_VARS['REQUEST_METHOD'];
       $cad_consulta=$HTTP_SERVER_VARS['QUERY_STRING'];
       $vars_formulario=($metodo=="GET")?$HTTP_GET_VARS:$HTTP_POST_VARS;

       echo "<H2>Formularios: método $metodo</H2>";
       echo "<I>Query String</I>: $cad_consulta <HR>";
       foreach ($vars_formulario as $clave => $valor)
            echo "<I>$clave</I> = $valor <BR>";
       echo "<HR>$marca $modelo $motor ($cc cc -$combustible-)<BR><HR>";
       echo "<PRE><A HREF='javascript:history.go(-1)'>volver</A></PRE>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

9.3.3 Formularios avanzados

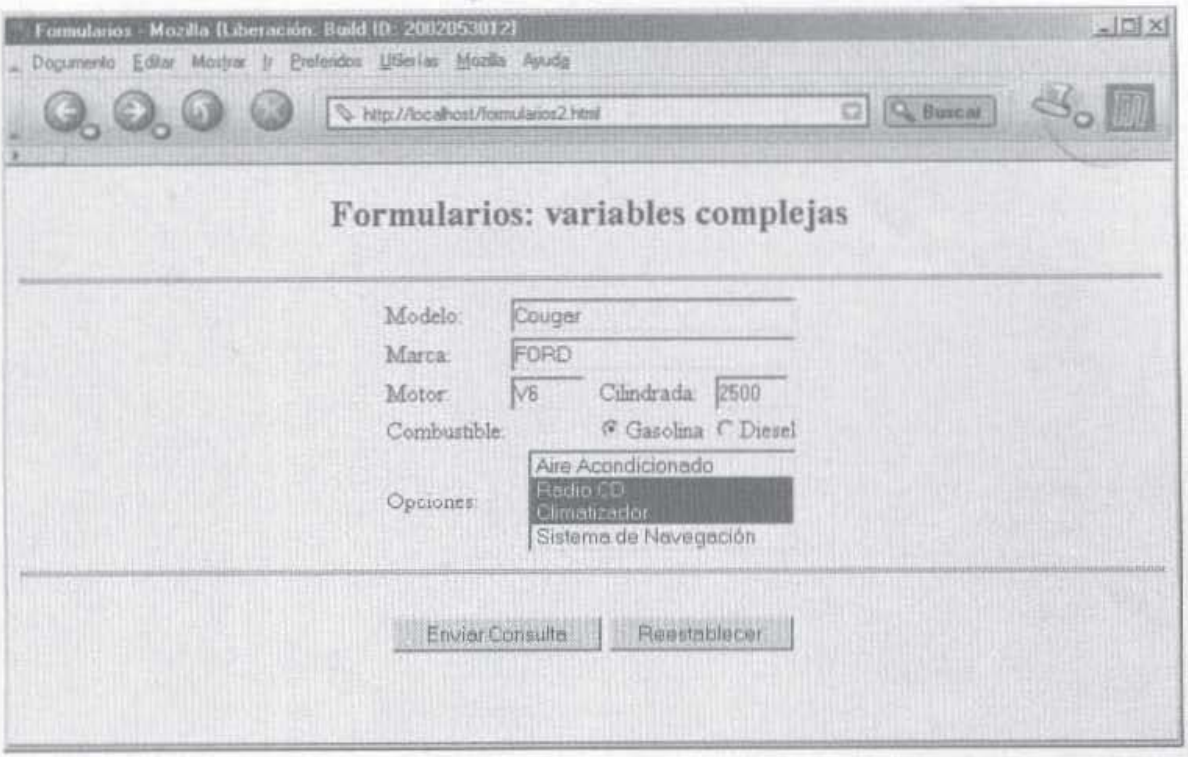
Además de la traslación directa de las variables de un formulario en variables de PHP, se puede hacer un uso avanzado de estas variables: en el formulario HTML podemos agrupar variables relacionadas en forma de *array*. Posteriormente, desde PHP podemos acceder a estos mismos *arrays* y, como tales, recorrerlos para obtener sus claves y respectivos valores. Esta característica también se puede usar para recuperar los valores de un campo `select` de tipo múltiple.

El siguiente ejemplo nos muestra dicha utilización:

```
<HTML>
 <HEAD>
   <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Formularios: variables complejas</H2><HR>
     <FORM METHOD="POST" ACTION="formularios2.php">
       <TABLE>
         <TR>
           <TD ALIGN="LEFT">Modelo:</TD>
           <TD ALIGN="RIGHT" COLSPAN="3">
```

```
            <INPUT TYPE="TEXT" NAME="coche[modelo]" SIZE=25">
          </TD>
        </TR>
        <TR>
          <TD ALIGN="LEFT">Marca:</TD>
          <TD ALIGN="RIGHT" COLSPAN="3">
            <INPUT TYPE="TEXT" NAME="coche[marca]" SIZE=25">
          </TD>
        </TR>
        <TR ALIGN='LEFT">
          <TD>Motor:</TD>
          <TD><INPUT TYPE="TEXT" NAME="coche[motor]" SIZE="5"></TD>
          <TD>Cilindrada:</TD>
          <TD><INPUT TYPE="TEXT" NAME="coche[cc]" SIZE="5"></TD>
        </TR>
        <TR>
          <TD ALIGN="LEFT">Combustible:</TD>
          <TD ALIGN="RIGHT" COLSPAN="3">
            <INPUT TYPE="RADIO" NAME="coche[combustible]"
                   VALUE="gasolina" CHECKED>Gasolina
            <INPUT TYPE="RADIO" NAME="coche[combustible]"
                   VALUE="diesel">Diesel
          </TD>
        </TR>
        <TR>
          <TD ALIGN="LEFT">Opciones:</TD>
          <TD ALIGN="RIGHT" COLSPAN="3">
            <SELECT MULTIPLE NAME="opciones[]">
              <OPTION VALUE="AA">Aire Acondicionado</OPTION>
              <OPTION VALUE="CD">Radio CD</OPTION>
              <OPTION VALUE="CA">Climatizador</OPTION>
              <OPTION VALUE="SN">Sistema de Navegación</OPTION>
            </SELECT>
          </TD>
        </TR>
      </TABLE><HR><BR>
      <INPUT TYPE="SUBMIT"> <INPUT TYPE="RESET">
    </FORM>
  </CENTER>
 </BODY>
</HTML>
```

Como podemos observar en el formulario, se definen dos variables de tipo *array* (uno de ellos de tipo asociativo). El formulario anterior se visualizaría del siguiente modo:



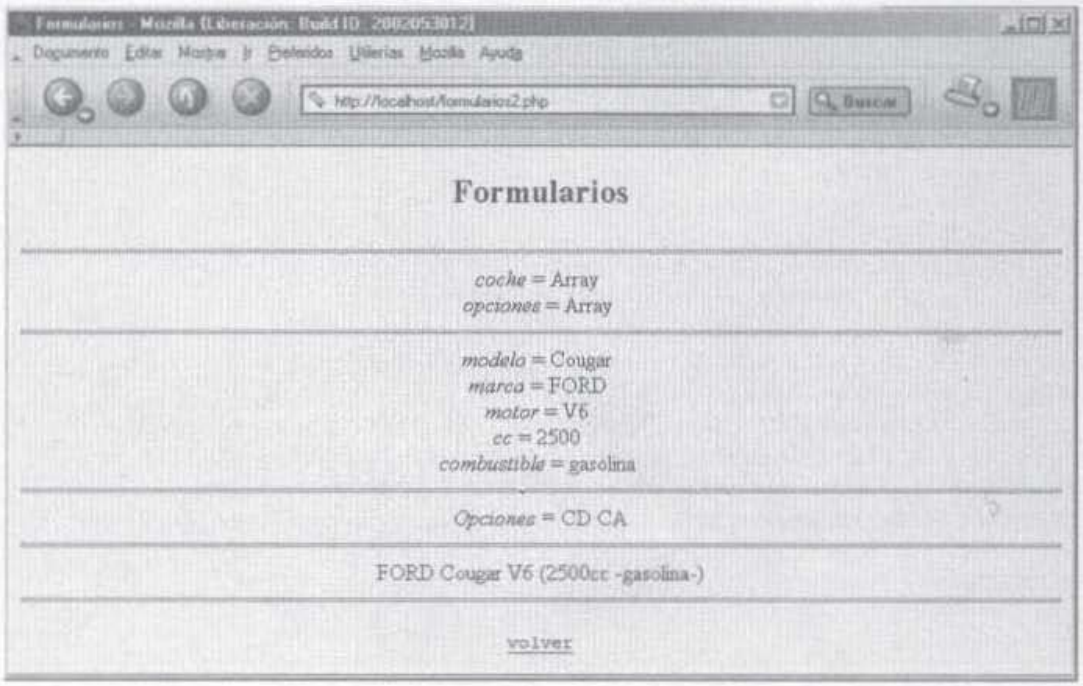
Para recuperar la información suministrada por el cliente, utilizamos el siguiente código:

```
<HTML>
 <HEAD>
   <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <?php
       echo "<H2>Formularios</H2>";
       echo "<HR>";
       foreach ($_POST as $clave => $valor)
            echo "<I>$clave</I> = $valor <BR>";
       echo "<HR>";
      foreach ($_POST['coche'] as $clave => $valor)
            echo "<I>$clave</I> = $valor <BR>";
       echo "<HR><I>Opciones = </I>";
      foreach ($_POST['opciones'] as $clave => $valor)
            echo "$valor ";
       echo "<HR>{$_POST['coche']['marca']} {$_POST['coche']['modelo']} ";
       echo "{$_POST['coche']['motor']} ({$_POST['coche']['cc']}cc -";
       echo "{$_POST['coche']['combustible']})-<BR><HR>";
       echo "<PRE><A HREF='javascript:history.go(-1)'>volver</A></PRE>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El código del ejemplo anterior para una versión de PHP anterior a la 4.2.0 sería el siguiente:

```
<HTML>
 <HEAD>
   <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <?php
       echo "<H2>Formularios</H2>";
       echo "<HR>";
       foreach ($HTTP_POST_VARS as $clave => $valor)
            echo "<I>$clave</I> = $valor <BR>";
       echo "<HR>";
      foreach ($coche as $clave => $valor)
            echo "<I>$clave</I> = $valor <BR>";
       echo "<HR><I>Opciones = </I>";
      foreach ($opciones as $clave => $valor)
            echo "$valor ";
       echo "<HR>$coche['marca'] $coche['modelo'] $coche['motor']
                 ($coche['cc'] cc -$coche['combustible']-)<BR><HR>";
       echo "<PRE><A HREF='javascript:history.go(-1)'>volver</A></PRE>";
     ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado, equivalente en ambos casos, se visualiza en la siguiente imagen:



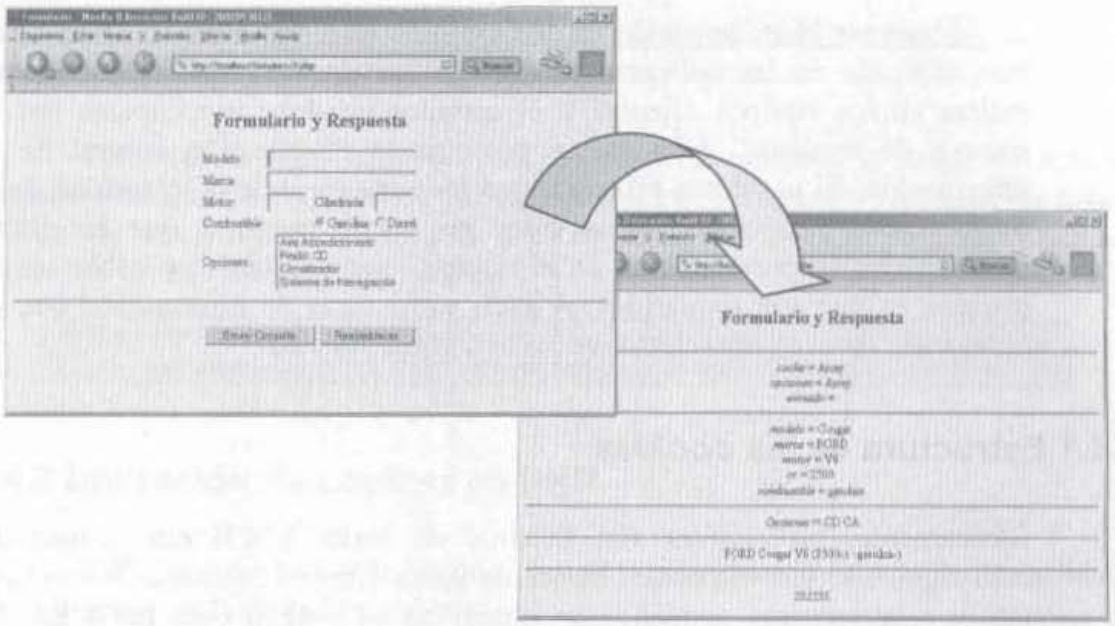
Una práctica muy habitual en PHP es agrupar en un solo fichero tanto el formulario que se le presenta al cliente, como su tratamiento. Aunque el funcionamiento interno es diferente, el resultado que observa el usuario es idéntico. El siguiente ejemplo muestra el uso de dicho modo de trabajo:

```
<HTML>
 <HEAD>
   <TITLE>Formularios</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
   <H2>Formulario y Respuesta</H2>
     <?php
       if(!$_POST){
     ?>
     <FORM METHOD="POST" ACTION="<?php echo $_SERVER['PHP_SELF'] ?>">
        <TABLE>
          <TR>
            <TD ALIGN="LEFT">Modelo:</TD>
            <TD ALIGN="RIGHT" COLSPAN="3">
                <INPUT TYPE="TEXT" NAME="coche[modelo]" SIZE=25">
            </TD>
          </TR>
          <TR>
            <TD ALIGN="LEFT">Marca:</TD>
            <TD ALIGN="RIGHT" COLSPAN="3">
               <INPUT TYPE="TEXT" NAME="coche[marca]" SIZE=25">
            </TD>
          </TR>
          <TR ALIGN="LEFT">
            <TD>Motor:</TD>
            <TD><INPUT TYPE="TEXT" NAME="coche[motor]" SIZE="5"></TD>
            <TD>Cilindrada:</TD>
            <TD><INPUT TYPE="TEXT" NAME="coche[cc]" SIZE="5"></TD>
          </TR>
          <TR>
            <TD ALIGN="LEFT">Combustible:</TD>
            <TD ALIGN="RIGHT" COLSPAN="3">
               <INPUT TYPE="RADIO" NAME="coche[combustible]"
                      VALUE="gasolina" CHECKED>Gasolina
               <INPUT TYPE="RADIO" NAME="coche[combustible]"
                      VALUE="diesel">Diesel
            </TD>
          </TR>
          <TR>
```

```
            <TD ALIGN="LEFT">Opciones:</TD>
            <TD ALIGN="RIGHT" COLSPAN="3">
               <SELECT MULTIPLE NAME="opciones[]">
                  <OPTION VALUE="AA">Aire Acondicionado</OPTION>
                  <OPTION VALUE="CD">Radio CD</OPTION>
                  <OPTION VALUE="CA">Climatizador</OPTION>
                  <OPTION VALUE="SN">Sistema de Navegación</OPTION>
               </SELECT>
            </TD>
         </TR>
      </TABLE><HR><BR>
      <INPUT TYPE="SUBMIT" NAME="enviado"> <INPUT TYPE="RESET">
   </FORM>
   <?php
      } else {
         echo "<HR>";
         foreach ($_POST as $clave => $valor)
               echo "<I>$clave</I> = $valor <BR>";
         echo "<HR>";
         foreach ($_POST['coche'] as $clave => $valor)
               echo "<I>$clave</I> = $valor <BR>";
         echo "<HR><I>Opciones = </I>";
         foreach ($_POST['opciones'] as $clave => $valor)
               echo "$valor ";
         echo "<HR>{$_POST['coche']['marca']} {$_POST['coche']['modelo']} ";
         echo $_POST['coche']['motor'];
         echo " ({$_POST['coche']['cc']}cc -
{$_POST['coche']['combustible']})<BR><HR>";
         echo "<PRE><A HREF='javascript:history.go(-1)'>volver</A></PRE>";
      }
   ?>
</CENTER>
</BODY>
</HTML>
```

Como podemos observar, este modo de trabajo consiste básicamente en dividir el código en dos secciones: cuando el usuario solicita la página en cuestión, lo primero que se encuentra el intérprete de PHP es una sentencia `if` que evalúa si el formulario ha sido enviado (preguntando por el contenido de la variable global encargada de almacenar la información transferida). En caso de que el formulario no haya sido enviado, el flujo de control evolucionará para tratar la parte del código que muestra el formulario HTML al cliente. En caso contrario, se evaluará la alternativa contraria en la que se procesan los datos enviados en el formulario.

El resultado de la ejecución de este *script* se muestra en la siguiente imagen:



9.4 *COOKIES* EN PHP

Como comentamos anteriormente, el protocolo HTTP está definido de forma que nunca se almacena información acerca de las conexiones y desconexiones que haya habido entre cliente y servidor, es un protocolo *stateless*. Por tanto, cuando necesitamos que algún dato o información de relevancia (preferencias del usuario, número de accesos, recursos visitados...) esté disponible entre diferentes conexiones, es necesario implementar un mecanismo que nos permita almacenar y acceder a dicha información. Para llevar a cabo dicha facilidad, tenemos varias soluciones posibles:

- Hacer uso de elementos de formulario ocultos. Efectivamente, si enviamos esta información en campos ocultos de un formulario (elementos `input` de tipo `hidden`), podremos manejarlos entre las diferentes páginas de nuestra aplicación y *arrastrar* esa información de una página a otra. El problema principal que presenta esta posible solución es que los datos tendrán vigencia, existirán, mientras el cliente esté navegando y, además, lo haga de una manera *ordenada*, esto es, siga la secuencia de páginas prevista en la aplicación.

- Almacenar la información en el servidor. Esta solución nos permite que, si el usuario deja de navegar, cuando retome el uso de nuestra aplicación Web, sus datos los tendremos disponibles. El único problema de esta solución está en el hecho de que un número muy grande de usuarios supone, por parte del servidor, la reserva de gran cantidad de recursos de almacenamiento, por tanto, es una solución válida cuando el número de posibles usuarios de nuestra aplicación Web no vaya a ser alto.

- Almacenar la información en los equipos clientes. Esta solución es la más utilizada en las aplicaciones Web puesto que el almacenamiento se realiza en los equipos clientes y el servidor no debe preocuparse por la reserva de recursos. Además, se potencia la distribución natural de la información. El problema principal que presenta es el de la integridad de la información, puesto que el servidor no puede asegurar que los datos almacenados (supuestamente) en el equipo cliente están disponibles en el instante en que son requeridos. A estas estructuras de información que se almacenan en el equipo cliente se las denomina ***cookies***.

9.4.1 Estructura de las *cookies*

Básicamente, las *cookies* son ficheros de texto ASCII que almacenan información siguiendo una estructura básica de pares `identificador = valor`. Su tamaño es relativamente pequeño, no superando en ningún caso los 4 Kb. El modo en que se almacenan, los directorios en que se almacenan y resto de características propias de una operación de lectura/escritura sobre disco dependen en gran manera del sistema operativo y del navegador que tenga instalado el equipo cliente. De igual forma, la posibilidad de hacer uso de *cookies* depende de que el *software* utilizado para acceder a la aplicación Web (normalmente un navegador) cuente con esta característica y que, además, esté habilitada.

La siguiente tabla nos muestra la estructura básica de una *cookie*:

Elemento	Significado
Nombre	Indica el nombre que se le ha dado a la *cookie*.
Valor	Indica el valor de la *cookie*, es decir, el contenido que tiene.
Caducidad	Indica cuál es el tiempo de validez de la *cookie*.
Dominio	Indica el rango de dominios en los cuales es válida la *cookie*.
Ruta	Contiene el directorio a partir del cual la *cookie* tiene validez.
Seguro	Indica que la *cookie* será transmitida únicamente por un canal seguro SSL.

Es muy importante tener en cuenta que todo el tratamiento de las *cookies* se realiza en la cabecera del mensaje HTTP y, por tanto, deben ser manejadas antes de

que se envíe cualquier otra información al navegador (la parte del cuerpo del mensaje HTTP); en caso contrario, obtendremos un error.

Por otra parte, el valor de una *cookie* tiene preferencia sobre los valores pasados mediante un formulario, es decir, cuando un formulario y una *cookie* hacen uso de los mismos identificadores, los valores de las *cookies* sobrescribirán los valores de las entradas del formulario.

> **NOTA:** Para obtener más información sobre las *cookies* visítese la hoja con su especificación oficial: http://www.netscape.com/newsref/std/cookie_spec.html.

9.4.2 Utilización de *cookies* en PHP

PHP propone una sola función para el manejo básico de *cookies* (creación y borrado) cuya sintaxis es la siguiente:

```
int setcookie (string nombre [, string valor]
               [, int caducidad]  [, string ruta]
               [, string dominio] [, int seguro]);
```

Para acceder al contenido, se hará uso de una variable global, tal y como veremos más adelante.

9.4.2.1 Creación de *cookies*

Como se puede observar en la definición de la función `setcookie()`, cada uno de los parámetros coincide con los elementos que componen la estructura básica de una *cookie*. También vemos que el único argumento obligatorio en la llamada a la función es el nombre que se le asigna a la *cookie*. Sin embargo, en el caso de la creación es necesario, al menos, que se le asigne un valor inicial.

Cuando no queramos hacer uso de los parámetros de tipo `string`, se deben reemplazar con la cadena vacía ("") y los de tipo `int`, con un valor 0. Si no usamos el parámetro `caducidad`, se tomará por defecto el tiempo que dure la sesión de trabajo activa en el navegador. De igual modo, si no se utilizan los parámetros `ruta` y `dominio`, se tomarán por defecto el camino y el dominio del servidor en los cuales se ha creado la *cookie*.

> **NOTA:** En PHP 3, cuando en un mismo *script* aparecían múltiples llamadas a la función `setcookie()`, éstas eran evaluadas en orden inverso al orden de su aparición.

9.4.2.2 Eliminación de *cookies*

Como ya sabemos, para borrar una *cookie*, usamos la misma función que para crearla: `setcookie()`, sólo que en este caso la llamada a la función sólo contendrá como parámetro el nombre de la *cookie* que deseamos eliminar del sistema.

```
setcookie("nombre_de_la_cookie");
```

9.4.2.3 Consulta de contenidos

Para poder acceder a los contenidos de las *cookies*, PHP proporciona una variable global consistente en un *array* asociativo formado por todas las variables pasadas a través de las *cookies*. Es la variable `$_COOKIE`:

```
$_COOKIE["nombre_de_la_cookie"]
```

Si se está usando una versión anterior a la 4.2.0, se deberá utilizar la variable `$HTTP_COOKIE_VARS`, cuya funcionalidad es exactamente igual a `$_COOKIE`:

```
$HTTP_COOKIE_VARS["nombre_de_la_cookie"]
```

Adicionalmente, en estas versiones anteriores a la 4.2.0, PHP, de forma automática, generaba, por cada una de las *cookies*, una variable global de igual nombre.

> **NOTA:** Esta característica estaba disponible al activar en el fichero de configuración, `php.ini`, la directiva `register_globals` (opción que aparecía configurada por defecto). Actualmente, por cuestiones de seguridad, esta directiva aparece deshabilitada, si bien, por cuestiones de compatibilidad, aún se permite su utilización.

Una vez que conocemos los tres pasos esenciales para trabajar con *cookies*, vamos a ver un ejemplo que hace uso de esta técnica para implementar un contador de accesos. Nuestro primer *script* contendrá la creación y actualización de la *cookie*:

```
<?php
  $accesos=1;
  if(isset($_COOKIE['num_accesos'])){   // si la cookie existe recupera su
valor
     $accesos=$_COOKIE['num_accesos']+1;
  }
  setcookie("num_accesos",$accesos,time()+3600); // crea/actualiza la cookie
?>
<HTML>
```

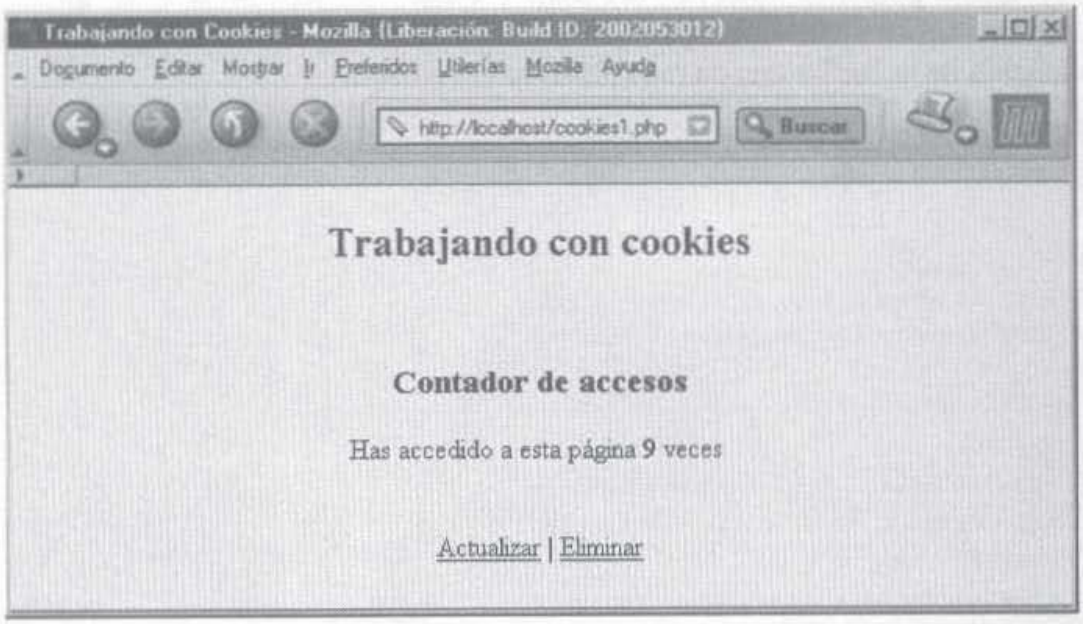
```
<HEAD>
  <TITLE>Trabajando con Cookies</TITLE>
</HEAD>
<BODY>
  <CENTER>
    <H2>Trabajando con cookies</H2><br>
    <H3>Contador de accesos</H3>
    <?php
      if($accesos>1)
        echo "Has accedido a esta página <B>$accesos</B> veces";
      else
        echo "Es la primera vez que accedes a esta página";
    ?>
    <BR><BR><BR>
    <A HREF="cookies1.php">Actualizar</A> |
        <A HREF="cookies2.php">Eliminar</A>
  </CENTER>
</BODY>
</HTML>
```

Como podemos observar, la *cookie* creada, num_accesos, tiene un tiempo de expiración de una hora (3.600 segundos). El segundo *script* contiene la eliminación de la *cookie* creada para albergar el contador:

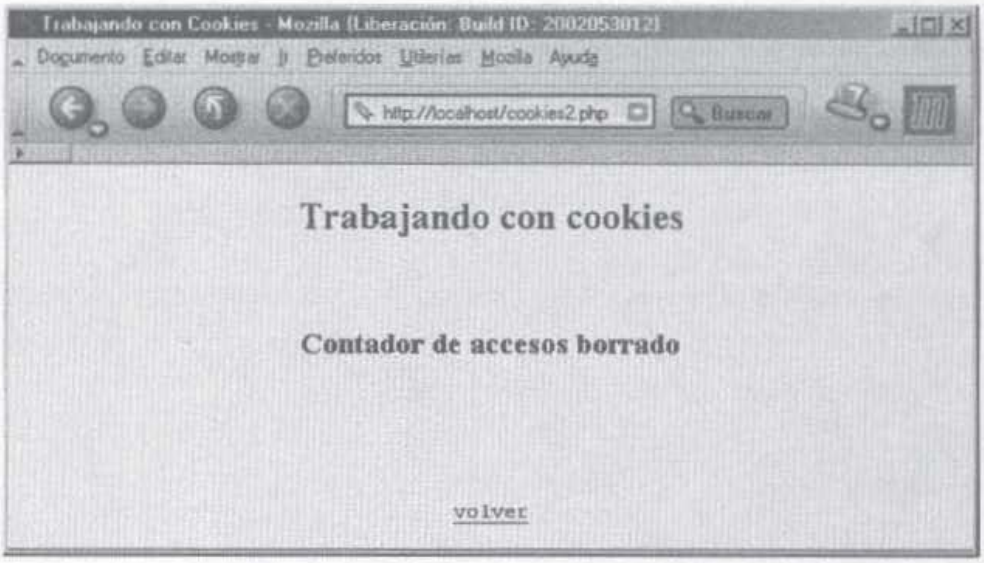
```
<?php
  setcookie("num_accesos");   //elimina la cookie
?>
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cookies</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con cookies</H2><br>
     <H3>Contador de accesos borrado</H3>
     <BR><BR><BR>
     <PRE><A HREF="cookies1.php">volver</A></PRE>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se puede mostrar a través de las siguientes imágenes. La primera vez que se accede obtenemos el siguiente resultado:

Los accesos posteriores a la misma página irán incrementando el número de accesos almacenado en la *cookie* `num_accesos`:



Finalmente, cuando pulsamos en el enlace `Eliminar` la *cookie*, `num_accesos` es borrada del sistema del cliente y, por tanto, cuando volvamos a la página anterior, no existirá:

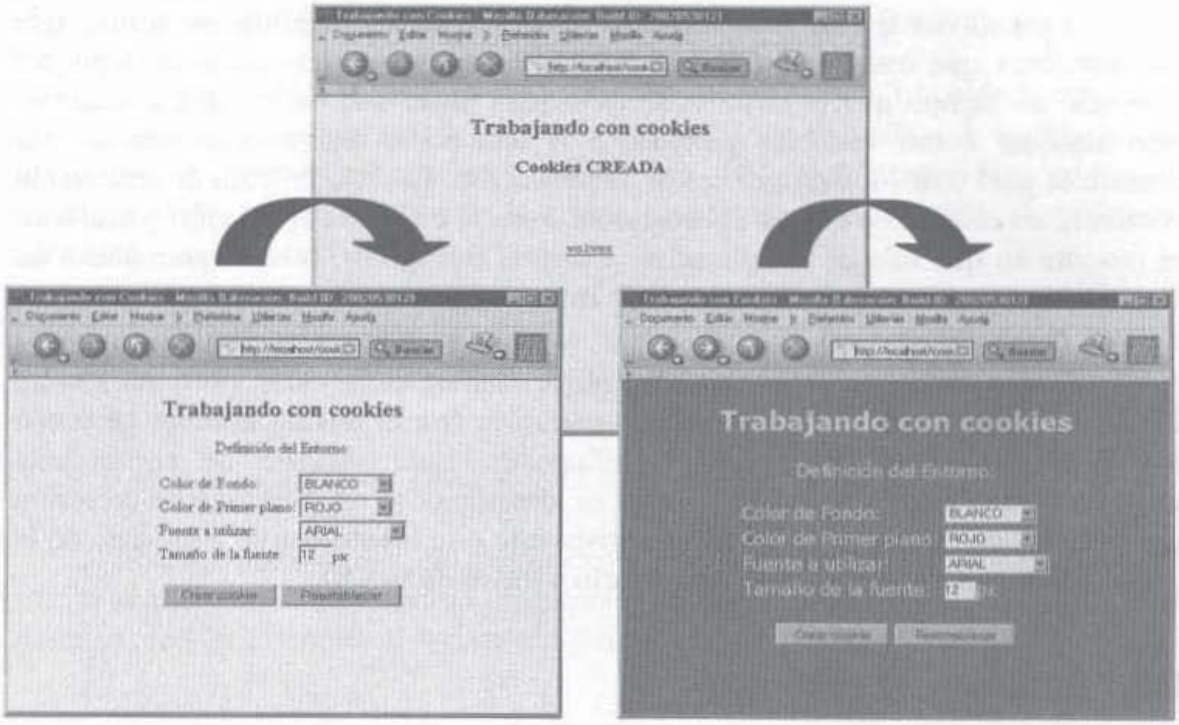


En el ejemplo anterior hemos visto que una *cookie* puede almacenar un único valor, pero también es posible que una misma *cookie* almacene múltiples valores. Para ello, definiremos la *cookie* como un *array*, utilizando la notación propia de éstos al especificar su nombre. El siguiente ejemplo muestra un uso avanzado de *cookies* con estructura de *array*:

```
<?php
  if($_POST){
     setcookie("entorno[color_fondo]",$_POST[color_fondo]);
     setcookie("entorno[color_plano]",$_POST[color_plano]);
     setcookie("entorno[letra_fuente]",$_POST[letra_fuente]);
     setcookie("entorno[letra_tamanio]",$_POST[letra_tamanio]);
?>
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cookies</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con cookies</H2>
     <H3>Cookies CREADA</H3><BR><BR>
     <PRE><A HREF="cookies3.php">volver</A></PRE>
   </CENTER>
 </BODY>
</HTML>
<?php
  } else {
?>
<HTML>
 <HEAD>
   <TITLE>Trabajando con Cookies</TITLE>
   <STYLE>
      BODY, TABLE, TD {
        font-family: '<?php echo $_COOKIE[entorno][letra_fuente]; ?>';
```

```
      font-size: <?php echo $_COOKIE[entorno][letra_tamanio]; ?>px;
      color: <?php echo $_COOKIE[entorno][color_plano]; ?>;
      background-color: <?php echo $_COOKIE[entorno][color_fondo]; ?>;
    }
  </STYLE>
</HEAD>
<BODY>
  <CENTER>
    <H2>Trabajando con cookies</H2>
    <P>Definición del Entorno:</P>
    <TABLE>
    <TR><TD>
    <FORM METHOD="POST" ACTION="cookies3.php">
     Color de Fondo:</TD><TD>
     <SELECT NAME="color_fondo">
        <OPTION VALUE="WHITE">BLANCO</OPTION>
        <OPTION VALUE="GREEN">VERDE</OPTION>
        <OPTION VALUE="BLUE">AZUL</OPTION>
        <OPTION VALUE="GRAY">GRIS</OPTION>
        <OPTION VALUE="YELLOW">AMARILLO</OPTION>
     </SELECT></TD></TR><TR><TD>
     Color de Primer plano:</TD><TD>
     <SELECT NAME="color_plano">
        <OPTION VALUE="RED">ROJO</OPTION>
        <OPTION VALUE="GREEN">VERDE</OPTION>
        <OPTION VALUE="BLUE">AZUL</OPTION>
        <OPTION VALUE="GRAY">GRIS</OPTION>
        <OPTION VALUE="YELLOW">AMARILLO</OPTION>
     </SELECT></TD></TR><TR><TD>
     Fuente a utilizar:</TD><TD>
     <SELECT NAME="letra_fuente">
        <OPTION VALUE="Arial">ARIAL</OPTION>
        <OPTION VALUE="Courier New">COURIER</OPTION>
        <OPTION VALUE="Comic Sans MS">COMIC</OPTION>
        <OPTION VALUE="Garamont">GARAMONT</OPTION>
        <OPTION VALUE="Tahoma">TAHOMA</OPTION>
     </SELECT><BR></TD></TR><TR><TD>
     Tamaño de la fuente:</TD><TD>
     <INPUT TYPE="TEXT" NAME="letra_tamanio"
            SIZE="3" MAXLENGTH="2" VALUE="12">px
     </TABLE><BR>
     <INPUT TYPE="SUBMIT" VALUE="Crear cookie"> <INPUT TYPE="RESET">
    </FORM>
  </CENTER>
 </BODY>
</HTML>
<?php
 }
?>
```

El resultado se puede observar en la siguiente imagen:



En el paso intermedio en que se informa al usuario de que la *cookie* ha sido creada con éxito, no es posible acceder a sus contenidos. Es necesario, por tanto, esperar a la siguiente carga de una página para poder acceder a los contenidos de la *cookie* y actualizar la visualización con los valores definidos por el usuario. Hay que tener muy en cuenta esta característica en el trabajo con *cookies*, puesto que un error habitual consiste en intentar establecer una *cookie* y acceder a su contenido en un mismo paso.

9.5 SESIONES EN PHP

A lo largo de este capítulo hemos visto que teníamos tres formas básicas de hacer que la información generada en un *script* estuviera disponible en *scripts* diferentes al de creación: utilizando formularios, pasando las variables y sus valores a través de la URL, o bien, definiendo *cookies*. Estos métodos no son todo lo útiles que desearíamos cuando los *scripts* que deben compartir la información no se ejecutan secuencialmente, o bien, están distantes los unos de los otros dentro del flujo de ejecución normal de nuestra aplicación. Aunque, a priori, el uso de *cookies* parece adecuado para solventar este problema (pueden ser consultadas en cualquier momento hasta su eliminación del sistema de cliente), hay que tener en cuenta que no todos los navegadores tienen disponible esta funcionalidad y que, incluso teniéndola, ésta puede aparecer deshabilitada por parte del usuario.

Para solventar esta situación, PHP proporciona las *variables de sesión*, que son variables que estarán disponibles en las diferentes peticiones a lo largo del intervalo de tiempo que el usuario necesite para hacer uso de nuestra aplicación. Son algo así como variables globales a la aplicación entera pues van a estar accesibles para todos los programas de la aplicación. El ciclo de vida de una sesión comienza en el instante en que el usuario se conecta a nuestra aplicación y acaba en el instante en que sale de la aplicación, cierra el navegador, o bien, permanece un lapso de tiempo (fijado por el sistema) sin interaccionar con la aplicación.

Todas las variables de sesión se almacenan en el servidor, por tanto, en un momento dado, podremos tener muchas variables con el mismo nombre pero con contenidos diferentes. Para evitar confusiones, cada variable de sesión está vinculada a una única sesión/usuario, que se identifica con un identificador de sesión único. El modo de mantener de forma persistente este identificativo a lo largo de la sesión es utilizar *cookies*, o bien, propagarlo a través de la URL.

9.5.1 Creación de sesiones

La configuración por defecto de PHP tiene deshabilitadas las sesiones; esto quiere decir que, cuando queramos hacer uso de ellas, deberemos indicárselo de forma explícita. Cambiando la configuración (modificando el valor de la directiva `session.auto_start` a 1 en el fichero de configuración `php.ini`), podemos activar por defecto las sesiones en PHP de modo que en el primer acceso de cada usuario a nuestro sistema se le cree una sesión.

El modo principal de activar el uso de sesiones cuando lo deseemos es hacer uso de la función `session_start()` cuya sintaxis es la siguiente:

```
bool session_start()
```

Esta función crea una nueva sesión y genera el nuevo identificador, o retoma la sesión en caso de que existiera, utilizando el identificador de sesión que se había propagado haciendo uso de la URL o de *cookies*.

> **NOTA:** Si se está haciendo uso de sesiones basadas en *cookies*, la función *session_start()* debe ser llamada al principio de nuestro *script* (antes de abrir cualquier etiqueta o de imprimir cualquier contenido). En caso contrario, se genera un error.

Como hemos dicho anteriormente, en el instante en que se crea una sesión, se genera un identificador único para ella. Haciendo uso de la función `session_id()`, podemos recuperar o modificar dicho valor. Su sintaxis es la siguiente:

```
string session_id ([string id])
```

Si llamamos a la función sin hacer uso del parámetro `id`, obtendremos una cadena con el identificador de la sesión actual (debe ser llamada una vez que exista la sesión). Si, por el contrario, hacemos uso del citado parámetro, estaremos reemplazando el identificativo de la sesión actual por el valor dado al parámetro (debe ser llamada antes de crear la sesión).

También es posible identificar a las sesiones asignándoles un nombre específico. Para ello, utilizaremos la función `session_name()` que tiene la siguiente sintaxis:

```
string session_name ([string nombre])
```

Si no hacemos uso del parámetro *nombre*, obtendremos el nombre de la sesión actual; si, por el contrario, proporcionamos dicho parámero, la sesión actual pasará a tener por nombre el argumento pasado.

9.5.2 Acceso a las variables de sesión

El modo de acceso a las variables de sesión es a través del *array* asociativo `$_SESSION` disponible como variable global (`$HTTP_SESSIONS_VARS` en las versiones anteriores a la 4.2.0). El siguiente ejemplo muestra la creación de una sesión y de una variable de sesión al igual que su acceso, actualización y eliminación para poder obtener un contador de accesos:

```
<?php
  session_start();
  if (isset($_SESSION['contador'])) {
     $_SESSION['contador']++;
  }
  else {
     $_SESSION['contador'] = 0;
  }
  $nombre_anterior = session_name('SESION_CONTADOR');
?>

<HTML>
 <HEAD>
   <TITLE>Trabajando con Sesiones</TITLE>
 </HEAD>
 <BODY>
```

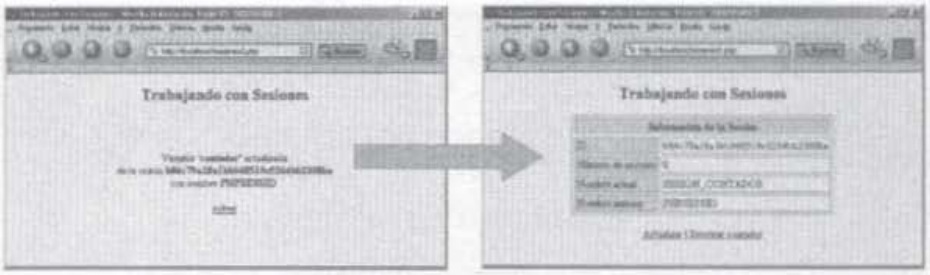
```
  <CENTER>
    <H2>Trabajando con Sesiones</H2>
    <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="4">
      <TR ALIGN="center" BGCOLOR="yellow">
        <TD COLSPAN="2"><B>Información de la Sesión</B></TD>
      </TR>
      <TR>
        <TD BGCOLOR="yellow">ID</TD>
        <TD><?php echo session_id() ?></TD>
      </TR>
      <TR>
        <TD BGCOLOR="yellow">Número de accesos</TD>
        <TD><?php echo $_SESSION['contador'] ?></TD>
      </TR>
      <TR>
        <TD BGCOLOR="yellow">Nombre actual</TD>
        <TD><?php echo session_name() ?></TD>
      </TR>
      <TR>
        <TD BGCOLOR="yellow">Nombre anterior</TD>
        <TD><?php echo $nombre_anterior ?></TD>
      </TR>
    </TABLE>
    <BR>
      <A HREF="sesiones1.php">Actualizar</A> |
      <A HREF="sesiones2.php">Resetear contador</A>
  </CENTER>
 </BODY>
</HTML>
```

Como podemos observar, para crear una nueva variable de sesión, sólo hay que definirla dentro del *array* `$_SESSION`. En el código se pregunta si la variable de sesión tiene valor (si está definida) y, de este modo, saber si hay que crearla o actualizar su valor. El ejemplo, además, obtiene el identificador de sesión y modifica el nombre asignado por defecto. La siguiente imagen muestra el resultado después de haber realizado 10 accesos a la página que contiene el contador:

El segundo *script* de este ejemplo se encarga de borrar la variable de sesión creada en el *script* anterior.

```
<?php
  session_start();
  unset($_SESSION['contador']);
?>
<HTML>
 <HEAD>
   <TITLE>Trabajando con Sesiones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con Sesiones</H2><BR><BR>
     <P>Variable <B>'contador'</B> actualizada<BR>
        de la sesión <B><?php echo session_id() ?></B><BR>
        con nombre <B><?php echo session_name() ?></B>
        <BR><BR><A HREF="sesiones1.php">volver</A></P>
   </CENTER>
 </BODY>
</HTML>
```

Como se puede observar en la siguiente imagen, la sesión sigue siendo la misma (su identificador es el mismo), pero ha perdido el nombre de la sesión. Este valor es necesario actualizarlo en cada petición; si no, se restaura con el valor almacenado por defecto en la directiva `session.name` del fichero de configuración `php.ini`:



Existe una forma alternativa para acceder a las variables de sesión sin utilizar la variable global `$_SESSION` que nos permite manejar estas variables directamente haciendo uso de su nombre. Para que esta opción esté disponible, será necesario activar la directiva `register_globals` en el fichero de configuración de PHP.

> **NOTA:** Por cuestiones de seguridad y rendimiento, es preferible hacer uso del *array* `$_SESSION`.

Este modo de trabajo está asociado con tres funciones PHP cuyas sintaxis y usos son los siguientes:

- `session_register(nombre [, nombre])`: Registra una nueva variable global para la sesión actual. En caso de que no exista una sesión, la crea realizando una llamada implícita a la función `session_start()`. Puede recibir múltiples argumentos, cada uno de los cuales será una nueva variable global para la sesión actual (devuelve `true` cuando todas las variables pasadas como argumentos han sido creadas sin ningún problema).

- `session_is_registered(nombre)`: Devuelve `true` cuando la variable cuyo nombre se ha pasado como argumento a la llamada está registrada dentro de la sesión actual.

- `session_unregister(nombre)`: Evita que la variable que se ha pasado como argumento a la llamada sea salvada como parte de la sesión actual, es decir, la elimina de la sesión, pero sigue siendo una variable del *script* donde se ejecuta (para esto tendríamos que llamar a la función `unset()`).

El siguiente código muestra el mismo ejemplo anterior, pero haciendo uso de este método de acceso a las variables de sesión:

```
<?php
  session_start();
  if (!session_is_registered('contador')) {
      session_register('contador');
      $contador = 0;
  }
  else {
```

```
    $contador++;
  }
  $nombre_anterior = session_name('SESION_CONTADOR');
?>
<HTML>
 <HEAD>
   <TITLE>Trabajando con Sesiones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con Sesiones</H2>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="4">
       <TR ALIGN="center" BGCOLOR="yellow">
          <TD COLSPAN="2"><B>Información de la Sesión</B></TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">ID</TD>
          <TD><?php echo session_id() ?></TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">Número de accesos</TD>
          <TD><?php echo $contador ?></TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">Nombre actual</TD>
          <TD><?php echo session_name() ?></TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">Nombre anterior</TD>
          <TD><?php echo $nombre_anterior ?></TD>
       </TR>
     </TABLE>
     <BR><PRE>
       <A HREF="sesiones3.php">Actualizar</A> |
       <A HREF="sesiones4.php">Resetear contador</A>
     </PRE>
   </CENTER>
 </BODY>
</HTML>
```

El fichero encargado de eliminar la variable de la sesión tiene el siguiente contenido:

```
<?php
  session_start();
  if(session_is_registered('contador'))
     session_unregister('contador');
?>
<HTML>
 <HEAD>
   <TITLE>Trabajando con Sesiones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con Sesiones</H2><BR><BR>
     <P>Variable <B>'contador'</B> actualizada<BR>
        de la sesión <B><?php echo session_id() ?></B><BR>
        con nombre <B><?php echo session_name() ?></B>
```

```
        <BR><BR><A HREF="sesiones3.php">volver</A></P>
    </CENTER>
  </BODY>
</HTML>
```

Los resultados son análogos a los mostrados en el ejemplo original.

9.5.3 Otras funciones asociadas al manejo de sesiones

En este apartado se muestran algunas de las funciones más representativas que PHP proporciona para el trabajo con sesiones.

- `session_destroy()`: Elimina todos los datos asociados con la sesión actual, sin modificar las variables globales asociadas a la sesión ni la *cookie* de sesión.

- `session_unset()`: Libera todos los recursos asociados a las variables de sesión actualmente registradas.

- `session_encode()`: Codifica los datos de la sesión actual en una cadena.

- `session_encode(datos)`: Decodifica los datos de una sesión pasados como argumento en una cadena, generando las variables guardadas en dicha sesión.

- `session_cache_expires([caducidad])`: Devuelve el tiempo que resta en minutos para que la sesión finalice. Si pasamos un valor entero como argumento en la llamada a la función, se actualizará el tiempo de caducidad de la sesión.

- `session_get_cookie_params()`: Obtiene los parámetros de la *cookie* de sesión (duración, camino, dominio y seguridad), en caso de que éste haya sido el método elegido para gestionar las sesiones.

- `session_set_cookie_params (tiempo [,camino [,dominio [,seguridad]]])`: Nos permite cambiar los parámetros de la *cookie* de sesión actual.

- `session_save_path ([string camino])`: Nos permite obtener o modificar la ruta donde se guardan los datos de la sesión actual.

El siguiente ejemplo muestra la utilización de algunas de estas funciones:

```
<?php
  session_start();
  $_SESSION['modelo'] = "Cougar V6";
  $_SESSION['cc'] = 2500;
?>
<HTML>
 <HEAD>
   <TITLE>Trabajando con Sesiones</TITLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Trabajando con Sesiones</H2>
     <TABLE BORDER="1" CELLPADDING="2" CELLSPACING="4">
       <TR ALIGN="center" BGCOLOR="yellow">
          <TD COLSPAN="2"><B>Información de la Sesión</B></TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">ID</TD>
          <TD><?php echo session_id() ?></TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">Tiempo de caducidad</TD>
          <TD><?php echo session_cache_expire()," minutos" ?></TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">Control de caché</TD>
          <TD><?php echo session_cache_limiter() ?></TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">Cookie de sesión</TD>
          <TD>
              <?php
                $parametros=session_get_cookie_params();
                while(list($par,$valor)=each($parametros)){
                   echo $par,"=",$valor,";  ";
                }
              ?>
          </TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">Codificación</TD>
          <TD><?php echo session_encode() ?></TD>
       </TR>
       <TR>
          <TD BGCOLOR="yellow">Directorio de salvado</TD>
          <TD><?php echo session_save_path() ?></TD>
       </TR>
     </TABLE>
   </CENTER>
 </BODY>
</HTML>
```

Se visualiza en la siguiente imagen:

9.5.4 Parámetros de configuración de sesiones

La siguiente tabla muestra un resumen de las directivas del fichero `php.ini` relacionadas con la gestión de sesiones:

Directiva	Valor por Defecto	Descripción
`session.auto_start`	0	Especifica si el módulo que gestiona las sesiones se inicia automáticamente al recibir una petición.
`session.name`	`PHPSESSID`	Especifica el nombre de la sesión que se usa como nombre de la *cookie*.
`session.save_handler`	`files`	Define el tipo de controlador que se usa para almacenar y recuperar los datos asociados a la sesión. Podemos especificar `user` si las operaciones las implementa el usuario.
`session.save_path`	`/tmp`	Ruta donde se almacenan los datos asociados a las *cookies*.
`session.serialize_handler`	`php`	Define el controlador que se utiliza para guardar y restaurar los datos de forma serializada.
`session.use_cookies`	1	Indica si el módulo puede usar *cookies* para guardar el identificativo de sesión en el lado del cliente.
`session.use_only_cookies`	0	Especifica si sólo se deben utilizar *cookies* para guardar el identificativo de sesión en el lado del cliente.
`session.cookie_lifetime`	0	Especifica la duración de la *cookie* en segundos que se manda al navegador. El valor 0 significa 'hasta que se cierra el navegador'.
`session.cookie_path`	/	Especifica la ruta a colocar en `session_cookie`.

Directiva	Valor por Defecto	Descripción
`session.cookie_domain`		Especifica el dominio a establecer en `session_cookie`.
`session.cookie_secure`		Especifica el dominio a establecer en `session_cookie`.
`session.referer_check`		Contiene la subcadena de comprobación de cada "HTTP Referer".
`session.cache_limiter`	`nocache`	Especifica el método de control del caché a usar en las páginas de la sesión (`none`, `nocache`, `private`, `private_no_expire`, `public`).
`session.cache_expire`	`180`	Especifica el tiempo de vida en minutos de las páginas de la sesión que se encuentran en caché.
`session.use_trans_sid`	`0`	Indica si la inclusión del `sid` transparente está activada o no.
`url_rewriter.tags`	`a=href,area=href,frame=src,input=src,form=fakeentry`	Especifica qué etiquetas html serán reescritas para incluir el identificativo de sesión si la inclusión del `sid` transparente está activada.
`session.gc_probability`	`1`	Especifica la probabilidad de que se inicie la rutina `gc` (*garbage collection* –recolección de basura–) en cada petición en porcentaje.
`session.gc_maxlifetime`	`1440`	Especifica el número de segundos tras los cuales los datos se considerarán como "basura" y serán eliminados.
`session.entropy_file`		Indica la ruta a un recurso externo (un archivo) que se usará como fuente adicional de entropía en el proceso de creación de identificativos de sesión.
`session.entropy_length`	`0`	Especifica el número de *bytes* que serán leídos del archivo indicado en la directiva anterior.

CAPÍTULO 10

FICHEROS Y DIRECTORIOS

Como es bien sabido, la información que se necesita guardar de manera permanente está almacenada en ficheros. Por esto, es bastante habitual que las aplicaciones, sean del tipo y del entorno que sean, necesiten realizar operaciones con ficheros del sistema local. Por ejemplo, poner un simple contador de visitas en una página nos obliga a guardar en un fichero el total de visitas recibidas para que, cuando llegue la siguiente, abramos el fichero, leamos el número que contiene y lo modifiquemos convenientemente.

Para poder implementar este tipo de tareas, PHP dispone de funciones predefinidas para la utilización y manejo de ficheros. Gracias a estas funciones, podremos acceder a un fichero que podamos haber creado previamente, podremos consultar los datos que contenga, añadir otros, modificarlos, o bien, borrarlos.

10.1 OPERACIONES CON FICHEROS (NIVEL INTERNO)

PHP nos provee de funciones que nos permiten realizar operaciones típicas de ficheros: abrir, cerrar, leer, escribir, recorrer...

10.1.1 Abrir un fichero

```
fopen (nombre, modoApertura [, ruta_busqueda])
```

Para poder abrir un fichero, disponemos de la función `fopen()`. PHP nos permite abrir ficheros locales o remotos. Que el fichero esté en el disco duro de nuestra máquina o en otra dependerá de cómo esté formado el parámetro `nombre`: si

éste comienza por `http://` o `ftp://`, indicará que es un fichero remoto y que se accederá a él vía el protocolo correspondiente. En caso contrario, se tratará de un fichero que PHP buscará en el sistema de ficheros del servidor, razón por la cual habrá que tener atención a la hora de indicar el camino o ruta de acceso al archivo deseado.

Si PHP no pudiera encontrar el fichero en la ruta especificada en `nombre` y el parámetro `ruta_busqueda` estuviera a 1, entonces PHP lo buscaría además en los directorios que estuvieran definidos en la directiva `include_path` del fichero de configuración `php.ini`.

El segundo parámetro (`modoApertura`) puede tener los siguientes valores expresados en la tabla de abajo:

Valor	Significado
r	Modo de sólo lectura. El puntero se coloca al inicio del fichero.
r+	Modo de lectura y escritura. El puntero se coloca al inicio del fichero.
w	Modo de sólo escritura. Si no existe el fichero, se crea. Si ya existe, se borra todo el contenido.
w+	Modo de lectura y escritura. Si no existe el fichero, se crea. Si ya existe, se borra todo su contenido.
a	Modo de sólo escritura. Si no existe el fichero, se crea. Si ya existe, el puntero se coloca al final del fichero para añadir datos.
a+	Modo de lectura y escritura. Si no existe el fichero, se crea. Si ya existe, el puntero se coloca al final del fichero para añadir datos.

La función `fopen()` devuelve un descriptor de fichero, esto es, el *canal* por el que vamos a poder acceder a él; por tanto, los descriptores serán imprescindibles a la hora de realizar operaciones sobre ficheros tales como lectura, escritura, cerrado, etc. De aquí deducimos que usaremos, tantas variables que almacenen descriptores, como ficheros abiertos simultáneamente vayamos a necesitar.

En el caso de que el fichero no se pudiera abrir por la causa que sea, `fopen()` devolverá `FALSE`. Característica por la cual la forma más habitual de abrir un fichero es:

```
<?php
    if (! $descriptor = fopen ("un_fichero_cualquiera", "r")) {
        echo "*** ERROR: el fichero no se puede abrir. <br>";
    } else {
        echo "podemos acceder al fichero a través de '\$descriptor'. <br>");
    }
?>
```

10.1.2 Cerrar un fichero

```
fclose (descriptor_fichero)
```

Con la función `fclose()` cerramos el fichero que está referenciado por el descriptor que pasamos como argumento. Por supuesto, este descriptor es el devuelto por la función `fopen()` vista anteriormente. `fclose()` devuelve `true`, si el fichero se cierra correctamente, y `false`, si no se ha podido cerrar.

Dado que un fichero abierto consume recursos del sistema, es conveniente cerrar todo aquél que se prevea que no se va a usar más. De todas formas, en el caso de que no se cierre expresamente un fichero, PHP lo hará automáticamente al terminar de ejecutar el *script*.

10.1.3 Lectura desde un fichero

PHP tiene distintas y variadas formas de leer el contenido de un fichero. Lógicamente, usaremos siempre la más conveniente para nuestras necesidades:

- `fgetc (descriptor)`: Devuelve un carácter del fichero referenciado por `descriptor`. Si se ha llegado al final del fichero, devuelve `FALSE`. De aquí, que una forma común de leer un fichero entero sea mediante el bucle:

```
while ($caracter = fgetc($df))
{
....
}
```

- `fgets (descriptor, [total_cars_a_leer ])`: Devuelve una cadena de `total_cars_a_leer-1` caracteres o de menor longitud si se ha encontrado un cambio de línea (que se incluiría en la cadena a devolver) o se ha llegado al final del fichero.
- `fgetss (descriptor, tamaño, etiq_permitidas)`: Es idéntica a `fgets()`, excepto en que purga del texto leído las etiquetas

HTML que haya encontrado. En el parámetro `etiq_permitidas` indicamos las etiquetas que sí queremos obtener.

- `fread (descriptor, total_cars_a_leer)`: Es idéntica a `fgets()`, excepto en que no deja de leer cuando encuentra un cambio de línea y en que devuelve exactamente `total_cars_a_leer` (si los hubiera).
- `fscanf (descriptor, formato [, var1...])`: Según lee la entrada, la procesa de acuerdo con el formato especificado en el segundo parámetro `formato` para asignarla convenientemente en las variables que se indiquen. Sigue el mismo formato que `printf()`.
- `feof (identificador)`: No es una función de lectura propiamente dicha, pero es muy útil cuando leemos ficheros: indica si se ha llegado al final (no quedan más datos por leer) o no.

```
while (!feof($df)) {
.....
}
```

- `file (nombre_fichero [,ruta_busqueda])`: Lee todo el contenido de un fichero y lo devuelve en forma de *array*: una línea en cada posición. Como puede apreciarse, esta función necesita el nombre del fichero y no un descriptor.
- `readfile (nomb_fichero [, ruta_busqueda])`: Lee el contenido de un fichero y lo muestra por la salida estándar. Devuelve el total de caracteres leídos.
- `fpassthru (descriptor)`: Igual a `readfile()`, excepto en que el parámetro que necesita es un descriptor de fichero, y en que lee y muestra a partir de donde se encuentre la posición del puntero de lectura.

10.1.4 Recorrer un fichero

En muchas ocasiones necesitamos colocarnos en una determinada parte del fichero para poder leer o escribir cadenas de texto a partir de esa posición. Para poder movernos a determinados puntos de fichero, PHP nos ofrece las siguientes funciones:

- `rewind (descriptor)`: Sitúa el puntero de lectura/escritura al principio del fichero.
- `fseek (descriptor, desplaz [,desde_donde])`: Desplaza la posición del puntero de lectura/escritura `desplaz` posiciones. El tercer parámetro puede tomar los valores `SEEK_SET`, `SEEK_CUR` y `SEEK_END`, lo que significará que los desplazamientos son relativos al principio del fichero, la posición actual del puntero o al final del fichero (entonces `desplaz` será negativo), respectivamente.
- `ftell (descriptor)`: Devuelve la posición del puntero.

10.1.5 Escritura en un fichero

Para escribir, disponemos de dos funciones: `fwrite()` y `fputs()`. Estas funciones se pueden usar de manera indistinta pues en la práctica son la misma, o sea, puede pensarse que una es un alias de la otra.

```
fwrite(descript, cadena [, total_cars]);

fputs (descript, cadena [, total_cars]);
```

Ambas funciones escriben la cadena (completa) pasada como parámetro. Si se hace uso del tercer parámetro, sólo se escribirán los `total_cars` indicados. Devuelve el total de caracteres escritos o `FALSE` en caso de producirse algún error.

En general, dado que son muy lentas todas las operaciones en las que está implicado un acceso al sistema de ficheros (al disco duro, para ser más exactos), se recurre a la escritura mediante *buffers* para hacerlas más eficaces. Esto quiere decir que la escritura no se hace efectiva en el fichero hasta que no se llena esta memoria intermedia. Por defecto, el *buffer* de escritura de PHP tiene un tamaño de 8 Kb, pero, si en algún momento necesitáramos cambiar esta cifra por otra, podríamos hacerlo con la función `set_file_buffer()`:

```
set_file_buffer (descript, buffer_tamaño);
```

> **NOTA**: En el anexo A del libro aparece un ejemplo de aplicación que nos muestra el uso de estas funciones.

10.2 INFORMACIÓN SOBRE FICHEROS

- `file_exists(nombr_fich)`: Comprueba si el fichero `nombr_fich` existe. Devuelve `TRUE` si `nombr_fich` existe; en caso contrario, devuelve `FALSE`. Esta función nos ahorrará desagradables mensajes de error si antes de abrir un fichero hacemos uso de ella.
- `is_file (fichero)`: Devuelve verdadero si el tipo del fichero es un fichero normal. En caso contrario, devuelve `FALSE`.
- `is_dir (fichero)`: Devuelve verdadero si el nombre del fichero pasado como argumento es un directorio. En caso de que no exista o no lo sea, devuelve `FALSE`.
- `is_executable (fichero)`: Devuelve verdadero si el nombre del fichero pasado como argumento tiene permisos de ejecución (Unix) o es un fichero ejecutable (extensión `exe`, `com` o `bat`).

- `is_readable (fichero)`: Devuelve verdadero si el fichero tiene permiso de lectura. Si no tiene dicho permiso o no existe, devuelve `FALSE`.
- `is_writeable (fichero)`: Devuelve verdadero si el fichero tiene permiso de escritura. Si no tiene dicho permiso o no existe, devuelve `FALSE`.
- `filemtime (fichero)`: Devuelve el instante (*timestamp*) de la última modificación hecha sobre el contenido del fichero.
- `filesize (fichero)`: Devuelve un número entero que indica el tamaño en bytes del fichero pasado como parámetro. En caso de error, devuelve `FALSE`.

Hay que resaltar que un fichero de texto tendrá tamaños diferentes en Windows y en Unix, siendo ligeramente inferior en este último: los cambios de línea se indican con dos caracteres en Windows (*carriage return* y *line feed*, `CR` y `LF`, respectivamente), mientras que en Unix sólo se usa uno (*line feed*).

10.3 OPERACIONES CON FICHEROS (NIVEL EXTERNO)

Con los ficheros podemos realizar operaciones para simplificar nuestros programas cuando utilizamos ficheros. Veamos algunas de estas operaciones:

- `copy (origen, destino)`: Copia un fichero destino. Si no ha habido ningún error, devuelve `TRUE`.
- `rename (nom_original, nom_final)` : Cambia el nombre a un fichero o directorio.
- `unlink (nom_fichero)`: Borra un fichero.
- `link (fich_original, fich_enlazado)`: Crea un enlace duro (válido sólo en Unix).
- `tempnam (directorio, prefijo)`: Crea un fichero único (no existente) en el directorio que se le indica como primer parámetro. Si el directorio no existiese o fuese la cadena vacía, el fichero se creará en el directorio temporal del sistema. El nombre generado empezará por lo que se pase como segundo parámetro. Devuelve el nombre del fichero creado. Es muy útil cuando necesitamos crear ficheros auxiliares en entornos multiusuario.
- `touch (fichero [, instante])`: Modifica las fechas de último acceso y modificación del fichero. Por defecto, esto es, sin el segundo parámetro, toma la fecha actual. Si no existe el fichero, lo crea.

10.4 MANEJO DE DIRECTORIOS

Al igual que tenemos funciones específicas para ficheros, también tenemos a nuestra disposición una serie de funciones predefinidas para el manejo y utilización de directorios. Con ellas podremos crear, borrar, leer las entradas que tiene un directorio, averiguar en qué directorio nos encontramos, cambiarnos a otro, etc.

- `opendir (nombre)`: Abre el directorio pasado como parámetro. Devuelve un descriptor de directorio.

Si se están escribiendo *scripts* para plataformas Windows, un par de consejos en beneficio de la portabilidad: usar el carácter "/" como separador de directorios (en lugar de "\") y no hacer referencia a la unidad de almacenamiento (ejemplos, `C:`, `A:`, etc.):

```
<?php
opendir("/mis documentos/libro ");
?>
```

- `readdir (descriptor)`: Lee una entrada del directorio a partir del descriptor devuelto por `opendir()`.

- `closedir (descriptor)`: Cierra el directorio indicador por el descriptor.

Aplicamos estas funciones para conseguir un sencillo *navegador* de archivos que nos permitirá movernos por la estructura de directorios del disco duro del servidor:

```
<?

if (isset($_GET['nuevo_dir']))
  $nuevo_dir=$_GET['nuevo_dir'];
else {                          // es la primera vez que se ejecuta el script
  $nuevo_dir=getcwd();
  if (isset($_SERVER["WINDIR"])); {
    $nuevo_dir=substr($nuevo_dir, 2); // quitamos las 2 letras de la unidad
    $nuevo_dir=strtr($nuevo_dir, "\\", "/");
  }
}

echo "<h2>CARPETA: <i>$nuevo_dir</i></h2>";

if (!$df_dir = opendir($nuevo_dir))
  die("<h3>*** ERROR: no se ha podido entrar en ($nuevo_dir)</h3>");
```

```
while (($item = readdir($df_dir)) !== false) {
  if ($item == ".") continue;

  if (es_directorio($nuevo_dir, $item)) {
    pon_url($nuevo_dir, $item);
  } else
    echo "<img src='/icons/generic.gif'>$item<br>";
}
closedir($df_dir);

function pon_url($un_dir, $un_item)
{
  $ini_etiq="<a href='$(_SERVER["PHP_SELF"])?nuevo_dir";
  $fin_etiq="</a><br>";

if ($un_item == "..") {
    if (substr_count($un_dir, "/") >= 1) {
      $un_dir=strtr(dirname($un_dir), "\\", "/"); // Win devuelve '/'
      echo "<img src='/icons/back.gif'>$ini_etiq=$un_dir'><font
size=+2>..</font>$fin_etiq";
    }
  } else {
    if ($un_dir=="/")
      echo "<img
src='/icons/folder.gif'>$ini_etiq=/$un_item'>$un_item$fin_etiq";
    else
      echo "<img
src='/icons/folder.gif'>$ini_etiq=$un_dir/$un_item'>$un_item$fin_etiq";
  }
}

function es_directorio($un_dir, $un_item)
{ // en windows no puede haber un nombre de directorio con dos '/': '//' mal
  if ($un_dir == '/')
    $fich_a_preguntar="/$un_item";
  else
    $fich_a_preguntar="$un_dir/$un_item";
  return (is_dir("$fich_a_preguntar"));
}
?>
```

El aspecto que tendría lo vemos en la figura de la página siguiente. Sólo los directorios tienen asociado un enlace (además de tener asociado un icono que representa una carpeta): para movernos por las carpetas de nuestro sistema de ficheros, bastará con hacer clic en ellos. Para ascender al directorio padre, seleccionaremos el directorio cuyo nombre es "..".

- `rewinddir (descriptor)`: Sitúa el puntero de lectura al principio del directorio.
- `getcwd()`: Devuelve el directorio actual de trabajo (también llamado *activo*) del *script*.
- `chdir (nuevo_dir)`: Establece `nuevo_dir` como nuevo directorio de trabajo. Devuelve `FALSE` en caso de error.
- `chroot(nuevoDirRaiz)`: Cambia el directorio raíz del proceso actual al directorio que viene indicado en el parámetro que se pasa a la función. Se suele usar para limitar el acceso a determinados recursos. No funciona en Windows. Sólo funciona correctamente cuando se usa CGI.

10.5 OPERACIONES CON DIRECTORIOS

Veamos algunas funciones básicas para trabajar con directorios:

- `mkdir (nombre_dir, permisos)`: Creamos el directorio `nombre_dir` con los permisos indicados en el segundo parámetro
- `rmdir (nombre_dir)`: Borra el directorio `nombre_dir`.

Dado que la operación de crear un directorio es atómica en cualquier sistema operativo, pues así se garantiza la consistencia de la estructura de árbol de un sistema de ficheros, podemos crear una ***región crítica*** haciendo uso de las funciones de creación y borrado de directorios, esto es, podemos garantizar el acceso *ordenado* de múltiples procesos a un recurso compartido[1].

[1] La función `flock()` parecería la más adecuada para esta tarea; sin embargo, no funciona correctamente en todas las implementaciones de servidores Web, ni en todos los sistemas operativos.

Este mecanismo se basa en el uso de *semáforos*: para que un proceso pueda acceder a un recurso, necesita previamente consultar si tiene *permiso*. Esta autorización se implementa de manera muy sencilla: si existe un directorio (de nombre convenido previamente), significará que hay otro proceso haciendo uso del recurso; en caso contrario, creará el directorio y accederá a dicho recurso. Una vez satisfechas las necesidades de uso del recurso, borrará el directorio.

```
<html>
<head>
<title>
prueba de region critica
</title>
</head>
<?php

$cerrojo='/tmp/cerrojo';
function crea_cerrojo()
{
  while (!@mkdir($GLOBALS['cerrojo'], 0700))
  {
    echo "me duermo, estoy esperando....<hr><br>\n";
    sleep(1);
  }
}

function libera_cerrojo()
{
  rmdir($GLOBALS['cerrojo']);
}
echo "intento crear el cerrojo<br>";
crea_cerrojo();
echo "ahora hago lo que quiera... ;)<br>";
sleep(10);

echo "libero el cerrojo :)<br>";
libera_cerrojo();


?>
```

Por ejemplo, podría implementarse un contador de accesos: bastaría con incluir el código de abajo entre las llamadas a las rutinas `crea_cerrojo()` y `libera_cerrojo()`:

```
$fich_contador='/tmp/contador';

$num_visitas=file($fich_contador);
$visita_mas_1=rtrim($num_visitas[0]);
$visita_mas_1++;
echo "Eres el visitante numero: <b>$visita_mas_1</b><br>";

$df=fopen($fich_contador, "w");
fwrite($df, "$visita_mas_1\n");
fclose($df);
```

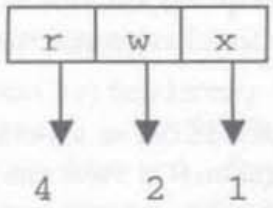
10.6 CONCEPTO DE PERMISOS Y DUEÑOS EN UNIX

Todas aquellas funciones que hacen referencia a usuarios, grupos y permisos sólo tienen aplicación en plataformas UNIX, razón ésta por la que mencionamos sus funciones relacionadas en un apartado separado.

Cada fichero o directorio en Unix tiene tres tipos de permisos: lectura, escritura y ejecución. El sistema operativo, en función de si se trata de un fichero o directorio, establece una semántica de operaciones, es decir, determinará para cada tipo de permiso lo siguiente:

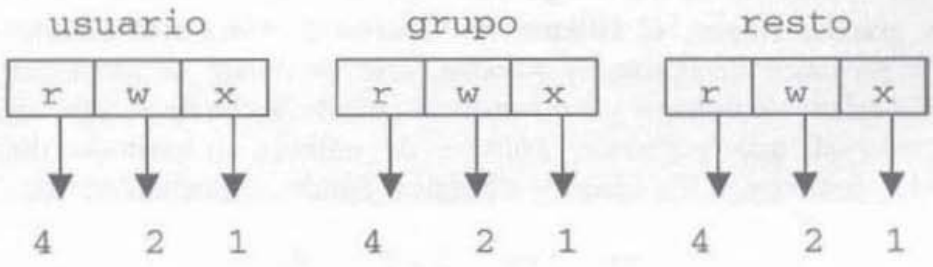
	lectura ("r")	**escritura ("w")**	**ejecución ("x")**
fichero	consulta	modificación	ejecución (independiente del contenido)
directorio	listado del contenido del directorio	creación y borrado de ficheros en el directorio	bajar al directorio

Por otra parte, para operar con los permisos, cada uno de éstos tiene una valoración o *peso*:



De manera que el total de permisos estará determinado por la suma de *pesos* que tenga. Así, por ejemplo, un *peso* igual a 7 equivaldrá a tener todos los permisos activados; un valor de 5 indicará que sólo se puede leer (4) y ejecutar (1); un 6 equivaldrá a poder leer (4) y modificar (2); etc.

Además de los permisos, se diferencian tres entidades: usuario, grupo y el resto (esto es, quien no es ni el usuario ni otro usuario que pertenezca al grupo). Por ello, los permisos siempre se darán en tríos.



Por ejemplo, el valor 640 indicará que el usuario tiene permiso de lectura y escritura, que los integrantes del grupo tienen permiso de lectura y que el resto de usuarios no tiene permiso alguno.

El superusuario no se ve afectado por los permisos de un fichero.

- `fileperms (fichero)`: Devuelve los permisos del nombre del fichero pasado como parámetro.
- `fileowner (fich_o_dir)`: Dicha función devuelve un número entero correspondiente al identificador de usuario del dueño del fichero o directorio pasado como parámetro, esto es, su UID.
- `filegroup (fich_o_dir)`: Devuelve un número entero correspondiente al identificador de grupo, su GID, del nombre del fichero pasado como parámetro.
- `chown (fich_o_direct, usuario)`: Cambia el propietario de un fichero o directorio. Sólo el superusuario (*root*) puede cambiar el dueño de un fichero. Devuelve verdadero si se ha realizado el cambio correctamente.
- `chgrp (fich_o_direct, grupo)`: Cambia el grupo al que pertenece un fichero o directorio. Para que tenga éxito esta función, se ha de cumplir, o bien, que el usuario sea dueño del fichero y que, además, pertenezca al grupo al que quiere cambiar el fichero, o bien, que el usuario que realiza esta operación sea el superusuario (`root`).
- `chmod (fichero, permisos)`: Cambia los permisos de un fichero. El segundo parámetro `permisos` se espera que sea un número octal; por esta razón, prefijaremos con un 0 el valor del permiso deseado. Ejemplo:

```
chmod ("/mysql/fich.txt", 0755);
```

- `umask (mascara)`: Establece qué permisos no tendrá un fichero o directorio al crearse. Al igual que en `chmod()` el parámetro `mascara` hay que expresarlo como un valor octal: empezará por 0.

10.7 INFORMACIÓN DE FICHEROS Y DIRECTORIOS EN UNIX

A grandes rasgos, el sistema de ficheros de Unix está basado en unas estructuras de datos denominadas ***i-nodos***, que es donde se almacena toda la información relativa a ficheros y/o directorios: tamaño, fecha de acceso, quién es el dueño, grupo al que pertenece, número de enlaces, dispositivo donde está almacenado, punteros a los bloques de datos donde se encuentra, etc. Toda la

información de los ficheros locales almacenada en el *i-nodo* puede obtenerse fácilmente usando la función `stat()`:

```
array stat (string nombre_fichero);
```

Para ello, le pasamos como parámetro la ruta del fichero y lo que obtendremos será un *array* con la siguiente información relativa al fichero:

- dispositivo[2]
- i-nodo
- permisos
- número de enlaces
- dueño del fichero (UID)
- grupo (GID)
- tipo de dispositivo
- tamaño del fichero en octetos
- fecha de último acceso
- fecha de última modificación
- fecha de último cambio
- tamaño del bloque
- número de bloques asignados

El *array* devuelto por `stat()` puede ser accedido bien a través de claves (*array* asociativo), en cuyo caso éstas son: `dev`, `ino`, `mode`, `nlink`, `uid`, `gid`, `rdev`, `blksize`, `size`, `atime`, `mtime`, `ctime`, `blocks`; o bien, como *array* numérico, en cuyo caso tendríamos los índices del 0 al 12 que se corresponden con las claves anteriores.

Como podrá apreciarse, las funciones siguientes están en realidad basadas en la información contenida en el *i-nodo*:

- `fileinode (fichero)`: Devuelve el número de *i-nodo* correspondiente al fichero que se pasa como parámetro. Si hay algún error, devuelve `FALSE`.

  Vemos con el ejemplo que, para averiguar el número del i-nodo, podemos llamar tanto a `fileinode()` como a `stat()`:

```
$f="entrada.txt";
$val=stat($f);
$el_inodo=fileinode($f);
echo "inodo de $f: ${val['ino']}==$el_inodo)";
```

[2] En Windows `a:` se corresponde con el `0`; `b:`, con el `1`; etc.

- `fileatime (fichero)`: Devuelve el instante (*timestamp*) del último acceso hecho al fichero. Si hay algún error, devuelve `FALSE`.

- `filectime (fichero)`: Devuelve el instante del último cambio que se ha hecho en alguno de los campos del i-nodo (número de enlaces, dueño, permisos, etc.) correspondiente al fichero pasado como parámetro. Ojo, esta función **no** devuelve la fecha de creación del fichero.

- `filetype (fichero)`: Devuelve una cadena indicando el tipo de fichero que se ha pasado como parámetro:
  - `file`: fichero (también en Windows).
  - `dir`: directorio (también en Windows).
  - `link`: enlace simbólico.
  - `fifo`: tubería (*pipe*) con nombre.
  - `char`: dispositivo carácter.
  - `block`: dispositivo especial de bloque.
  - `unknown`: tipo sin determinar.

- `is_link (fichero)`: Devuelve verdadero si el tipo del fichero es un enlace simbólico. En caso contrario, devuelve `FALSE`.

- `readlink (fichero)`: Devuelve el nombre del fichero apuntado en el fichero simbólico pasado como parámetro.

Como ya se ha comentado anteriormente, las operaciones más lentas que puede realizar un ordenador son los accesos al disco duro. Si lo único que queremos es pedir información sobre un fichero (por ejemplo, necesitamos saber quién es el dueño), una opción de implementación para aumentar el rendimiento de estas operaciones consiste en guardar en memoria el resultado de la primera petición. Si, más tarde, se necesitara preguntar algo más sobre el mismo fichero, el sistema devolvería lo que tiene en memoria en lugar de volver a hacer un acceso al disco. En definitiva, lo que se hace es *cachear* dicha información.

Por tanto, cuando usamos cualquiera de las funciones `stat()`, `file_exists()`, `is_writeable()`, `is_readable()`, `is_executable()`, `is_file()`, `is_dir()`, `is_link()`, `filectime()`, `fileatime()`, `filemtime()`, `fileinode()`, `filesize()`, `filetype()`, `filegroup()`, `fileowner()` y `fileperms()` para preguntar sobre un determinado fichero, PHP

guarda en memoria caché el resultado obtenido (que es la información completa que hay en el i-nodo). La siguiente vez que se utiliza cualquiera de las funciones enumeradas arriba, PHP usará lo que haya guardado en memoria y no accederá al sistema de ficheros (siendo así mucho más rápida la respuesta).

Esto tiene dos efectos: consumir recursos del sistema y (lo más importante) podemos no llegar a tener información real del fichero porque otro proceso haya modificado cualquiera de sus propiedades o incluso lo haya borrado. Por esta razón, PHP cuenta con la función:

```
void clearstatcache (void);
```

La cual lo que hace es borrar o liberar esta memoria caché donde se almacena la información que hemos pedido sobre los ficheros que sean.

Pero los datos contenidos en la caché sólo son válidos durante el tiempo de ejecución del programa. Una vez que se vuelva a cargar, estos datos se reinicializan.

10.8 OTRAS FUNCIONES

En este apartado incluimos, a modo de miscelánea, funciones que de alguna manera están relacionadas con los ficheros:

- `basename (ruta)`: Ésta no es una operación de ficheros propiamente dicha, sino, más bien, de cadenas de caracteres ya que tanto lo que pasamos a la función, como lo que obtenemos de ella, es un *string*: dada una cadena de caracteres que definen una ruta o posición de un fichero en el disco duro (separados los directorios por el carácter "/" o "\"), `basename()` nos devuelve los caracteres que están después del último carácter que usamos como separador. Por ejemplo, si le aplicamos esta función a la cadena `/hola/que/tal`, obtendremos `tal`

- `dirname (ruta)`: Hace el inverso de la función `basename()`, es decir, devuelve la ruta de directorios y subdirectorios donde está ubicado el fichero: los directorios por los que hay que pasar para encontrarlo; aplicado a "`/hola/que/tal`", obtendremos "`/hola/que`".

- `disk_free_space(dir)`: Esta función toma una ruta a un directorio y devuelve el espacio libre en octetos (*bytes*) disponibles en el sistema de ficheros que contiene al directorio.

- `disk_total_space(dir)`: Toma una ruta a un directorio y devuelve el tamaño del sistema de ficheros donde está situado dicho directorio.

10.9 TRANSFERENCIA DE FICHEROS ENTRE CLIENTE Y SERVIDOR

Por último, las operaciones en las que están involucrados ficheros que nos quedan por ver son aquéllas en que el cliente le envía un fichero al servidor y a la inversa: el servidor le envía un fichero al cliente.

10.9.1 *Subir* ficheros al servidor

No es raro encontrarse con páginas que nos permiten transmitir ficheros desde el cliente al servidor, es decir, lo que vulgarmente se conoce como *subir* ficheros o hacer *uploads*. Casos como éste lo encontramos, por ejemplo, en páginas donde se ha de rellenar un *curriculum vitae* y uno de los campos a *rellenar* es incluir una fotografía de la persona interesada. También en los clientes de correo vemos esta posibilidad: en la página de componer mensajes lo normal es poder adjuntar archivos para enviarlos junto con la redacción del mensaje.

El problema de subir ficheros al servidor se divide en dos escenarios: uno es la página HTML donde el usuario indicará (a través de etiquetas) qué fichero de su disco local quiere subir; el otro se encuentra en la parte programática: gracias a la página HTML (en realidad, gracias al protocolo HTTP), el fichero ya ha sido transferido al servidor y ahora lo que tenemos que hacer es poder acceder a él para realizar las tareas que consideremos oportunas.

En la página HTML sólo necesitamos dos etiquetas: la etiqueta `<input>` (en realidad, tantas como ficheros queramos subir), gracias a la cual el navegador mostrará una cajita, y un botón de navegación (*browsing*), para que el usuario seleccione el fichero deseado:

```
<input file='file' name='nombre_fichero_a_subir'>
```

La segunda etiqueta necesaria es la que nos permite construir un formulario (`<form>`), pero con una serie de cláusulas adicionales:

```
<form action='subir_fich.php'
      method="post"
      enctype="multipart/form-data">
```

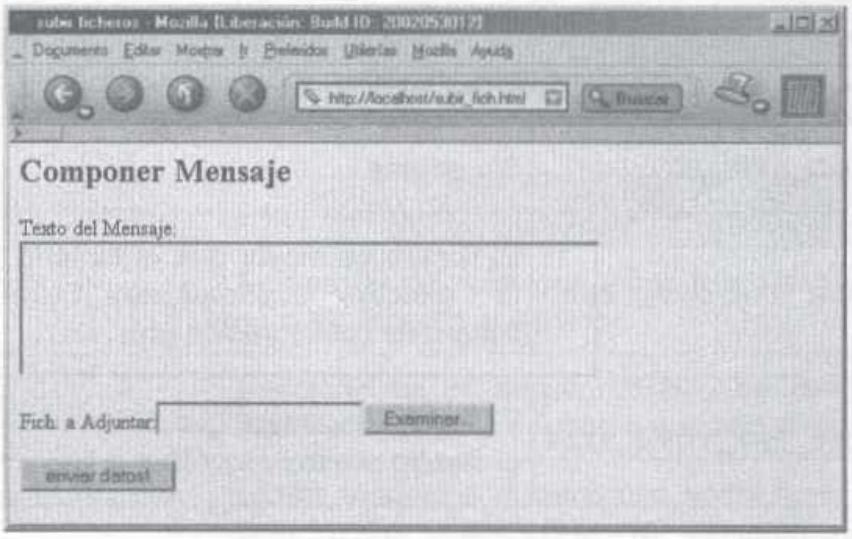
Es decir, tenemos que indicar que el método de transferencia de los datos del formulario es mediante `POST` (viajan junto a la página), y que viajarán codificados como `multipart/form-data`.

Opcionalmente, se puede expresar el tamaño máximo que podrá tener un fichero para poder ser enviado con un campo de datos oculto (tendríamos entonces tres etiquetas) al que hay que identificarlo como `max_file_size` (si no se indica, se tomará la directiva `upload_max_filesize`, comentada más abajo). Por ejemplo, limitamos el tamaño máximo a 1.000 caracteres:

```
<input type='hidden' name='max_file_size' value='1000'>
```

Veamos un sencillo ejemplo de página HTML desde la que se podría enviar un mensaje junto con un fichero adjunto:

```
<HTML>
<HEAD>
<TITLE> subir ficheros </TITLE>
</HEAD>
<BODY>
  <h2>Componer Mensaje</h2>
  <FORM method='post' action='subir_fich.php'
        enctype='multipart/form-data'>
  <INPUT type='hidden' name='max_file_size' value='250000'>

  Texto del Mensaje:<br>
  <textarea cols='50' rows='5' name='texto'>
  </textarea>
  <br><br>Fich. a Adjuntar:<input type='file' name='el_fich_adjunto'>
  <br><br><input type='submit' value='enviar datos!'>
  </FORM>
</BODY>
</HTML>
```



Después de pulsar el botón de enviar, el fichero se encuentra en el servidor. Evidentemente, el fichero transferido no está en un directorio al azar: se almacena en

el directorio que se especifica en el fichero `php.ini` a través de la directiva `upload_tmp_dir`.

Para poder trabajar con los ficheros así transmitidos, PHP nos proporciona el *array* asociativo `$_FILES`. Este *array* tiene dos dimensiones: en la primera columna se indican los ficheros que han sido subidos usando como claves los identificadores que usamos en la etiqueta `<input>`, mientras que las claves de la segunda columna se corresponderán con las características de cada fichero:

Clave	Descripción
`name`	Nombre del fichero transferido.
`tmp_name`	Nombre del fichero temporal donde está guardado el fichero recién subido. Si no se pudiera haber cumplido esta operación (por ejemplo, el fichero es demasiado grande), entonces obtendremos `none`.
`size`	Tamaño en caracteres del fichero.
`type`	El tipo MIME del fichero (indicado por el cliente).
`error`	Código de error devuelto: indica cómo ha ido la transferencia[3].

Por su parte, los códigos de error que podremos encontrarnos son los que se muestran en la tabla de abajo:

Valor	Constante	Significado
0	`UPLOAD_ERR_OK`	Sin errores.
1	`UPLOAD_ERR_INI_SIZE`	El fichero es mayor que el tamaño indicado en la directiva `upload_max_filesize` del fichero de configuración `php.ini`.
2	`UPLOAD_ERR_FORM_SIZE`	El fichero es mayor que el tamaño indicado en el campo oculto `max_file_size`.

[3] Esta clave ha sido incluida a partir de la versión 4.2.0.

Valor	Constante	Significado
3	UPLOAD_ERR_PARTIAL	El fichero ha sido parcialmente transferido.
4	UPLOAD_ERR_NO_FILE	No se ha transferido el fichero.

También disponemos de un par de funciones que son muy útiles para trabajar con estos ficheros. Una nos vale para saber si el fichero ha sido *subido*: `is_uploaded_file($_FILES['identif_fich']['tmp_name']);` y la otra nos sirve para renombrar o mover el fichero temporal a otro que será permanente: `move_uploaded_file(nom_fich_original, destino).`

Veamos un ejemplo de cuál sería el programa que trataría los datos introducidos en la página HTML anterior:

```
<?
foreach ($_FILES['f_adjunto'] as $clave => $valor)
  echo "\$_FILES[$clave]: ($valor)<br>";

if (!is_uploaded_file($_FILES['f_adjunto']['tmp_name']))
{
  $error=$_FILES['f_adjunto']['error'];
  die("<h3>** ERROR: fichero no transferido: $error </h3>");
}

if ($_FILES['f_adjunto']['type'] != 'application/x-zip-compressed')
  echo "<h3>** ERROR: el fichero enviado no está comprimido</h3>";
?>
```

10.9.2 Directivas de `PHP.INI` involucradas

Por último, hay que considerar tres directivas del fichero de configuración `php.ini` que nos determinarán el comportamiento del intérprete PHP en lo que a transferencia de ficheros al servidor se refiere:

Directiva	Descripción
`file_uploads`	En función del valor que tenga (`on`, `off`), permitirá o no *subir* ficheros al servidor.
`upload_tmp_dir`	Directorio en el servidor donde se almacenan de manera temporal los ficheros que han sido *subidos*.
`upload_max_filesize`	Limita el tamaño máximo que podrá tener un fichero para poder transferirse al servidor. Este valor prevalece sobre el valor dado en el campo oculto identificado como `max_file_size`.

10.9.3 *Bajar* ficheros del servidor

La transferencia de ficheros entre cliente y servidor también puede hacerse en sentido contrario al visto hasta ahora: desde el servidor podemos enviar un fichero al cliente forzándo a éste a que lo almacene en su disco duro. Por ejemplo, queremos enviarle un fichero gráfico (por ejemplo, un fichero de tipo `jpg`) que no queremos que lo visualice en el navegador, sino que lo guarde automáticamente. Para realizar esta transferencia, enviamos al cliente, gracias a la función `header()`, los siguientes comandos del protocolo HTTP:

```
<?php
$nom_fich="saco";
header("Content-Disposition: attachment; filename='$nom_fich'");
header("Content-Length: ", filesize($nom_fich));
header("Content-Type: application/octet-stream; name='$nom_fich'");
header("Content-Description: mensaje que quiera");  // opcional
?>
```

10.10 CONTROL DE LA SALIDA ESTÁNDAR

Hemos visto que, por defecto, la salida generada por cualquiera de las funciones `readfile()`, `fpassthru()`, `echo()`, `print()`, etc., se envía directamente a la salida estándar, es decir, la recibe directamente el cliente. Por ello, el resultado de estas funciones no podemos asignarlo a una variable o pasarlo como parámetro a otra.

PHP nos ofrece toda una *familia* de funciones para poder controlar todo aquello que se dirige a la salida estándar. En la práctica, lo que hacen es trabajar con el contenido de una serie de memorias intermedias (*buffers*):

- `ob_start()`: Activa el almacenamiento en memoria intermedia, por lo tanto, cualquier texto enviado a la salida estándar no llegará al navegador del cliente.
- `ob_end_flush()`: Envía el contenido del *buffer* a la salida estándar y luego lo vacía. Además, da por terminado el almacenamiento intermedio.
- `ob_end_clean()`: Borra el contenido del *buffer* y da por terminado el *buffering*.
- `ob_clean()`: Borra el contenido del *buffer*.
- `ob_get_contents()`: Devuelve el contenido de la memoria intermedia.
- `ob_get_length()`: Devuelve el número de octetos de lo que hay en el *buffer*.
- `ob_flush()`: Envía el contenido del *buffer* a la salida estándar. Esta función no vacía el *buffer*.

En el ejemplo que proponemos queremos enviar una imagen en una página HTML pero, en lugar de hacerlo como siempre, nos vamos a ahorrar una conexión HTTP pues vamos a empotrar la imagen en la propia página haciendo uso del esquema:

```
<img src="_en_base_64">
```

```
<?
echo "<h2>Ejemplo de control de la salida</h2>";

ob_start();
readfile("world2.png");
$la_img=ob_get_contents();
ob_end_clean();

$en_base64=chunk_split(base64_encode($la_img));
echo "<img src='$en_base64' width='20'>";

?>
```

Vemos en el código de ejemplo que podemos capturar la salida de la función `readfile()`, recuperarla del buffer, tratarla como mejor nos convenga y enviarla cuando nos parezca.

CAPÍTULO 11

BASES DE DATOS

Tal como podemos ver en el anexo A donde hemos desarrollado una agenda, utilizamos un fichero como estructura para almacenar la información. Las teóricas ventajas de esta implementación se basan en la rapidez con que podemos programar las rutinas de acceso a los registros (insertar, eliminar, modificar, etc.) e, incluso, que en cualquier momento podemos ver o modificar dicho fichero con cualquier editor de texto. Sin embargo, usar ficheros de texto tiene muchos inconvenientes. El más evidente está en que, conforme crece el volumen de información almacenado, la gestión de los datos se vuelve cada vez más lenta pues los accesos a ellos son secuenciales, esto es, siempre desde el principio del fichero. Más razones de peso para **no** usar ficheros de texto como elementos de almacenamiento de información: los accesos concurrentes son muy poco eficientes, resulta muy complicado (imposible en ocasiones) limitar los accesos de manera selectiva (en función de usuarios y en función de determinadas áreas), la generación de inconsistencias es muy fácil que aparezcan y muy difícil de gestionarlas, etc., etc. Por lo tanto, si necesitamos que el rendimiento sea óptimo, tendremos que usar un gestor de base de datos.

11.1 BASES DE DATOS RELACIONALES

Existen muchos tipos de base de datos en función del modo en que almacenan y acceden a la información que guardan: relacional, jerárquica, en red, orientada a objetos, etc. Ejemplos de gestores de bases de datos relacionales o RDBMS (*Relational Database Management System*) hay muchos: MySQL, SQLite, Oracle, Informix, SyBase, Microsoft SQL Server, PostGres, mSQL, etc.

Básicamente, un gestor de base de datos relacional almacena los datos en **tablas**, cada una de las cuales está formada por **filas** (o registros), y éstas, a su vez, están formadas por **columnas** (o campos). Antes de definir una tabla, hay que **normalizarla**, proceso que consiste en evitar redundancias, es decir, que la información esté duplicada ya que, si hubiera que cambiar un dato que estuviera repetido, habría que cambiarlo varias veces.

Por otra parte, el acceso a los gestores de bases de datos se hace a través del lenguaje SQL (*Structured Query Language*), del cual no haremos una descripción exhaustiva dado que no es el objetivo de este libro dar una visión profunda de este lenguaje. Por tanto, presuponemos una cierta familiaridad del lector con el entorno de trabajo con bases de datos relacionales y su lenguaje de consulta.

11.2 MySQL

De entre todos los gestores anteriormente mencionados, la elección de MySQL como gestor de base de datos radica en que es gratuito tanto para usos privados, como comerciales (sólo hay que pagar en el caso de que se desarrolle un producto comercial que esté basado en MySQL), en su disponibilidad para distintos sistemas operativos (la mayor parte de los *sabores* Unix, Windows 9x/NT/2000/XP, OS/2, etc.), en que es capaz de trabajar con millones de registros y porque, además, es muy rápido y no necesita grandes recursos de máquina.

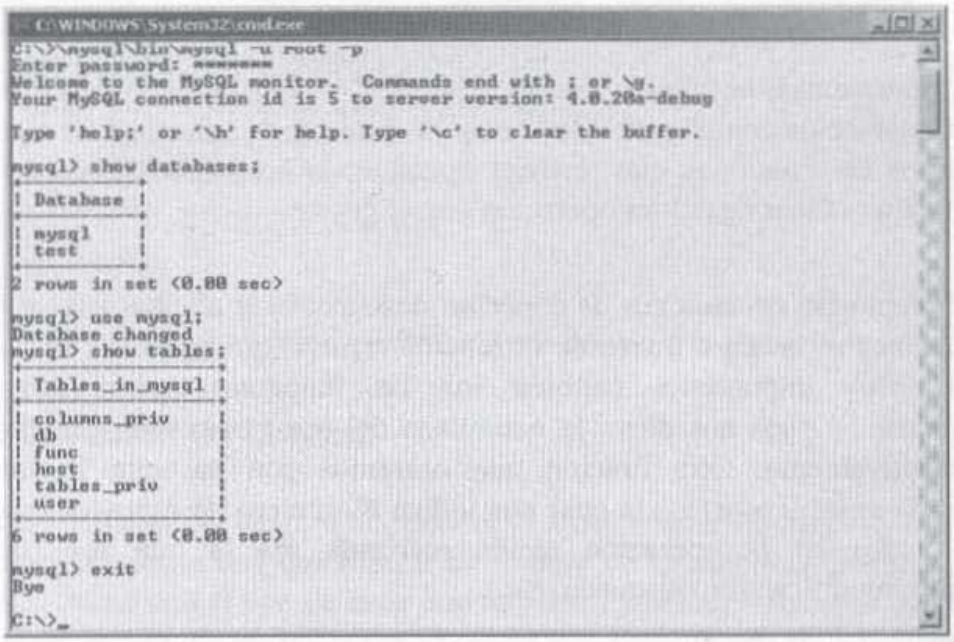
El nuevo sistema de almacenamiento para las bases de datos gestionadas por MySQL es *InnoDB*, desarrollado y mantenido por *Heikki Tuuri*. Este sistema de almacenamiento proporciona bloqueos de columnas, lecturas no bloqueantes, múltiples niveles de aislamiento, integridad referencial, recuperación automática y todas las garantías ACID (*Atomic, Consistent, Isolated, Durable*).

11.2.1 Conexión con el gestor de la base de datos

Como ya es bien sabido, las aplicaciones que siguen la arquitectura cliente-servidor (Web, correo, dns, ftp, *news*, etc.) basan su funcionamiento en dos extremos: un servidor que se mantiene a la *escucha* de peticiones en un puerto determinado y, en el otro, los clientes que, cuando quieren contactar con el servidor, realizan conexiones a ese puerto.

MySQL sigue esta misma arquitectura y, por tanto, para poder realizar operaciones, es necesario tener arrancado el programa servidor tal como vimos en el capítulo dedicado a la instalación. Por defecto, el servidor de MySQL escucha peticiones en el puerto TCP 3306. En la misma distribución donde viene el servidor

encontramos un programa cliente (`mysql`) que nos permite hacer la conexión y trabajar interactivamente con el gestor:



Al comando le indicamos el usuario con el que nos vamos a conectar (`-u root`) y que la clave la introduciremos mediante el teclado (`-p`). Las operaciones que hemos ejecutado en la pantalla anterior son las correspondientes a ver todas las bases de datos que contiene MySQL, poner activa la base de datos `mysql` y ver las tablas que contiene ésta. Para salirnos de la sesión, tecleamos `exit`.

También podemos hacer la conexión al servidor a través de una serie de funciones implementadas en PHP, puesto que los comandos que podemos introducir en la sesión con MySQL tienen su equivalente como función de PHP (todas estas funciones empiezan con el prefijo `mysql_`). El ejemplo de sesión anterior lo podemos hacer desde un *script* PHP exactamente igual con las funciones `mysql_connect()`, `mysql_select_db()`, `mysql_list_tables()` y `mysql_close()`:

```
<?
$dbd = mysql_connect('localhost', 'root', 'root');

$res = mysql_list_dbs($dbd);                    // show databases
while ($fila = mysql_fetch_array($res, MYSQL_NUM))
    echo "Base de datos: $fila[0]<br>";

$base_datos='mysql';
mysql_select_db($base_datos, $dbd);             // use mysql

$res = mysql_list_tables("$base_datos");        // show tables

echo "<br>Tablas en <b>$base_datos</b>:<ul>";
while ($fila = mysql_fetch_array($res, MYSQL_NUM))
```

```
  echo "<li>$fila[0]<br>";
echo "</ul>";

mysql_close($dbd);                          // exit
?>
```

Como vemos, la función `mysql_connect()` devuelve un descriptor de la conexión establecida con el gestor de la base de datos. Este descriptor es equivalente al usado por las funciones que realizan operaciones con ficheros, es decir, será necesario para realizar cualquier operación con el gestor.

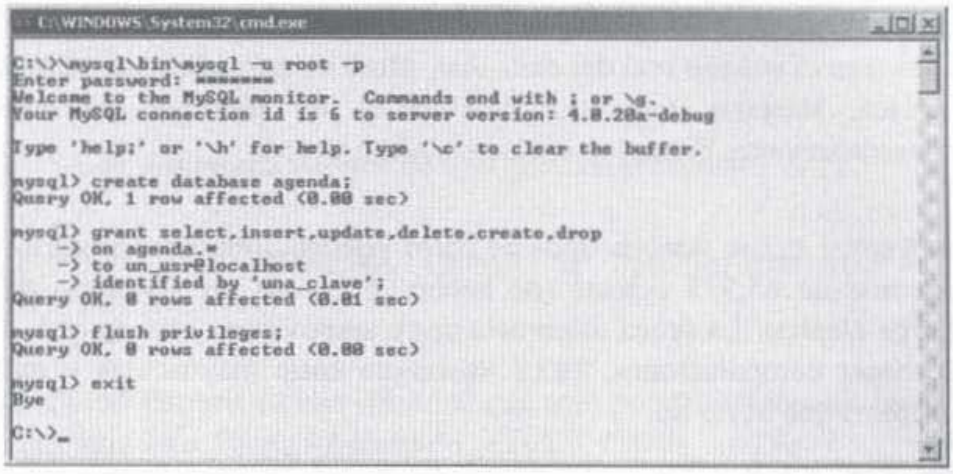
Excepto las operaciones de conectar, desconectar y alguna más, el resto de las operaciones se hacen a través de la función `mysql_query($setencia_sql, $descr)`. Muy importantes también son las funciones `mysql_error()` y `mysql_errno()` pues nos dicen la naturaleza del error ocurrido y el número de éste, respectivamente. Otra función que usaremos con bastante frecuencia es `mysql_affected_rows()`, la cual nos indica el número de registros que se han visto *afectados* en la operación recién realizada sea la que sea (inserción, eliminación, modificación, búsqueda, etc.).

11.3 IMPLEMENTACIÓN DE UNA AGENDA CON MySQL

Como ya sabemos, las operaciones que podemos realizar con una base de datos son crearla, borrarla y manipular sus datos (insertar, modificar, borrar y buscar). En este apartado vamos a simultanear la descripción de los comandos de MySQL que nos permiten ejecutar las operaciones mencionadas junto con la descripción de las funciones implementadas en PHP y su aplicación en la implementación de la agenda del anexo A.

11.3.1 Creación de la base de datos

Por seguridad, la operación de crear una base de datos está restringida al usuario `root`. Por esto, y dado que esta operación sólo se realiza una vez, la base de datos que albergará la agenda la crearemos desde el programa cliente `mysql`. Así, en el código de abajo vemos cómo nos conectamos con el usuario `root` y, una vez dentro, creamos la base de datos `agenda` y, a continuación, añadimos el usuario `un_usr`, que sólo tendrá acceso desde la máquina donde reside el gestor (determinado por la cláusula `TO un_usr@localhost`); tendrá como palabra clave `una_clave`, y todos los permisos sobre la base de datos `agenda` que acabamos de crear:



11.3.2 Creación de la tabla

En la creación de la tabla donde se almacenarán las filas (o registros) de nuestra base de datos hay que indicar el nombre de la tabla, las distintas columnas que tendrá (junto con el tipo de cada uno de ellos), y la descripción de qué columnas serán usadas como claves primarias. La sintaxis simplificada de la sentencia SQL que nos permite crear una tabla es la siguiente:

```
CREATE TABLE tabla (
  [campo tipo [NOT NULL|NULL] [DEFAULT valor]
  [AUTO_INCREMENT]]
  [, PRIMARY KEY([campo])]
  [, INDEX [nomb_ind] (campo)]
  [, UNIQUE [INDEX] [nomb_ind] (campo)]
  [, FOREIGN KEY campo (nomb_ind)])
  [ON DELETE RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT]
  [ON UPDATE RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT]
```

11.3.2.1 Tipos de datos de las columnas

Los identificadores de las columnas pueden empezar por cualquier letra o número, pero no pueden estar constituidos sólo por números; además, pueden tener una longitud máxima de 64 caracteres. A continuación, resumimos algunos de los distintos tipos de datos que MySQL es capaz de manipular:

- `CHAR(L)`: Lo usamos para almacenar cadenas de `L` caracteres. Una columna de este tipo puede tener un tamaño máximo de 255 caracteres. Internamente, se reservan siempre 255 caracteres. Ejemplo: `nombre CHAR(50)`

- `VARCHAR(L)`: Es igual que `CHAR`, excepto en que internamente sólo se almacena el tamaño real del dato. Así, `CHAR` es más eficiente en términos de rapidez, mientras que `VARCHAR` lo es en términos de espacio de almacenamiento. Ejemplo: `nombre VARCHAR(50)`

- `TEXT` / `BLOB`: Ambos tipos de datos pueden almacenar hasta un tamaño máximo de 65,535 octetos (de hecho, `BLOB` es un acrónimo de *Binary Large Object*). La única diferencia entre ambos tipos está en que, a la hora de hacer comparaciones, `TEXT` distingue entre mayúsculas y minúsculas mientras que `BLOB` no.

- `INT [unsigned]`: Especifica un número entero cuyos valores van desde -2,147,483,648 hasta 2,147,483,648. Si especificamos `unsigned`, entonces los valores irán desde 0 hasta 4,294,967,295. Ejemplo: `edad INT unsigned`.

- `TINYNT [unsigned]`: Especifica un entero pequeño cuyos valores van desde -127 hasta 128, o desde 0 a 255, si se especifica `unsigned`.

- `SMALLINT [unsigned]`: Especifica un entero pequeño cuyos valores van desde -32,768 hasta 32,767, o desde 0 a 65535, si se especifica `unsigned`.

- `FLOAT[(E,D)]`: Especifica valores en coma flotante de precisión simple. Dado que hay una parte entera y otra real en estos números, se puede especificar el total de dígitos para ambas cantidades. Ejemplos: `media_aritmet FLOAT`, `temperatura FLOAT(2,1)`, `altura FLOAT(1,2)`.

- `DOUBLE[(E,D)]` o `REAL[(E,D)]`: Es igual que `FLOAT`, excepto en que especifica valores en coma flotante de precisión doble.

- `DECIMAL[(E,D)]`: Igual que `FLOAT`, excepto en que no hace redondeo de la cantidad puesto que almacena los números como cadenas de caracteres (es útil, por ejemplo, para tratar cantidades de dinero).

- `DATE`: Almacena valores de tipo fecha. Los valores que admite son cadenas de caracteres con distintos formatos: `YYYY-MM-DD`, `YY-MM-DD`, o `YYMMDD`. Los valores que puede tomar van desde `1000-01-01` hasta `9999-31-12`

- `TIME`: Almacena valores de tipo `hora`. Admite los formatos `HH:MM:SS`, `HHMMSS` o `HHMM`.

- `YEAR`: Almacena valores de tipo *año* con el formato `YYYY`.

- `TIMESTAMP`: Almacena valores de tipo *instante* con el formato `YYYYMMDDhhmmss`.

- `ENUM`: Especifica una columna que sólo podrá contener un único valor de entre una lista. Ejemplo: `estado_civil ENUM('soltero', 'casado', 'otros')`

- `SET`: Es igual que el anterior excepto en que la columna podrá tener cualquier combinación de uno o más valores. Ejemplo: `aficiones SET('snowboard', 'natación', 'senderismo', 'viajar')`

Al especificar qué tipo de datos pueden almacenar las columnas, podemos, además, indicar una serie de características que pueden cumplir como son:

- `PRIMARY KEY`: Si una columna tiene esta característica, no podrá haber dos registros que tengan el mismo valor en dicha columna, es decir, lo usaremos para diferenciar una fila de otra. Dado que este campo es la **clave primaria** de la tabla, MySQL indexará automáticamente la tabla en función de esta columna (esta característica es la que da el calificativo de *relacional* al modelo de datos, ya que relacionaremos unas tablas con otras a través de las claves).

- `AUTOINCREMENT`: Sólo se aplica a campos de tipo entero ya que su efecto es, al insertar un registro en la tabla y dar el valor `NULL` para dicho campo, asignar al correspondiente campo del registro nuevo el valor más grande que haya en esa columna más uno. No se puede definir más de una columna de este tipo por tabla.

- `DEFAULT valor_por_defecto`: Al insertar una fila, se asignará (a la columna que tenga esta característica) `valor_por_defecto` cuando se especifique `NULL` como valor a asignar.

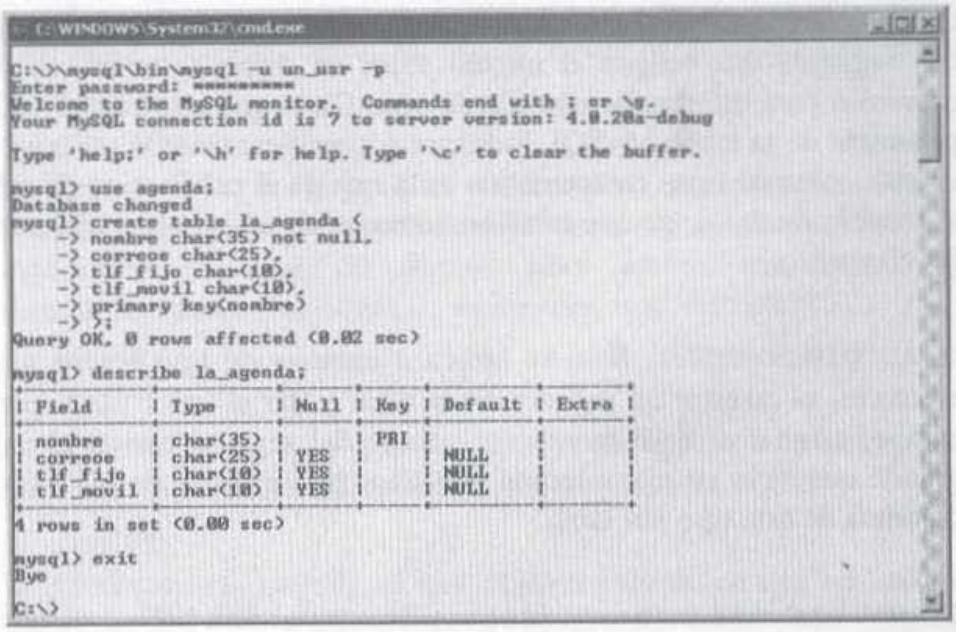
- `NOT NULL`: Un campo que tenga esta característica no podrá nunca asignársele un valor `NULL`.

- NULL: Son campos cuyos valores serán NULL o aquéllos especificados por DEFAULT.

11.3.2.2 Creación de la tabla

Comparando con la agenda realizada con ficheros, la única operación que resulta más *compleja* es la de crear la tabla que nos servirá como base de datos puesto que, como se desprende de la sintaxis del comando CREATE TABLE, hay que indicar expresamente el nombre, tipo y longitud de cada campo, y qué campo será el usado como clave de la tabla (en el caso de hacerlo con ficheros, basta con crearlo vacío).

Dado que en el punto anterior al crear la base de datos agenda, le dimos (con el comando GRANT) los privilegios suficientes al usuario un_usr sobre ella, ahora, para la creación de la tabla, lo haremos como dicho usuario. También, en este caso, esta operación sólo necesitaremos hacerla una única vez. Entonces, la secuencia de comandos es: conectarnos como un_usr, hacer que la base de datos agenda sea la que esté activa por defecto y crear la tabla la_agenda. La última instrucción que ejecutamos es la de comprobar la estructura de la tabla recién creada. Las sentencias SQL necesarias son:



El *script* PHP que habría hecho esta misma operación:

```
<?
$la_bd    = "agenda";
$la_tabla = "la_agenda";

$db=mysql_connect("localhost", "un_usr", "una_clave");
```

```
if (!$db)
  die ("<h3>*** ERROR al conectar...");

echo "creando base de datos...";
$sql="CREATE DATABASE $la_bd;";
if (!$result=mysql_query($sql, $db))
  die ("<h3>*** ERROR al crear la BD: '$sql'(".mysql_errno());

if (!mysql_select_db($la_bd, $db))
  die("<h3>ERROR: al seleccionar BD $la_bd</h3>".mysql_errno());

echo "creando tabla $la_tabla...";
$sql="CREATE TABLE $la_tabla (
      nombre CHAR(35) NOT NULL,
      correoe CHAR(25),
      tlf_fijo CHAR(10),
      tlf_movil CHAR(10),
      PRIMARY KEY(nombre)
      );";
if (!$result=mysql_query($sql, $db))
  die ("<h3>*** ERROR al ejecutar: '$sql' (".mysql_error());

$res        = mysql_query("SELECT * FROM $la_tabla");
$num_campos = mysql_num_fields($res);
echo "La tabla <b>$la_tabla</b> tiene $num_campos Campos:<br>
      <table border=1>
      <tr
bgcolor='lightgray'><th>Nombre<th>Tipo<th>Tamaño<th>Opciones</tr>\n";
for ($i=0; $i < $num_campos; $i++) {
  $nombre  = mysql_field_name($res, $i);
  $tipo    = mysql_field_type($res, $i);
  $tam     = mysql_field_len($res, $i);
  $flags   = mysql_field_flags($res, $i);
  echo  "<tr><td>$nombre<td>$tipo<td>$tam<td>$flags</tr>\n";
}
echo "</table>";

mysql_close($db);
```

11.3.3 Fichero de apoyo

Dado que habrá una serie de rutinas que serán utilizadas por casi todos los *scripts*, las situaremos en un fichero aparte (`agbd_datos.inc`) que será *importado* por éstos. La primera de ellas es la función de establecer una conexión con el servidor MySQL (a la que habrá que indicarle tanto el usuario, como la clave de éste) que nos devolverá un descriptor de la conexión.

```
function conecta()
{
  $dbd=mysql_connect("localhost", "un_usr", "una_clave");
  if (!$dbd)
    die ("<h3>*** ERROR al conectar... :((");

  if (!mysql_select_db("agenda", $dbd))
    die("<h3>ERROR: al seleccionar</h3>".mysql_errno());
  return $dbd;
}
```

Definimos tres funciones más relacionadas con la generación de texto HTML: `pinta_entrada_datos()`, que nos permitirá sacar por pantalla el formulario donde el usuario introducirá los datos para ser insertados o modificados en la base de datos; `página_anterior()` que imprimirá en el navegador un enlace para volver a la página anterior, y `pon_pie_de_vuelta()` que sacará otro enlace que nos llevará a la página principal.

```
//la usan: agbd_suma1_plantilla.php y agbd_modif_plantilla.php
function pinta_entrada_datos($el_php, $leyenda_submit, $la_victima,
                             $el_n, $el_c, $el_tf, $el_tm)
{
echo "<form action='$el_php' method='get'>";

if ($la_victima)
  echo "<input type='hidden' name='victima_modificacion'
         value='$la_victima'>";

echo "
<table border='0'>
<tr>
  <td>Nombre:
  <td><input type='text' size='35' name='nombre' value='$el_n'>
</tr>
<tr>
  <td>Correo-e:
  <td><input type='text' size='25' name='correoe' value='$el_c'>
</tr>
<tr>
  <td>Teléfono fijo:
  <td><input type='text' size='10' name='tlf_fijo' value='$el_tf'>
</tr>
<tr>
  <td>Teléfono móvil:
  <td><input type='text' size='10' name='tlf_movil' value='$el_tm'>
</tr>
<tr>
  <td colspan='2'><input type='submit' value='$leyenda_submit'>
</tr>
</table>
</form>
";
}

function pagina_anterior()
{
echo "<a href='javascript:history.go(-1)'>&lt;&lt;volver atrás</a><br>";
}

function pon_pie_de_vuelta()
{
  echo "<br><a href='agbd_opers.php'>Volver a la Agenda</a>";
}

?>
```

11.3.4 Listado de registros

La sentencia SQL que nos permite realizar búsquedas de datos que satisfagan una serie de condiciones es la sentencia `SELECT`, cuya sintaxis simplificada es:

```
SELECT lista_de_campos FROM tabla
                WHERE condicion
                [GROUP BY (campos) [HAVING condicion]]
                [ORDER BY (campos) [ASC|DESC]]
```

Donde `lista_de_campos` indica las columnas que se desean obtener en la consulta (puede ser el nombre de un campo o una lista de ellos separados por comas; también puede ser el carácter "*", en cuyo caso se obtendrán todas las columnas). Obviamente, `tabla` es la tabla donde se va a hacer la consulta.

En la cláusula `WHERE`, la `condicion` normalmente será una expresión del tipo `nombre_de_campo='un_valor'`, si queremos filas cuyo campo de búsqueda tenga una coincidencia exacta con el valor indicado, o `nombre_de_campo LIKE 'expresion_regular'`, si queremos encontrar registros cuyo campo de búsqueda coincida con un patrón dado por una expresión regular (los metacaracteres que podemos usar son "`%`" y "`_`", los cuales indican cualquier cadena de caracteres, incluida la nula, y un solo carácter, respectivamente). Por supuesto, las condiciones de búsqueda se pueden hacer más complejas usando los operadores lógicos `AND` o `&&`, `OR` o `||` y `NOT` o `!`. Las cláusulas `GROUP BY` y `ORDER BY` nos permiten agrupar u ordenar el resultado.

En el ejemplo de la agenda que nos ocupa, hay una serie de operaciones con los registros de la base de datos que nos requieren un listado de registros: para elegir uno de entre todos para modificarlo, para elegir todos aquellos que queramos borrar, para mostrar el resultado de una búsqueda o, simplemente, para mostrarlos. Por este motivo, hemos juntado en un único *script* (`agbd_listados.php`) la generación de todos estos listados.

No obstante, cada uno de estos listados tiene unos requisitos distintos. Por ejemplo, no es lo mismo generar un listado que tenga la posibilidad de elegir un grupo de registros para ser eliminados que un listado normal de registros: mientras que este último no requiere nada especial, el primero debe contar con elementos de formulario de tipo *checkbox* para que el usuario pueda marcarlos según sus necesidades y, por supuesto, este formulario debe indicar el nombre del *script* que va a hacer efectiva la eliminación. Por tanto, las llamadas a este *script* habrá que *parametrizarlas* adecuadamente pues hay que saber qué tipo de listado debe generar

y con qué características. En la práctica, esto significa que en las llamadas a este programa expresaremos esas particularidades en forma de valores dados en el QUERY_STRING. Así, los datos que obtendremos serán: el tipo de operación a realizar, la cadena de caracteres a buscar (en su caso), el campo por el que haremos la ordenación de los registros y el tipo de ordenación (ascendente o descendente):

```
<?
$operacion=$_GET['oper'];
switch($operacion) {
  case 'LISTAR' :
    $cabecera="Agenda (version BD): Listado de entradas en la agenda";
    $cgi='';
    $elem_form='';
    $leyenda='';
    $a_buscar='';
    break;
  case 'BUSCAR' :
    $cabecera="Agenda (version BD): búsqueda de registros";
    $cgi='';
    $elem_form='';
    $leyenda='';
    $a_buscar=$_GET['a_buscar'];
    break;
  case 'MODIFICAR' :
    $cabecera="Agenda (version BD): Selección de Entradas a Modificar";
    $cgi='agbd_modif_plantilla.php';
    $elem_form='r';
    $leyenda='Modificar!';
    $a_buscar='';
    break;
  case 'BORRAR' :
    $cabecera="Agenda: (versión BD) Selección de Entradas a Borrar";
    $cgi='agbd_borrar.php';
    $elem_form='c';
    $leyenda='Borrar!';
    $a_buscar='';
    break;
}
echo "<title>$cabecera</title> </head>
      <body>
      <h2>$cabecera</h2>";

if (isset($_GET['campo'])){
  $campo_ord=$_GET['campo'];
  $sentido=$_GET['dir'];
} else {
  $campo_ord='nombre';
  $sentido='ASC';
}

include("agbd_datos.inc");
$dbd=conecta();

$sql="SELECT * from la_agenda ";
if ($a_buscar)
  $sql .= "WHERE nombre LIKE '%$a_buscar%' ";
$sql .= "ORDER BY ($campo_ord) $sentido";
$res = mysql_query($sql, $dbd);
if (!$res)
  die ("*** ERROR en la búsqueda: " . mysql_error());
```

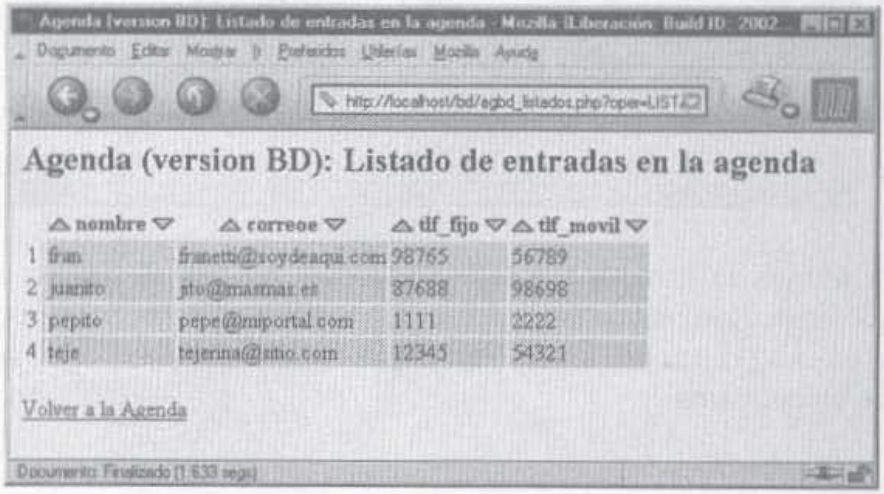
```
pinta_listado($res, $operacion,
            $cgi, $elem_form, $leyenda, $a_buscar);

pon_pie_de_vuelta();
```

Por tanto, una vez unidos los distintos valores y hecha la consulta a la base de datos, llamamos a una función que hemos declarado en el mismo *script* (con el nombre `pinta_listado()`) para que, con el resultado de la consulta, haga la impresión del listado. Esta función está definida con los siguientes parámetros:

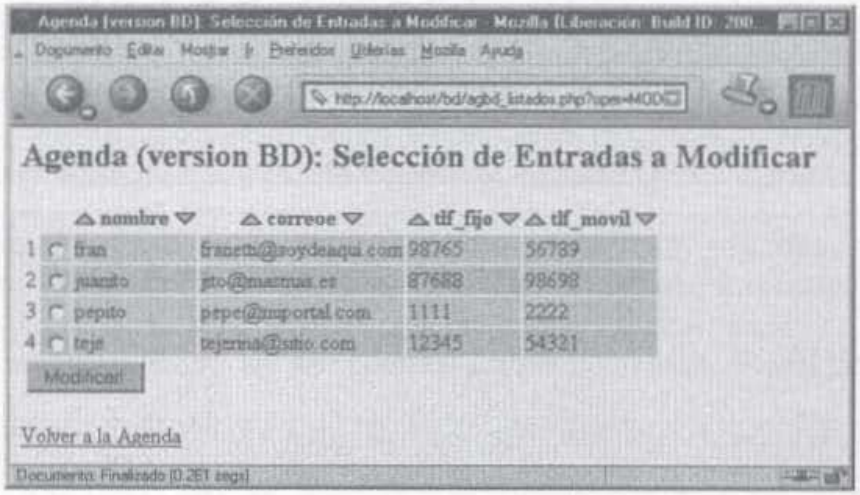
```
pinta_listado($cursor, $operac, $cgi, $elem_form, $bot_submit, $busco)
```

Los parámetros que hay que pasarle son el cursor donde están los resultados de la búsqueda realizada, el tipo de operación o listado a realizar (`LISTAR`, `BUSCAR`, `MODIFICAR` o `BORRAR`), el nombre del programa que va a tratar los datos del formulario (si es el caso), el tipo de elemento de formulario que aparecerá (*checkbox*, *radio* o nada) en la primera columna de la tabla, la leyenda que aparecerá en el botón de tipo *submit* que efectúa la llamada al siguiente *script* PHP (si es el caso) y, por último, la cadena a buscar, si es que la operación es de búsqueda.

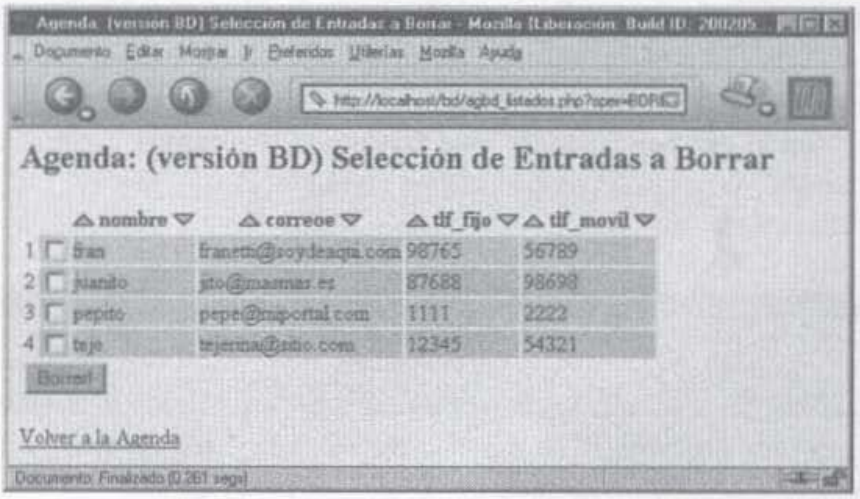
Entonces, para generar un simple listado de todas las entradas en la base de datos, o un listado del resultado de una búsqueda, llamamos a esta función de la siguiente manera: `pinta_listado($res, 'LISTADO', '', '', '', '')`, cuyo resultado sería:



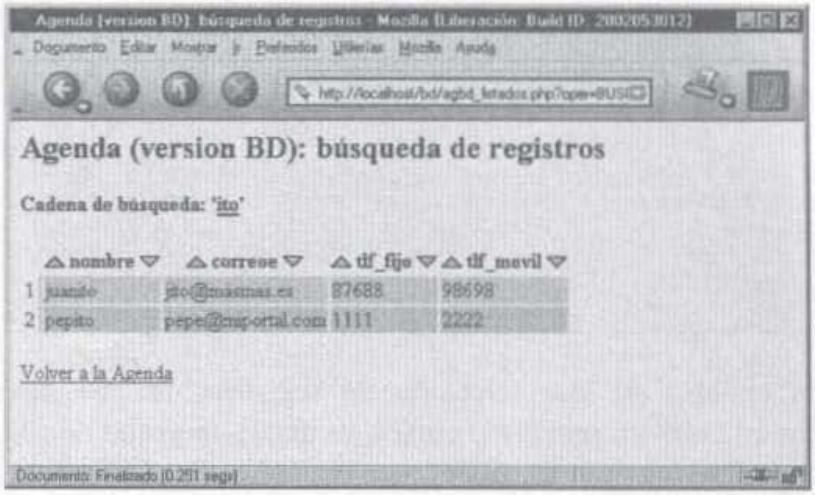
La llamada que generará un listado de todas las entradas para poder seleccionar aquella que queramos modificar será de la siguiente manera: `pinta_listado($res, 'MODIFICAR', 'agbd_modif_plantilla.php', 'r' , 'Modificar!','')`, y generará:



Para borrar registros, al usuario se le muestra un listado de todos los que hay y así pueda seleccionar (mediante *checkboxes*) aquellos que quiera borrar. En este caso, la llamada a la función sería `pinta_listado($res, 'BORRAR', 'agbd_borrar.php', 'c', 'Borrar!', '')`, y la salida:



Por último, la llamada a esta función para obtener el listado de aquellos registros que contengan una determinada cadena en el campo `nombre`, por ejemplo, `'ito'`, sería `pinta_listado($res, 'BUSCAR', '' , '' , '', 'ito')` y la pantalla que obtendríamos:



El código correspondiente a esta función es muy sencillo, básicamente se trata de imprimir una tabla HTML donde cada fila se corresponde con un registro y tener en cuenta el tipo de elemento de formulario que se va a imprimir para la elección de registros:

```
function pinta_listado($cursor, $operac,
                       $cgi, $elem_form, $bot_submit, $busco)
{
  global $campos;
  $colores_filas=array('#cccccc', 'lightblue');
  $ind_colores=0;
  $cont_lineas=1;

  echo "<table>";
  if ($cgi) echo "<form action='$cgi' method='post'>";

  if ($busco)
     echo "<h4>Cadena de búsqueda: '<u>$busco</u>'</h4>";

  pinta_cabecera_tabla($operac, $busco);

  while ($reg = mysql_fetch_array($cursor, MYSQL_ASSOC))
  {
    $ind_colores++;
    $ind_colores %= 2;
    echo "<tr bgcolor=${colores_filas[$ind_colores]}>";
    switch($elem_form){
      case 'c':
        $valor=$reg['nombre'];
        $seg_col="
         <input type='checkbox' name='victima$cont_lineas' value='$valor'>";
        break;
      case 'r':
        $valor=$reg['nombre'];
        $seg_col="<input type='radio' name='victima' value='$valor'>";
        break;
      default: $seg_col="";  // caso de listar y buscar
    }

    echo "<td bgcolor='white'>$cont_lineas</td><td>$seg_col</td>";
    for ($i=0; $i<count($reg); $i++) {
      echo "<td>", $reg[$campos[$i]], "</td>";
```

```
    }
    echo "</tr>";
    $cont_lineas++;
  }
  if ($cgi)
    echo "<tr>
          <td colspan='6'><input type='submit' value='$bot_submit'></td>
          </tr>
          </form>";
  echo "</table>";
}
```

En el caso de una selección de registros, lo que devuelve la función `mysql_query()` es un *recurso* o cursor, es decir, un grupo de filas almacenadas en una estructura interna de la que habrá que extraerlas con la función:

```
mysql_fetch_array($recurso, TIPO_ARRAY)
```

Si queremos que nos devuelva las filas en forma de *array* asociativo, numérico o con ambos tipos de índices, pondremos como segundo parámetro: `MYSQL_ASSOC`, `MYSQL_NUM` o `MYSQL_BOTH`, respectivamente.

11.3.4.1 Ordenación de registros

Como se podrá observar en el código anterior, desde la función `pinta_listado()` llamamos a la función `pinta_cabecera_tabla()` la cual lo que hace es imprimir la cabecera de la tabla con los nombres de los campos y un par de iconos que representan la ordenación de los registros en función de esos campos, pero en orden ascendente o descendente:

△ nombre ▽ △ correoe ▽ △ tlf_fijo ▽ △ tlf_movil ▽

De nuevo, con respecto a la búsqueda en la base de datos con ficheros, esta operación se simplifica enormemente pues la operación de ordenación no tenemos que implementarla nosotros ya que la sentencia `SELECT` lo hace por nosotros. Como vimos anteriormente, gracias a la cláusula `ORDER BY(campo) [ASC|DESC]`, la devolución de la consulta se hará ordenada alfabéticamente en orden ascendente (`ASC`) o descendente (`DESC`). Por esta razón, vemos en un extracto del código mostrado anteriormente cómo construimos la orden de búsqueda o listado de registros:

```
$sql="SELECT * from la_agenda ";
if ($a_buscar)
  $sql .= "WHERE nombre LIKE '%$a_buscar%' ";
$sql .= "ORDER BY ($campo_ord) $sentido";
```

Dado que la función `pinta_cabecera_tabla()` genera los enlaces para obtener el mismo listado que tenemos pero ordenado por otro campo y/o en otro sentido, necesita para ello conocer los datos de la consulta, esto es, el tipo de listado y la cadena de búsqueda. Por este motivo, la hemos declarado con dos parámetros que se corresponden con el tipo de listado y la cadena de búsqueda.

```
function pinta_cabecera_tabla($op, $op2)
{
  global $campos;

  echo "<tr><td></td><td></td>";  // 2 celdas: numeracion & elem de
formulario

  if ($op2) {
    $param="&a_buscar=$op2";
  } else {
    $param="";
  }
  foreach ($campos as $un_campo) {
    $prefijo  = "<a href='".$_SERVER['PHP_SELF']."?oper=$op&campo=$un_campo";
    $pref_arriba   = "$prefijo&dir=ASC$param'>";
    $pref_abajo    = "$prefijo&dir=DESC$param'>";
    $sufijo_arriba = "<img src='up.gif' border='0'></a>";
    $sufijo_abajo  = "<img src='down.gif' border='0'></a>";
    echo "<th> $pref_arriba$sufijo_arriba $un_campo
$pref_abajo$sufijo_abajo</th>";
  }
  echo "</tr>";
}

?>
```

En el caso del ejemplo con ficheros se podría mostrar los registros de manera ordenada puesto que podríamos ordenar el *array* donde leíamos los datos. El problema estaría en las operaciones de modificar y borrar, puesto que se basaban en la posición que ocupaban en el fichero y no en el nombre, como es el caso.

11.3.5 Borrar un registro

La sintaxis simplificada de la sentencia del lenguaje SQL que nos permitirá borrar filas de una tabla de la base de datos es:

```
DELETE FROM tabla WHERE condicion
```

Donde la cláusula `condicion` es idéntica a la que indicamos cuando hacemos una búsqueda (borrar implica hacer un primer examen de los registros que cumplen la condición indicada y, acto seguido, ejecutar la operación de borrar sobre aquellos registros encontrados).

En la agenda, para borrar registros, llamamos al programa de listados: `agbd_listados?op=BORRAR`, el cual generará un listado de todos los registros de la agenda llamando a la rutina `pinta_listado()` de la siguiente manera:

```
pinta_listado($res, 'BORRAR',
            'agbd_borrar.php', 'c', 'Borrar!', '');
```

Como ya sabemos, se genera un formulario con casillas de verificación (*checkboxes*) para que el usuario seleccione los registros que van a ser borrados. Mostramos un extracto de la función `pinta_listado()` para ver cómo se generan estos elementos de formulario, puesto que es importante conocer cómo es el mecanismo por el cual se seleccionan los registros que se van a eliminar:

```
switch($tipo_elem_form){
  case 'c':
    $valor=$reg['nombre'];
    $elem_form="
    <td>
    <input type='checkbox' name='victima$cont_lineas' value='$valor'>
    </td>";
    break;
```

Es decir, cuando borramos, lo hacemos por el nombre de la clave (que es única) y no por posición (como en la agenda implementada con ficheros), lo que hace esta operación más fiable.

Como apreciamos arriba, el nombre del *script* que hará efectiva la eliminación es `agbd_borrar.php`:

```
<html>
<head> <title>Agenda: Borrar entradas</title> </head>
<body>
<h2>Agenda (versión BD): Borrar entradas</h2>
<?php
include("agbd_datos.inc");

if (count($_GET) > 0) {
  $sql="DELETE FROM la_agenda WHERE ";
  foreach ($_GET as $clave=>$valor) {
    $sql.= "nombre='$valor' OR ";
  }
  $sql=substr($sql, 0, strlen($sql)-3); // quito el ultimo 'OR '

  $dbd = conecta();
  $res = mysql_query($sql, $dbd);
  $total_victimas=mysql_affected_rows($dbd);
  echo "<h3>Borrados <u>$total_victimas</u> registros </h3>";
} else {
  echo "<h4>NO ha seleccionado ningún registro para borrar :(</h4>";
  pagina_anterior();
}

pon_pie_de_vuelta();
?>
```

11.3.6 Modificar registros

Para realizar la operación de modificar o actualizar datos, SQL proporciona la sentencia `UPDATE`, cuya sintaxis simplificada es:

```
UPDATE tabla SET campo1=val1,... [WHERE condicion]
```

La operación de modificar un registro la hacemos en tres pasos: primero ejecutamos `agbd_listados?op=MODIFICAR`, por lo que la generación del listado de registros será:

```
pinta_listado($res, 'MODIFICAR',
              'agbd_modif_plantilla.php', 'r', 'Modificar!', '');
```

A continuación, el *script* `agbd_modif_plantilla.php` (expuesto abajo) mostrará el contenido actual del registro seleccionado para que podamos introducir los valores oportunos:

```
<html>
<head>
<title>Agenda: seleccionar registro para modificarlo</title>
</head>
<body>
<h2>Agenda (versión BD): Seleccionar elementos para su modificación</h2>
<?php
if (!isset($_GET['victima'])) {
  echo "<h4>NO ha seleccionado ningún registro para modificar</h4>";
  pagina_atras();
  pon_pie_de_vuelta();
  exit;
}
$victima=$_GET['victima'];

include("agbd_datos.inc");

$dbd = conecta();
$sql = "SELECT * FROM la_agenda WHERE nombre='$victima'";
$res = mysql_query($sql, $dbd);
if (! $res){
  echo "*** ERROR en el SELECT: ", mysql_error();
  pon_pie_de_vuelta();
  exit;
}

$campos=mysql_fetch_array($res, MYSQL_ASSOC);
pinta_entrada_datos("agbd_modificar.php", "modifica!",
                    "$victima",
                    "${campos['nombre']}",   "${campos['correoe']}",
                     "${campos['tlf_fijo']}", "${campos['tlf_movil']}");

pagina_anterior();
pon_pie_de_vuelta();
?>
```

Por último, al igual que en la operación de insertar datos, el *script* que hace efectiva la modificación también se beneficia de la característica de que el campo `nombre` sea clave primaria (será imposible introducir un campo repetido pues el gestor generará un error) y de que, al no existir caracteres *no permitidos*, no será necesario comprobar lo que ha teclado el usuario en el formulario previo. Este *script* se llama `agbd_modificar.php` y su contenido es el siguiente:

```
<html>
<head> <title>Agenda: Modificación de entradas</title> </head>
<body>
<h2>Agenda (versión BD): Modificación de entradas</h2>
<?php

$nombre    = $_GET['nombre'];
if (strlen(trim($nombre))==0) {
  echo ("*** ERROR: el campo nombre no puede estar vacio<br>");
  pagina_anterior;
  pon_pie_de_vuelta();
  exit;
}
$victima_modificacion = $_GET['victima_modificacion'];
$correoe              = $_GET['correoe'];
$tlf_fijo             = $_GET['tlf_fijo'];
$tlf_movil            = $_GET['tlf_movil'];

include("agbd_datos.inc");
$dbd=conecta();

$sql="UPDATE la_agenda SET ";
$sql.="nombre='$nombre', correoe='$correoe', tlf_fijo='$tlf_fijo',
      tlf_movil='$tlf_movil' ";
$sql.="WHERE nombre='$victima_modificacion'";

$res=mysql_query($sql, $dbd);
if (! $res) {
  echo "*** ERROR al actualizar $victima_modificacion: ", mysql_error();
  pon_pie_de_vuelta();
  exit;
}
echo "Modificado <u>", mysql_affected_rows(), "</u> registro<br>";
pon_pie_de_vuelta();
?>
```

11.3.7 Insertar registros

La sentencia SQL `INSERT` presenta la siguiente sintaxis:

```
INSERT INTO tabla [(campo1, campo2,...)]
       VALUES (valor1, valor2,...)
```

Si no hacemos uso de la parte condicional en la que se deben indicar los nombres de los campos a insertar, la lista de valores deberá contemplar a todos los elementos de la fila en el mismo orden en que se declararon cuando se creó la tabla. En caso de indicar los campos a insertar, la lista podrá tener cualquier orden e,

incluso, no incluir algún campo, siempre teniendo en cuenta que se asociarán los valores a sus respectivos campos siguiendo el orden en que aparezcan estos últimos.

Existe una segunda forma alternativa de la sentencia `INSERT` en la que los nombres de las columnas se especifican expresamente junto con su valor:

```
INSERT INTO tabla SET col1=val1, col2=val2, ...
```

Lo que sí hay que tener en cuenta en ambos casos es que, al insertar un valor de tipo cadena de caracteres (`CHAR`, `VARCHAR`, `TEXT`, etc.), éste hay que englobarlo entre comillas.

A diferencia de la agenda del anexo, donde teníamos que programar una rutina específica para asegurarnos de que no introducíamos nombres repetidos, con MySQL no hace falta preguntar si está duplicado ya que, al haber definido este campo como clave primaria, el gestor hace este trabajo por nosotros: nos impedirá introducir claves repetidas generando un error que tendremos que capturar y mostrar al usuario para que tenga noticia de ello. Además, puesto que no existe el concepto de *carácter delimitador* de campo, tampoco tenemos que comprobar que el usuario lo introduce en los datos del formulario. Lo único que tendremos en cuenta es recoger adecuadamente los valores introducidos por el usuario en el formulario previo e introducirlo en la base de datos:

```
<?
$nombre    = $_GET['nombre'];
if (strlen(trim($nombre))==0) {
  echo ('*** ERROR: el campo nombre no puede estar vacio<br>');
  pagina_anterior();
  pon_pie_de_vuelta();
  exit;
}
$correoe   = $_GET['correoe'];
$tlf_fijo  = $_GET['tlf_fijo'];
$tlf_movil = $_GET['tlf_movil'];

include('agbd_datos.inc');
$dbd=conecta();

$sql="INSERT INTO la_agenda VALUES ('$nombre', '$correoe', '$tlf_fijo',
'$tlf_movil')";

$res = mysql_query($sql, $dbd);
if ($res)
  echo "Añadido ", mysql_affected_rows(), "registro/s a la bd<br>";
else
  echo "ERROR al añadir: ", mysql_error();

pon_pie_de_vuelta();
?>
```

11.3.8 Total de registros

Para mostrar el total de registros que hay en la base de datos usaremos una de las funciones agregadas que tiene la sentencia SELECT:

- COUNT(*): Devuelve el total de filas seleccionadas.
- COUNT(DISTINCT campo): Devuelve el total de filas seleccionadas sin tener en cuenta aquellas donde campo esté repetido.
- AVG(campo): Devuelve la media aritmética de campo.
- MAX(campo): Devuelve el valor máximo.
- MIN(campo): Devuelve el valor mínimo.

Por tanto, averiguar el número total de registros también es muy sencillo si hacemos uso de la cláusula COUNT:

```
<html>
<head> <title>Agenda: Total de registros</title> </head>
<body>
<h2>Agenda (versión BD): Total de registros en la agenda</h2>
<?php
include('agbd_datos.inc');
$dbd=conecta();

$res = mysql_query("SELECT COUNT(*) from la_agenda", $dbd);
$total=mysql_fetch_array($res);
echo "<h4>Total de registros en al agenda: <u>",$total[0], "</u></h4>";

pon_pie_de_vuelta();
?>
```

11.3.9 Modificar una tabla

Adicionalmente, si en algún momento necesitáramos modificar la estructura de una tabla de nuestra base de datos por la razón que fuese, podríamos hacerlo con la sentencia SQL ALTER TABLE, cuya sintaxis simplificada es:

```
ALTER TABLE tabla [
                  [ADD campo tipo [FIRST|AFTER campo]]
                  [ADD PRIMARY KEY (campo)]
                  [CHANGE campo_ant campo nuevo tipo]
                  [DROP col_name]
                  [DROP PRIMARY KEY]
                  [RENAME [TO] nomb_tabla_nuevo] ]
```

Por ejemplo, en el siguiente código vemos cómo haríamos para añadir una nueva columna en la agenda donde introducir la fecha de cumpleaños:

```
ALTER table la_agenda ADD cumple DATE;
```

Hacer más grande el campo de la dirección de correo electrónico:

```
ALTER table la_agenda CHANGE correoe correoe VARCHAR(30);
```

Eliminar el campo destinado al teléfono fijo:

```
ALTER table la_agenda DROP tlf_fijo;
```

Cambiar el nombre a la base de datos:

```
ALTER table la_agenda RENAME agenda;
```

Gracias al empleo de una base de datos, y tal como está diseñada la aplicación, realizar una modificación en la estructura de la agenda no supondría mayores problemas para reflejarlos en los *scripts*, suposición que no es necesariamente cierta en el caso de la agenda implementada con ficheros.

11.4 SEGURIDAD EN MySQL

11.4.1 Usuarios

MySQL, cada vez que recibe una petición, comprueba que, tanto el usuario (reconocido por su identificador más palabra clave), como el equipo desde que se realiza la conexión, tienen permiso de conexión a la base de datos.

Además, una vez permitido el acceso a la base de datos, MySQL comprueba qué tipo de privilegios (lectura, escritura, borrado, etc.) tiene el usuario para realizar qué operaciones sobre qué tablas. Es decir, MySQL permite asignar privilegios para cada una de las operaciones básicas del SQL para manejo de datos (`SELECT`, `INSERT`, `UPDATE` y `DELETE`) y para definición y gestión de datos (`ALTER`, `CREATE`, `DROP`, `GRANT`, `FILE`, `INDEX`, `PROCESS`, `REFERENCES`, `RELOAD`, `SHUTDOWN` y `USAGE`).

Como ya se ha visto a lo largo del capítulo, el usuario `root` es el usuario que tiene todos los privilegios sobre la base de datos completa. El problema es que, nada más instalar el paquete MySQL, este *superusuario* no tiene palabra clave asignada, por lo que es urgente asignarle una (por ejemplo, `clave_del_root`):

```
mysqladmin -u root password clave_del_root
```

Mientras que en Unix, el comando `scripts/mysql_install_db` establece una serie de medidas que pueden catalogarse como de *prudentes*, en Windows no existe tal comando y, por defecto, se permite la conexión de usuarios anónimos a la base de datos, esto es, sin especificar ningún usuario. Además, estos usuarios tienen privilegios sobre toda la base de datos. Obviamente, esta situación no es la ideal desde el punto de vista de la seguridad: por defecto, cualquier usuario puede realizar cualquier operación sin indicar palabra clave alguna. Por este motivo, impedimos los accesos anónimos desde la máquina donde está instalado el servidor.

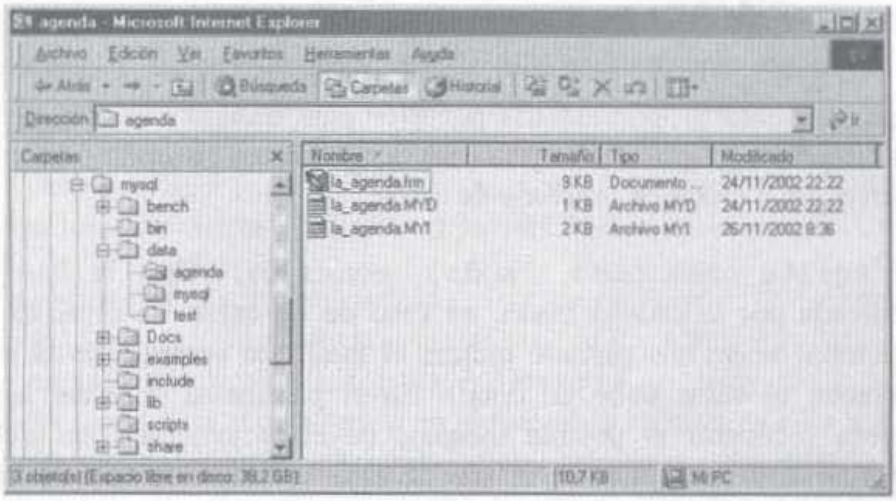
```
c:\>\mysql\bin\mysql -u root -p
mysql> DELETE FROM User WHERE Host='localhost' AND User='';
mysql> exit
c:\>\mysql\bin\mysqladmin -u root -p reload
```

11.4.2 Copias de seguridad

Como todo buen administrador conoce bien, es recomendable (y prudente) guardar una copia de respaldo de la base de datos cada cierto tiempo. MySQL nos lo pone muy fácil ya que nos proporciona una utilidad que nos permite hacer esta tarea de una manera muy sencilla. El comando en cuestión se llama `mysqldump` y la salida que genera son los comandos SQL necesarios para regenerar la base de datos completa (creación de las bases de datos, tablas e inserciones de filas). Este programa tiene multitud de opciones de las cuales sólo comentaremos un par de ellas: la que nos permite *salvar* todas las bases de datos de MySQL (para ello habrá que ejecutarla como `root`) y otra que nos permitirá *guardar* la base de datos de la agenda:

```
mysqldump -q -uroot -proot --all-databases > todas_las_bds.sql
mysqldump -q -uun_usr -puna_clave --databases agenda > agenda.sql
```

Otra forma alternativa de guardar la base de datos consiste en aprovecharse de la estructura del árbol de directorios de MySQL: debajo del directorio `data`, por cada base de datos crea un directorio con su nombre de ésta en el que encontraremos tres ficheros otra vez con el mismo nombre pero con extensiones distintas: `*.FRM` que contiene la estructura de la tabla; `*.MYD`, para los datos de la tabla y `*.MYI`, fichero que contiene los índices (información sobre las claves y otros datos que MySQL usa para las búsquedas). Por tanto, bastará con hacer una copia del directorio que alberga la base de datos deseada.



11.5 SQLITE

A pesar de que MySQL sigue siendo una de las opciones más utilizadas para el desarrollo de aplicaciones PHP que trabajan con bases de datos, debido a problemas de licencia de utilización no forma parte del paquete por defecto de PHP 5. En su lugar se ha optado por incorporar por defecto la extensión para trabajar con bases de datos SQLite.

SQLite es una librería C que implementa una base de datos SQL empotrada, de forma que los programas que trabajan con esta librería pueden acceder a bases de datos SQL sin necesidad de trabajar con un gestor SGBDR externo. SQLite no es una librería cliente utilizada para conectarse contra un servidor, como ocurría en el caso de MySQL: es en sí misma un servidor de bases de datos. Entre sus realizaciones destaca:

- Implementar casi completamente el estándar SQL92.
- Toda la base de datos (tablas, índices...) se almacena en un solo fichero en el disco. Soporta bases de datos de hasta 2 terabytes.
- Permite transacciones ACID (*Atomic*, *Consistent*, *Isolated*, *Durable*).
- Es dos veces más rápida que PostgreSQL y MySQL en algunas de las operaciones más habituales.
- Su código fuente es de dominio público y puede ser utilizado para cualquier propósito.

NOTA: Para obtener más información sobre SQLite se puede recurrir a la dirección http://www.sqlite.org/.

11.5.1 Interfaz de SQLite

Las funciones proporcionadas por PHP para trabajar con SQLite se pueden dividir en diferentes grupos. Estas primeras nos permiten las operaciones básicas de apertura, creación y cierre de una base de datos:

- `sqlite_open(bbdd, modo, mensaje)`: Abre la base de datos indicada por la cadena `bbdd`, en caso de no existir la crea. El parámetro opcional `modo` nos permite indicar el modo de apertura de la misma (por defecto, el valor 0666 en octal). En el parámetro opcional `mensaje` se puede recuperar el posible mensaje de error generado en dicha acción. Devuelve un manejador de la base de datos o `false` en caso de error.

- `sqlite_popen(bbdd, modo, mensaje)`: Tiene el mismo funcionamiento que la función anterior, con la diferencia de que devuelve un manejador persistente.

- `sqlite_close(bbdd)`: Cierra la base de datos asociada al manejador `bbdd` proporcionado.

El siguiente conjunto de funciones nos permiten realizar consultas SQL sobre la base de datos:

- `sqlite_query(bbdd, consulta)`: Ejecuta la `consulta` SQL sobre la base de datos referenciada por el manejador `bbdd`. En caso de que la consulta sea correcta y dependiendo del tipo de operación que se haya ejecutado, devuelve un cursor asociado a los datos recuperados o `true`. Devuelve `false` en caso de error.

  Esta función almacena los datos recuperados en un buffer intermedio, permitiendo de este modo el acceso aleatorio a los mismos. Por ello suele utilizarse conjuntamente con las funciones `sqlite_seek()`, `sqlite_rewind()`, `sqlite_next()`, `sqlite_current()`, y `sqlite_num_rows()`. En caso de sólo necesitar acceso secuencial a los datos es preferible utilizar la función `sqlite_unbuffered_query()` que se comporta de una forma más óptima por no hacer uso de memoria intermedia.

NOTA: Permite una sintaxis alternativa, cambiando de orden los parámetros proporcionados, para ser compatibles con otras extensiones como MySQL.

❑ `sqlite_unbuffered_query(bbdd, consulta)`: Realiza la misma labor que la función anterior, con la diferencia de que los datos devueltos sólo se pueden acceder de forma secuencial. Las funciones `sqlite_seek()`, `sqlite_rewind()`, `sqlite_next()`, `sqlite_current()`, y `sqlite_num_rows()` no pueden ser utilizadas con los cursores devueltos por `sqlite_unbuffered_query()`.

❑ `sqlite_array_query(bbdd, consulta, tipo_array, decodificar)`: Ejecuta la `consulta` SQL sobre la base de datos referenciada por el manejador `bbdd`. Devuelve un *array* escalar con tantos índices como filas recuperadas, en el que cada entrada es a su vez un *array* asociativo con tantas entradas como columnas o campos se hayan recuperado. En caso de producirse un error devuelve `false`. Se puede indicar el `tipo_array` del *array* devuelto y si se debe o no `decodificar` los datos que aparezcan en binario. La siguiente tabla muestra los diferentes tipos de *arrays* que se pueden obtener:

tipo_array	descripción
SQLITE_ASSOC	*Array* asociativo cuyos índices son los nombres de las columnas recuperadas en la consulta.
SQLITE_BOTH	*Array* con índices numéricos y con los nombres de las columnas recuperadas en la consulta.
SQLITE_NUM	*Array* sólo con índices numéricos asociados a las columnas recuperadas por la consulta. La primera columna es la 0.

El siguiente conjunto de funciones nos permiten trabajar con la información recuperada desde una consulta SQL, en ellas el primer parámetro no es un descriptor de conexión a la base de datos sino los datos sobre los que vamos a trabajar:

❑ `sqlite_fetch_array (cursor, tipo_array, decodificar)`: Devuelve un *array* con la **siguiente fila** del `cursor` proporcionado. Se puede indicar el `tipo_array` del *array* devuelto y si se debe o no `decodificar` los datos que aparezcan en binario.

❑ `sqlite_current(cursor, tipo_array, decodificar)`: Es idéntica a `sqlite_fetch_array()` con la única diferencia de que no avanza a la siguiente fila antes de devolver los datos, es decir, devuelve los datos de la **fila actual**.

❑ `sqlite_fetch_single (cursor, tipo_array, decodificar)`: Devuelve una cadena con la primera columna de la siguiente fila del `cursor` proporcionado. Se puede indicar el `tipo_array` del *array* devuelto y si se debe o no `decodificar` los datos que aparezcan en binario. Es especialmente útil cuando trabajamos con consultas que sólo devuelven una columna de datos.

- `sqlite_fetch_string(cursor, tipo_array, decodificar)`: Es un alias de la función anterior.

- `sqlite_column(cursor, tipo_array, decodificar)`: Devuelve una cadena con la primera columna de la fila actual del `cursor` proporcionado. Se puede indicar el `tipo_array` del *array* devuelto y si se debe o no `decodificar` los datos que aparezcan en binario. Es especialmente útil cuando trabajamos con consultas que sólo devuelven una columna de datos.

- `sqlite_has_more(cursor)`: Devuelve `true` si quedan más filas que procesar en el `cursor` proporcionado o `false` en caso contrario.

- `sqlite_num_rows(cursor)`: Devuelve el número de filas almacenadas en el `cursor` proporcionado. No funciona con los cursores devuletos por `sqlite_unbuffered_query()`.

- `sqlite_num_fields(cursor)`: Devuelve el número de columnas o campos recuperados en el `cursor` proporcionado.

- `sqlite_field_name(cursor, índice)`: Devuelve el nombre de la columna o campo cuyo `índice` se proporciona.

- `sqlite_changes(bbdd)`: Devuelve el número de filas que se modificaron en la última consulta ejecutada sobre la base de datos referenciada por `bbdd`.

El siguiente ejemplo muestra el uso de algunas de estas funciones para recuperar la información almacenada en la base de datos (`pruebas_sqlite.db`) cuya una única tabla (`tornillos`) tiene la siguiente estructura:

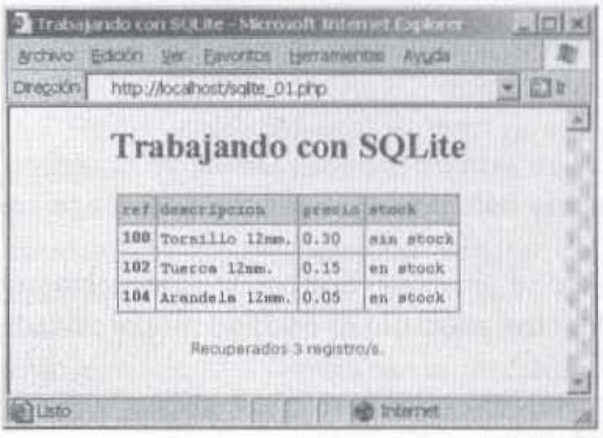
Campo	Estructura	Descripción
REF	*SMALLINT UNSIGNED*	Referencia de la pieza.
DESCRIPCION	*CHAR(30)*	Descripción básica de la pieza.
PRECIO	*FLOAT(8,2)*	Precio en Euros.
STOCK	*CHAR(30)*	Información sobre el *stock* en almacén.

El *script* es el siguiente:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con SQLite</TITLE>
   <STYLE>
      TABLE, P, A {font: 14px "monospace"}
      P {font: 12px "Verdana"; color: red;}
   </STYLE>
 </HEAD>
```

```
<BODY>
  <CENTER>
  <H2>Trabajando con SQLite</H2>
 <?php
    if (!$bd = sqlite_open('pruebas_sqlite.db', 0666, $error))
       die($error);
    $datos = sqlite_query($bd,'SELECT * FROM tornillos');
    $num_filas = sqlite_num_rows($datos);
    if($num_filas!=0)  {
       echo '<TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0">';
       echo '<TR BGCOLOR="yellow">';
      for($i=0;$i<sqlite_num_fields($datos);$i++){
         echo "<TD>",sqlite_field_name($datos,$i),"</TD>";
      }
       echo '</TR>';
       while(sqlite_has_more($datos)){
          $pieza=sqlite_fetch_array($datos,SQLITE_ASSOC);
          echo "<TR>";
          echo "<TD><B>$pieza['ref']</B></TD>";
          echo "<TD>$pieza['descripcion']</TD>";
          echo "<TD>$pieza['precio']</TD>";
          echo "<TD>$pieza['stock']</TD>";
          echo "</TR>";
       }
       echo "</TABLE>";
       echo "<P>Recuperados $num_filas registro/s.</P>";
    } else {
       echo"<P>La consulta no devuelve ningún registro</P>";
    }
    sqlite_close($bd);
 ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



El siguiente conjunto de funciones nos permite movernos por los datos presentes en los manejadores devueltos por la función `sqlite_query()`:

- `sqlite_seek(datos, num_fila)`: Permite situarnos en la `num_fila` deseada del manejador de `datos` proporcionado.

- `sqlite_rewind(datos)`: Permite situarnos en la primera fila del manejador de `datos` proporcionado.

- `sqlite_next(datos)`: Permite situarnos en la siguiente fila del manejador de `datos` proporcionado.

Si sustituimos el bucle `while` del ejemplo anterior por la siguiente porción de código, en la que se hace uso de estas funciones, obtendríamos el mismo resultado:

```
sqlite_rewind($datos);
while(sqlite_has_more($datos)){
   $pieza=sqlite_current($datos,SQLITE_ASSOC);
    echo "<TR>";
    echo "<TD><B>$pieza['ref']</B></TD>";
    echo "<TD>$pieza['descripcion']</TD>";
    echo "<TD>$pieza['precio']</TD>";
    echo "<TD>$pieza['stock']</TD>";
    echo "</TR>";
   sqlite_next($datos);
}
```

Con el siguiente código también se obtiene el mismo resultado:

```
for($fila=0;$fila<$num_filas;$fila++){
   sqlite_seek($datos,$fila);
   $pieza=sqlite_current($datos,SQLITE_ASSOC);
    echo "<TR>";
    echo "<TD><B>$pieza['ref']</B></TD>";
    echo "<TD>$pieza['descripcion']</TD>";
    echo "<TD>$pieza['precio']</TD>";
    echo "<TD>$pieza['stock']</TD>";
    echo "</TR>";
}
```

Para el control de errores específicos de bases de datos SQLite, PHP cuenta con las dos funciones siguientes:

- `sqlite_last_error(bbdd)`: Devuelve el código del último error que se haya producido trabajando con la base de datos referenciada por `bbdd`.

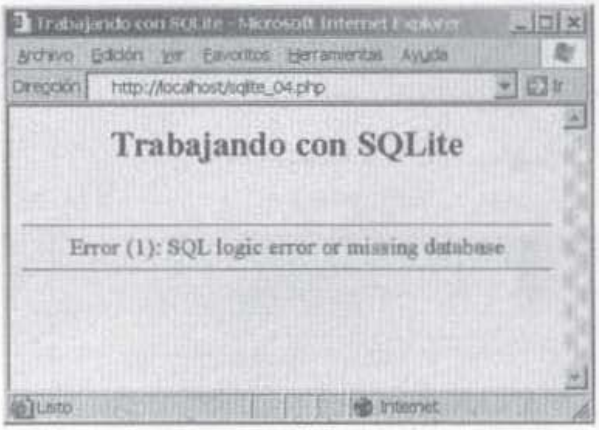
- `sqlite_error_string(código)`: Devuelve una cadena con la descripción del error asociado al `código` proporcionado.

El siguiente ejemplo nos muestra cómo utilizar estas funciones para generar un completo mensaje de error:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con SQLite</TITLE>
   <STYLE>
      TABLE, P, A {font: 14px "monospace"}
      P {font: 12px "Verdana"; color: red;}
   </STYLE>
```

```
</HEAD>
<BODY>
  <CENTER>
  <H2>Trabajando con SQLite</H2>
 <?php

    if (!$bd = sqlite_open('pruebas_sqlite.db', 0666, $error))
       die($error);
    if($datos = @sqlite_query($bd, "SELECT * FROM")){    // error
       echo "<P>Consulta correcta...</P>";
    } else {
       $cod_error = sqlite_last_error($bd);
       $str_error = sqlite_error_string($cod_error);
       echo "<br><hr><span style='color:red;'>";
       echo "Error ($cod_error): $str_error</span><hr>";
    }
 ?>
  </CENTER>
 </BODY>
</HTML>
```

Como podemos observar del código, la sentencia SQL que se solicita está incompleta y como resultado genera un error que se visualiza en la siguiente imagen:



Además de todas las funciones anteriores, PHP proporciona las siguientes funciones de carácter genérico:

- `sqlite_libversion()`: Devuelve la versión de la librería SQLite actualmente utilizada.
- `sqlite_libencoding()`: Devuelve la codificación de la librería SQLite actualmente utilizada.
- `sqlite_udf_decode_binary(cadena)`: Decodifica los datos binarios pasados en la `cadena` proporcionada a UDF.
- `sqlite_udf_encode_binary(cadena)`: Codifica los datos binarios pasados en la `cadena` proporcionada a UDF.

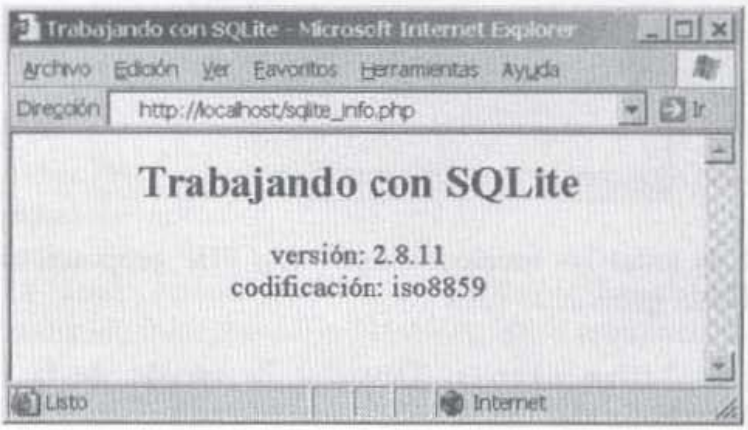
- `sqlite_escape_string(cadena)`: Enmascara la `cadena` proporcionada para utilizarla como parámetro de una consulta.

- `sqlite_busy_timeout(bbdd, milisegundos)`: Fija el máximo tiempo en `milisegundos` que se debe esperar para que la base de datos referenciada por `bbdd` se encuentre disponible. Superado este tiempo se genera el correspondiente error.

El siguiente ejemplo nos muestra cómo utilizar alguna de estas funciones:

```
<HTML>
  <HEAD>
    <TITLE>Trabajando con SQLite</TITLE>
  </HEAD>
  <BODY>
   <CENTER>
   <H2>Trabajando con SQLite</H2>
   <?php
       echo 'versión: ',sqlite_libversion(),"<br>";
       echo 'codificación: ',sqlite_libencoding(),"<br>";
   ?>
   </CENTER>
  </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



11.5.2 Interfaz orientada a objetos de SQLite

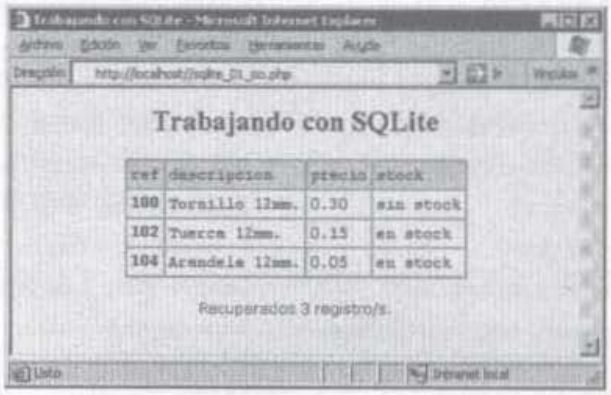
La extensión de SQLite nos permite trabajar como si se tratara de un entorno orientado a objetos. De este modo, la conexión a una base de datos se convierte en un objeto sobre el que podremos invocar métodos o acceder a sus propiedades. La siguiente tabla muestra los nombres de algunas de las funciones principales de SQLite y sus equivalencias en formato orientado a objetos:

Formato Procedural	Formato OO
`$bd = sqlite_open($tabla)`	`$bd = new SQLiteDatabase($tabla)`
`sqlite_close($bd)`	`unset($bd)`
`$rs = sqlite_query($bd, $sql)`	`$rs = $bd->query($sql)`
`$rs = sqlite_query_array($bd, $sql)`	`$rs = $bd->arrayQuery($sql)`
`$rs = sqlite_query_unbuffered($bd,sql)`	`$rs = $bd->unbufferedQuery($sql)`
`sqlite_fetch_array($rs)`	`$rs->fetch( )`
`sqlite_fetch_single($rs)`	`$rs->fetchSingle( )`
`$cad = sqlite_escape_string($s)`	`$cad = $bd->escapeString($s)`
`$num = sqlite_last_insert_rowid($rs)`	`$num = $bd->lastInsertRowid($rs)`

El siguiente ejemplo nos muestra cómo recuperar la información de la tabla `tornillos` haciendo uso de la interfaz orientada a objetos de SQLite:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con SQLite</TITLE>
   <STYLE>
      TABLE, P, A {font: 14px "monospace"}
      P {font: 12px "Verdana"; color: red;}
   </STYLE>
 </HEAD>
 <BODY>
   <CENTER>
   <H2>Trabajando con SQLite</H2>
  <?php
     if (!$bd = new SQLiteDatabase('pruebas_sqlite.db', 0666,$error))
        die($error);
     $datos = $bd->query("SELECT * FROM tornillos");
     $num_filas = $datos->numRows();
     if($num_filas!=0)  {
        echo '<TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0">';
        echo '<TR BGCOLOR="yellow">';
       for($i=0;$i<$datos->numFields();$i++){
          echo "<TD>",$datos->fieldName($i),"</TD>";
       }
        echo '</TR>';
        while($pieza=$datos->fetch()){
           echo "<TR>";
           echo "<TD><B>$pieza[ref]</B></TD>";
           echo "<TD>$pieza[descripcion]</TD>";
           echo "<TD>$pieza[precio]</TD>";
           echo "<TD>$pieza[stock]</TD>";
           echo "</TR>";
        }
        echo "</TABLE>";
        echo "<P>Recuperados $num_filas registro/s.</P>";
     } else {
        echo"<P STYLE='background-color:yellow;'>
              La consulta no devuelve ningún registro</P>";
     }
     unset ($bd);
 ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



11.5.3 Diferencias entre SQLite y MySQL

Tal y como hemos comentado las funciones proporcionadas por MySQL y SQLite tienen nombres muy parecidos, pero no idénticos. La siguiente tabla muestra los nombres de las funciones principales de ambos sistemas:

SQLite	MySQL
`sqlite_connect()`	`mysql_connect()`
`sqlite_close()`	`mysql_close()`
`sqlite_query()`	`mysql_query()`
`sqlite_fetch_array()`	`mysql_fetch_row()`
	`mysql_fetch_assoc()`
`sqlite_num_rows()`	`mysql_num_rows()`
`sqlite_last_inserted_row_id()`	`mysql_insert_id()`
`sqlite_escape_string()`	`mysql_real_escape_string()`

11.5.4 Ejemplo completo con SQLite

Este apartado muestra un ejemplo completo de una herramienta para la gestión básica (altas, bajas, modificaciones y consultas) de la tabla `tornillos` vista en el ejemplo anterior.

Todos los *scripts* hacen uso del siguiente código (`conectar_BBDD.php`) que se encarga de abrir/crear la base de datos utilizada y que además define una función para mostrar los errores que se puedan producir:

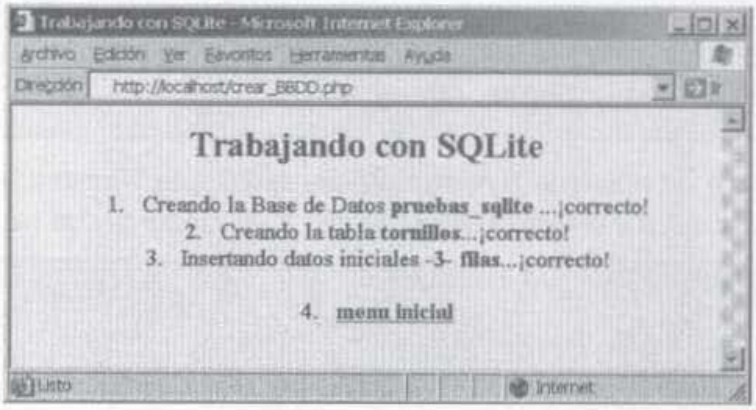
```
<?php
    function mostrar_error($bbdd){
      $cod_error = sqlite_last_error($bbdd);
      $str_error = sqlite_error_string($cod_error);
```

```
        return "<br><hr><span>Error ($cod_error): $str_error</span><hr>";
        }
    // si la base de datos no existe, se crea
    if (!$bd = sqlite_open('pruebas_sqlite.db', 0666))
        die(mostrar_error($bd));
?>
```

El primero de nuestros *scripts* (`crear_BBDD.php`) genera la base de datos (`pruebas_sqlite.bd`), la tabla (`tornillos`) y la inicializa con tres registros:

```
<HTML>
  <HEAD>
    <TITLE>Trabajando con SQLite</TITLE>
  </HEAD>
  <BODY>
   <CENTER>
   <H2>Trabajando con SQLite</H2>
   <ol>
     <li>Creando la Base de Datos <b>pruebas_sqlite</b>
   <?php
       include("conectar_bbdd.php");
       echo "...¡correcto!</li>";
       if (@sqlite_query($bd, "SELECT * FROM tornillos;"))
          sqlite_query($bd, "DROP TABLE tornillos;");
       $sql = "CREATE TABLE tornillos (
                  ref SMALLINT UNSIGNED NOT NULL,
                  descripcion CHAR(30),
                  precio FLOAT(8,2),
                  stock CHAR(30),
              PRIMARY KEY(ref));";
       echo "<li>Creando la tabla <b>tornillos</b>";
       if (!sqlite_query($bd, $sql))
          die(mostrar_error($bd));
       echo "...¡correcto!</li>";
       $sql =  'INSERT INTO tornillos
                     VALUES (100,"Tornillo 12mm.",0.30,"sin stock");';
       if (!sqlite_query($bd, $sql))
          die(mostrar_error($bd));
       $sql = 'INSERT INTO tornillos
                     VALUES (102,"Tuerca 12mm.",0.15,"en stock");';
       if (!sqlite_query($bd, $sql))
          die(mostrar_error($bd));
       $sql = 'INSERT INTO tornillos
                     VALUES (104,"Arandela 12mm.",0.05,"en stock");';
       echo "<li>Insertando datos iniciales";
       if (!sqlite_query($bd, $sql))
          die(mostrar_error($bd));
       echo "<b> -3- filas</b>...¡correcto!</li>";
       sqlite_close($bd);
   ?>
     <br><br><li><a href="menu_BBDD.html"><b>menu inicial</b></a></li>
   </ol>
   </CENTER>
  </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



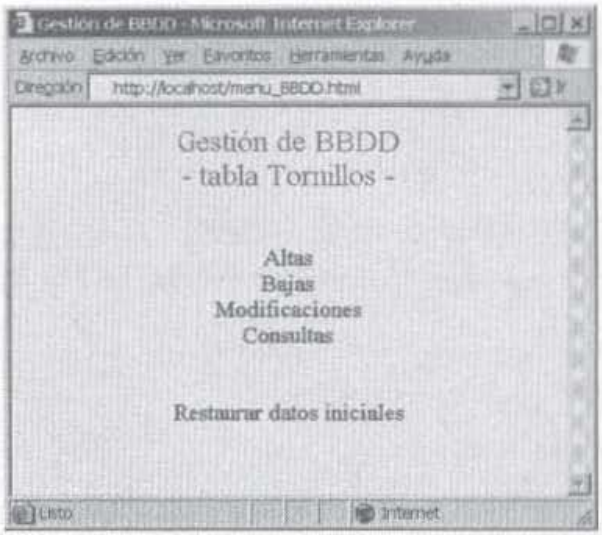
Después de la ejecución, la tabla `tornillos` tendrá el siguiente contenido:

ref	descripcion	precio	stock
100	Tornillo 12mm.	0.30	sin stock
102	Tuerca 12mm.	0.15	en stock
104	Arandela 12mm.	0.05	en stock

La siguiente página HTML muestra el menú de opciones que proporciona la herramienta:

```
<HTML>
 <HEAD>
   <TITLE>Gestión de BBDD</TITLE>
   <STYLE>
      H2 {color: gray; font: 26px "monoespace";}
      A {text-decoration: none; color: blue; font: 20px "monoespace";}
      A:hover {background-color: yellow;}
   </STYLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Gestión de BBDD<BR>- tabla Tornillos -</H2><BR>
     <A HREF="alta1_BBDD.html">Altas</A><BR>
     <A HREF="baja1_BBDD.php">Bajas</A><BR>
     <A HREF="modi_BBDD.php">Modificaciones</A><BR>
     <A HREF="con1_BBDD.php">Consultas</A><BR>
     <BR><BR><A HREF="crear_BBDD.php">Restaurar datos iniciales</A><BR>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



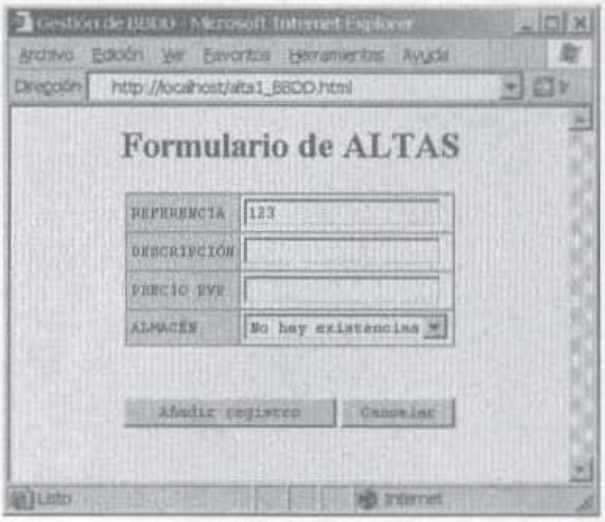
El proceso de altas se compone de dos *scripts*, el primero de ellos es una página HTML (`alta1_BBDD.html`) que muestra un formulario en el que se introducirán los datos de la nueva fila a insertar en la tabla:

```
<HTML>
 <HEAD>
   <TITLE>Gestión de BBDD</TITLE>
   <STYLE>
      TABLE, A, INPUT, SELECT {font: 14px "monospace"}
   </STYLE>
   <SCRIPT LANGUAGE="Javascript">
   <!--
   function comprobar_ref(){
       if(document.forms[0].REF.value=="" ||
          isNaN(document.forms[0].REF.value)){
              alert("Debes codificar una referencia válida...");
              document.forms[0].REF.value="";
              document.forms[0].REF.focus();
              return false;
       }
       if (document.forms[0].DESCRIPCION.value=="")
           document.forms[0].DESCRIPCION.value="Sin descripción";
       if (document.forms[0].PRECIO.value=="")
           document.forms[0].PRECIO.value="0.0";
       return true;
   }
   //-->
   </SCRIPT>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Formulario de ALTAS</H2>
     <FORM METHOD="POST" ACTION="alta2_BBDD.php"
                         ONSUBMIT="return comprobar_ref()">
     <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0">
        <TR>
           <TD BGCOLOR="yellow">REFERENCIA</TD>
           <TD>
              <INPUT TYPE="TEXT" NAME="REF">
           </TD>
        </TR>
```

```
        <TR>
          <TD BGCOLOR="yellow">DESCRIPCIÓN</TD>
          <TD>
            <INPUT TYPE="TEXT" NAME="DESCRIPCION" VALUE="">
          </TD>
        </TR>
        <TR>
          <TD BGCOLOR="yellow">PRECIO PVP</TD>
          <TD>
            <INPUT TYPE="TEXT" NAME="PRECIO" VALUE="">
          </TD>
        </TR>
        <TR>
          <TD BGCOLOR="yellow">ALMACÉN</TD>
          <TD>
            <SELECT NAME="STOCK">
             <OPTION VALUE="sin stock">No hay existencias</OPTION>
             <OPTION VALUE="en stock">Hay existencias</OPTION>
            </SELECT>
          </TD>
        </TR>
    </TABLE><BR><BR>
  <INPUT TYPE="SUBMIT" VALUE="Añadir registro">
  <INPUT TYPE="BUTTON" VALUE="Cancelar" ONCLICK="location='menu_BBDD.html';">
    </FORM>
  </CENTER>
 </BODY>
</HTML>
```

Como se puede observar, además de recoger la información introducida por el usuario, se realiza un control básico de errores con la función *javascript* `comprobar_ref()` que se encarga de comprobar que se ha introducido un código de referencia (`ref`) válido para la nueva pieza. También se encarga de fijar una descripción (`descripcion`), un precio y un valor de almacén (`stock`) por defecto.

La página HTML se visualiza de la siguiente forma:

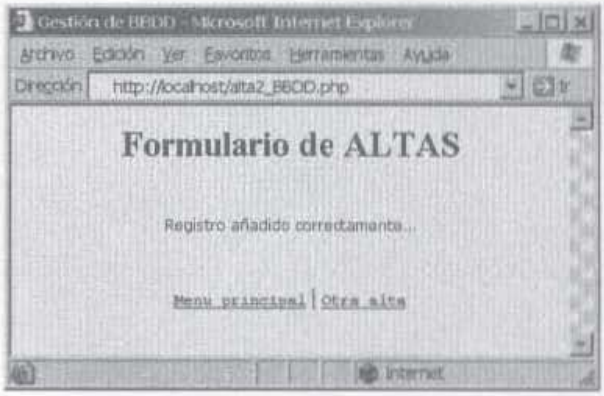


En caso de que la referencia proporcionada por el usuario no sea correcta se muestra el siguiente mensaje:

Por otra parte el proceso de altas también tiene el siguiente *script* (`alta2_BBDD.php`), encargado de añadir en la tabla los datos introducidos en la página anterior:

```
<HTML>
 <HEAD>
   <TITLE>Gestión de BBDD</TITLE>
   <STYLE>
      A {font: 14px "monospace"}
      P {font: 12px "Verdana"; color: red;}
   </STYLE>
 </HEAD>
 <BODY>
   <CENTER>
   <H2>Formulario de ALTAS</H2><BR>
   <?php
     include("conectar_bbdd.php");
     $sql="INSERT INTO TORNILLOS VALUES ($_POST[REF], '$_POST[DESCRIPCION]',
                                        $_POST[PRECIO], '$_POST[STOCK]');";
     if (!sqlite_unbuffered_query($bd, $sql))
       die(mostrar_error($bd));
     sqlite_close ($bd);
   ?>
   <P>Registro añadido correctamente...</P><BR>
   <A HREF="menu_BBDD.html">Menu principal</A> |
   <A HREF="alta1_BBDD.html">Otra alta</A>
   </CENTER>
 </BODY>
</HTML>
```

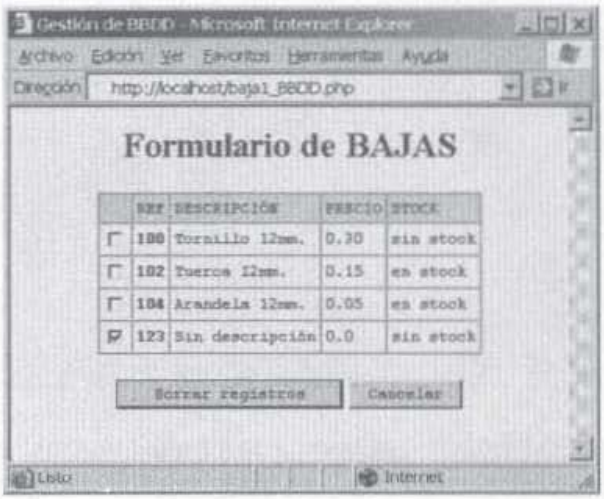
El resultado, si se ha realizado correctamente la inserción, se puede observar en la siguiente imagen:



Para realizar las bajas se utilizan de nuevo dos *scripts*, uno primero (`baja1_BBDD.php`) que muestra los registros que actualmente contiene la tabla en un formulario para que el usuario pueda decidir cuál eliminar:

```
<HTML>
 <HEAD>
   <TITLE>Gestión de BBDD</TITLE>
   <STYLE>
      TABLE, A, INPUT (font: 14px "monospace")
   </STYLE>
 </HEAD>
 <BODY>
   <?php include("conectar_bbdd.php"); ?>
   <CENTER>
     <H2>Formulario de BAJAS</H2>
     <FORM METHOD="POST" ACTION="baja2_BBDD.php">
     <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0">
        <TR BGCOLOR="yellow">
           <TD> </TD>
           <TD>REF</TD>
           <TD>DESCRIPCIÓN</TD>
           <TD>PRECIO</TD>
           <TD>STOCK</TD>
        </TR>
  <?php
     $sql="SELECT * FROM TORNILLOS";
     if (!$datos=sqlite_unbuffered_query($bd, $sql))
       die(mostrar_error($bd));
     while(sqlite_has_more($datos)){
       $pieza=sqlite_fetch_array($datos,SQLITE_ASSOC);
       echo "<TR>";
       echo "<TD ALIGN='center'>";
       echo "<INPUT TYPE='checkbox' NAME='registros[]' VALUE='$pieza[ref]'>";
       echo "</TD>";
       echo "<TD><B>$pieza['ref']</B></TD>";
       echo "<TD>$pieza['descripcion']</TD>";
       echo "<TD>$pieza['precio']</TD>";
       echo "<TD>$pieza['stock']</TD>";
       echo "</TR>";
     }
     sqlite_close ($bd);
   ?>
     </TABLE><BR>
     <INPUT TYPE="SUBMIT" VALUE="Borrar registros">
     <INPUT TYPE="BUTTON" VALUE="Cancelar"
                          ONCLICK="location='menu_BBDD.html';">
     </FORM>
   </CENTER>
 </BODY>
</HTML>
```

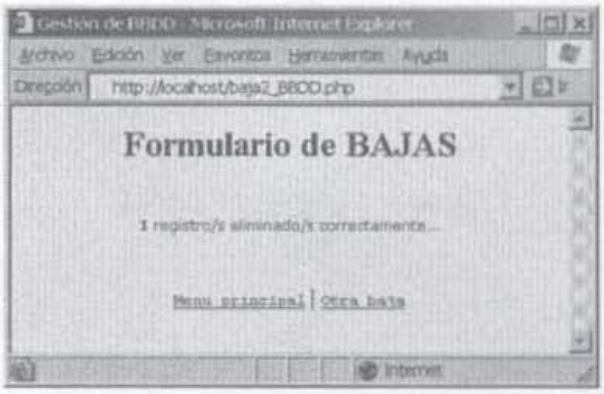
El resultado se muestra en la siguiente imagen:



Y un segundo *script* que realiza la baja de los registros seleccionados en el paso anterior:

```
<HTML>
 <HEAD>
   <TITLE>Gestión de BBDD</TITLE>
   <STYLE>
      A {font: 14px "monospace"}
      P {font: 12px "Verdana"; color: red;}
   </STYLE>
 </HEAD>
 <BODY>
   <?php include("conectar_bbdd.php"); ?>
   <CENTER>
   <H2>Formulario de BAJAS</H2><BR>
   <?php
     if (isset($_POST['registros'])>0) {
       $lista_bajas = join(",",$_POST['registros']);
       $sql = "DELETE FROM tornillos
                    WHERE ref IN($lista_bajas)";
       if (!$datos=sqlite_unbuffered_query($bd, $sql))
          die(mostrar_error($bd));
       echo "<P><B>".sqlite_changes($bd).
            "</B> registro/s eliminado/s correctamente...</P><BR>";
     } else
       echo "<P>No se ha seleccionado ningún registro...</P><BR>";
     sqlite_close ($bd);
   ?>
   <A HREF="menu_BBDD.html">Menu principal</A> |
   <A HREF="baja1_BBDD.php">Otra baja</A>
   </CENTER>
 </BODY>
</HTML>
```

El resultado, si se ha realizado correctamente la eliminación de algún registro, se puede observar en la siguiente imagen:

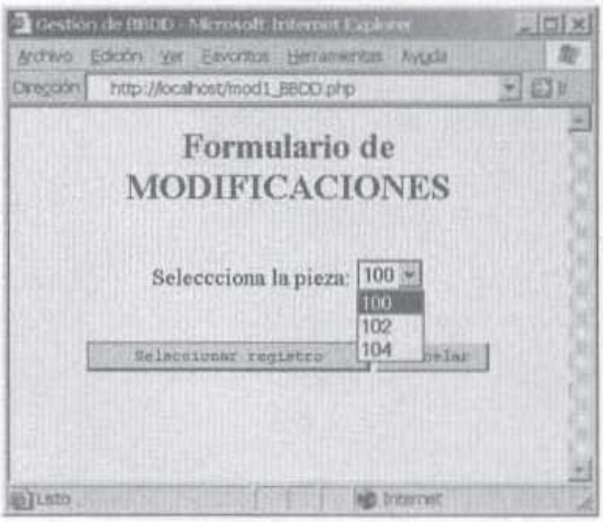


El proceso de modificaciones cuenta también con dos *scripts*. En el primero de ellos (`modl_BBDD.php`), se muestra una lista con los códigos de las piezas (`ref`) disponibles en la tabla para que el usuario elija el componente a modificar:

```
<HTML>
 <HEAD>
   <TITLE>Gestión de BBDD</TITLE>
   <STYLE>
      TABLE, INPUT {font: 14px "monospace"}
   </STYLE>
 </HEAD>
 <BODY>
   <?php include("conectar_bbdd.php"); ?>
   <CENTER>
     <H2>Formulario de MODIFICACIONES</H2>
   <?php
     if(empty($_GET['ref'])){

     $sql="SELECT ref FROM TORNILLOS";
     if (!$datos=sqlite_unbuffered_query($bd, $sql))
       die(mostrar_error($bd));
   ?>
        <FORM METHOD="GET" ACTION="modl_BBDD.php">
          <BR>Seleccciona la pieza:
          <SELECT NAME="ref">
           <?php
             while(sqlite_has_more($datos)){
                $pieza=sqlite_fetch_string($datos);
                echo "<OPTION VALUE='$pieza'>$pieza</OPTION>";
             }
           ?>
          </SELECT><BR><BR><BR>
          <INPUT TYPE="SUBMIT" VALUE="Seleccionar registro">
          <INPUT TYPE="BUTTON" VALUE="Cancelar"
                               ONCLICK="location='menu_BBDD.html';">
        </FORM>
   <?php
     } else {
       $sql="SELECT * FROM TORNILLOS WHERE ref = $_GET['ref']";
       if (!$pieza=sqlite_array_query($bd,$sql,SQLITE_ASSOC))
         die(mostrar_error($bd));
   ?>
     <FORM METHOD="POST" ACTION="mod2_BBDD.php">
     <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0">
       <TR>
         <TD BGCOLOR="yellow">REFERENCIA</TD>
```

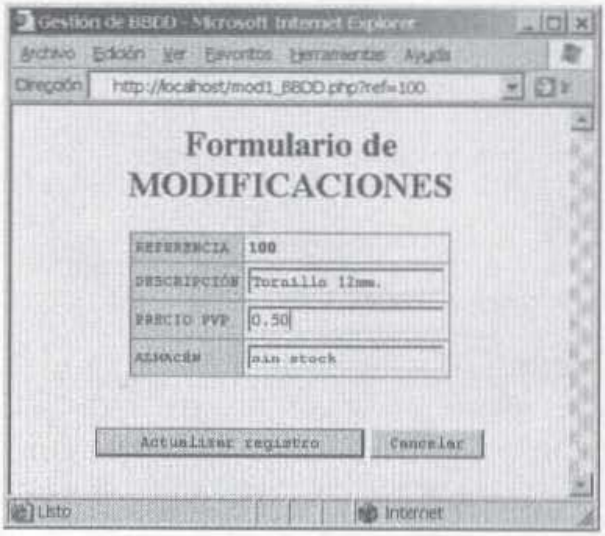
```
            <TD>
               <B><?php echo $pieza[0]['ref']?></B>
               <INPUT TYPE="hidden" NAME="REF"
                      VALUE="<?php echo $pieza[0]['ref']?>">
            </TD>
         </TR>
         <TR>
            <TD BGCOLOR="yellow">DESCRIPCIÓN</TD>
            <TD>
              <INPUT TYPE="TEXT" NAME="DESCRIPCION"
                     VALUE="<?php echo $pieza[0]['descripcion']?>">
            </TD>
         </TR>
         <TR>
            <TD BGCOLOR="yellow">PRECIO PVP</TD>
            <TD>
               <INPUT TYPE="TEXT" NAME="PRECIO"
                      VALUE="<?php echo $pieza[0]['precio']?>">
            </TD>
         </TR>
         <TR>
            <TD BGCOLOR="yellow">ALMACÉN</TD>
            <TD>
               <INPUT TYPE="TEXT" NAME="STOCK"
                      VALUE="<?php echo $pieza[0]['stock']?>">
            </TD>
         </TR>
      </TABLE><BR><BR>
      <INPUT TYPE="SUBMIT" VALUE="Actualizar registro">
      <INPUT TYPE="BUTTON" VALUE="Cancelar"
             ONCLICK="location='menu_BBDD.html';">
      </FORM>
   <?php  }   sqlite_close ($bd);  ?>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:



Este mismo *script* también se encarga de mostrar un formulario con la información de la pieza solicitada en el paso anterior, para que el usuario pueda

modificar cualquiera de sus campos, menos el campo clave (`ref`). El resultado se muestra en la siguiente imagen:



El segundo *script* (`mod2_BBDD.php`), es el encargado de actualizar la tabla con las modificaciones realizadas por el usuario en el paso anterior:

```
<HTML>
 <HEAD>
   <TITLE>Gestión de BBDD</TITLE>
   <STYLE>
     A {font: 14px "monospace"}
     P {font: 12px "Verdana"; color: red;}
   </STYLE>
 </HEAD>
 <BODY>
   <CENTER>
   <H2>Formulario de MODIFICACIONES</H2><BR>
   <?php
     include("conectar_bbdd.php");
     $sql="UPDATE TORNILLOS SET
             DESCRIPCION='$_POST[DESCRIPCION]',
             PRECIO=$_POST[PRECIO],
             STOCK='$_POST[STOCK]'
           WHERE REF=$_POST[REF]";
     if (!sqlite_unbuffered_query($bd, $sql))
       die(mostrar_error($bd));
     sqlite_close ($bd);
   ?>
   <P>Registro actualizado correctamente...</P><BR>
   <A HREF="menu_BBDD.html">Menu principal</A> |
   <A HREF="mod1_BBDD.php">Otra modificación</A>
   </CENTER>
 </BODY>
</HTML>
```

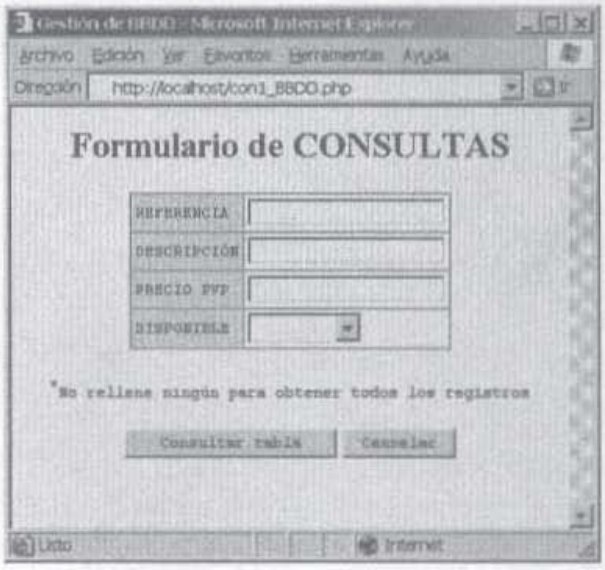
El resultado, si se ha realizado correctamente la modificación del registro seleccionado, se puede observar en la siguiente imagen:

Finalmente, el proceso de consultas también cuenta con dos *scripts*. El primero de ellos (`con1_BBDD.php`), muestra al usuario un formulario en el que se pueden ajustar algunos de los parámetros de la consulta. También se encarga de construir una consulta SQL correcta, que almacena en una variable de sesión, a partir de dichos parámetros:

```
<HTML>
 <HEAD>
   <TITLE>Gestión de BBDD</TITLE>
   <STYLE>
      TABLE, A, P, INPUT {font: 14px "monospace"}
   </STYLE>
 </HEAD>
 <BODY>
   <CENTER>
     <H2>Formulario de CONSULTAS</H2>
<?php
  include("conectar_bbdd.php");
  if(isset($_POST['REF'])){
   $consultaSQL = "SELECT * FROM TORNILLOS";
   $condicion = "";
   if ($_POST['REF']!="")
      $condicion .= "(REF=$_POST[REF])";
   if ($_POST['DESCRIPCION']!=""){
      if ($condicion!="") $condicion .= " OR ";
      $condicion .= "(DESCRIPCION LIKE '$_POST[DESCRIPCION]')";
   }
   if ($_POST['PRECIO']!=""){
      if ($condicion!="") $condicion .= " OR ";
      $condicion .= "(PRECIO=$_POST[PRECIO])";
   }
   if ($_POST['STOCK']!=""){
      if ($condicion!="") $condicion .= " OR ";
      $condicion .= "(STOCK LIKE '$_POST[STOCK]')";
   }
   if ($condicion!="") $consultaSQL .= " WHERE $condicion";

   session_start();
   $_SESSION['consultaSQL']=$consultaSQL;
   header("Location: con2_BBDD.php");

} else { ?>
     <FORM METHOD="POST" ACTION="con1_BBDD.php">
     <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0">
```

```
        <TR>
          <TD BGCOLOR="yellow">REFERENCIA</TD>
          <TD>
            <INPUT TYPE="TEXT" NAME="REF">
          </TD>
        </TR>
        <TR>
          <TD BGCOLOR="yellow">DESCRIPCIÓN</TD>
          <TD>
            <INPUT TYPE="TEXT" NAME="DESCRIPCION">
          </TD>
        </TR>
        <TR>
          <TD BGCOLOR="yellow">PRECIO PVP</TD>
          <TD>
            <INPUT TYPE="TEXT" NAME="PRECIO">
          </TD>
        </TR>
        <TR>
          <TD BGCOLOR="yellow">DISPONIBLE</TD>
          <TD>
            <SELECT NAME="STOCK">
             <OPTION></OPTION>
             <OPTION>sin stock</OPTION>
             <OPTION>en stock</OPTION>
            </SELECT>
          </TD>
        </TR>
      </TABLE>
      <P><SUP>*</SUP>No rellene ningún para obtener todos los registros</P>
      <INPUT TYPE="SUBMIT" VALUE="Consultar tabla">
      <INPUT TYPE="BUTTON" VALUE="Cancelar"
                           ONCLICK="location='menu_BBDD.html'">
    </FORM>
  <?php } ?>
  </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:

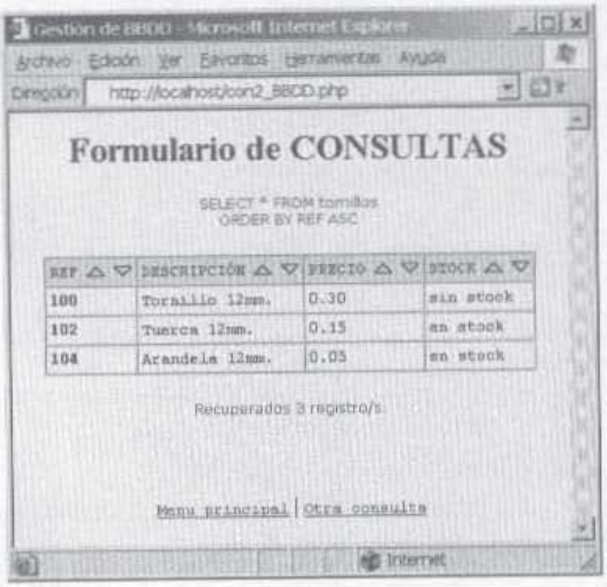


Por su parte el segundo *script* (`con2_BBDD.php`), se encarga de ejecutar la consulta SQL construida en el paso anterior y mostrar los resultados obtenidos. En caso de que el usuario no haya ajustado, en el paso anterior, ningún parámetro de consulta se mostrarán todos los registros presentes en la tabla:

```
<HTML>
 <HEAD>
   <TITLE>Gestión de BBDD</TITLE>
   <STYLE>
      TABLE, P, A {font: 14px "monospace"}
      P {font: 12px "Verdana"; color: red;}
   </STYLE>
 </HEAD>
 <BODY>
   <CENTER>
   <H2>Formulario de CONSULTAS</H2>
  <?php
     include("conectar_bbdd.php");

     session_start();
     $consultaSQL_txt=$_SESSION['consultaSQL'];

     if(empty($_GET['campo'])) $_GET['campo']="REF";
     if(empty($_GET['orden'])) $_GET['orden']="ASC";
     $consultaSQL_orden=" ORDER BY ".$_GET['campo']." ".$_GET['orden'];

     $consultaSQL=$consultaSQL_txt.$consultaSQL_orden;
     if (!$datos=sqlite_query($bd, $consultaSQL))
       die(mostrar_error($bd));

     $consultaSQL_base="SELECT * FROM tornillos";
     if($condiciones=strstr($consultaSQL_txt,"WHERE")){
        echo "<P>$consultaSQL_base
              <br>$condiciones<br>$consultaSQL_orden</P>";
     } else
        echo "<P>$consultaSQL_base<br>$consultaSQL_orden</P>";

     $num_registros = sqlite_num_rows($datos);
     if($num_registros!=0)  {
 ?>
     <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0">
        <TR BGCOLOR="yellow">
           <TD>REF
              <A HREF="con2_bbdd.php?campo=REF&orden=ASC">
                <IMG SRC="up.gif" BORDER="0"></A>
              <A HREF="con2_bbdd.php?campo=REF&orden=DESC">
                <IMG SRC="down.gif" BORDER="0"></A>
           </TD>
           <TD>DESCRIPCIÓN
              <A HREF="con2_bbdd.php?campo=DESCRIPCION&orden=ASC">
                <IMG SRC="up.gif" BORDER="0"></A>
              <A HREF="con2_bbdd.php?campo=DESCRIPCION&orden=DESC">
                <IMG SRC="down.gif" BORDER="0"></A>
           </TD>
           <TD>PRECIO
              <A HREF="con2_bbdd.php?campo=PRECIO&orden=ASC">
                <IMG SRC="up.gif" BORDER="0"></A>
              <A HREF="con2_bbdd.php?campo=PRECIO&orden=DESC">
                <IMG SRC="down.gif" BORDER="0"></A>
           </TD>
```

```
            <TD>STOCK
               <A HREF="con2_bbdd.php?campo=STOCK&orden=ASC">
                  <IMG SRC="up.gif" BORDER="0"></A>
               <A HREF="con2_bbdd.php?campo=STOCK&orden=DESC">
                  <IMG SRC="down.gif" BORDER="0"></A>
            </TD>
         </TR>
<?php
    while(sqlite_has_more($datos)){
       $pieza=sqlite_fetch_array($datos,SQLITE_ASSOC);
       echo "<TR>";
       echo "<TD><B>$pieza[ref]</B></TD>";
       echo "<TD>$pieza[descripcion]</TD>";
       echo "<TD>$pieza[precio]</TD>";
       echo "<TD>$pieza[stock]</TD>";
       echo "</TR>";
    }
    echo "</TABLE>";
    echo "<P>Recuperados $num_registros registro/s.</P>";
 } else
      echo"<P>La consulta no devuelve ningún registro</P>";
 sqlite_close ($bd);
?>
  <BR><BR><A HREF="menu_BBDD.html">Menu principal</A> |
  <A HREF="con1_BBDD.php">Otra consulta</A>
  </CENTER>
</BODY>
</HTML>
```

Como se puede observar en la siguiente imagen, este *script* también permite que el usuario reordene los datos obtenidos por el campo que desee:



11.5.5 Instalación en Unix/Linux

Aunque la librería para utilizar SQLite desde PHP viene incluida por defecto en la distribución de PHP 5.0 en ocasiones puede ser interesante instalarla por

separado, puesto que ante nuevas versiones del producto podremos actualizarlas sin necesidad de modificar la instalación de PHP. Como complemento a realizar la instalación por separado contaremos con un programa que permite la manipulación de la base de datos desde la línea de comandos (`/usr/local/bin/sqlite`).

Para instalar SQLite sobre un sistema Unix/Linux, simplemente hay que seguir los siguientes pasos:

```
$ tar xzf sqlite-2.8.15.tar.gz
$ mkdir bld
$ cd bld
$ ../sqlite/configure
$ make
$ su
# make install
```

Es decir, desempaquetar el producto (en formato *tarball*), crear un directorio para la compilación, ejecutar la configuración desde el directorio de compilación, compilar la distribución y, como superusuario, hacer la instalación.

Posteriormente, hay que indicarle a PHP que haga uso de la nueva librería. Para ello, al comando de configuración hay que añadirle la opción `--with-sqlite=/usr/local`, y seguir los mismos pasos que vimos en el capítulo de instalación.

```
$ ./configure --with-sqlite=/usr/local \
              resto_de_opciones_deseadas
```

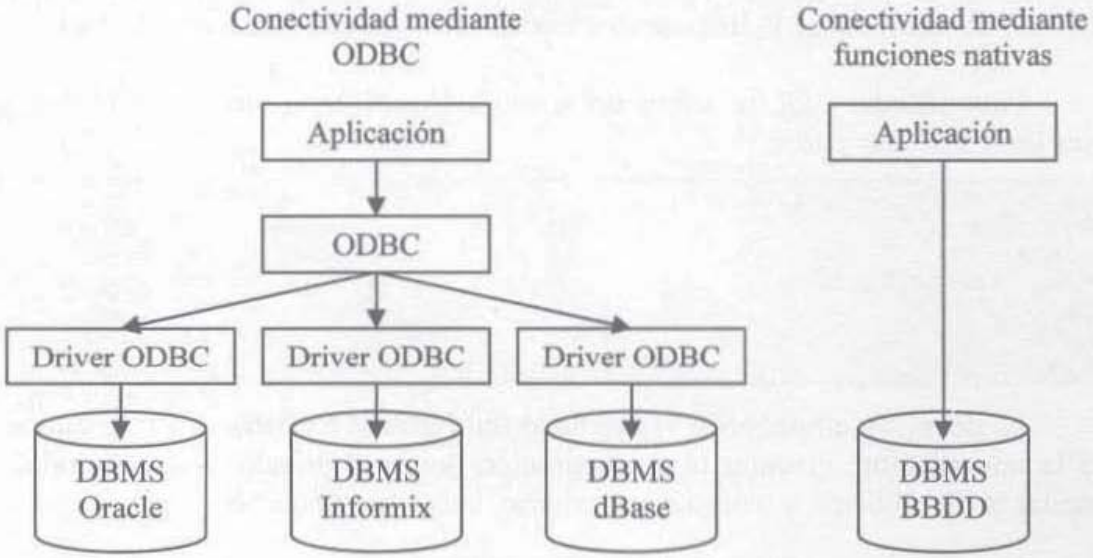
Si lo prefiere también se puede compilar `SQLite` realizando las modificaciones que desee en el fichero "`Makefile.linux-gcc`" y ejecutando el `makefile`. Este es el método utilizado para todos los desarrollos y pruebas oficiales de `SQLite` y para construir los ficheros binarios precompilados que se encuentran en la web. Los ficheros binarios de Windows son generados utilizando el compilador cruzado de Linux, MinGW.

11.6 USO DE ODBC

A lo largo de este capítulo hemos visto algunas de las funciones (las más frecuentemente utilizadas) que PHP proporciona para trabajar con MySQL; de igual forma existen otras muchas que nos dan soporte para diferentes bases de datos: Adabas D, Ingres, Oracle (OCI7 y OCI8), dBase, InterBase, Ovrimos, Empress, FrontBase, PostgreSQL, mSQL, Solid, Hyperwave, Direct MS-SQL, Sybase, IBM DB2, Informix y Unix dbm, entre otras.

En líneas generales, dicho soporte se implementa de dos modos diferenciados, o bien, mediante funciones nativas propias de cada uno de los diferentes gestores de bases de datos (tal y como hemos visto para MySQL), o bien,

mediante ODBC (*Open DataBase Conectivity*), la API estándar de conectividad de base de datos desarrollada por Microsoft.



ODBC presenta un nivel intermedio de *software*, un conjunto de llamadas a alto nivel, que permite el intercambio de instrucciones y datos con cualquier gestor de base de datos sin la necesidad de conocer sus detalles de implementación. Para utilizar ODBC, sólo es necesario que la base de datos (la fuente de datos ODBC puede incluso no ser una base de datos: una hoja de cálculo, un editor de texto, etc.) tenga el *driver* apropiado.

ODBC, por tanto, presenta un modo homogéneo de manipulación de datos independiente de gestor específico (DBMS) que se esté utilizando. Además, también es independiente del sistema operativo sobre el que se esté ejecutando nuestra aplicación favoreciendo, de este modo, la independencia y transportabilidad del código generado respecto del entorno de ejecución. La interfaz definida por ODBC se basa principalmente en los siguientes elementos:

- Una librería de llamadas a funciones ODBC que nos permiten conectarnos a un gestor de base de datos, ejecutar sentencias SQL y recuperar resultados.

- Una representación estándar de los diferentes tipos de datos a manejar.

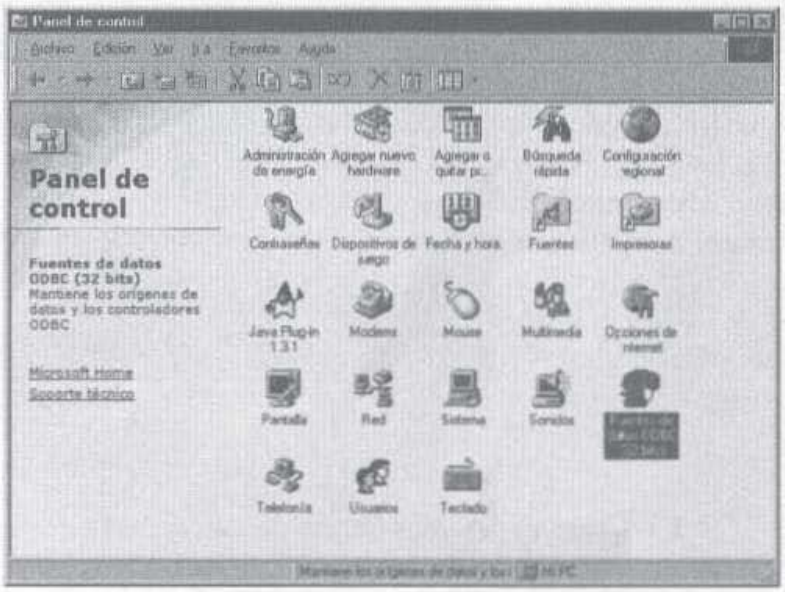
La contrapartida más importante que presenta ODBC es la sobrecarga que supone el uso de esta capa intermedia de *software*. Cuando hacemos uso de las funciones propias de cada gestor, obtenemos un método de acceso mucho más eficiente, si bien perdemos la transparencia proporcionada por ODBC.

11.6.1 Ejemplo de uso sobre Access

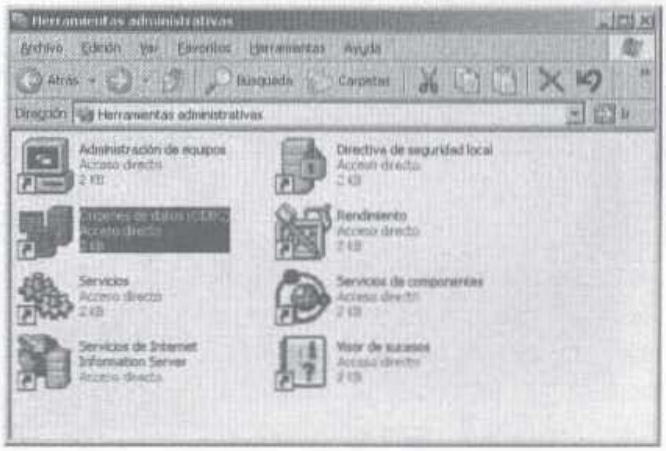
Los pasos para utilizar ODBC para conectarnos a una base de datos definida con Access serán los siguientes:

1. Arrancar el *Administrador de ODBC*. Este componente de todos los sistemas operativos de la familia Windows nos permite administrar las fuentes de datos y los controladores ODBC.

   a. Normalmente se encuentra ubicado directamente dentro del *Panel de control*, en los sistemas Windows 95/98.



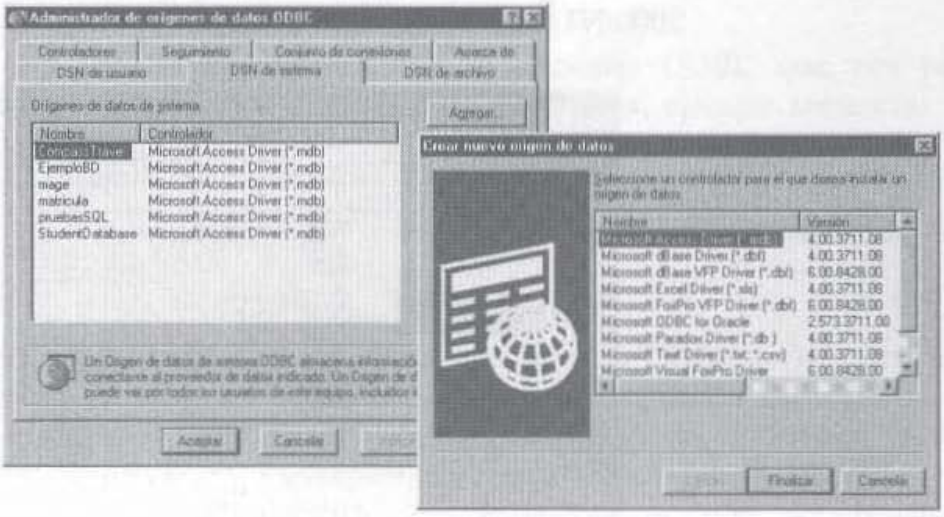
   b. O bien dentro de las *Herramientas administrativas* accesibles desde el *Panel de control*, en sistemas Windows 2000/NT/XP.



2. Comprobar que nuestro sistema tiene instalado el *driver* ODBC necesario, en nuestro caso el de Access. Para ello accederemos al contenido de la pestaña *Controladores* y comprobaremos que el *driver* necesario se encuentra instalado en nuestro sistema. En caso de que no esté, sería necesario instalarlo. Los *drivers* ODBC habitualmente los proporciona el constructor de la base de datos.

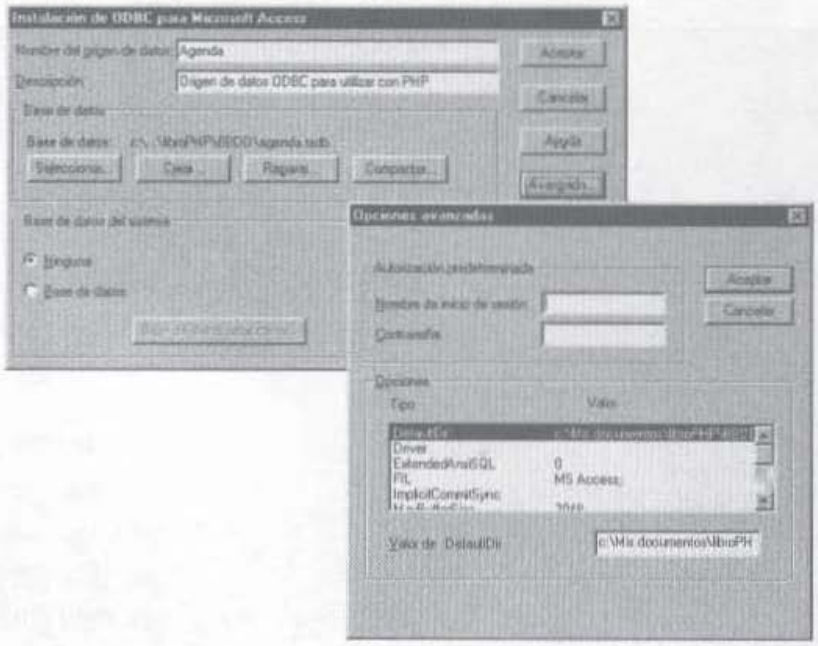
NOTA: El controlador de ODBC de Microsoft Access se proporciona por defecto en todas las versiones de Windows. En caso de no tenerlo disponible, se puede instalar desde el CD de Office o descargarlo desde el sitio Web de Microsoft.

3. Crear una nueva fuente de datos desde el Administrador de ODBC. Para ello mostraremos el contenido de la pestaña *DSN de sistema* (el administrador del sistema, o cualquier usuario que tenga privilegios puede utilizar un origen de datos configurado con un DSN de sistema) y agregaremos un nuevo origen de datos.



4. Asociar a la nueva fuente de datos ODBC una base de datos.

5. Configurar el origen de datos ODBC recién creado, fijando los valores requeridos para las opciones disponibles mostradas en la ventana de diálogo de *Opciones avanzadas*. Entre otros valores, podemos fijar el usuario y la contraseña necesarios para iniciar una sesión con nuestra base de datos.



6. Hacer uso de las funciones proporcionadas por PHP para la conectividad con fuentes de datos ODBC.

En nuestro caso el código desarrollado para la versión de la Agenda que utiliza MySQL es totalmente válido. Sólo deberíamos cambiar las llamadas a las funciones nativas de MySQL por sus correspondientes llamadas ODBC. La

siguiente tabla muestra las funciones ODBC equivalentes a las funciones nativas de MySQL que hemos utilizado a lo largo del capítulo:

funciones nativas MySQL	funciones ODBC
mysql_connect()	odbc_connect()
mysql_query()	odbc_do()
mysql_fetch_row()	odbd_fetch_row()
mysql_free_result()	odbc_free_result()
mysql_close()	odbc_close()

La siguiente tabla muestra las correspondencias entre las funciones de SQLite y ODBC:

funciones nativas SQLite	funciones ODBC
sqlite_connect()	odbc_connect()
sqlite_query()	odbc_do()
sqlite_fetch_array()	odbd_fetch_row()
sqlite_close()	odbc_close()

A modo de ejemplo se muestra el *script* encargado de realizar la conexión con la base de datos y mostrar un listado con todos los registros:

```
<HTML>
 <HEAD>
   <TITLE>Trabajando con ODBC</TITLE>
   <STYLE>
      TABLE, P, A {font: 14px "monospace"}
      P {font: 12px "Verdana"; color: red;}
   </STYLE>
 </HEAD>
 <BODY>
   <CENTER>
   <H2>Trabajando con ODBC</H2>
   <table border="1" cellpadding="2">
     <tr bgcolor="yellow">
      <th>Nombre</th>
      <th>Correo-e</th>
      <th>Tlf Fijo</th>
      <th>Tlf Móvil</th>
    </tr>

<?

// me conecto a la base de datos
$dbd = odbc_connect("Agenda", "un_usr", "una_clave");

// construyo la consulta
$sql = "SELECT * FROM la_agenda";

// ejecuto la consulta
$res = odbc_do($dbd, $sql);

// recorro el resultado
while(odbc_fetch_row($res)) {
```

```
  $nombre    = odbc_result($res, 1);
  $correoe   = odbc_result($res, 2);
  $tlf_fijo  = odbc_result($res, 3);
  $tlf_movil = odbc_result($res, 4);

  echo "<tr>
        <td>$nombre</td>
        <td>$correoe</td>
        <td>$tlf_fijo</td>
        <td>$tlf_movil</td>
        </tr>";
}

// cierro la conexion con la base de datos
odbc_close($dbd);
?>
   </table>
   </CENTER>
 </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:

11.6.2 Instalación de ODBC en Linux

Para instalar soporte ODBC en Linux, podemos instalarnos el gestor iODBC (http://www.iodbc.org), que es un gestor *Open Source* mantenido por OpenLink Software (http://www.openlinksw.com).

Para compilar PHP con iODBC Driver Manager como un módulo compartido de Apache, realizamos los siguientes pasos:

1. Bajamos el paquete desde la página http://www.iodbc.org/opliodbc.htm. Allí buscamos el paquete correspondiente a `Linux glibc2.1 (Intel)` (lo que se

corresponde con el fichero `13kozzzz.taz`). No es necesario bajarse el `Generic Installation Script`).

2. Como el usuario `root`, descomprimimos el fichero que acabamos de obtener (y que lo hemos salvado en el directorio `/tmp`) debajo del directorio `/usr/local`:

```
# cd /usr/local
# tar ztvf /tmp/13kozzzz.taz
```

3. Tenemos que volver a reconfigurar, recompilar y reinstalar nuestra distribución de PHP para que incluya el soporte iODBC (consultar el capítulo de instalación):
   A) En la línea de configuración de PHP (comando `./configure`) añadimos la opción `--with-iodbc=/usr/local/odbcsdk`.
   B) Compilamos `# make`.
   C) Instalamos `#make install`.

4. Por último, para que el servidor Apache encuentre las librerías de iODBC, hay que indicarle dónde se encuentran; por tanto, hay que modificar el fichero de arranque de Apache y en la línea donde ponía:

```
/usr/local/apache/bin/apachectl start
```

tiene que poner ahora:

```
LD_LIBRARY_PATH=/usr/local/odbcsdk/lib \
   /usr/local/apache/bin/apachectl start
```

CAPÍTULO 12

PHP Y XML

PHP 5 proporciona varias extensiones del lenguaje (módulos de software) que permiten implementar analizadores gramaticales de documentos XML. Estos analizadores, también conocidos como procesadores o *parsers* XML, permiten a las aplicaciones el acceso a la estructura y al contenido de este tipo de documentos. Haciendo uso del conjunto de funciones proporcionadas por PHP, podemos definir *scripts* que interpreten nuestros documentos XML.

Este capítulo no pretende ser un resumen exhaustivo de estas tecnologías, simplemente muestra una aproximación a las mismas presentando ejemplos tipo con los tres procesadores de XML principales que presenta PHP 5: SimpleXML, SAX (*Simple API for XML*) y DOM (*Document Object Model*).

12.1 INTRODUCCIÓN A XML

A pesar de que el objetivo de este capítulo no es explicar en profundidad la composición y funcionalidad del lenguaje XML, conviene hacer una breve introducción a él, para que los que se acercan por primera vez a este lenguaje entiendan la naturaleza de las funciones proporcionadas por PHP para el tratamiento de este tipo de documentos.

12.1.1 ¿Qué es XML?

XML es el acrónimo de *eXtensible Markup Language*, es decir, "Lenguaje de Marcas Extendido". Es un estándar desarrollado por el consorcio de la "World Wide Web" (W3C) que define un meta-lenguaje basado en marcas, que nos permite crear lenguajes para estructurar información, es decir, para describir la información que manejan. También describe parcialmente el comportamiento de los programas de computador que pueden procesar los documentos XML.

XML está siendo ampliamente utilizado para el intercambio de documentos entre entornos heterogéneos puesto que define un formato independiente de la plataforma y del lenguaje utilizados. Esto explica el gran auge que está teniendo su utilización como base del intercambio de información en la Red.

> **NOTA:** Para obtener más información sobre XML y tecnologías afines, se puede recurrir a la dirección http://www.w3c.org/XML.

12.1.2 Estructura de un documento XML

Los documentos XML se componen de etiquetas de marcado y de contenido. Cuenta con diferentes etiquetas de marcado: elementos, referencias a entidades, comentarios, secciones CDATA, instrucciones de procesamiento, notaciones y definiciones de tipo de documento.

- **Elementos** (*elements*): son las etiquetas principales de la arquitectura, pues son las encargadas de estructurar el documento XML. La mayoría están destinadas a contener datos u otros elementos, identificando e informando de la naturaleza del contenido que encierran. Son las diferentes piezas de información en las que podemos dividir un documento. Normalmente constan de etiquetas de apertura y cierre aunque también pueden estar configurados como una única etiqueta vacía. Pueden contener en su etiqueta de apertura atributos para aumentar la semántica del elemento.

- **Comentarios** (*comments*): permiten la inserción de comentarios dentro de un documento XML, pero sin formar parte del contenido del mismo. Comienzan con la cadena "`<!--`" y terminan con la cadena "`-->`".

- **Secciones CDATA** (*CDATA sections*): son usadas para enmascarar bloques de texto que contengan caracteres que de otro modo serían reconocidos como marcas. Comienzan con la cadena "`<![CDATA[`" y terminan con la cadena "`]]>`".

- **Instrucciones de procesamiento** (*processing instructions* o *PI's*): permiten a los documentos incluir instrucciones para las aplicaciones encargadas de procesarlos. No forman parte del contenido textual del documento pero deben ser pasadas a las aplicaciones encargadas del procesamiento.

- **Referencias a entidades** (*entity references*): las entidades son las unidades de texto de las que se compone todo documento. Pueden ir desde un solo carácter hasta todo un documento completo. También se utilizan para hacer referencia a caracteres especiales que de otro modo serían considerados como marcado.

- **Notaciones** (*notations*): básicamente, identifican mediante un nombre XML a una aplicación auxiliar capaz de procesar cierto tipo de datos y que podrá ser utilizada por el procesador XML o la aplicación encargada de procesar el documento. Las notaciones también se utilizan para realizar negociación de contenidos.

- **Definiciones de tipo de documento** (*DTD – Document Type Definition*): proporcionan un mecanismo para definir reglas que nos permiten describir restricciones en la estructura lógica de los documentos XML. Un documento de XML es válido si tiene asociada una declaración de tipo de documento y es conforme con las restricciones que en ella se expresan. También nos permiten especificar las entidades a utilizar dentro del documento.

El siguiente documento XML muestra cada uno de los componentes básicos mencionados:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Ejemplo de fichero XML -->
<!DOCTYPE notas [
        <!ELEMENT notas (alumno*, comentarios?)>
        <!ELEMENT alumno (matricula, nombre, apellidos, teoria, practicas)>
        <!ELEMENT matricula (#PCDATA)>
        <!ELEMENT nombre (#PCDATA)>
        <!ELEMENT apellidos (#PCDATA)>
        <!ELEMENT teoria (#PCDATA)>
        <!ELEMENT practicas (#PCDATA)>
        <!ELEMENT comentarios (#PCDATA)>
        <!ATTLIST notas asignatura CDATA "SSOO">
        <!ENTITY SSOO "Sistemas Operativos">
        <!ENTITY logoUPM SYSTEM "logoupm.bmp" NDATA imagen>
        <!NOTATION imagen SYSTEM "lector.exe">
]>
<notas asignatura="&SSOO;">
        <alumno>
                <matricula>x0503</matricula>
                <nombre>Inés</nombre>
                <apellidos>Bueno Gutiérrez</apellidos>
```

```
            <teoria>8</teoria>
            <practicas>9</practicas>
        </alumno>
        <alumno>
            <matricula>x0504</matricula>
            <nombre>Violeta</nombre>
            <apellidos>Duro Gutiérrez</apellidos>
            <teoria>4.5</teoria>
            <practicas>7</practicas>
        </alumno>
        <comentarios><![CDATA[Convocatoria extraordinaria de <Junio>]]>
        </comentarios>
</notas>
```

Los elementos que forman el cuerpo del documento:

```
<notas asignatura="&SSOO;">
        <alumno>
            <matricula>x0503</matricula>
            <nombre>Inés</nombre>
            <apellidos>Bueno Gutiérrez</apellidos>
            <teoria>8</teoria>
            <practicas>9</practicas>
        </alumno>
        <alumno>
            <matricula>x0504</matricula>
            <nombre>Violeta</nombre>
            <apellidos>Duro Gutiérrez</apellidos>
            <teoria>4.5</teoria>
            <practicas>7</practicas>
        </alumno>
        <comentarios>
          <![CDATA[Convocatoria extraordinaria de <Junio>]]>
        </comentarios>
</notas>
```

Los comentarios:

```
<!-- Ejemplo de fichero XML -->
```

Las secciones CDATA, podemos observar cómo la etiqueta `<Junio>` no se interpreta, no se considera un elemento, por formar parte de la sección:

```
          <![CDATA[Convocatoria extraordinaria de <Junio>]]>
```

Las instrucciones de procesamiento, en este caso la PI obligatoria que indica que el documento es XML y utiliza una codificación `ISO-8859-1`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Las declaraciones de DTD (en este caso interno), que establecen las reglas de validez del documento:

```
<!DOCTYPE notas [
        <!ELEMENT notas (alumno*, comentarios?)>
        <!ELEMENT alumno (matricula, nombre, apellidos, teoria, practicas)>
        <!ELEMENT matricula (#PCDATA)>
        <!ELEMENT nombre (#PCDATA)>
        <!ELEMENT apellidos (#PCDATA)>
```

```
        <!ELEMENT teoria (#PCDATA)>
        <!ELEMENT practicas (#PCDATA)>
        <!ELEMENT comentarios (#PCDATA)>
        <!ATTLIST notas asignatura CDATA "SSOO">
        <!ENTITY SSOO "Sistemas Operativos">
        <!ENTITY logoUPM SYSTEM "logoupm.bmp" NDATA imagen>
        <!NOTATION imagen SYSTEM "lector.exe">
]>
```

La declaración de las entidades, dentro de la declaración del DTD (una interna y otra externa):

```
        <!ENTITY SSOO "Sistemas Operativos">
        <!ENTITY logoUPM SYSTEM "logoupm.bmp" NDATA imagen>
```

El uso de las entidades dentro del documento:

```
<notas asignatura="&SSOO;">
```

Las notaciones, declaradas dentro del DTD. En este caso se indica que los datos de tipo `imagen` deben ser procesados por el programa `lector.exe`:

```
        <!NOTATION imagen SYSTEM "lector.exe">
```

12.2 XML EN PHP

La relación entre PHP y XML comienza en PHP 3, pues esta versión (en la 3.0.6) ya contaba con una extensión para SAX soportada por la librería `expat` de James Clark. Las extensiones para SimpleXML y para el DOM soportadas por la librería `libxml2` (la librería de XML de GNOME) tuvieron que esperar hasta la versión 4. De este modo, PHP 4 incorporaba tres modos diferentes de trabajar con los documentos XML, si bien, la extensión más utilizada fue la de SAX (incluida por defecto en la versión Win32 de PHP 4) porque SimpleXML y DOM siempre fueron extensiones experimentales, es decir, propensas a cambios y modificaciones sin previo aviso, lo que hizo que los desarrolladores no las usaran ampliamente.

> **NOTA:** Para obtener más información sobre `libxml2`, se puede recurrir a la dirección http://www.xmlsoft.org/. Más información sobre `expat`, en la dirección http://www.jclark.com/xml/expat.html.

- La extensión **SimpleXML** proporcionaba un conjunto sencillo de herramientas que permitía convertir un documento XML en un objeto con el que poder trabajar. El problema principal de esta extensión, además de su carácter experimental, era el reducido conjunto de posibilidades que ponía a disposición del programador.

- La extensión para **SAX** proporcionaba un procesador XML orientado a eventos. El trabajo con SAX era relativamente sencillo, consistía en

recorrer linealmente todo el documento XML llamando a funciones, especificadas por el programador, ante ciertos eventos normalmente asociados a la apertura y cierre de etiquetas o a la aparición de determinados componentes. El principal inconveniente que presentaba SAX es que no nos proveía de una forma de validar el documento respecto a una DTD. Y que no permitía modificar el documento con el que se estaba trabajando.

- La extensión **DOM** de PHP proporcionaba un procesador XML completo, permitía validar un documento respecto a una DTD, que transformaba el documento XML en una estructura de árbol que seguía de forma aproximada el modelo de objetos de documento del W3C. El problema principal de esta extensión, además de su carácter experimental, era que hacía uso de funciones no presentes en el estándar y que no podía dar algunas de las prestaciones especificadas en el DOM, principalmente por la naturaleza del trabajo con objetos en PHP 4.

Sin embargo, en PHP 5 se han reescrito completamente las funcionalidades de XML, haciendo uso de la librería `libxml2`, para dar soporte tanto a SimpleXML y DOM como a SAX. La nueva extensión del DOM sigue completamente el estándar publicado por el W3C y además las nuevas extensiones de SimpleXML y DOM tienen interfaces para compartir información.

12.3 SIMPLEXML

Tal y como hemos dicho, SimpleXML nos proporciona un conjunto sencillo de herramientas para trabajar con documentos XML. Es una buena herramienta para realizar un primer contacto con XML, puesto que implementa las funcionalidades más comunes de XML, dejando el resto para otras extensiones como SAX y DOM.

A través de SimpleXML, un documento XML se convierte en una estructura de datos sobre la que podemos trabajar haciendo uso de las colecciones de *arrays* y de objetos que la componen. Las funciones proporcionadas para realizar estas operaciones son las siguientes:

- `simplexml_load_file(fichero)`: convierte el documento XML almacenado en el `fichero` en un objeto de clase `simplexml_element` que contiene los datos almacenados en el documento. El documento XML deberá estar bien formado.

- `simplexml_load_string(cadena)`: convierte la `cadena` XML en un objeto de clase `simplexml_element` que contiene los datos almacenados en dicha cadena. La cadena XML deberá estar bien formada.

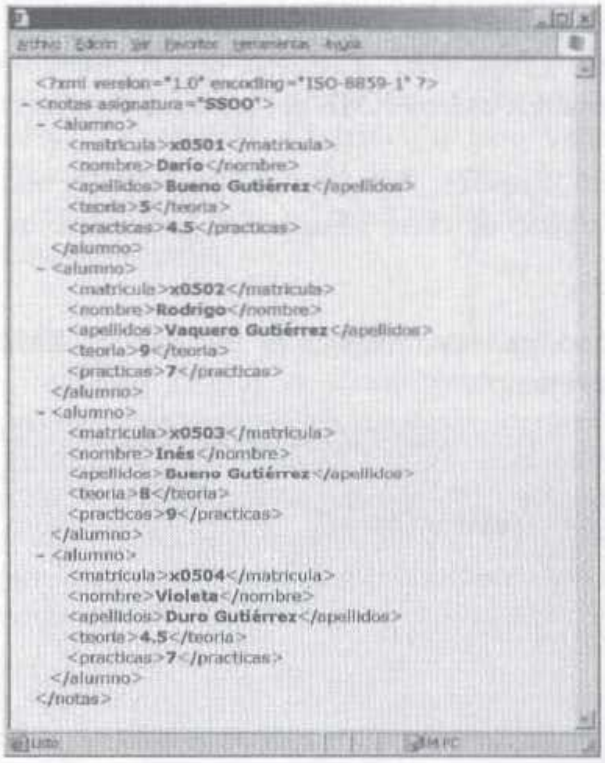
- `simplexml_import_dom(nodo)`: convierte el `nodo`[1] de un documento DOM en un objeto de clase `simplexml_element` que representa a dicho nodo.

El siguiente código nos muestra el documento XML (`notas.xml`) que vamos a utilizar en los ejemplos:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<notas asignatura="SSOO">
   <alumno>
      <matricula>x0501</matricula>
      <nombre>Darío</nombre>
      <apellidos>Bueno Gutiérrez</apellidos>
      <teoria>5</teoria>
      <practicas>4.5</practicas>
   </alumno>
   <alumno>
      <matricula>x0502</matricula>
      <nombre>Rodrigo</nombre>
      <apellidos>Vaquero Gutiérrez</apellidos>
      <teoria>9</teoria>
      <practicas>7</practicas>
   </alumno>
   <alumno>
      <matricula>x0503</matricula>
      <nombre>Inés</nombre>
      <apellidos>Bueno Gutiérrez</apellidos>
      <teoria>8</teoria>
      <practicas>9</practicas>
   </alumno>
   <alumno>
      <matricula>x0504</matricula>
      <nombre>Violeta</nombre>
      <apellidos>Duro Gutiérrez</apellidos>
      <teoria>4.5</teoria>
      <practicas>7</practicas>
   </alumno>
</notas>
```

En un navegador nuestro documento sería visualizado de la siguiente forma:

[1] En el apartado 12.5 se explica qué es un nodo DOM.



Desde el punto de vista de SimpleXML obtendríamos la siguiente estructura de datos:

```
simplexml_element Object
(
    [alumno] => Array
        (
            [0] => simplexml_element Object
                (
                    [matricula] => x0501
                    [nombre] => Darío
                    [apellidos] => Bueno Gutiérrez
                    [teoria] => 5
                    [practicas] => 4.5
                )
            [1] => simplexml_element Object
                (
                    [matricula] => x0502
                    [nombre] => Rodrigo
                    [apellidos] => Vaquero Gutiérrez
                    [teoria] => 9
                    [practicas] => 7
                )

            [2] => simplexml_element Object
                (
                    [matricula] => x0503
                    [nombre] => Inés
                    [apellidos] => Bueno Gutiérrez
                    [teoria] => 8
                    [practicas] => 9
                )
            [3] => simplexml_element Object
```

```
            (
                [matricula] => x0504
                [nombre] => Violeta
                [apellidos] => Duro Gutiérrez
                [teoria] => 4.5
                [practicas] => 7
            )
    )
)
```

Es decir, un objeto de la clase `simplexml_element` que contiene una única propiedad llamada `alumno`, que es un *array* compuesto de cuatro objetos de la clase `simplexml_element`, cada uno de los cuales tiene las propiedades `matricula`, `nombre`, `apellidos`, `teoria` y `practicas` de tipo cadena.

La interfaz `simplexml_element` proporciona los siguientes métodos:

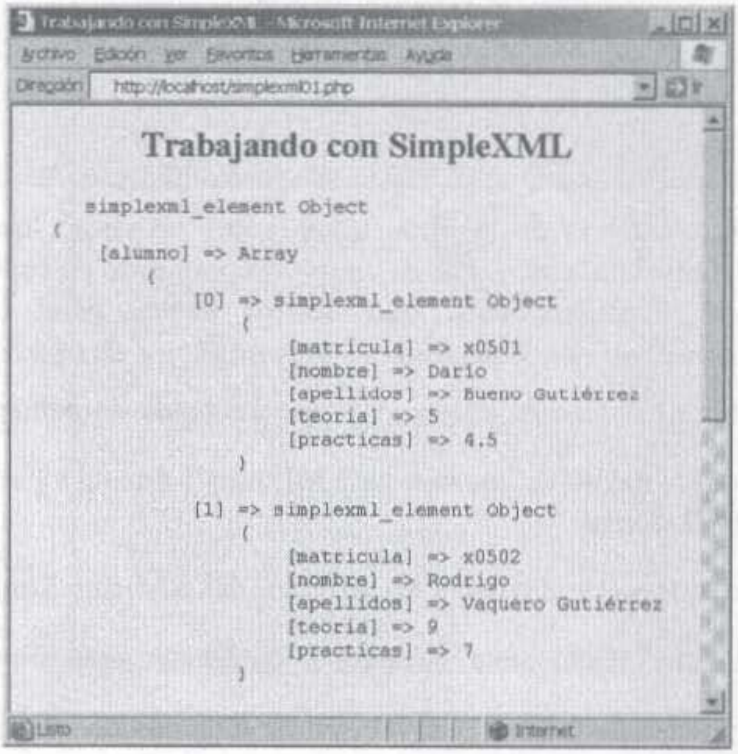
- `asXML()`: devuelve una cadena XML bien formada a partir del elemento SimpleXML actual.
- `attributes()`: devuelve los atributos del elemento SimpleXML actual.
- `children()`: devuelve los nodos hijos del elemento SimpleXML actual.
- `xpath(camino)`: nos permite aplicar métodos de búsqueda basados en XPath[2] sobre los nodos hijos del elemento SimpleXML actual.

El siguiente ejemplo muestra cómo abrir un documento XML y transformarlo con SimpleXML. Además, visualiza la estructura de datos explicada anteriormente:

```
<html>
 <head>
   <title>Trabajando con SimpleXML</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con SimpleXML</h2>
   <table>
   <tr><td><pre>
   <?php
     if (file_exists('notas.xml')) {
        $notas = simplexml_load_file('notas.xml');
        print_r($notas);
     } else {
        die('No se encuentra el fichero "notas.xml".');
     }
   ?>
   </pre></td></tr></table>
   </center>
  </body>
</html>
```

[2] XPATH es una tecnología asociada a XML que permite realizar búsquedas en los documentos.

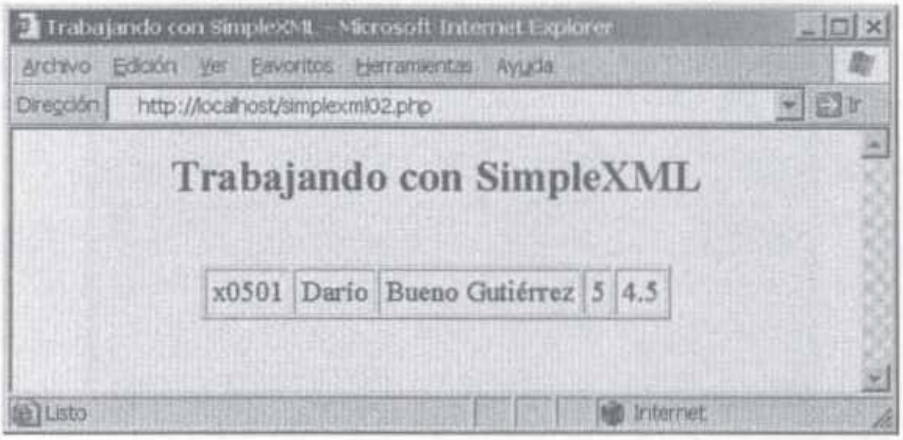
El resultado se muestra en la siguiente imagen:



Para recuperar la información simplemente hay que recorrer la estructura hasta llegar al dato deseado. El siguiente código nos muestra una forma sencilla de recuperar los datos del primero de los alumnos almacenados en el documento XML:

```
<html>
 <head>
   <title>Trabajando con SimpleXML</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con SimpleXML</h2><br>
   <?php
     if (!file_exists('notas.xml')) {
        exit('No se encuentra el fichero "notas.xml".');
     }
   ?>
   <table border="1" cellpadding="4">
   <tr>
   <?php
       $notas = simplexml_load_file('notas.xml');
       echo '<td>',$notas->alumno[0]->matricula,'</td>';
       echo '<td>',$notas->alumno[0]->nombre,'</td>';
       echo '<td>',$notas->alumno[0]->apellidos,'</td>';
       echo '<td>',$notas->alumno[0]->teoria,'</td>';
       echo '<td>',$notas->alumno[0]->practicas,'</td>';
   ?>
   </tr></table>
   </center>
  </body>
</html>
```

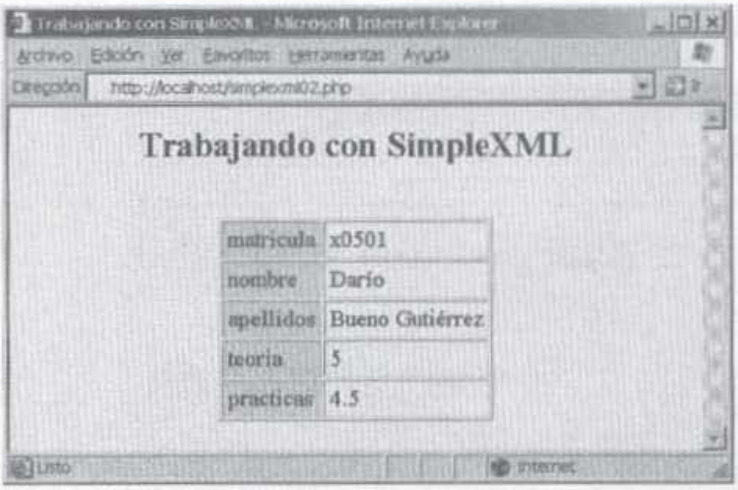
El resultado se muestra en la siguiente imagen:



El siguiente código nos muestra una forma equivalente de recuperar la misma información:

```
<html>
 <head>
   <title>Trabajando con SimpleXML</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con SimpleXML</h2><br>
   <?php
     if (!file_exists('notas.xml')) {
        exit('No se encuentra el fichero "notas.xml".');
     }
   ?>
   <table border="1" cellpadding="4">
   <?php
       $notas = simplexml_load_file('notas.xml');
       foreach ($notas->alumno[0] as $prop => $valor) {
              echo '<tr>';
              echo '<td bgcolor="yellow">',$prop,'</td>';
              echo '<td>',$valor,'</td>';
              echo '</tr>';
       }
   ?>
   </table>
   </center>
  </body>
</html>
```

El resultado es el que se muestra en la siguiente imagen:



Para recuperar la información de todos los alumnos del documento tendríamos que modificar el código anterior tal y como se muestra en el siguiente ejemplo:

```
<html>
 <head>
   <title>Trabajando con SimpleXML</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con SimpleXML</h2><br>
   <?php
     if (!file_exists('notas.xml')) {
        exit('No se encuentra el fichero "notas.xml".');
     }
   ?>
   <table border="1" cellpadding="4">
   <?php
        $notas = simplexml_load_file('notas.xml');
        foreach ($notas->alumno as $alumno) {
          echo '<tr>';
          echo '<td>',$alumno->matricula,'</td>';
          echo '<td>',$alumno->nombre,'</td>';
          echo '<td>',$alumno->apellidos,'</td>';
          echo '<td>',$alumno->teoria,'</td>';
          echo '<td>',$alumno->practicas,'</td>';
          echo '</tr>';
        }
   ?>
   </table>
   </center>
  </body>
</html>
```

Un bucle equivalente al anterior sería el siguiente:

```
   <?php
        $notas = simplexml_load_file('notas.xml');
        foreach ($notas->alumno as $alumno) {
           echo '<tr>';
           foreach ($alumno as $valor) {
              echo '<td>',$valor,'</td>';
           }
```

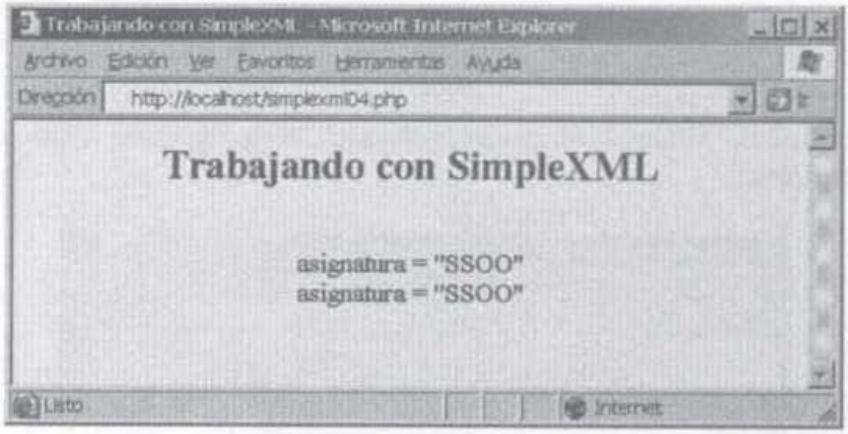
```
            echo '</tr>';
        }
    ?>
```

La siguiente imagen muestra el resultado de ejecutar cualquiera de los ejemplos anteriores:

Para recuperar cualquiera de los atributos presentes en un elemento contamos con dos posibilidades: hacer uso del método `attributes()`, disponible en los objetos de tipo `simplexml_element`, que recuperaría todos sus atributos o bien acceder al atributo deseado como si se tratase de un componente de un *array* asociativo. El siguiente ejemplo nos lo muestra de forma práctica:

```
<html>
 <head>
   <title>Trabajando con SimpleXML</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con SimpleXML</h2><br>
   <?php
     if (!file_exists('notas.xml')) {
        exit('No se encuentra el fichero "notas.xml".');
     }
   ?>
   <?php
        $notas = simplexml_load_file('notas.xml');
        foreach($notas->attributes() as $atributo => $valor) {
           echo $atributo,' = "',$valor,'"<br>';
        }
        echo 'asignatura = "',$notas['asignatura'],'"<br>';
   ?>
   </center>
  </body>
</html>
```

Como se puede observar en la siguiente imagen, el resultado obtenido es el mismo por cualquiera de los dos métodos:



El uso de XPath para realizar consultas en un documento XML nos lo proporciona, tal y como ya hemos comentado, el método `xpath()` presente en los objetos `simplexml_element`. El siguiente ejemplo nos muestra su utilización:

```
<html>
 <head>
   <title>Trabajando con SimpleXML</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con SimpleXML</h2><br>
   <?php
     if (!file_exists('notas.xml')) {
        exit('No se encuentra el fichero "notas.xml".');
     }
   ?>
   <table border="1" cellpadding="4"><tr>
   <?php
       $notas = simplexml_load_file('notas.xml');
       foreach($notas->xpath('//nombre') as $nombre) {
          echo "<td>",$nombre," </td>";
       }
   ?>
   </tr></table>
   </center>
  </body>
</html>
```

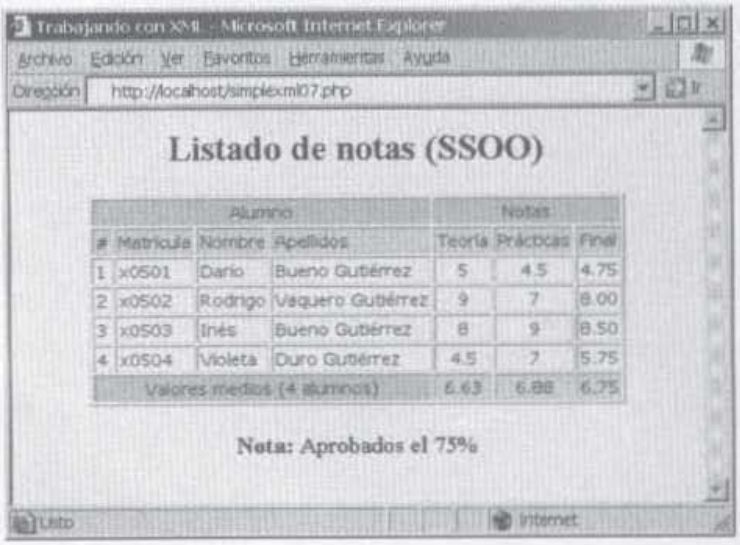
El parámetro '`//nombre`' pasado al método `xpath()` indica que se debe buscar de forma recursiva desde la raíz del documento la aparición de todos los elementos cuya etiqueta tenga el valor `nombre`. El resultado se observa en la siguiente imagen:

El siguiente ejemplo nos muestra el uso conjunto de las funcionalidades proporcionadas por SimpleXML para obtener un listado completo de las notas de los alumnos junto con información estadística:

```
<?php

if (file_exists('notas.xml')) {
      $notas = simplexml_load_file('notas.xml');

      echo "<HTML>";
      echo "<HEAD><TITLE>Trabajando con XML</TITLE>";
      echo "<STYLE TYPE='text/css'>";
      echo "  TABLE {font:14px Verdana; border:1px; cell-padding:2px;}";
      echo "  .azul {background-color:cyan; text-align:center;}";
      echo "  .amarilla {background-color:yellow;}";
      echo "</STYLE></HEAD>";
      echo "<BODY><CENTER><H2>Listado de notas ({$notas[asignatura]})</H2>";
      echo "<TABLE>";
      echo "<TR CLASS='azul'><TD COLSPAN='4'>Alumno</TD>";
      echo "<TD COLSPAN='3'>Notas</TD></TR>";
      echo "<TR CLASS='amarilla'><TD>#</TD><TD>Matrícula</TD>";
      echo "<TD>Nombre</TD><TD>Apellidos</TD>";
      echo "<TD>Teoría</TD><TD>Prácticas</TD><TD>Final</TD></TR>";

      $num_alumnos=0;
      $num_aprobados=0;
      $notas_teoria=0;
      $notas_practicas=0;
      $notas_finales=0;
      foreach($notas->alumno as $alumno) {
            $num_alumnos++;
            echo "<TR><TD>$num_alumnos</TD>";
            echo "<TD>$alumno->matricula</TD>";
            echo "<TD>$alumno->nombre</TD>";
            echo "<TD>$alumno->apellidos</TD>";
            echo "<TD ALIGN='center'>$alumno->teoria</TD>";
            echo "<TD ALIGN='center'>$alumno->practicas</TD>";
           $notas_teoria+=floatval($alumno->teoria);
           $notas_practicas+=floatval($alumno->practicas);
           $nota_final_alumno=
              (floatval($alumno->teoria)+floatval($alumno->practicas))/2;
           $notas_finales+=$nota_final_alumno;
           printf("<TD ALIGN='center'>%01.2f</TD>",$nota_final_alumno);
           if($nota_final_alumno>5){
              $num_aprobados++;
           }
      }
```

```
        echo "<TR CLASS='amarilla'>";
        echo "<TD CLASS='azul' COLSPAN='4'>";
        echo "Valores medios ($num_alumnos alumnos)</TD>";
        echo "<TD>";
        printf("%01.2f",$notas_teoria/$num_alumnos);
        echo "</TD><TD>";
        printf("%01.2f",$notas_practicas/$num_alumnos);
        echo "</TD><TD>";
        printf("%01.2f",$notas_finales/$num_alumnos);
        echo "</TD></TR></TABLE>";
        echo "<P><B>Nota: </B>Aprobados el ";
        printf("%01.0f",($num_aprobados/$num_alumnos)*100);
        echo "%</P></CENTER></BODY></HTML>";

} else {
    exit('No se encuentra el fichero "notas.xml".');
}
?>
```

El resultado se observa en la siguiente imagen:



12.4 SAX

A pesar de que los analizadores basados en SAX no son capaces de validar documentos XML, puesto que no tienen en cuenta los posibles DTD's asociados con estos documentos, su rapidez y pequeño tamaño los hacen especialmente adecuados para ser utilizados en la Red.

SAX proporciona analizadores basados en eventos, es decir, con SAX se visualiza el documento XML como un conjunto de eventos. Es, por tanto, un analizador centrado en los datos presentes en el documento XML y no en su estructura. Procesa el documento desde su inicio hasta su final informando a la

aplicación que lo usa de los eventos que va encontrando (tales como las etiquetas de apertura y cierre de cada elemento, las instrucciones de procesamiento encontradas...) por medio de la llamada a funciones definidas por el propio desarrollador.

Las funciones proporcionadas por PHP para trabajar con analizadores de tipo SAX se pueden agrupar según sus funcionalidades. Así, para la creación y manejo básico de analizadores tenemos las funciones:

- `xml_parser_create(codificación)`: crea un analizador de XML y devuelve un valor de tipo entero que identifica al analizador creado. El parámetro opcional `codificación` nos permite indicar la codificación de caracteres a utilizar, en caso de no proporcionarse se utilizará por defecto la `ISO-8859-1`.

- `xml_parser_create_ns(codificación, separador)`: crea un analizador de XML con soporte para espacios de nombres y devuelve un valor de tipo entero que identifica al analizador creado. El parámetro opcional `codificación` nos permite indicar la codificación de caracteres a utilizar, en caso de no proporcionarse se utilizará por defecto la `ISO-8859-1`. También se puede indicar el `separador` a utilizar (por defecto ':').

> **NOTA:** Para utilizar esta función sin problemas será necesario tener instalada una versión de `libxml2` igual o superior a la `2.2.6`.

- `xml_parser_free(analizador)`: libera los recursos asignados al `analizador` proporcionado.

- `xml_parser_set_option(analizador, opción, valor)`: permite fijar las opciones básicas de funcionamiento del `analizador`. La siguiente tabla muestra las opciones básicas de los analizadores SAX:

opción	utilización
`XML_OPTION_CASE_FOLDING`	indica si el analizador debe convertir los nombres de elementos del documento XML a analizar en mayúsculas
`XML_OPTION_TARGET_ENCODING`	indica la codificación destino que utilizará el analizador
`XML_OPTION_SKIP_WHITE`	indica si el analizador debe mantener los espacios en blanco presentes en el documento XML a analizar

- `xml_parser_get_option(analizador, opción)`: permite recuperar las opciones básicas de funcionamiento del `analizador`.

Para la definición de los diferentes manejadores o gestores de eventos que serán invocados a lo largo del análisis tenemos las funciones:

- `xml_set_element_handler(analizador, f_apertura, f_cierre)`: asigna al `analizador` proporcionado las funciones encargadas de manejar las etiquetas de apertura (`f_apertura`) y cierre (`f_cierre`) asociadas a cada elemento del documento XML. Se deben definir funciones diferenciadas para las etiquetas de apertura y cierre. La función `f_apertura` deberá aceptar tres parámetros: el `analizador` para el que ha sido definida, el `nombre` del elemento para el que se llama a la función, y un *array* asociativo donde se pasan los `atributos` en caso de que existan. Por su parte, la función `f_cierre` sólo tendrá como parámetros el `analizador` y el `nombre` del elemento.

- `xml_set_character_data_handler(analizador, función)`: asigna al `analizador` proporcionado la `función` encargada de gestionar los datos de caracteres encontrados al analizar el documento XML. Dicha `función` deberá aceptar dos parámetros: el `analizador` para el que ha sido definida y los datos de carácter como una cadena.

- `xml_set_processing_instruction_handler (analizador, función)`: asigna al `analizador` proporcionado la `función` encargada de gestionar las instrucciones de procesamiento encontradas al analizar el documento XML. Dicha `función` deberá aceptar tres parámetros: el `analizador` para el que ha sido definida, el objetivo de la PI y los datos de la PI.

- `xml_set_external_entity_ref_handler (analizador, función)`: asigna al `analizador` proporcionado la `función` encargada de gestionar las referencias a entidades externas encontradas al analizar el documento XML. Dicha `función` deberá aceptar cinco parámetros: el `analizador` para el que ha sido definida, los nombres de las entidades que se abren para el análisis de esta entidad, la base para resolver las referencias, el identificador de sistema (`systemId`) y el identificador público (`publicId`) de la entidad.

- `xml_set_unparsed_entity_decl_handler (analizador, función)`: asigna al `analizador` proporcionado la `función` encargada de gestionar las referencias a entidades no analizadas (entidades externas con una declaración NDATA) encontradas al analizar el documento XML. Dicha `función` deberá aceptar seis parámetros: el `analizador` para el que ha sido definida, los nombres de las entidades que se abren para el análisis de esta entidad, la base para resolver las referencias, el identificador de sistema (`systemId`), el identificador público (`publicId`) y el nombre de la notación de la entidad.

- `xml_set_notation_decl_handler (analizador, función)`: asigna al `analizador` proporcionado la `función` encargada de gestionar las declaraciones de notación encontradas al analizar el documento XML. Dicha `función` deberá aceptar cinco parámetros: el `analizador` para el que ha sido definida, el nombre de la notación, la base para resolver las referencias, el identificador de sistema (`systemId`) y el identificador público (`publicId`) de la declaración de notación.

- `xml_set_default_handler (analizador, función)`: asigna al `analizador` proporcionado una función encargada de gestionar aquellos componentes del documento XML para los que no exista un manejador específico. Dicha `función` deberá aceptar dos parámetros: el `analizador` para el que ha sido definida y una cadena con los datos.

Para llevar a cabo el análisis del documento tenemos las dos funciones siguientes:

- `xml_parse(analizador, datos, final)`: esta función comienza a realizar el análisis del documento utilizando el `analizador` y la cadena de `datos` proporcionados. El parámetro opcional `final`, sirve para indicar si ésta será la última porción de datos a procesar (los bloques de información a analizar tienen un tamaño máximo de 4.096 *bytes*). Cuando el documento XML es analizado, se hacen llamadas a los gestores para los eventos configurados tantas veces como sea necesario. Devuelve `TRUE` si el análisis se realiza con éxito, `FALSE` si no tiene éxito, o si se hace referencia a un `analizador` no válido.

- `xml_parse_into_struct (analizador, datos, componentes, índices)`: hace que el `analizador` procese los `datos` proporcionados creando dos estructuras de tipo *array* que contienen los `componentes` encontrados y los `índices` a dichos componentes (opcional). Ambos *arrays* hay que pasarlos por referencia. Esta función produce una estructura semejante, pero no igual, a la obtenida mediante SimpleXML.

Para recuperar información de análisis fallidos y controlar los posibles errores que se produzcan cuenta con las funciones:

- `xml_get_error_code(analizador)`: obtiene el código de error que se ha producido en el `analizador` proporcionado.

- `xml_error_string(código)`: devuelve una cadena con una descripción textual del `código` de error proporcionado, o `FALSE` si no se encontró ninguna descripción asociada a dicho `código`.

- `xml_get_current_line_number(analizador)`: devuelve la línea actual del buffer de datos en la que se encuentra realizando el análisis el `analizador` proporcionado, o `FALSE` en cualquier otro caso.

- `xml_get_current_column_number(analizador)`: devuelve la columna de la línea actual del buffer de datos en la que se encuentra realizando el análisis el `analizador` proporcionado, o `FALSE` en cualquier otro caso.

- `xml_get_current_byte_index(analizador)`: devuelve el índice del *byte* actual del buffer de datos en la que se encuentra realizando el análisis el `analizador` proporcionado, o `FALSE` en cualquier otro caso.

Por último, PHP proporciona dos funciones que permiten modificar la codificación de caracteres con los que está trabajando nuestro analizador SAX:

- `utf8_encode(cadena)`: codifica la `cadena` ISO-8859-1 proporcionada a UTF-8.

- `utf8_decode(cadena)`: codifica la `cadena` UTF-8 proporcionada a ISO-8859-1.

> **NOTA:** La extensión XML de PHP soporta el conjunto de caracteres Unicode a través de las codificaciones `US-ASCII`, `ISO-8859-1` y `UTF-8`. La codificación `UTF-16` no está soportada.

La sintaxis de las funciones anteriores y su utilidad dentro del proceso de análisis de documentos XML se muestran con el siguiente conjunto de ejemplos en los que poco a poco, se van ampliando sus usos básicos para dar respuesta a soluciones más complejas.

En este primer ejemplo se crea un analizador SAX básico que nos *avisa* (genera un evento) de la aparición de las etiquetas de inicio y cierre de cada elemento, al igual que de la aparición de las secciones con datos de tipo texto (*character data*) que contienen la información del documento:

```
<?
//datos XML a analizar
  $datos = "<alumno>".
              "<matricula>x0501</matricula>".
              "<nombre>Dario</nombre>".
              "<apellidos>Bueno Gutierrez</apellidos>".
              "<teoria>5</teoria>".
              "<practicas>4.5</practicas>".
           "</alumno>";

//manejadores
  function inicio_elemento($parser,$etiqueta,$atributos){
     echo "    elemento  : <B>&lt;$etiqueta&gt;</B><BR>";
```

```
  }

  function fin_elemento($parser,$etiqueta){
     echo "    elemento  : <B>&lt;/$etiqueta&gt;</B><BR>";
  }

  function datos($parser,$dato){
     echo "       datos  : <B>$dato</B><BR>";
  }

//1er paso
  $parser = xml_parser_create();
  xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
  xml_parser_set_option($parser, XML_OPTION_SKIP_WHITE, 1);
  xml_parser_set_option($parser, XML_OPTION_TARGET_ENCODING, "ISO-8859-1");

// 2º paso
  xml_set_element_handler($parser, "inicio_elemento","fin_elemento");
  xml_set_character_data_handler($parser,"datos");

?>

<html>
 <head>
   <title>Trabajando con SAX</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con SAX</h2></center><pre>
   <?php
   //3er paso
     if (!xml_parse($parser,$datos,true)){
         echo "<span style='color:red; font: bold 18px Verdana;'>";
         echo "Error XML (",xml_get_error_code($parser),"): ";
         echo "'",xml_error_string(xml_get_error_code($parser)), "'<BR>";
         echo " --- línea ",xml_get_current_line_number($parser);
         echo ", columna ",xml_get_current_column_number($parser);
         echo " ---</span>";
     }
     xml_parser_free($parser);
   ?>
   </pre>
  </body>
</html>
```

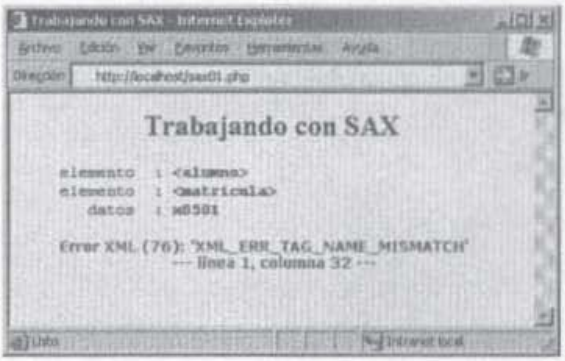
Como podemos observar en el código, la forma de operar con este tipo de analizadores es muy sencilla. En un primer paso se crea el analizador y se le fijan las opciones básicas de funcionamiento. Posteriormente, se le asignan los manejadores de los componentes que queremos controlar y sus correspondientes funciones asociadas. Por último, se realiza el análisis. En nuestro ejemplo, en este último paso, también se lleva a cabo el control de errores.

El resultado de ejecutar el *script* se visualiza en la siguiente imagen:

En caso de que los datos a analizar estuvieran mal formados, habríamos obtenido el error correspondiente. Por ejemplo, si hubiéramos intentado analizar los siguientes datos en los que se abre una etiqueta cuyo nombre no coincide con su etiqueta de cierre correspondiente:

```
$datos = "...<matricula>x0501</Matricula>...";
```

El resultado hubiera sido el mostrado por la siguiente imagen:



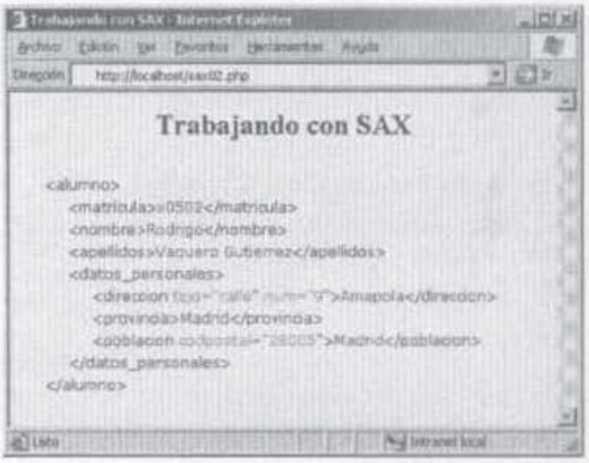
El siguiente ejemplo muestra un uso más avanzado de las funciones proporcionadas por SAX para manejar documentos XML. El *script* visualiza el documento XML analizado mostrando de forma diferenciada cada uno de sus componentes:

```
<?
// nivel de anidamiento de la etiqueta
$nivel=1;
$cambiar_linea=false;

// documento XML a analizar
$documento = "prueba.xml";
```

```
// manejadores
function inicio_elemento($parser,$etiqueta,$atributos){
  global $nivel, $cambiar_linea;

  echo "<P STYLE='padding-left:",$nivel*20,"px;'>&lt;";
  echo "<SPAN class='etq'>$etiqueta";
  while (list ($atributo, $valor) = each ($atributos)) {
       echo " <SPAN class='atr'>$atributo=\"$valor</SPAN>\"";
  }
  echo "</SPAN>&gt;";
  $cambiar_linea=false;
  $nivel++;
}

function fin_elemento($parser,$etiqueta){
  global $nivel, $cambiar_linea;

  $nivel--;
  if($cambiar_linea)
     echo "<P STYLE='padding-left:",$nivel*20,"px;'>";
  echo "&lt;/<SPAN class='etq'>$etiqueta</SPAN>&gt;</P>";
  $cambiar_linea=true;
}

function datos($parser,$dato){
  echo "<SPAN class='val'>$dato</SPAN>";
}

function resto_info($parser,$datos){
    echo "<I>$datos</I><BR>";
}

?>
<html>
 <head>
   <meta http-equiv='content-type' content='text/html;'>
   <title>Trabajando con SAX</title>
   <style type="text/css">
      .etq {color:blue;}
      .atr {color:green;}
      .val {color:red;}
      .err {color:red; font: bold 12px Verdana;}
      code {font: 12px Verdana; line-height:0px; color:blue;}
   </style>
 </head>
 <body>
   <center>
   <h2>Trabajando con SAX</h2></center>
   <br><code>
<?php
  // 1er paso
  $parser = xml_parser_create();
  xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
  xml_parser_set_option($parser, XML_OPTION_SKIP_WHITE, 1);
  xml_parser_set_option($parser, XML_OPTION_TARGET_ENCODING, "ISO-8859-1");

  // 2º paso
  xml_set_element_handler($parser, "inicio_elemento","fin_elemento");
  xml_set_character_data_handler($parser,"datos");
  xml_set_default_handler($parser,"resto_info");

  //3er paso
```

```
    if(!file_exists($documento)){
        die("No se puede encontrar el fichero\"$documento\".");
    }
    if(!($fp = @fopen($documento, "r"))){
        die("No se puede leer el fichero\"$documento\".");
    }
    while($data = fread($fp, 4096)){
        if(!xml_parse($parser, $data, feof($fp))){
            echo "<center><span class='err'>";
            echo "Error XML (",xml_get_error_code($parser),"): ";
            echo "'",xml_error_string(xml_get_error_code($parser)),"'<BR>";
            echo " --- línea ",xml_get_current_line_number($parser);
            echo ", columna ",xml_get_current_column_number($parser);
            echo " ---</span></center>";
            break;
        }
    }
    fclose($fp);
    xml_parser_free($parser);
?>
  </code>
 </body>
</html>
```

El resultado se puede visualizar en la siguiente imagen:



En caso de que no queramos utilizar el sistema de eventos proporcionado por SAX, podemos recurrir a recuperar la información en estructuras de *arrays* haciendo uso de la función `xml_parse_into_struct()`. Para el mismo documento XML (`prueba.xml`) del ejemplo anterior los *arrays* definidos serían:

Array de índices

```
Array(
 [alumno] => Array
     (
         [0] => 0
         [1] => 9
     )
 [matricula] => Array
```

```
        (
            [0] => 1
        )
    [nombre] => Array
        (
            [0] => 2
        )

    [apellidos] => Array
        (
            [0] => 3
        )
    [datos_personales] => Array
        (
            [0] => 4
            [1] => 8
        )
    [direccion] => Array
        (
            [0] => 5
        )
    [provincia] => Array
        (
            [0] => 6
        )
    [poblacion] => Array
        (
            [0] => 7
        )
)
```

Si nos fijamos, la estructura está compuesta por un *array* asociativo (cada entrada corresponde a un elemento) compuesto a su vez por *arrays* que indican la posición relativa de los elementos respecto al *array* de componentes.

Por su parte la estructura del *array* de contenidos sería:

```
Array (
  [0] => Array
      (
          [tag] => alumno
          [type] => open
          [level] => 1
      )
  [1] => Array
      (
          [tag] => matricula
          [type] => complete
          [level] => 2
          [value] => x0502
      )
  [2] => Array
      (
          [tag] => nombre
          [type] => complete
          [level] => 2
          [value] => Rodrigo
      )
  [3] => Array
      (
```

```
            [tag] => apellidos
            [type] => complete
            [level] => 2
            [value] => Vaquero Gutierrez
        )
    [4] => Array
        (
            [tag] => datos_personales
            [type] => open
            [level] => 2
        )
    [5] => Array
        (
            [tag] => direccion
            [type] => complete
            [level] => 3
            [attributes] => Array
                (
                    [tipo] => calle
                    [num] => 9
                )

            [value] => Amapola
        )
    [6] => Array
        (
            [tag] => provincia
            [type] => complete
            [level] => 3
            [value] => Madrid
        )
    [7] => Array
        (
            [tag] => poblacion
            [type] => complete
            [level] => 3
            [attributes] => Array
                (
                    [codpostal] => 28005
                )

            [value] => Madrid
        )
    [8] => Array
        (
            [tag] => datos_personales
            [type] => close
            [level] => 2
        )

    [9] => Array
        (
            [tag] => alumno
            [type] => close
            [level] => 1
        )

)
```

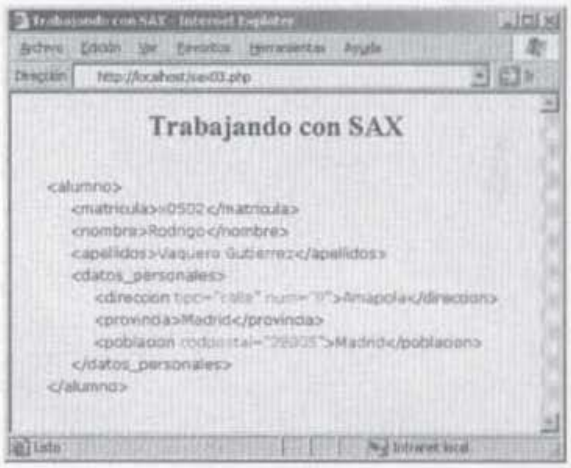
Es decir, un *array* en el que cada índice corresponde con un elemento del documento. Cada entrada es, a su vez, un *array* asociativo con las entradas `tag` (nombre de la etiqueta), `type` (tipo de etiqueta, apertura -`open`-, cierre -`close`- o completa -`complete`-), `level` (nivel de anidamiento) y `attributes` (*array* asociativo con los atributos de la etiqueta si éstos existen). Además, en el caso de que la etiqueta sea de tipo completa aparece la entrada `value` (valor textual de la etiqueta).

El siguiente código obtiene los mismos resultados que el ejemplo anterior, pero esta vez trabajando con las estructuras recuperadas con la llamada `xml_parse_into_struct()`:

```
<?
// documento XML a analizar
$documento = "prueba.xml";

//lee un documento y lo almacena en un array
function leer_documento($documento){
    if(!file_exists($documento))
        die("No se puede encontrar el fichero\"$documento\".");
    if(!($fp = @fopen($documento, "r")))
        die("No se puede leer el fichero\"$documento\".");
    return fread($fp, 4096);
}

// 1er paso
  $parser = xml_parser_create();
  xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
  xml_parser_set_option($parser, XML_OPTION_SKIP_WHITE, 1);
  xml_parser_set_option($parser, XML_OPTION_TARGET_ENCODING, "ISO-8859-1");

//2º paso
$data = leer_documento($documento);
if(!xml_parse_into_struct($parser, $data, $componentes)){
        echo "<p class='err'>";
        echo "Error XML (",xml_get_error_code($parser),"): ";
        echo "'",xml_error_string(xml_get_error_code($parser)), "'<BR>";
        echo " --- línea ",xml_get_current_line_number($parser);
        echo ", columna ",xml_get_current_column_number($parser)," ---</p>";
}
xml_parser_free($parser);
?>

<html>
 <head>
   <meta http-equiv='content-type' content='text/html;'>
   <title>Trabajando con SAX</title>
   <style type="text/css">
      .etq {color:darkblue;}
      .atr {color:green;}
      .val {color:red;}
      .err {color:red; text-align:center; line-height:20px;}
      code {font: 12px Verdana; line-height:0px; color:blue;}
   </style>
 </head>
 <body>
   <center>
   <h2>Trabajando con SAX</h2></center>
```

```
    <br><code>
    <?php
      for($i=0; $i<count($componentes); $i++){
        $nivel = $componentes[$i]['level']*20;
        echo "<P STYLE='padding-left:",$nivel,"px;'>&lt;";
        echo "<SPAN class='etq'>";
        if($componentes[$i]['type']=="close") echo "/";
        echo $componentes[$i]['tag'];
        if (array_key_exists('attributes', $componentes[$i]))
              while (list ($atributo, $valor) =
                    each ($componentes[$i]['attributes'])) {
                    echo " <SPAN class='atr'>$atributo=\"$valor</SPAN>\"";
              }
        if($componentes[$i]['type']=="complete"){
              echo "&gt;<SPAN class='val'>",$componentes[$i]['value'];
              echo "</SPAN>&lt;/",$componentes[$i]['tag'],"";
        }
         echo "</SPAN>&gt;</p>";
      }
    ?>
    </code>
  </body>
</html>
```

Como podemos observar en la siguiente imagen, el resultado obtenido es idéntico al del ejemplo anterior:



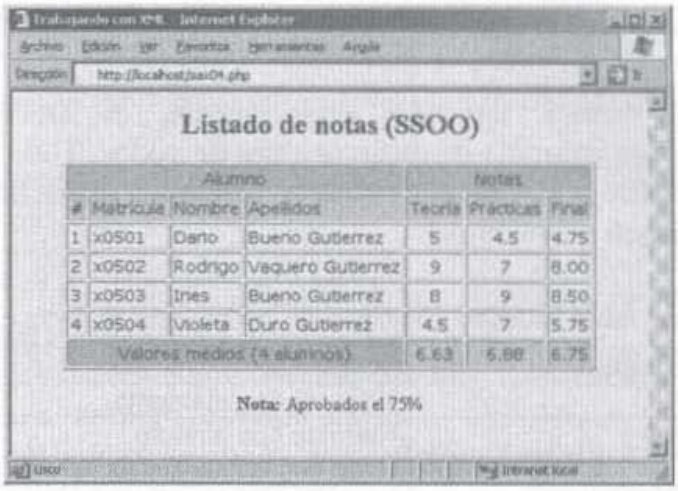
También podemos crear nuestras propias estructuras partiendo de un analizador SAX básico. El siguiente ejemplo nos muestra un *script* que analiza un documento XML y recupera su información en un conjunto de *arrays* que posteriormente se utilizarán para generar el listado completo de alumnos:

```
<?
// variables
$matricula=array();
$nombre=array();
$apellidos=array();
$teoria=array();
$practicas=array();
$elemento_actual = "";
$asignatura = "";
```

```
// documento XML a analizar
$documento = 'notas.xml';

// manejadores
function inicio_elemento($parser,$etiqueta,$atributos){
   global $elemento_actual,$asignatura;

   if($etiqueta=="notas")
     $asignatura=$atributos['asignatura'];
   $elemento_actual=$etiqueta;
}

function fin_elemento($parser,$etiqueta){
}

function datos($parser,$dato){
   global $elemento_actual,$matricula,$nombre,$apellidos,$teoria,$practicas;

   ${$elemento_actual}[]=$dato;
}

// 1er paso
  $parser = xml_parser_create();
  xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
  xml_parser_set_option($parser, XML_OPTION_TARGET_ENCODING, "ISO-8859-1");
  xml_parser_set_option($parser, XML_OPTION_SKIP_WHITE, 1);

// 2ª paso
  xml_set_element_handler($parser, "inicio_elemento","fin_elemento");
  xml_set_character_data_handler($parser,"datos");

// 10ª paso
function analizar_documento($parser,$documento){
    if(!file_exists($documento)){
        die("No se puede encontrar el fichero\"$documento\".");
    }
    if(!($fp = @fopen($documento, "r"))){
        die("No se puede leer el fichero\"$documento\".");
    }
    while($data = fread($fp, 4096)){
        if(!xml_parse($parser, $data, feof($fp))){
            echo "<p class='err'>";
            echo "Error XML (",xml_get_error_code($parser),"): ";
            echo "'",xml_error_string(xml_get_error_code($parser)), "'<BR>";
            echo " --- línea ",xml_get_current_line_number($parser);
            echo ", columna ",xml_get_current_column_number($parser);
            echo " ---</p>";
           break;
        }
    }
    fclose($fp);
    xml_parser_free($parser);
}

// 3er paso
analizar_documento($parser,$documento);

echo "<HTML><HEAD>";
echo "<meta http-equiv='content-type' content='text/html;'>";
echo "<TITLE>Trabajando con XML</TITLE></HEAD>";
echo "<BODY><CENTER><H2>Listado de notas ($asignatura)</H2>";
```

```
echo "<TABLE STYLE='font:14px Verdana;' BORDER='1' CELLPADDING='2'>";
echo "<TR BGCOLOR='cyan' ALIGN='center'><TD COLSPAN='4'>Alumno</TD>";
echo "<TD COLSPAN='3'>Notas</TD></TR>";
echo "<TR BGCOLOR='yellow'><TD>#</TD><TD>Matr&iacute;cula</TD>";
echo "<TD>Nombre</TD><TD>Apellidos</TD>";
echo "<TD>Teor&iacute;a</TD><TD>Pr&aacute;cticas</TD><TD>Final</TD></TR>";

$num_aprobados=0;
$num_alumnos=count($matricula);
for($i=0;$i<$num_alumnos;$i++){
    echo "<TR><TD>",$i+1,"</TD>";
    echo "<TD>",$matricula[$i],"</TD>";
    echo "<TD>",$nombre[$i],"</TD>";
    echo "<TD>",$apellidos[$i],"</TD>";
    echo "<TD ALIGN='center'>$teoria[$i]</TD>";
    echo "<TD ALIGN='center'>$practicas[$i]</TD>";
    $notas_finales[]=($teoria[$i]+$practicas[$i])/2;
    printf("<TD ALIGN='center'>%01.2f</TD>",$notas_finales[$i]);
    if ($notas_finales[$i]>5) $num_aprobados++;
}

echo "<TR BGCOLOR='cyan' ALIGN='center'>";
echo "<TD COLSPAN='4'>Valores medios ($num_alumnos alumnos)</TD>";
echo "<TD BGCOLOR='yellow'>";
printf("%01.2f",array_sum($teoria)/$num_alumnos);
echo "</TD>";
echo "<TD BGCOLOR='yellow'>";
printf("%01.2f",array_sum($practicas)/$num_alumnos);
echo "</TD>";
echo "<TD BGCOLOR='yellow'>";
printf("%01.2f",array_sum($notas_finales)/$num_alumnos);
echo "</TD>";
echo "</TR></TABLE>";
echo "<P><B>Nota: </B>Aprobados el ";
printf("%01.0f",($num_aprobados/$num_alumnos)*100);
echo "%</P></CENTER></BODY></HTML>";

?>
```

En el código se parte de cinco *arrays* vacíos que hacen referencia a los cinco tipos de datos que se almacenan de cada alumno. Estos *arrays* se van rellenando, dentro del manejador de los datos de tipo carácter, con los datos según van apareciendo (se utiliza una variable de variable para referenciar a los *arrays*). El resultado de ejecutar el *script* lo podemos observar en la siguiente imagen:



12.5 DOM

El DOM, *Document Object Model* o modelo de objetos de documento, se originó como una especificación que permitiera a los *scripts* Javascript y a los programas Java ser portables entre los diferentes navegadores Web. Podemos considerar, por tanto, al HTML dinámico o DHTML como su antedecesor. Pero ha evolucionado hasta convertirse en una interfaz estándar para la programación de aplicaciones (API) orientadas a trabajar con documentos HTML y documentos XML bien formados. El DOM define la estructura lógica de los documentos y el modo en que pueden ser accedidos y manipulados. De este modo, los programadores pueden definir documentos, navegar a través de su estructura y añadir, modificar o eliminar sus componentes y/o contenidos. Este modelo puede ser utilizado en una gran variedad de entornos y aplicaciones, pues es independiente de la plataforma y del lenguaje de desarrollo.

A través del DOM cualquier documento HTML o XML bien formado puede verse como una estructura de tipo arborescente, en la que cada nodo del árbol representa a un componente del documento. Además, cada elemento se modela utilizando objetos (de ahí el nombre de *modelo de objetos*), es decir, los nodos del árbol que representan al documento no hacen referencia a estructuras de datos sino a objetos que tienen métodos y propiedades. Como modelo de objetos el DOM identifica:

- las interfaces y los objetos usados para representar y manipular un documento.
- la semántica de esas interfaces y objetos, incluyendo sus atributos y comportamiento.
- las relaciones y colaboraciones entre esas interfaces y objetos.

12.5.1 Interfaces del DOM

Tal como hemos indicado anteriormente, el DOM representa los documentos como una jerarquía de distintos tipos de nodos. Todo documento tendrá un nodo raíz, diferentes nodos hijos y lo que se denomina nodos hoja o nodos finales (aquellos que no tienen nodos hijos).

La siguiente tabla muestra los tipos de nodos que maneja el DOM:

	tipo de nodo	utilización
1	ELEMENT_NODE	Nodo de tipo elemento
2	ATTRIBUTE_NODE	Nodo de tipo atributo
3	TEXT_NODE	Nodo con contenido de texto
4	CDATA_SECTION_NODE	Nodo de tipo sección CDATA
5	ENTITY_REFERENCE_NODE	Nodo de tipo referencia a entidad
6	ENTITY_NODE	Nodo de tipo entidad
7	PROCESSING_INSTRUCTION_NODE	Nodo de tipo instrucción de procesamiento
8	COMMENT_NODE	Nodo de tipo comentario
9	DOCUMENT_NODE	Nodo de tipo documento
10	DOCUMENT_TYPE_NODE	Nodo que representa al tipo de documento
11	DOCUMENT_FRAGMENT_NODE	Nodo que representa a un fragmento de documento
12	NOTATION_NODE	Nodo de tipo notación

En esta sección se muestran algunas de las interfaces más importantes del DOM, si bien éste cuenta con un juego mucho más amplio.

> **NOTA:** Para obtener más información sobre DOM y tecnologías afines, se puede recurrir a la dirección http://www.w3.org/TR/DOM-Level-3-Core .

12.5.2 Interfaz *node*

En general a cada tipo de nodo le va a corresponder un tipo de objeto y un tipo de interfaz específica, sin embargo, existe una interfaz primaria, la interfaz `node` que es implementada por la mayor parte de los componentes del DOM.

PROPIEDADES:

- `attributes`: colección de nodos (interfaz `NamedNodeMap`) que contiene los atributos del nodo actual.
- `baseURI`: URI de la base absoluta del nodo actual o `null` si no está definido.
- `childNodes`: lista de nodos (interfaz `nodeList`) hijos del nodo actual.
- `firstChild`: el primero de los nodos hijos del nodo actual.

- `lastChild`: último de los nodos hijos del nodo actual.
- `localName`: parte local del nombre cualificado del nodo actual.
- `namespaceURI`: URI del espacio de nombres utilizado por el nodo actual.
- `nextSibling`: nodo inmediatamente siguiente al nodo actual, si no existe devuelve `null`.
- `nodeName`: nombre del nodo actual.
- `nodeType`: tipo del nodo actual.
- `nodeValue`: valor del nodo actual, dependerá del tipo de nodo. La siguiente tabla muestra cómo varían los valores de `nodeName`, `nodeType` y `nodeValue` en función del tipo de nodo:

	tipo de nodo	nodeName	nodeValue
1	ELEMENT_NODE	Nombre de la etiqueta	null
2	ATTRIBUTE_NODE	Nombre del atributo	Valor del atributo
3	TEXT_NODE	"#text"	Contenido del nodo del nodo de texto
4	CDATA_SECTION_NODE	"#cdata-section"	Contenido de la sección CDATA
5	ENTITY_REFERENCE_NODE	Nombre de la entidad referenciada	null
6	ENTITY_NODE	Nombre de la entidad	null
7	PROCESSING_INSTRUCTION_NODE	Destino	Contenido del destino
8	COMMENT_NODE	"#comment"	Contenido del comentario
9	DOCUMENT_NODE	"#document"	null
10	DOCUMENT_TYPE_NODE	Nombre del tipo de documento	null
11	DOCUMENT_FRAGMENT_NODE	"#document-fragment"	null
12	NOTATION_NODE	Nombre de la notación	null

- `ownerDocument`: documento (interfaz `Document`) asociado al nodo actual.
- `parentNode`: nodo padre del nodo actual; si no existe, devuelve `null`.
- `prefix`: prefijo del espacio de nombres utilizado por el nodo actual; si no está definido, devuelve `null`.
- `previousSibling`: nodo inmediatamente anterior al nodo actual; si no existe, devuelve `null`.

- `textContent`: contenido de texto del nodo actual y todos sus descendientes.

MÉTODOS:

- `appendChild(nodo)`: añade el `nodo` proporcionado al final de la lista de hijos del nodo actual. Si el nuevo hijo ya existe en dicha lista, primero se elimina.

- `cloneNode(clonar_hijos)`: devuelve una copia del nodo actual. Si `clonar_hijos` es `true`, también se clona el subárbol bajo el nodo actual.

- `compareDocumentPosition(nodo)`: compara el `nodo` pasado como parámetro con el nodo actual teniendo en cuenta su orden dentro del documento.

- `getFeature(realización, versión)`: devuelve el objeto que implementa la API que contiene la `realización` con la `versión` especificada.

- `getUserData(clave)`: devuelve el objeto asociado a la `clave` indicada dentro del nodo actual (ver `setUserData()`).

- `hasAttributes()`: devuelve `true`, si el nodo actual, de tipo elemento, tiene atributos.

- `hasChildNodes()`: devuelve `true`, si el nodo actual tiene nodos hijos.

- `insertBefore(nodo_ins, nodo_ref)`: inserta el `nodo_ins` antes del `nodo_ref`. Si este último nodo no existe, `nodo_ins` se insertará al final de la lista de nodos hijos del nodo actual. Si `nodo_ins` es de tipo `DOCUMENT_FRAGMENT_NODE`, todos sus nodos hijos serán insertados en el orden en que aparecen. Si `nodo_ins` existe actualmente en el subárbol del nodo actual, primero es eliminado.

- `isDefaultNamespace(espacio_de_nombres)`: comprueba si el `espacio_de_nombres` proporcionado es el utilizado por defecto.

- `isEqualNode(nodo)`: comprueba si el `nodo` pasado como parámetro es igual al nodo actual.

- `isSupported(realización, versión)`: comprueba si la implementación del DOM incluye la `realización` con la `versión` solicitada y si el nodo actual la soporta.

- `lookupNamespacesURI(prefijo)`: busca los espacios de nombres asociados al `prefijo` proporcionado comenzando por el nodo actual.

- `lookupPrefix(espacio_de_nombres)`: busca los prefijos asociados al `espacio_de_nombres` dado comenzando por el nodo actual.

- `normalize()`: une todos los nodos de texto adyacentes (interfaz `Text`) del nodo actual, en un solo nodo por cada bloque de texto.

- `removeChild(nodo)`: elimina el `nodo` hijo pasado como parámetro de la lista de nodos hijos del nodo actual. Devuelve el `nodo` hijo eliminado.

- `replaceChild(nodo_nuevo, nodo_viejo)`: reemplaza el nodo hijo `nodo_viejo` con un `nodo_nuevo`. Si el `nodo_nuevo` es de tipo `DOCUMENT_FRAGMENT_NODE`, todos sus nodos hijos serán insertados en el orden en que aparecen.

- `setUserData(clave, objeto, manejador)`: asocia a un `objeto` del nodo actual una `clave`. Se puede indicar el `manejador` asociado a la clave. Si el `objeto` pasado es `null`, se eliminará cualquier asociación existente con la `clave` dada (ver `getUserData()`).

12.5.3 Interfaz *Document*

La interfaz `Document` representa al documento XML o HTML completo. Conceptualmente es el nodo raíz del árbol que representa al documento. El resto de los diferentes tipos de nodos no pueden existir fuera del contexto de un documento. Además, esta interfaz contiene los métodos necesarios para poder crear los diferentes tipos de nodos. Extiende la interfaz `node`.

PROPIEDADES:

- `actualEncoding`: codificación actual del documento o `null` en caso de que ésta no exista.

- `config`: objeto con la configuración actual del documento.

- `doctype`: DTD, tipo del documento, asociado con el documento actual. En caso de que no exista devuelve `null`.

- `documentElement`: proporciona un acceso directo al nodo raíz del árbol que representa al documento, al elemento principal (*document element*) del documento actual.

- `documentURI`: localización del documento si existe, en otro caso `null`.
- `encoding`: valor del atributo `encoding` de la declaración `xml`, con la codificación del documento o `null` en caso de que éste no exista.
- `implementation`: objeto con la implementación actual del DOM que está siendo utilizada.
- `standalone`: valor del atributo `standalone` de la declaración `xml`, o `null` en caso de que éste no exista.
- `strictErrorChecking`: indica si se fuerza o no la validación de errores.
- `version`: valor del atributo `version` de la declaración `xml`, con la versión del documento o `null` en caso de que éste no exista.

MÉTODOS:

- `adoptNode(nodo)`: cambia la propiedad `ownerDocument` del `nodo` proporcionado y de todos sus nodos hijos. Si el `nodo` tiene padre, será eliminado primero de su lista de nodos hijos.
- `createAttribute(nombre)`: crea un nuevo nodo de tipo atributo (interfaz `Attr`) con el `nombre` dado. Esta nueva instancia deberá ser asignada a un elemento (interfaz `Element`) utilizando el método `setAttributeNode()`.
- `createAttributeNS(espacio_de_nombres, nombre)`: crea un nuevo nodo de tipo atributo (interfaz `Attr`) con el `nombre` dado y calificado con el `espacio_de_nombres` proporcionado.
- `createCDATASection(cadena)`: crea un nodo de tipo sección CDATA (interfaz `CDATASection`) con el valor especificado en la `cadena` proporcionada.
- `createComment(cadena)`: crea un nodo de tipo comentario (interfaz `Comment`) con el valor especificado en la `cadena` proporcionada.
- `createDocumentFragment()`: crea un nodo de tipo fragmento de documento (interfaz `DocumentFragment`) vacío.

- `createElement(nombre)`: crea un nodo de tipo elemento (interfaz `Element`) cuyo atributo `nodeName` es `nombre`.

- `createElementNS(espacio_de_nombres, nombre)`: crea un nodo de tipo elemento (interfaz `Element`) con el `nombre` dado y calificado con el `espacio_de_nombres` proporcionado.

- `createEntityReference(nombre)`: crea un nodo de tipo referencia entidad (interfaz `entityReference`) a la entidad indicada por `nombre`.

- `createProcessingInstruction(destino, datos)`: crea un nodo de tipo instrucción de procesamiento (interfaz `ProcessingInstruction`) con el `destino` y los `datos` proporcionados.

- `createTextNode(cadena)`: crea un nodo de tipo texto (interfaz `Text`) cuyo valor es la `cadena` dada.

- `getElementById(identificador)`: devuelve el elemento (interfaz `Element`) cuyo atributo `ID` coincide con el `identificador` proporcionado. Si este elemento no existe, devuelve `null`.

- `getElementsByTagName(nombre)`: devuelve una lista de nodos (interfaz `nodeList`) con los elementos cuyo nombre de etiqueta coincida con el `nombre` proporcionado. El valor especial "`*`" hace referencia a todas las etiquetas.

- `getElementsByTagNameNS(espacio_de_nombres, nombre)`: devuelve una lista de nodos (interfaz `nodeList`) con los elementos cuyo nombre de etiqueta coincida con el `nombre` local y `espacio_de_nombres` proporcionado. El valor especial "`*`" hace referencia a todas las etiquetas.

- `importNode(nodo, importar_hijos)`: importa un `nodo` desde otro documento, creando una copia del nodo original. El nodo devuelto no tiene padre. Si `importar_hijos` es `true`, se importarán todos los hijos del nodo seleccionado.

- `normalizeDocument()`: une todos los nodos de texto adyacentes (interfaz `Text`) del documento actual en un solo nodo por cada bloque de texto.

- `renameNode(nodo, espacio_de_nombres, nombre)`: renombra el `nodo` dado con el `nombre` calificado por el `espacio_de_nombres`

proporcionado. Sólo se pueden renombrar los nodos de tipo `ELEMENT_NODE` o `ATTRIBUTE_NODE`

12.5.4 Interfaz *Element*

La interfaz `Element` representa a los elementos básicos que configuran los documentos XML o HTML, es decir, cada uno de los componentes delimitados por etiquetas de apertura y cierre o etiquetas vacías. Extiende interfaz `node`.

PROPIEDADES:

- `schemaTypeInfo`: tipo de información asociada con el elemento.
- `tagName`: nombre del elemento, nombre de la etiqueta.

MÉTODOS:

- `getAttribute(nombre)`: devuelve el valor del atributo cuyo `nombre` coincide con el proporcionado o la cadena vacía en caso de que no esté especificado su valor.
- `getAttributeNS(espacio_de_nombres, nombre)`: devuelve el valor del atributo especificado por el `nombre` local y `espacio_de_nombres` proporcionados o la cadena vacía en caso de que no esté especificado su valor.
- `getAttributeNode(nombre)`: devuelve el nodo asociado al atributo cuyo `nombre` coincide con el proporcionado (interfaz `Attr`).
- `getAttributeNodeNS(espacio_de_nombres, nombre)`: devuelve el nodo asociado al atributo especificado por el `nombre` local y `espacio_de_nombres` proporcionados (interfaz `Attr`).
- `getElementsByTagName(nombre)`: devuelve una lista de nodos (interfaz `nodeList`) con los elementos cuyo nombre de etiqueta coincida con el `nombre` proporcionado. El valor especial "`*`" hace referencia a todas las etiquetas.
- `getElementsByTagNameNS(espacio_de_nombres, nombre)`: devuelve una lista de nodos (interfaz `nodeList`) con los elementos cuyo nombre de etiqueta coincida con el `nombre` local y `espacio_de_nombres` proporcionados. El valor especial "`*`" hace referencia a todas las etiquetas.

- `hasAttribute(nombre)`: devuelve `true` si el elemento presenta un atributo cuyo `nombre` coincide con el proporcionado o dicho atributo tiene asociado un valor por defecto en el DTD asociado.

- `hasAttributeNS(espacio_de_nombres, atributo)`: devuelve `true` si el elemento presenta un atributo con el `nombre` local y `espacio_de_nombres` proporcionados o dicho atributo tiene asociado un valor por defecto en el DTD asociado.

- `removeAttribute(nombre)`: elimina el atributo que coincide con el `nombre` proporcionado. Si este atributo tiene un valor por defecto asignado en el DTD asociado, se creará de forma automática un nuevo atributo con dicho valor.

- `removeAttributeNS(espacio_de_nombres, nombre)`: elimina el atributo que coincide con el `nombre` local y `espacio_de_nombres` proporcionados. Si este atributo tiene un valor por defecto asignado en el DTD asociado, se creará de forma automática un nuevo atributo con dicho valor.

- `removeAttributeNode(nodo)`: elimina el atributo correspondiente al `nodo` proporcionado. Si este atributo tiene un valor por defecto asignado en el DTD asociado, se creará de forma automática un nuevo nodo atributo con dicho valor.

- `setAttribute(nombre, valor)`: crea un nuevo atributo con el `nombre` y `valor` proporcionados. Si actualmente existe un atributo con dicho `nombre`, simplemente modifica su valor con el `valor` proporcionado.

- `setAttributeNS(espacio_de_nombres, nombre, valor)`: crea un nuevo atributo con el `nombre` local y `espacio_de_nombres` proporcionados. Si actualmente existe un atributo con dicho nombre local dentro del mismo espacio de nombres, simplemente modifica su valor con el `valor` proporcionado.

- `setAttributeNode(nodo)`: crea un nuevo `nodo` de tipo atributo (interfaz `Attr`). Si actualmente existe un nodo atributo con igual valor en el atributo `nodeName`, es reemplazado.

- `setAttributeNodeNS(nodo)`: crea un nuevo `nodo` de tipo atributo (interfaz `Attr`). Si actualmente existe un nodo atributo con el mismo nombre local y espacio de nombres, es reemplazado.

- `setIdAttribute(nombre, esID)`: indica, si el parámetro `esID` vale `true`, que el atributo especificado por el `nombre` proporcionado es de tipo ID. Si el valor especificado en el atributo es único, el elemento podrá ser recuperado posteriormente haciendo uso del método `getElementsById()` presente en el documento (interfaz `Document`) que lo contiene.

- `setIdAttributeNS(espacio_de_nombres, nombre, esID)`: indica, si el parámetro `esID`, vale `true`, que el atributo especificado por el `nombre` local y `espacio_de_nombres` proporcionados es de tipo ID. Si el valor especificado en el atributo es único, el elemento podrá ser recuperado posteriormente haciendo uso del método `getElementsById()` presente en el documento (interfaz `Document`) que lo contiene.

- `setIdAttributeNode(nodo, esID)`: indica, si el parámetro `esID` vale `true`, que el atributo especificado en el `nodo` proporcionado es de tipo ID. Si el valor especificado en el atributo es único, el elemento podrá ser recuperado posteriormente haciendo uso del método `getElementsById()` presente en el documento (interfaz `Document`) que lo contiene.

12.5.5 Interfaz *Attr*

La interfaz `Attr` representa un atributo dentro de un objeto de tipo `Element`. Generalmente los valores permitidos para el atributo estarán definidos dentro de un DTD. Extiende la interfaz `node`, pero como los atributos no tienen actualmente nodos hijos, el DOM no los considera como parte del árbol del documento, sino que los considera como propiedades de los elementos. Por ello, los atributos heredados `parentNode`, `previousSibling` y `nextSibling` tienen el valor `null` en los objetos de tipo `Attr`.

PROPIEDADES:

- `name`: nombre del atributo actual.

- `ownerElement`: nodo asociado al elemento (interfaz `Element`) en el cual está presente el atributo actual.

- `schemaTypeInfo`: tipo de información asociada con el atributo actual.

- `specified`: toma el valor `true` cuando al atributo se le ha asignado explícitamente un valor en la instancia del documento actual.

- `value`: valor del atributo. Los caracteres y las referencias a entidades generales son reemplazadas por sus correspondientes valores.

MÉTODOS:

- `isID()`: devuelve `true` si el atributo actual es de tipo ID.

12.5.6 Interfaz *ProcessingInstruction*

La interfaz `ProcessingInstruction` representa a las instrucciones de procesamiento utilizadas en XML como medio para mantener información específica del procesador dentro del texto del documento. Extiende la interfaz `node`.

PROPIEDADES:

- `data`: contenido de la instrucción de procesamiento.

- `target`: destino de la instrucción de procesamiento. Representa a la propiedad `target` definida en XML

12.5.7 Interfaz *characterData*

Extiende la interfaz `node` añadiendo un conjunto de atributos y métodos para facilitar el manejo de datos de tipo carácter. No existen en el DOM objetos directamente definidos a partir de esta interfaz, es extendida a su vez por otros componentes del DOM.

PROPIEDADES:

- `data`: los datos de tipo carácter presentes en el nodo que implementa esta interfaz. El DOM no pone límites a la cantidad de datos que se pueden almacenar en un nodo de este tipo.

- `length`: número de caracteres, unidades de 16 bits (se utiliza codificación UTF-16).

MÉTODOS:

- `appendData(cadena)`: añade la `cadena` proporcionada al final de los datos de tipo carácter del nodo.

- `deleteData(desplazamiento, cuenta)`: elimina, de los datos de tipo carácter del nodo, el número de caracteres fijados por `cuenta` a partir de la posición indicada por `desplazamiento`.

- `insertData(desplazamiento, cadena)`: añade, a los datos de tipo carácter del nodo, la `cadena` proporcionada a partir de la posición indicada por `desplazamiento`.

- `replaceData(desplazamiento, cuenta, cadena)`: reemplaza, en los datos de tipo carácter del nodo, el número de caracteres fijados por `cuenta` a partir de la posición indicada por `desplazamiento` por la `cadena` de caracteres proporcionada.

- `substringData(desplazamiento, cuenta)`: extrae, de los datos de tipo carácter del nodo, el número de caracteres fijados por `cuenta` a partir de la posición indicada por `desplazamiento`.

12.5.8 Interfaz *Text*

Extiende la interfaz `characterData` y representa el contenido textual de un elemento (interfaz `Element`) o de un atributo (interfaz `Attr`). Aunque lo habitual es que sólo exista un nodo de tipo `Text` por cada bloque de texto, es posible crear nodos de texto adyacentes para representar los contenidos de un elemento sin la necesidad de la utilización de marcado.

PROPIEDADES:

- `wholeText`: contiene el texto asociado al nodo de texto y a todos los nodos de texto adyacentes al mismo en caso de que existan. El contenido se concatena en base al orden de aparición dentro del documento.

MÉTODOS:

- `isWhitespaceInElementContent()`: devuelve `true` si el nodo de texto contiene espacios en blanco dentro del contenido del elemento.

- `replaceWholeText(cadena)`: reemplaza el contenido actual del nodo de texto por la `cadena` proporcionada.

- `splitText(posición)`: divide el nodo de texto actual, por la `posición` indicada, en dos nuevos nodos adyacentes que permanecerán como hermanos dentro del árbol del documento.

12.5.9 Interfaz *CDATASection*

Extiende la interfaz `Text` y representa a las secciones CDATA utilizadas para enmascarar bloques de texto que contiene caracteres especiales que de otro modo pueden ser considerados como parte del marcado.

12.5.10 Interfaz *Comment*

Extiende la interfaz `characterData` y representa el contenido de un comentario, es decir, todos los caracteres comprendidos entre las cadenas "`<!--`" y "`-->`".

12.5.11 Interfaz *Entity*

Extiende la interfaz `node` y representa a una entidad conocida dentro del documento XML. Modela a la entidad, no a su declaración. El procesador XML puede optar por expandir completamente todas las entidades antes de definir el modelo de estructura DOM, en este caso, no existirán nodos de tipo referencia a entidad (interfaz `EntityReference`) dentro del árbol del documento. Los nodos de tipo entidad no tienen padre.

PROPIEDADES:

- `actualEncoding`: codificación actual de la entidad, en caso de que ésta sea una entidad externa, o `null` en otro caso.
- `encoding`: codificación presente en la declaración de la entidad, en caso de que ésta sea una entidad externa, o `null` en otro caso.
- `notationName`: nombre de la notación para la entidad, en caso de que ésta sea una entidad sin procesar, o `null` en otro caso.
- `publicId`: identificador público asociado a la entidad en caso de que esté especificado o `null` en otro caso.
- `systemId`: identificador de sistema asociado a la entidad en caso de que esté especificado o `null` en otro caso.
- `version`: número de versión de la entidad, en caso de que ésta sea una entidad externa, o `null` en otro caso.

12.5.12 Interfaz *EntityReference*

Extiende la interfaz `node` y se utiliza para representar las referencias a entidades dentro del árbol del documento.

12.5.13 Interfaz *Notation*

Esta interfaz representa una notación declarada dentro de un DTD. Las notaciones son utilizadas en las declaraciones del formato de entidades sin procesar y de los destinos en las instrucciones de procesamiento. Extiende la interfaz `node`.

PROPIEDADES:

- `publicId`: identificador público de la notación en caso de que esté especificado o `null` en otro caso.
- `systemId`: identificador de sistema asociado a la notación en caso de que esté especificado o `null` en otro caso.

12.5.14 Interfaz *DocumentType*

Extiende la interfaz `node` y se utiliza para acceder a las entidades definidas para un documento.

PROPIEDADES:

- `entities`: colección de las entidades generales, internas y externas, declaradas en el DTD. No contiene entidades de parámetro. Cada elemento de la colección implementa la interfaz `Entity`.
- `internalSubset`: cadena que representa el subconjunto interno (DTD interno) o `null` en otro caso.
- `name`: nombre del DTD.
- `notations`: colección de las notaciones declaradas en el DTD. Cada elemento de la colección implementa la interfaz `Notation`.
- `publicId`: identificador público del subconjunto externo (DTD externo).

- `systemId`: identificador de sistema asociado al subconjunto externo (DTD externo).

12.5.15 Interfaz *DocumentFragment*

Extiende la interfaz `node` y se utiliza para representar una porción del árbol asociado a un documento. Puede considerarse como una versión "mínima" de los objetos de tipo documento (interfaz `Document`).

12.5.16 Interfaz *nodeList*

Proporciona una abstracción de una colección ordenada de nodos, sin definir o restringir cómo debe implementarse dicha colección.

PROPIEDADES:

- `lenght`: número de nodos de la lista.

MÉTODOS:

- `item(posición)`: devuelve el nodo con la `posición` indicada dentro de la lista. El primer nodo de la lista tiene la posición 0. Si la `posición` indicada es mayor que el número de nodos en la lista, devuelve `null`.

12.5.17 Interfaz *NamedNodeMap*

Proporciona una colección de nodos, sin un orden en particular, a los que se puede acceder por su nombre. También pueden ser accedidos en base a un índice ordinal.

PROPIEDADES:

- `length`: número de nodos de la colección.

MÉTODOS:

- `getNamedItem(nombre)`: devuelve el nodo cuyo `nombre` coincide con el proporcionado o `null` si no encuentra el nodo en la colección.

- `getNamedItemNS(espacio_de_nombres,nombre)`: devuelve el nodo cuyo `nombre` local y `espacio_de_nombres` coinciden con los proporcionados o `null` si no encuentra el nodo en la colección.

- `item(posición)`: devuelve el nodo con la `posición` indicada dentro de la colección. El primer nodo de la lista tiene la posición 0. Si la `posición` indicada es mayor que el número de nodos en la lista, devuelve `null`.

- `removeNamedItem(nombre)`: elimina de la colección el nodo cuyo `nombre` coincide con el proporcionado.

- `removeNamedItemNS(espacio_de_nombres, nombre)`: elimina de la colección el nodo cuyo `nombre` local y `espacio_de_nombres` coinciden con los proporcionados.

- `setNamedItem(nodo)`: añade a la colección el `nodo` proporcionado. Si actualmente existe en la colección un nodo con el mismo nombre, es reemplazado.

- `setNamedItemNS(nodo)`: añade a la colección el `nodo` proporcionado haciendo uso de su nombre local (atributo `localName`) y espacio de nombres (atributo `namespacesURI`). Si actualmente existe en la colección un nodo con el mismo nombre local y espacio de nombres, es reemplazado.

12.5.18 Ejemplos usando DOM

A través de los siguientes ejemplos vamos a ver cómo trabajar desde PHP con el DOM y la estructura arborescente que impone a los documentos. Tal y como hemos visto en la descripción de la interfaz *Document* del estándar DOM, los objetos que derivan de ella representan a un documento XML o HTML completo.

En PHP la clase que implementa esta interfaz es la clase *DOMDocument* y, a partir de ella, podremos invocar todos sus métodos y usar todas sus propiedades. Además, para que la gestión de almacenamiento/carga de ficheros hacia o desde el documento sea sencilla, esta clase cuenta con los siguientes métodos:

- `load(fichero)`: carga el contenido del `fichero` XML en el objeto de tipo documento DOM. El `fichero` XML debe estar bien formado.

- `loadXML(cadenaXML)`: carga la `cadenaXML` en el objeto de tipo documento DOM. La `cadenaXML` debe estar bien formada.

- `loadHTMLFile(fichero)`: carga el contenido del `fichero` HTML en el objeto de tipo documento DOM. El `fichero` HTML no tiene que estar bien formado.

- `loadHTML(cadenaHTML)`: carga la `cadenaHTML` en el objeto de tipo documento DOM. La `cadenaHTML` no tiene que estar bien formada.

- `save(fichero)`: crea un `fichero` XML con la información contenida en el objeto de tipo documento DOM.

- `saveXML([nodo])`: crea una cadena XML con la información contenida en el objeto de tipo documento DOM. Puede indicarse el `nodo` del documento DOM a partir del cual generar la cadena.

- `saveHTMLFile(fichero)`: crea un `fichero` HTML con la información contenida en el objeto de tipo documento DOM.

- `saveHTML()`: crea una cadena HTML con la información contenida en el objeto de tipo documento DOM.

Dentro de las propiedades añadidas cabe destacar `preserveWhiteSpace` utilizada para indicar si los espacios en blanco no significativos de un documento XML o HTML se deben o no tener en cuenta.

En el primero de los ejemplos simplemente creamos un nuevo objeto de tipo *DOMDocument*, que utilizamos para cargar y visualizar el documento XML `notas.xml`:

```
<?php
        $documento = new domDocument();
        $documento->load('notas.xml');
        $cadaena_xml = $documento->saveXML();
        echo $cadena_xml;
?>
```

Como podemos observar en el código, utilizamos el constructor `domDocument()` para crear el nuevo objeto de tipo documento DOM (*DOMDocument*). El resultado se muestra en la siguiente imagen:

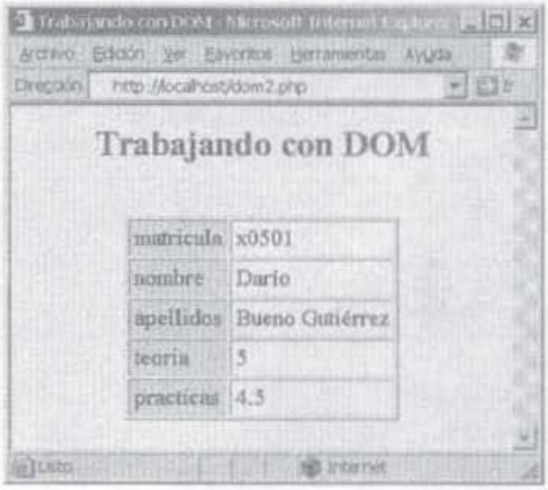
En el siguiente ejemplo se recuperan los datos del primero de los alumnos contenidos en el fichero anterior:

```
<html>
 <head>
   <title>Trabajando con DOM</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con DOM</h2><br>
   <?php
     if (!file_exists('notas.xml')) {
        exit('No se encuentra el fichero "notas.xml".');
     }
     $docXML = new domDocument();
     $docXML->preserveWhiteSpace=false;
     $docXML->load('notas.xml');
   ?>
   <table border="1" cellpadding="4">
   <?php
        $notas=$docXML->documentElement;
        $primer_alumno = $notas->firstChild;
        foreach ($primer_alumno->childNodes as $datos_alumno) {
              echo '<tr>';
              echo '<td bgcolor="yellow">',$datos_alumno->nodeName,'</td>';
```

```
                echo '<td>',$datos_alumno->nodeValue,'</td>';
                echo '</tr>';
            }
        ?>
        </table>
        </center>
    </body>
</html>
```

Como observamos del *script* la variable `$notas` contiene el nodo raíz del documento XML, de este modo al recuperar su primer hijo, en la variable `$primer_alumno`, obtenemos el nodo del que cuelgan todos los datos del primero de los alumnos. A partir de este último se recuperan sus nodos hijos, en la variable `$datos_alumno`, de los que se van obteniendo tanto el nombre del nodo (`nodeName`) como su valor (`nodeValue`).

Si lo vemos desde el punto de vista del documento DOM, lo que hemos hecho ha sido recorrer desde el primer hijo del nodo raíz todos los nodos que cuelgan de él. El resultado se puede observar en la siguiente imagen:

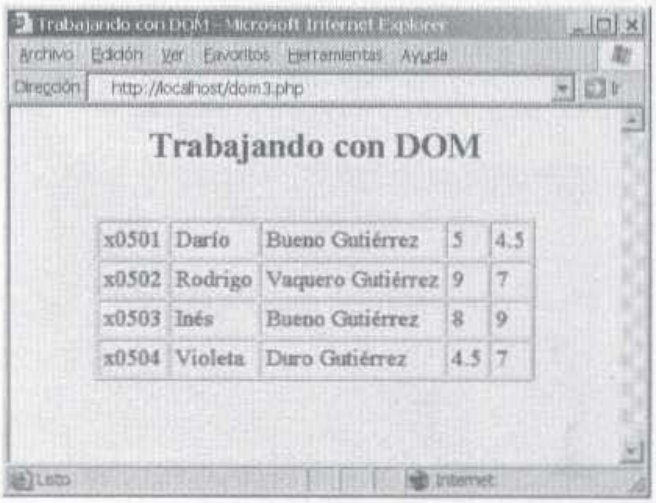


Para recuperar los datos de todos los alumnos deberíamos modificar el *script* de la siguiente manera:

```
<html>
 <head>
   <title>Trabajando con DOM</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con DOM</h2><br>
   <?php
     if (!file_exists('notas.xml')) {
        exit('No se encuentra el fichero "notas.xml".');
     }
     $docXML = new domDocument();
     $docXML->preserveWhiteSpace=false;
     $docXML->load('notas.xml');
   ?>
   <table border="1" cellpadding="4">
```

```
    <?php
        $notas=$docXML->documentElement;
        $lista_alumnos = $notas->childNodes;
        foreach ($lista_alumnos as $alumno) {
                echo '<tr>';
                foreach ($alumno->childNodes as $datos_alumno) {
                        echo '<td>',$datos_alumno->nodeValue,'</td>';
                }
                echo '</tr>';
        }
    ?>
    </table>
    </center>
  </body>
</html>
```

El resultado se observa en la siguiente imagen:

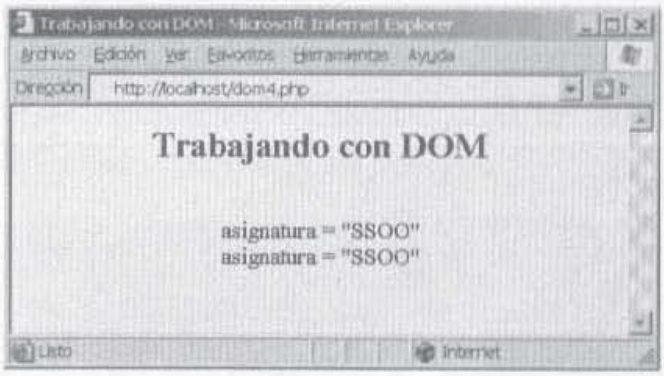


Para recuperar cualquiera de los atributos presentes en un elemento contamos con dos posibilidades: hacer uso de la propiedad `attributes` disponible en los objetos de tipo nodo (interfaz `node`) que recuperaría todos sus atributos como una colección de nodos o bien acceder al atributo deseado mediante el método `getAttribute()`. El siguiente ejemplo nos lo muestra de forma práctica:

```
<html>
 <head>
   <title>Trabajando con DOM</title>
 </head>
 <body>
   <center>
   <h2>Trabajando con DOM</h2><br>
   <?php
     if (!file_exists('notas.xml')) {
        exit('No se encuentra el fichero "notas.xml".');
     }
     $docXML = new domDocument();
     $docXML->preserveWhiteSpace=false;
     $docXML->load('notas.xml');
   ?>
   <?php
        $notas=$docXML->documentElement;
```

```
        foreach($notas->attributes as $atributo) {
           echo $atributo->name,' = "',$atributo->value,'"<br>';
        }
        echo 'asignatura = "',$notas->getAttribute('asignatura'),'"<br>';
    ?>
    </center>
  </body>
</html>
```

Como se puede observar en la siguiente imagen, el resultado obtenido es el mismo por cualquiera de los dos métodos:



El siguiente ejemplo muestra cómo obtener sólo ciertos elementos de nuestro documento, en nuestro caso cómo recuperar los nombres de los alumnos:

```
<html>
 <head>
   <title>Trabajando con DOM</title>
   <meta http-equiv='content-type' content='text/html; charset=UTF-8'>
</head>
 <body>
   <center>
   <h2>Trabajando con DOM</h2><br>
   <?php
     if (!file_exists('notas.xml')) {
        exit('No se encuentra el fichero "notas.xml".');
     }
     $docXML = new domDocument();
     $docXML->preserveWhiteSpace=false;
     $docXML->load('notas.xml');
   ?>
   <table border="1" cellpadding="4" cellspacing="4"><tr>
   <?php
        $notas=$docXML->documentElement;
        $nombres = $notas->getElementsByTagName("nombre");
        foreach($nombres as $nombre) {
           echo '<td>',$nombre->nodeValue,'</td>';
        }
   ?>
   </tr></table>
   </center>
  </body>
</html>
```

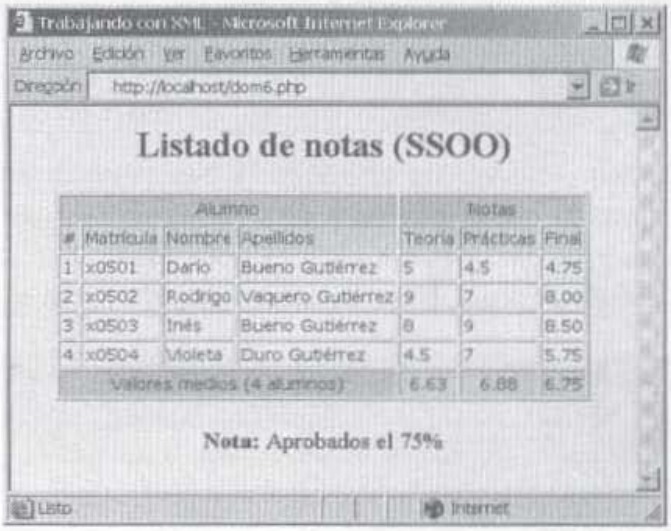
El resultado se muestra en la siguiente imagen:

El siguiente ejemplo nos muestra cómo resolver, haciendo uso de los métodos proporcionados por el DOM, el ejemplo del listado de notas completo:

```
<?php
        $docXML = new domDocument();
        $docXML->preserveWhiteSpace=false;
        $docXML->load('notas.xml');
        $notas=$docXML->documentElement;

        echo "<HTML><HEAD>";
        echo "<meta http-equiv='content-type' content='text/html;>";
        echo "<TITLE>Trabajando con XML</TITLE></HEAD>";
        echo "<BODY><CENTER><H2>Listado de notas (";
        echo $notas->getAttribute('asignatura'),")</H2>";
        echo "<TABLE STYLE='font:14px Verdana;' BORDER='1' CELLPADDING='2'>";
        echo "<TR BGCOLOR='cyan' ALIGN='center'><TD COLSPAN='4'>Alumno</TD>";
        echo "<TD COLSPAN='3'>Notas</TD></TR>";
        echo "<TR BGCOLOR='yellow'><TD>#</TD><TD>Matr&iacute;cula</TD>";
        echo "<TD>Nombre</TD><TD>Apellidos</TD>";
        echo "<TD>Teoría</TD><TD>Prácticas</TD><TD>Final</TD></TR>";

        $num_alumnos=0;
        $num_aprobados=0;
        $notas_teoria=0;
        $notas_practicas=0;
        $notas_finales=0;
        $alumnos = $notas->getElementsByTagName("alumno");
        foreach($alumnos as $alumno) {
             $num_alumnos++;
             echo "<TR><TD>$num_alumnos</TD>";
             foreach($alumno->childNodes as $datos) {
                 echo "<TD>",$datos->nodeValue,"</TD>";
             }
             $nota_teoria =
                          floatval($alumno->childNodes->item(3)->nodeValue);
             $nota_practica =
                          floatval($alumno->childNodes->item(4)->nodeValue);
              $notas_teoria+=$nota_teoria;
              $notas_practicas+=$nota_practica;
              $nota_final_alumno=($nota_teoria+$nota_practica)/2;
              $notas_finales+=$nota_final_alumno;
              printf("<TD ALIGN='center'>%01.2f</TD>",$nota_final_alumno);
              if($nota_final_alumno>5){
                 $num_aprobados++;
              }
        }
```

```
        echo "<TR BGCOLOR='cyan' ALIGN='center'>";
        echo "<TD COLSPAN='4'>Valores medios ($num_alumnos alumnos)</TD>";
        echo "<TD BGCOLOR='yellow'>";
        printf("%01.2f",$notas_teoria/$num_alumnos);
        echo "</TD>";
        echo "<TD BGCOLOR='yellow'>";
        printf("%01.2f",$notas_practicas/$num_alumnos);
        echo "</TD>";
        echo "<TD BGCOLOR='yellow'>";
        printf("%01.2f",$notas_finales/$num_alumnos);
        echo "</TD>";
        echo "</TR></TABLE>";
        echo "<P><B>Nota: </B>Aprobados el ";
        printf("%01.0f",($num_aprobados/$num_alumnos)*100);
        echo "%</P></CENTER></BODY></HTML>";
?>
```

El resultado se observa en la siguiente imagen:

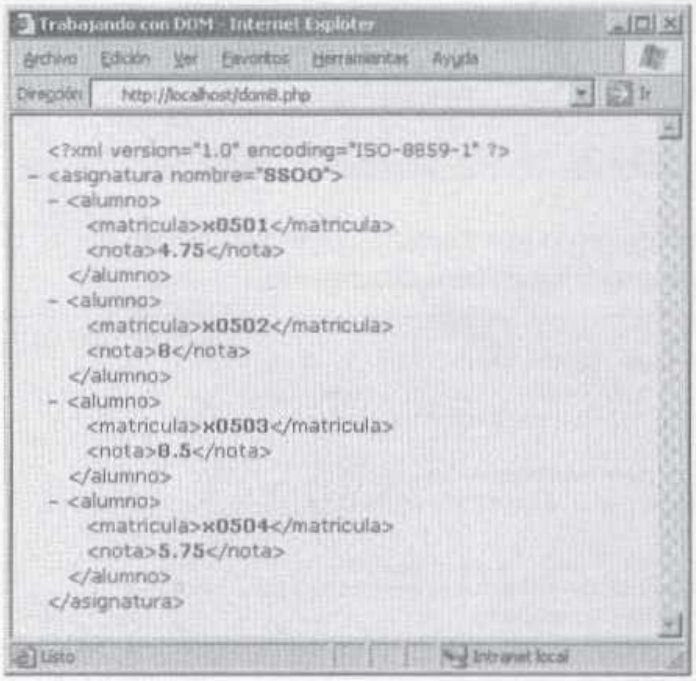


El siguiente ejemplo nos muestra cómo generar un nuevo documento XML a partir de los datos contenidos en otro documento:

```
<?php
  $docXMLorigen = new domDocument();
  $docXMLorigen->preserveWhiteSpace=false;
  $docXMLorigen->load('notas.xml');

  $docXMLdestino = new domDocument();
  $docXMLdestino->encoding = "ISO-8859-1";


  $asig = $docXMLdestino->createElement("asignatura");
  $asig->setAttribute("nombre",
        $docXMLorigen->documentElement->getAttribute('asignatura'));

  $lista_alumnos = $docXMLorigen->documentElement-
>getElementsByTagName("alumno");
  foreach($lista_alumnos as $un_alumno) {

    $alumno = $docXMLdestino->createElement("alumno");
```

```
    $matricula = $docXMLdestino->createElement("matricula");
    $nota = $docXMLdestino->createElement("nota");

    $matriculaTXT = $docXMLdestino->createTextNode($un_alumno->childNodes-
>item(0)->nodeValue);

    $nota_teoria = floatval($un_alumno->childNodes->item(3)->nodeValue);
    $nota_practica = floatval($un_alumno->childNodes->item(4)->nodeValue);
    $nota_final_alumno=($nota_teoria+$nota_practica)/2;
    $notaTXT = $docXMLdestino->createTextNode($nota_final_alumno);

    $matricula->appendChild($matriculaTXT);
    $nota->appendChild($notaTXT);

        $alumno->appendChild($matricula);
        $alumno->appendChild($nota);
        $asig->appendChild($alumno);
    }

    $docXMLdestino->appendChild($asig);
    print $docXMLdestino->saveXML();

    unset($docXMLorigen);
    unset($docXMLdestino);
?>
```

Como podemos observar en el *script* el nuevo documento cuenta con un elemento raíz denominado `asignatura` y un nodo `alumno`, que contiene a su vez un nodo `matricula` y un nodo `nota` (con la nota final), por cada nodo `alumno` presente en el documento original. El resultado se muestra en la siguiente imagen:



Por último, el siguiente ejemplo nos muestra cómo recuperar todo un documento XML sin conocer a priori su contenido:

```
<?php

function imprimirArbolDOM($nodoDOM) {
    static $nivel = 0;

    if ($nodoDOM_hijo = $nodoDOM->firstChild){
        $esEtiqueta = 0;

        $espaciado = "\n<br>".str_repeat(" ", ($nivel * 2));
        while($nodoDOM_hijo){
            if ($nodoDOM_hijo->nodeType == XML_TEXT_NODE)
                echo trim($nodoDOM_hijo->nodeValue);
            elseif ($nodoDOM_hijo->nodeType == XML_ELEMENT_NODE){
                $esEtiqueta = 1;
                echo $espaciado;
                echo '&lt;', $nodoDOM_hijo->nodeName;
                if ($nodoDOM_hijo->hasAttributes()){
                    $atributos = $nodoDOM_hijo->attributes;
                    foreach ($atributos as $atributo){
                        echo ' ',$atributo->name,'="',$atributo->value,'"';
                    }
                }
                echo '&gt;';
                if ($nodoDOM_hijo->hasChildNodes()){
                    $nivel++;
                    if (imprimirArbolDOM($nodoDOM_hijo)){
                        echo $espaciado;
                    }
                    $nivel--;
                }
                echo '&lt;/', $nodoDOM_hijo->nodeName, '&gt;';
            }
            $nodoDOM_hijo = $nodoDOM_hijo->nextSibling;
        }
        return $esEtiqueta;
    }
}

$docXML = new domDocument();
$docXML->preserveWhiteSpace=false;
$docXML->load('notas.xml');

?>

<HTML>
  <HEAD>
    <meta http-equiv='content-type' content='text/html; charset=UTF-8'>
    <TITLE>Trabajando con XML</TITLE>
  </HEAD>
  <BODY>
    <CENTER><H2>Trabajando con DOM</H2></CENTER>
    <PRE>
      <?php imprimirArbolDOM($docXML); ?>
    </PRE>
  </BODY>
</HTML>
```

El resultado se muestra en la siguiente imagen:


Trabajando con XML - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Dirección http://localhost/dom7.php
Trabajando con DOM
<notas asignatura="ESOO">
<alumno>
<matricula>x0501</matricula>
<nombre>Dario</nombre>
<apellidos>Bueno Gutiérrez</apellidos>
<teoria>5</teoria>
<practicas>4.5</practicas>
</alumno>
<alumno>
<matricula>x0502</matricula>
<nombre>Rodrigo</nombre>
<apellidos>Vaquero Gutiérrez</apellidos>
<teoria>9</teoria>
<practicas>7</practicas>
</alumno>
<alumno>
<matricula>x0503</matricula>
<nombre>Inés</nombre>
<apellidos>Bueno Gutiérrez</apellidos>
<teoria>8</teoria>
<practicas>9</practicas>
</alumno>
<alumno>
<matricula>x0504</matricula>
<nombre>Violeta</nombre>
<apellidos>Duro Gutiérrez</apellidos>
<teoria>4.5</teoria>
<practicas>7</practicas>
</alumno>
</notas>
Listo
Internet

CAPÍTULO 13

EJEMPLO DE APLICACIÓN: *WEBMAIL*

Como aplicación práctica en la que poder ver las posibilidades de PHP, vamos a implementar una aplicación muy usada actualmente. Se trata de un cliente de correo electrónico que nos permita manipular los mensajes de nuestra cuenta de correo a través de un navegador Web. Las ventajas que este tipo de servicio ofrece son:

- Los mensajes están siempre almacenados en el servidor. Esto significa que podremos leer el correo desde cualquier sitio (casa de un amigo, un cybercafé, el trabajo, etc.) sin que los mensajes se descarguen en el ordenador desde donde accedemos.

- No es necesario disponer de un cliente de correo específico (Netscape Messenger, Outlook, Eudora, Pegasus, etc.) en el ordenador donde queramos revisar el correo, pues basta con cualquier navegador (programa ubicuo donde los haya, ya que es prácticamente imprescindible para el acceso a Internet).

- Existe menos probabilidad de contagio de virus tipo aplicaciones eml, vbs, etc., ya que este cliente no los ejecuta por el simple hecho de posicionarnos encima de un mensaje.

Vistas las ventajas de este tipo de servicio, antes de abordar el desarrollo e implementación de la aplicación, lo siguiente que necesitamos es conocer los requisitos que ha de cumplir, es decir, qué funcionalidades debe proporcionar nuestro servicio *webmail*:

- Poder conectarse a servidores IMAP[1].
- Disponer de tres carpetas: *Entrada*, *Enviados* y *Papelera*.
- Ordenar las cabeceras de los mensajes por remitente, asunto, tamaño o fecha de recepción.
- Los mensajes no leídos aparecen resaltados en negrita.
- Visualizar los mensajes.
- Indicación del número de ficheros adjuntos por mensaje.
- Responder al remitente de un mensaje.
- Responder a todos los destinatarios de un mensaje.
- Reenviar un mensaje a otro usuario.
- Adjuntar ficheros a un mensaje para su envío.
- Guardar en disco los ficheros adjuntos de un mensaje recibido.
- Mover mensajes de una carpeta a cualquiera de las otras dos.
- Borrar mensajes.
- Seleccionar mensajes para ser eliminados o movidos a otra carpeta.

13.1 ESTRUCTURA GENERAL

De las especificaciones anteriores se desprende que necesitaremos al menos los siguientes programas:

- Entrada al *webmail*.
- Listado de mensajes.

[1] No confundir con el protocolo de correo SMTP (*Simple Mail Transfer Protocol*), el cual sólo se usa para enviar correo. Algunas de las implementaciones más comunes de este protocolo son los programas *sendmail* y *postfix*.

- Ver un mensaje.
- Componer mensajes.
- Salir de la aplicación.

13.1.1 Variables de sesión

Dado que el protocolo HTTP por el cual se comunican cliente y servidor Web no guarda estado de ninguna conexión previa, y dado que necesitamos guardar ciertos datos del usuario (nombre, clave, opciones de ordenación de los mensajes, etc.) durante todo el tiempo que esté dentro de la aplicación, estos datos han de ser guardados en variables de sesión, pues, de esta forma, lograremos que se comporten como variables *globales* para todos los *scripts*. Las variables que usaremos en la aplicación son:

- `WM`: Hace de centinela para comprobar que se ha entrado en la aplicación.
- `WM_USR`: Contiene el nombre del usuario (*login*).
- `WM_CLAVE`: Palabra clave del usuario (*password*).
- `WM_BUZON_ACTUAL`: Buzón o carpeta que está siendo revisada por el usuario.
- `WM_CAMPO_ORD`: Campo por el que se están ordenando las cabeceras.
- `WM_SENTIDO`: Ordenación ascendente o descendente de las cabeceras.
- `WM_NUM_FICH_ADJUNTOS`: Número máximo de ficheros a adjuntar.
- `WM_TAM_FICH_ADJUNTOS`: Tamaño máximo de fichero para adjuntar.
- `WM_CUOTA`: Espacio máximo en disco asignado al usuario.
- `WM_SRV_IMAP`: Servidor IMAP donde el usuario tiene su correo.

- `WM_DIR_BUZONES`: Directorio en el servidor donde se guardan las carpetas.

Para evitar que se entre en la aplicación sin haber proporcionado el nombre de usuario y clave correctas, en todos los programas se pregunta por la existencia de una variable sesión (`WM`) introducida para el caso. Si no estuviera registrada dicha variable, se redirige al navegador a la página inicial de la aplicación:

```
session_start();
if (!session_is_registered(WM))
  header("Location: index.php");
```

Hay que resaltar que en este esquema de variables *globales*, en el que se almacena el usuario y su clave en variables de sesión, existe un riesgo potencial para la seguridad del servidor, ya que estos valores estarán almacenados *en claro* en un fichero físico. Por esta razón, el directorio donde se guardan estos ficheros (determinado en el fichero de configuración `php.ini`) debe estar protegido con los permisos adecuados para evitar poner en grave peligro la seguridad del sistema.

La máquina servidora de correo no tiene por qué ser la misma que aquélla donde se ejecuta el servidor Web, esto es, aquélla donde se va a ejecutar la aplicación *webmail*. Por tanto, disponemos de una variable donde almacenar la máquina donde se encuentra el servidor IMAP: `WM_SRV_IMAP`.

Para no aumentar la complejidad y, por tanto, el seguimiento, del ejemplo de aplicación no hemos usado una base de datos en la que se almacenen las particularidades de cada usuario: espacio máximo de disco del que se puede disponer, número máximo de ficheros a adjuntar, tamaño máximo de éstos, preferencias de ordenación de los mensajes, etc. En particular, para este ejemplo, tanto la última carpeta visualizada como las preferencias de ordenación sí que deberían guardarse para que en la siguiente conexión el usuario tuviera la configuración que dejó al salir.

13.1.2 Botonera

Vamos a proporcionar en cada página de la aplicación una botonera en la que estarán todas las operaciones que pueda realizar el usuario (ver pantalla en la página siguiente).

Ahora bien, estos botones estarán activados según tengan sentido o no; por ejemplo, el botón de eliminar un mensaje no estará activo cuando se esté escribiendo un mensaje para su envío o cuando se esté en la pantalla donde aparecen las cabeceras de los mensajes.

Se ha previsto una función `pinta_botonera()` que imprimirá dicha botonera con los valores que se le pasen como parámetro. Por ello, todos los *scripts* que generen la mencionada cabecera inicializarán el *array* con los valores convenientes a cada caso. Por ejemplo:

```
$botonera=array();
$botonera[0]="1"; // Actualizar
$botonera[1]="1"; // Redactar
$botonera[2]="0"; // Responder
$botonera[3]="0"; // Responder Todos
$botonera[4]="0"; // Reenviar
$botonera[5]="0"; // Eliminar
$botonera[6]="1"; // Logout

pinta_botonera($botonera);
```

La función al completo:

```
function pinta_botonera($valores_botones) {
  global $i_arial, $f_arial;
  // $botones=array (array(href, leyenda, imagen)
  $botones=array (
          array("wm_ver_cabs.php",       "Actualizar",
                "images/checkmail.gif"),
          array("wm_componer.php",       "Redactar",
                "images/msg_new.gif"),
          array("javascript:reenvia(1)", "Responder",
                "images/msg_reply.gif"),
```

```
            array("javascript:reenvia(2)", "Responder Todos",
                  "images/msg_reply.gif"),
            array("javascript:reenvia(3)", "Reenviar",
                  "images/msg_forward.gif"),
            array("wm_selecs.php",         "Eliminar",
                  "images/msg_delete.gif"),
            array("wm_salida.php",         "",
                  "images/logout.gif"),
            );

  $celda="<td bgcolor=\"#FFFBAD\">$i_arial" ;
  $fin_celda="$f_arial</td>" ;

  echo '<center>';
  echo '<table><tr>';

  echo "$celda
        <form name='f_selec_buzon' method='get' action='wm_ver_cabs'>
        <img src='images/checkmail.gif'>

        <select name=s_buzon
         onChange='document.f_selec_buzon.submit();'>";

  echo "<option value=\"\">Actualizar</option>
        <option value=\"Entrada\">Entrada</option>
        <option value=Enviados>Enviados</option>
        <option value=Papelera>Papelera</option>
        </select>
        </td>
        </form>";

  for ($i=1; $i<count($valores_botones); $i++) {
    if($valores_botones[$i]) {
      if($valores_botones[$i]!=1) {
        $i_ref="<a href=\"{$botones[$i][0]}?$valores_botones[$i]\">";
      } else  {
        $i_ref="<a href=\"{$botones[$i][0]}\">";
        $f_ref="</a>";
      }
    } else {
      $i_ref="<font color=\"#CCCCCC\">";
      $f_ref="</font>";
    }
    $imagen="<img src=\"{$botones[$i][2]}\" alt=\"{$botones[$i][1]}\"
border=no>";
    echo "\n$celda$i_ref$imagen{$botones[$i][1]}$f_ref</td>";
  }
  echo '</tr></table>';
  echo '</center>';
}?>
```

13.1.3 *Software* necesario en el servidor

El *software* necesario para la implementación de esta aplicación es:

- Servidor Web: Apache versión 1.3.23[2] o superior.
- PHP 4.0.6 o superior.
- Biblioteca de funciones de IMAP (*Internet Message Access Protocol*) para PHP[3].

Partimos de la existencia de las tres carpetas que hay por usuario (`Entrada`, `Enviados`, `Papelera`). La creación y borrado de tales carpetas es un asunto que estaría dentro de la administración del servidor de correo y, por tanto, está fuera del alcance de este capítulo.

> **NOTA:** En una distribución RedHat Linux, las carpetas de correo se almacenan en el directorio `/var/spool/mail`. Por tanto, las tres carpetas mencionadas, para la cuenta `un_usr`, se corresponden con los ficheros:
>
> ```
> /var/spool/mail/un_usr,
> /var/spool/mail/un_usr.Enviados
> /var/spool/mail/un_usr.Papelera
> ```

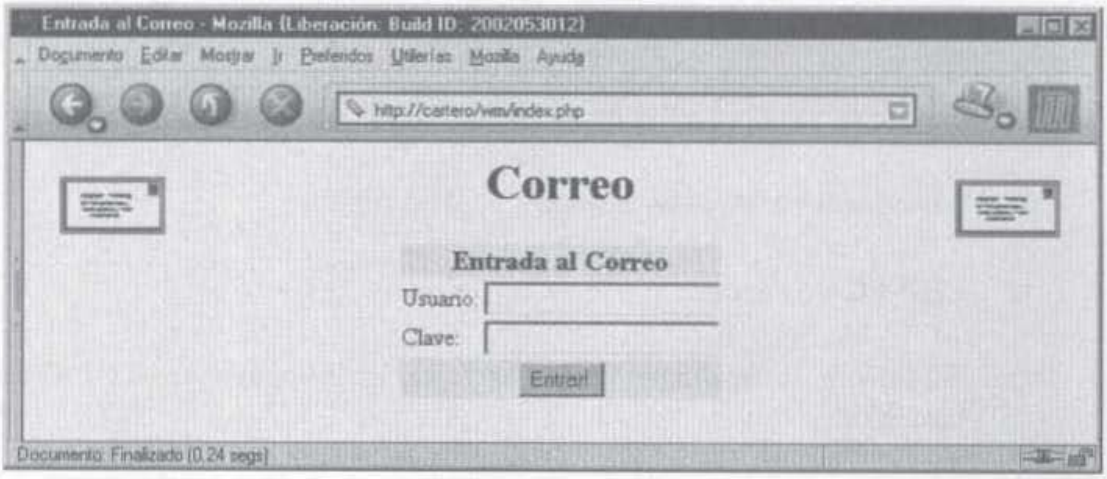
13.2 ENTRADA AL CORREO

La entrada a la aplicación la hemos separado en dos ficheros: un fichero HTML donde está definido el aspecto de la página de entrada y el propio programa de entrada al *webmail* que comprueba que los datos introducidos en el formulario son correctos. De esta forma, conseguimos una separación entre la programación y el interfaz: si queremos modificar el aspecto, nos basta con modificar el fichero HTML.

---

[2] Si instalamos un servidor Web con las librerías SSL (*Secure Socket Layer*), podremos leer el correo de forma totalmente segura a salvo de *miradas* indiscretas en la red. Véase http://www.modssl.org/ y http://www.apache-ssl.org.

[3] Estas funciones también se pueden usar para atacar a servidores POP y NNTP. En sistemas U*ix será necesario descargarse esta biblioteca de la dirección ftp://ftp.cac.washington.edu/imap/ y proceder a su instalación.



Esta separación entre aspecto y programa nos permite implementar código JavaScript que realice pequeñas comprobaciones de manera más cómoda. Por ejemplo, para evitar conexiones fallidas al servidor de correo y esperas inútiles al usuario, la página HTML contiene unas líneas de código JavaScript que verifican, antes de enviar la página con los datos del formulario al servidor, que los campos de usuario y clave del formulario no están vacíos. El contenido del fichero HTML responsable de la apariencia de la figura anterior es el siguiente:

```
<html>
<head>
<title>
Entrada al Correo
</title>
<script language='javascript'>
  function campos_ok()
  {
    if  (!document.fEntrada.login.value)
    {
      alert('No ha rellenado el campo de usuario');
      return false;
    }
    if  (!document.fEntrada.password.value)
    {
      alert('No ha rellenado el campo de la clave');
      return false;
    }
    return true;
  }
</script>
</head>
<body>
<center>
<img align='left' src='images/MAIL.GIF'><img align='right'
src='images/MAIL.GIF'><h1>Correo</h1>
<form name='fEntrada' action='index.php' method='post'
      onsubmit='return campos_ok();'>
<table border='0'>
```

```
<tr>
        <td bgcolor='#fffbad' colspan='2' align='center'>
                <font size='+1'><b>Entrada al Correo</b></font>
        </td>
</tr>
<tr>
  <td width='50'>Usuario:</td>
  <td><input type='text' name='login' ></td>
</tr>
<tr>
<td width='50'>Clave:</td>
<td><input type='password' name='password' ></td>
</tr>
<tr>
        <td bgcolor='#fffbad' colspan='2' align='center'>
                <input type='submit' name='submit' value='Entrar!' >
        </td>
</tr>
</table>
</form>
```

Por su parte, el *script* PHP de entrada intenta realizar una primera conexión, vía la función `imap_open()`, con el servidor IMAP para que valide el nombre del usuario y la clave introducidas. Si así lo fuesen, se crea la sesión y sus variables, se almacenan en éstas los valores oportunos y, por último, se redirige al navegador a la página que muestra las cabeceras de los mensajes con la línea:

```
header("Location: wm_ver_cabs.php");
```

Si la conexión no puede realizarse, se emite un mensaje de error y se vuelve a la pantalla inicial. El código del programa de entrada sería:

```
<?

if (!$submit)
{
  require "wm_entrada.html";
}
else
{

   $serv_imap='localhost:143';

   if (!$kk = @imap_open("\{$serv_imap}",$login,$password))
   {
     require "wm_entrada.html";
     echo "<br><br><h3>** ERROR: login incorrecto</h3>";
   }
   else
   {
     session_start();
     session_register("WM");      // para acceso controlado
```

```
        session_register("WM_USR");
        session_register("WM_CLAVE");
        session_register("WM_SRV_IMAP");

        session_register("WM_BUZON_ACTUAL");
        session_register("WM_CAMPO_ORD");
        session_register("WM_SENTIDO");

        session_register("WM_CUOTA");
        session_register("WM_DIR_BUZONES");
        session_register("WM_MAX_NUM_FICH_ADJUNTOS");
        session_register("WM_MAX_TAM_FICH_ADJUNTO");

        $WM             = TRUE;
        $WM_USR         = $login;
        $WM_CLAVE       = $password;
        $WM_SRV_IMAP    = $serv_imap;

        $WM_BUZON_ACTUAL = 'Entrada';
        $WM_CAMPO_ORD    = 'pos_de';
        $WM_SENTIDO      = '0';

        $WM_CUOTA                   = 5242880; // 5Mb (5 * 1024 * 1024)
        $WM_DIR_BUZONES             = '/var/spool/mail';
        $WM_MAX_NUM_FICH_ADJUNTOS = 6;
        $WM_MAX_TAM_FICH_ADJUNTO  = 45760;

        // redirijo a la pagina de cabeceras
        header("Location: wm_ver_cabs.php");
    }
}
?>
```

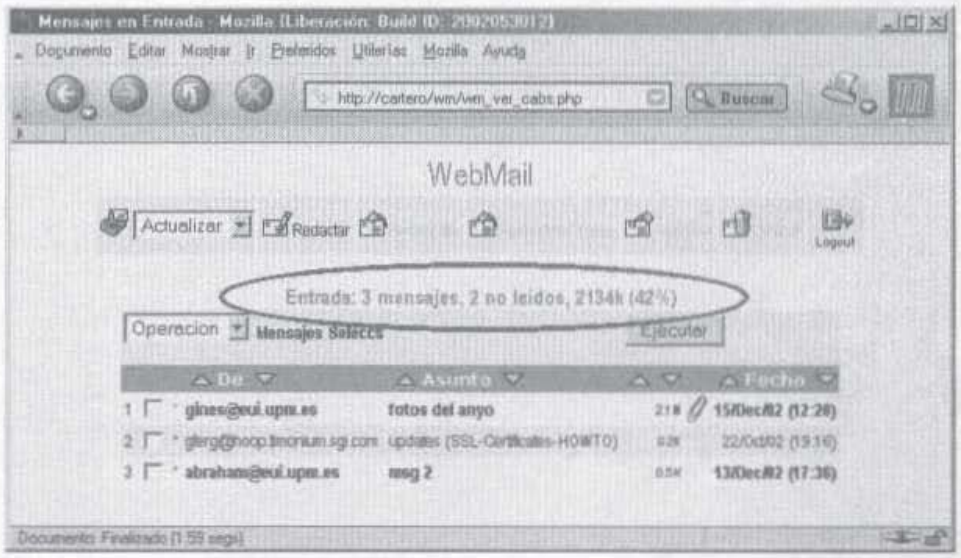
13.3 SALIDA DEL SISTEMA

Como no podía ser de otra forma, éste es el programa más sencillo de todos los que componen la aplicación. Las acciones que realiza son eliminar las variables de sesión, destruir la sesión y, por último, redirigir al navegador a la página de entrada a la aplicación. El texto del *script* `wm_salida.php` es:

```
<?
// inicializo la sesion
session_start();

// elimino todas las variables de sesion
session_unset();

// destruyo la sesion
session_destroy();

// redirijo a la pagina de entrada
header("Location: index.php");

?>
```

13.4 REVISIÓN DE LOS MENSAJES EN LAS CARPETAS

En esta pantalla, para cada mensaje, se muestra el remitente, el asunto, el tamaño del mensaje, si tiene ficheros adjuntos, y la fecha. Además, aquellos mensajes que no han sido leídos se muestran en negrita. Adicionalmente a las cabeceras de los mensajes, se muestra información suplementaria, como el número total de mensajes en la carpeta actual, el número de mensajes no leídos, el total de espacio en disco ocupado y el tanto por ciento que esto representa sobre la cuota de disco asignada.



13.4.1 Opciones de ordenación y cambio de carpeta

Dado que se permite al usuario visualizar la lista de mensajes ordenada según distintos criterios, este *script* puede ser llamado con las nuevas opciones de ordenación al pinchar el usuario en los iconos dispuestos en la cabecera de la lista de mensajes.

Los enlaces de las flechas de ordenación llaman al propio *script* `wm_ver_cabs.php` con los valores adecuados en los parámetros `wm_sentido` y/o `wm_campo_ord`. Por esta razón, al principio del *script*, comprobamos la existencia de estas variables y actualizamos las correspondientes variables de sesión.

```
if (isset($wm_buzon_actual))
   $WM_BUZON_ACTUAL=$wm_buzon_actual;
if (isset($wm_sentido))
   $WM_SENTIDO=$wm_sentido;
if (isset($wm_campo_ord))
   $WM_CAMPO_ORD=$wm_campo_ord;
```

Lo mismo ocurre si queremos revisar los mensajes de una carpeta distinta a la actual: se llama al *script* con el parámetro `wm_buzon_actual` asignado con cualquiera de los valores `Entrada`, `Enviados` o `Papelera`, que, luego, son guardados en su correspondiente variable de sesión.

Los mensajes no pueden empezar a imprimirse en el navegador sin más: han de guardarse en una estructura que permita su ordenación antes de ser transmitidos al cliente. Para ello, contamos con un *array* numérico que tendrá un elemento por mensaje y a su vez cada elemento, es decir, cada mensaje, almacenará otro *array* donde guardará el remitente, el asunto, el tamaño, el número de adjuntos, el *epoch*, la fecha, si es nuevo, y la posición original en el buzón.

```
$filas[$fila_i] = array();
....
$filas[$fila_i][$pos_de]=
                $cab_aux->from[0]->mailbox."@".$cab_aux-
>from[0]->host;
$filas[$fila_i][$pos_asunto] = $cab_aux->Subject;
$filas[$fila_i][$pos_tam]    =
dime_tam_total_msg_bytes($estruc);
$filas[$fila_i][$pos_adjs]   = dime_num_adjuntos($estruc);
$filas[$fila_i][$pos_epoch]  = dame_epoch($cab_aux->date);

if ($cab_aux->date) {
  $filas[$fila_i][$pos_fecha]= repinta_fecha($cab_aux->date);
}
else
{
  $filas[$fila_i][$pos_fecha]= haz_fecha($cab_aux->udate);
}
$filas[$fila_i][$pos_es_nuevo]      = es_msg_nuevo($cab_aux);
$filas[$fila_i][$pos_pos_en_buzon] = "$n_msg";
....
usort($filas, "ordena_por");
pinta_tabla($filas);
```

Para hacer efectiva la ordenación, usamos la función PHP `usort()`, la cual nos permite definir una función propia de ordenación La función definida para ordenar los mensajes en función del campo y el sentido requeridos es la siguiente:

```
function ordena_por($a, $b) {
  global $posiciones, $WM_CAMPO_ORD, $WM_SENTIDO;
  $posicion = $posiciones[$WM_CAMPO_ORD];
```

```
  if ($WM_SENTIDO == 1) {
    $si = -1;
    $no =  1;
  } else {
    $si =  1;
    $no = -1;
  }
  $aa = strtoupper($a[$posicion]);
  $bb = strtoupper($b[$posicion]);
  if ($aa == $bb) return 0;
  return ($aa <  $bb)  ? $si : $no;
}
```

13.4.2 Selección de mensajes para ser borrados o movidos

Este apartado está implementado en JavaScript dado que la interacción con el usuario se hace en el navegador:

- Las operaciones relacionadas con la selección de mensajes para ser movidos o borrados se realizan con elementos de formulario: casillas *checkbox* para marcar aquéllos que deseemos y menús desplegables de tipo *select* para indicar la carpeta destino.

```
function submit_con_params() {
  if (document.fOperacs.sLaOperac.selectedIndex == 0) {
    alert("Debe seleccionar Mover o Eliminar");
    return;
  }
  var max = parseInt(document.f_cabs.total_msgs.value);
  var lista="";
  var destino="";

  switch(max) {
  case 0:
      break;
  case 1:
      if (document.f_cabs.msg.checked)
        lista=document.f_cabs.msg.value
      break;
  default:
      for (var i = 0; i < max; i++) {
        if (document.f_cabs.msg[i].checked) {
          lista += document.f_cabs.msg[i].value+"," ;
        }
      }
      lista = lista.substr(0,(lista.length-1));
  }

  if (lista.length > 0) {
    lista = "&wm_msgs=" + lista;
```

```
    if (esDOM) {
      ind=document.forms.fCarpetas.sLaCarpeta.selectedIndex
      buz_destino=document.forms.fCarpetas.sLaCarpeta[ind].value
      with (document.forms.fOperacs)
        operac=sLaOperac[sLaOperac.selectedIndex].value;
    } else if (esNN4) {
      with (window.document.cdestino.document.fCarpetas)
        buz_destino=sLaCarpeta[sLaCarpeta.selectedIndex].value;
      with (window.document.fOperacs)
      operac=sLaOperac[sLaOperac.selectedIndex].value;
    } else {
buz_destino=fCarpetas.sLaCarpeta[fCarpetas.sLaCarpeta.selectedIndex].value;
      with (document.fOperacs)
        operac=sLaOperac[sLaOperac.selectedIndex].value;
      }

    if (operac == "Mover") {
      destino="&wm_buzon_destino=" + buz_destino;
    }
    operac="wm_operacion=" + operac;
    locat="wm_selecs.php?" + operac+destino+lista;
    window.location=locat;
  } else {
    alert("NO ha seleccionado ningún mensaje");
  }
}
```

- Si la operación a realizar con los mensajes seleccionados es `Mover` hacemos aparecer una capa HTML que alberga un menú desplegable donde poder indicar la carpeta destino de los mensajes. Por el contrario, si la operación es `Borrar`, ocultamos dicha capa.

```
function quita_pon_buzon_destino() {
if (esDOM) {
    if (document.forms.fOperacs.sLaOperac.selectedIndex == 1)
      document.getElementById("cdestino").style.visibility="visible"
    else
      document.getElementById("cdestino").style.visibility="hidden"
  } else if (esNN4) {
    if (document.fOperacs.sLaOperac.selectedIndex == 1) {
      document.cdestino.visibility="show";
    } else {
      document.cdestino.visibility="hide";
    }
  } else { // esIE4
    if (document.all.fOperacs.sLaOperac.selectedIndex == 1) {
```

```
      document.all.cdestino.style.visibility="visible";
    } else {
      document.all.cdestino.style.visibility="hidden";
    }
  }}
```

El código completo del *script* `wm_ver_cabs.php` es el siguiente:

```
<?
session_start();

/*
 * posibles parametros:
 *
 * wm_buzon_actual: Entrada, Enviados, Papelera
 * wm_campo_ord: campo por el que se ordenan las cabeceras (pos_tam,
 *               pos_fecha, pos_de, pos_asunto)
 * wm_sentido: ascendente (1)/descendente (0)
 *
 */

if (!session_is_registered(WM))
  header("Location: index.php");

if (isset($wm_buzon_actual))
   $WM_BUZON_ACTUAL=$wm_buzon_actual;
if (isset($wm_sentido))
   $WM_SENTIDO=$wm_sentido;
if (isset($wm_campo_ord))
   $WM_CAMPO_ORD=$wm_campo_ord;

$script='
 <style type="text/css">
 #cdestino {position:relative;top:0px;visibility:hidden;}
 </style>

<script language="javascript">

var esNN4  = (document.layers)          ? 1 : 0
var esIE4  = (document.all)             ? 1 : 0
var esDOM  = (document.getElementById) ? 1 : 0
function ver_msg(n_msg)
{
  document.f_oculto.wm_que_msg.value=n_msg;
  document.f_oculto.submit();
}

function quita_pon_buzon_destino() {
if (esDOM) {
    if (document.forms.fOperacs.sLaOperac.selectedIndex == 1)
      document.getElementById("cdestino").style.visibility="visible"
    else
      document.getElementById("cdestino").style.visibility="hidden"
  } else if (esNN4) {
    if (document.fOperacs.sLaOperac.selectedIndex == 1) {
      document.cdestino.visibility="show";
    } else {
```

```
      document.cdestino.visibility="hide";
    }
  } else { // esIE4
    if (document.all.fOperacs.sLaOperac.selectedIndex == 1) {
      document.all.cdestino.style.visibility="visible";
    } else {
      document.all.cdestino.style.visibility="hidden";
    }
  }
}

function submit_con_params() {
  if (document.fOperacs.sLaOperac.selectedIndex == 0) {
    alert("Debe seleccionar Mover o Eliminar");
    return;
  }
  var max = parseInt(document.f_cabs.total_msgs.value);
  var lista="";
  var destino="";

  switch(max) {
  case 0:
      break;
  case 1:
      if (document.f_cabs.msg.checked)
        lista = document.f_cabs.msg.value
      break;
  default:
      for (var i = 0; i < max; i++) {
        if (document.f_cabs.msg[i].checked) {
         lista += document.f_cabs.msg[i].value+"," ;
        }
      }
      lista = lista.substr(0,(lista.length-1));
  }

  if (lista.length > 0) {
    lista = "&wm_msgs=" + lista;
    if (esDOM) {
      ind=document.forms.fCarpetas.sLaCarpeta.selectedIndex
      buz_destino=document.forms.fCarpetas.sLaCarpeta[ind].value
      with (document.forms.fOperacs)
        operac=sLaOperac[sLaOperac.selectedIndex].value;
    } else if (esNN4) {
      with (window.document.cdestino.document.fCarpetas)
        buz_destino=sLaCarpeta[sLaCarpeta.selectedIndex].value;
      with (window.document.fOperacs)
      operac=sLaOperac[sLaOperac.selectedIndex].value;
    } else {

buz_destino=fCarpetas.sLaCarpeta[fCarpetas.sLaCarpeta.selectedIndex].value;
      with (document.fOperacs)
        operac=sLaOperac[sLaOperac.selectedIndex].value;
    }

    if (operac == "Mover") {
      destino="&wm_buzon_destino=" + buz_destino;
    }
    operac="wm_operacion=" + operac;
    locat="wm_selecs.php?" + operac+destino+lista;
    window.location=locat;
  } else {
```

```
    alert("NO ha seleccionado ningún mensaje");
  }
}

</script>
';

if ($WM_BUZON_ACTUAL == "")  // esto ocurre en el inicio de sesion
  $WM_BUZON_ACTUAL = 'Entrada';

include("./wm_cabeceras.php");
pinta_cabecera_html("Mensajes en $WM_BUZON_ACTUAL", "$script");

$botonera=array();
$botonera[0]="1"; // Actualizar
$botonera[1]="1"; // Redactar
$botonera[2]="0"; // Responder
$botonera[3]="0"; // Responder Todos
$botonera[4]="0"; // Reenviar
$botonera[5]="0"; // Eliminar
$botonera[6]="1"; // Logout

pinta_botonera($botonera);

include("./wm_lib_imap.php");

$buzon = dame_canal("$WM_BUZON_ACTUAL");

$info_buzon = imap_mailboxmsginfo ($buzon);
echo "<br><center><b>
      <font face='arial narrow' color=orange>
      $WM_BUZON_ACTUAL: $info_buzon->Nmsgs mensajes,
      $info_buzon->Unread no leidos,
      ".dame_ocupado_y_quota("$WM_DIR_BUZONES", "$WM_USR", "$WM_CUOTA").")
      </font></b></center>";

if ($info_buzon->Nmsgs == 0) {
 return 0;
}

$l_cabs = imap_headers($buzon);
if(is_array($l_cabs)) {

/*
 * necesito informacion adicional a cada fila:
 *     - el epoch para poder ordenar por fecha
 *     - si es nuevo el mensaje para ponerlo en negrita
 *     - la posicion del mensaje en el buzon
 *
 * $filas[$i]=array(from, subject,tamanyo,adjuntos,epoch,fecha,nuevo, posic);
 * $filas[$i]=array(0,    1,       2,       3,       4,    5,    6,     7);
 *
 */

  $posiciones = array ();
  $pos_de          = "0";   $posiciones["pos_de"]          = "0";
  $pos_asunto      = "1";   $posiciones["pos_asunto"]      = "1";
  $pos_tam         = "2";   $posiciones["pos_tam"]         = "2";
  $pos_adjs        = "3";   $posiciones["pos_adjs"]        = "3";
  $pos_epoch       = "4";   $posiciones["pos_epoch"]       = "4";
  $pos_fecha       = "5";   $posiciones["pos_fecha"]       = "5";
  $pos_es_nuevo    = "6";   $posiciones["pos_es_nuevo"]    = "6";
```

```
  $pos_pos_en_buzon = "7";   $posiciones["pos_en_buzon"]      = "7";

  $filas=array();
  $fila_i=0;
  reset($l_cabs);
  while (list($n_msg, $val) = each($l_cabs)) {

    $cab_aux=imap_header($buzon, ++$n_msg);
    $estruc=imap_fetchstructure($buzon, $n_msg);

// $filas[$fila_i]=array(from,subj,tam,adjuntos,epoch,fecha,nuevo,pos_buzon);
// $filas[$fila_i]=array( 0,   1,    2,   3,       4,    5,     6,     7);

    $filas[$fila_i] = array();
    $filas[$fila_i][$pos_de]=$cab_aux->from[0]->mailbox."@".$cab_aux-
>from[0]->host;
    $filas[$fila_i][$pos_asunto] = $cab_aux->Subject;
    $filas[$fila_i][$pos_tam]    = dime_tam_total_msg_bytes($estruc);
    $filas[$fila_i][$pos_adjs]   = dime_num_adjuntos($estruc);
    $filas[$fila_i][$pos_epoch]  = dame_epoch($cab_aux->date);

    if ($cab_aux->date) {
      $filas[$fila_i][$pos_fecha]  = repinta_fecha($cab_aux->date);
    }
    else
    {
      $filas[$fila_i][$pos_fecha]  = haz_fecha($cab_aux->udate);
    }
    $filas[$fila_i][$pos_es_nuevo] = es_msg_nuevo($cab_aux);
    $filas[$fila_i][$pos_pos_en_buzon]="$n_msg";
    $fila_i++;

  }

  usort($filas, "ordena_por");
  pinta_tabla($filas);

} else
  print "imap_listmailbox failed: ".imap_last_error()."\n";

imap_close($buzon);

echo '
    <form name="f_oculto" action="wm_info_msg" method="get">
    <input type="hidden" name="wm_que_msg">
    </form>';

/*
 * --- Rutinas ----------------------------------------
 */

function ordena_por($a, $b) {
global $posiciones, $WM_CAMPO_ORD, $WM_SENTIDO;
  $posicion = $posiciones[$WM_CAMPO_ORD];
  if ($WM_SENTIDO == 1) {
    $si = -1;
    $no =  1;
  } else {
    $si =  1;
    $no = -1;
  }
  $aa = strtoupper($a[$posicion]);
```

```
  $bb = strtoupper($b[$posicion]);
  if ($aa == $bb) return 0;
  return ($aa <  $bb)  ? $si : $no;
}

function pinta_tabla($t) {
  global $info_buzon;
  global $WM_BUZON_ACTUAL, $naranja;
  global $pos_de, $pos_asunto, $pos_tam, $pos_adjs, $pos_epoch,
         $pos_fecha, $pos_es_nuevo, $pos_pos_en_buzon;

  $i_arial ='<font face="arial narrow" size=-1>';
  $i_arial2='<font face="arial narrow" size=-2>';
  $f_font='</font>';

  $col_filas = array();
  $col_filas[0]="bgcolor=#FFFBAD";
  $col_filas[1]="bgcolor=#FFFFFF";
  $ind_col_filas=0;

  echo "<center>
        <table BORDER=0 CELLPADDING=2 CELLSPACING=0>";

  pinta_fila_operaciones() ;
  pinta_cabecera_tabla() ;

  for ($i=0; $i<count($t); $i++) {
    $ind_col_filas++;
    $ind_col_filas%=2;

    /* compruebo si este mensaje no ha sido visto */
    if ($t[$i][$pos_es_nuevo]) {
      $t[$i][$pos_es_nuevo]='<font color="green">*</font>';
      $i_negr="<b>";
      $f_negr="</b>";
    } else {
      $t[$i][$pos_es_nuevo]='<font color="red" size=-1>*</font>';
      $i_negr="";
      $f_negr="";
    }

    // de
    $t[$i][$pos_de]=pon_url($t[$i][$pos_pos_en_buzon],
                            $t[$i][$pos_de]);
    $t[$i][$pos_de]="$i_arial$i_negr{$t[$i][$pos_de]}$f_negr$f_font";

    // asunto
    if (!$t[$i][$pos_asunto]) {
      $t[$i][$pos_asunto]=" ";
    } else
      $t[$i][$pos_asunto]=pon_url($t[$i][$pos_pos_en_buzon],
                                  $t[$i][$pos_asunto]);
    $t[$i][$pos_asunto]=$i_arial.$i_negr.$t[$i][$pos_asunto].$f_negr.$f_font;

    // fecha
    $t[$i][$pos_fecha]=$i_arial.$i_negr.$t[$i][$pos_fecha].$f_negr.$f_font;

    // tamanyo
    $el_tam="$i_arial2$i_negr".dime_tam($t[$i][$pos_tam])."$f_negr$f_font";

    // ficheros adjuntos
    if ($t[$i][$pos_adjs]) {
```

```
      if ($t[$i][$pos_adjs] > 1)
        $el_alt="($t[$i][$pos_adjs]) fichs adjuntos";
      else
        $el_alt="($t[$i][$pos_adjs]) fich adjunto";
      $img_adj="<img src=\"images/attachment.gif\" alt=\"$el_alt\">";
                // height=\"17\" width=\"15\">";
    } else
      $img_adj=" ";


    echo "<tr $col_filas[$ind_col_filas]>

        <td align=right><font size=-1>".($i+1)."</font></td>
        <td><input type=\"checkbox\" name=\"msg\"
value=\"($t[$i][$pos_pos_en_buzon])\"></td>
        <td>($t[$i][$pos_es_nuevo])</td>
        <td>($t[$i][$pos_de])</td>
        <td>($t[$i][$pos_asunto])</td>
        <td align=right>$el_tam$f_font</td>
        <td>$img_adj</td>
        <td align=right>($t[$i][$pos_fecha])</td>
        </tr>";
  }

  echo "</table>
       <input type='hidden' name=total_msgs value='$info_buzon->Nmsgs'>
       </form>
       </center>";
       // $total_msgs lo voy a usar para el recorrido por el checkbox 'msg'
}

function pon_url($n_msg, $txt) {
  $str ="<a href='javascript:ver_msg($n_msg)'>$txt</a>";
  return $str;
}

function dime_num_adjuntos($str) {
  if ($str->parts) {
    if (is_array($str->parts)){
      return (count($str->parts) -1);
    }
  }
  else return "";
}

function dime_tam($un_tam) {
  $un_mega=1048576;
  $un_k=1024;

  if ($un_tam < 100)
    return "$un_tam"."b" ;
  else if ($un_tam > $un_mega)
    return round ($un_tam / $un_mega, 1)."M" ;
  else
    return round ($un_tam / $un_k, 1)."K" ;
}

function dime_tam_total_msg_bytes($str) {
  $tamanyo=0;
  if ($str->parts) {
    if (is_array($str->parts)){
      for ($pos=1; $pos<count($str->parts); $pos++) {
```

```
        $subestr=$str->parts[$pos];
        $tamanyo+=$subestr->bytes;
      }
    }
  } else
    $tamanyo=$str->bytes;

  return $tamanyo;
}

function haz_fecha($el_timestamp)
{
  $meses[0]='Ene';$meses[1]='Feb';$meses[2]='Mar';$meses[3]='Abr';
  $meses[4]='May';$meses[5]='Jun';$meses[6]='Jul';$meses[7]='Ago';
  $meses[8]='Sep';$meses[9]='Oct';$meses[10]='Nov';$meses[11]='Dic';

  $array_fecha=localtime($el_timestamp,1);
  $anyo=(1900 + $array_fecha['tm_year']) % 2000;
  if ($anyo<10) {
    $anyo = "0$anyo";
  }
  $fecha="{$array_fecha['tm_mday']}/{$meses[$array_fecha['tm_mon']]}/$anyo
  ({$array_fecha['tm_hour']}:{$array_fecha['tm_min']})";
  return $fecha;
}

function repinta_fecha($un_str) {
  $elems_fecha=explode(" ", $un_str);
  $anyo=substr($elems_fecha[3], 2, 2);
  $elems_hora=explode(":", $elems_fecha[4]);
  return "$elems_fecha[1]/$elems_fecha[2]/$anyo
          ($elems_hora[0]:$elems_hora[1])";
}

function dame_epoch($str_fecha) {
  $meses=array();
  $meses["Jan"]=0; $meses["Feb"]=1; $meses["Mar"]=2;  $meses["Apr"]=3;
  $meses["May"]=4; $meses["Jun"]=5; $meses["Jul"]=6;  $meses["Aug"]=7;
  $meses["Sep"]=8; $meses["Oct"]=9; $meses["Nov"]=10; $meses["Dec"]=11;

  // ej de fecha: Mon, 12 Mar 2001 11:28:42 +0100
  // ej de fecha: 0    1  2   3    4
  $fechax=explode(" ", $str_fecha);
  $horax=explode(":", $fechax[4]);

  return mktime($horax[0], $horax[1], $horax[2], $meses[$fechax[2]],
                $fechax[1], $fechax[3]);
}  // fin de dame_epoch($str_fecha)

function dame_ocupado_y_quota($dir, $usuario, $cuota)
{
  // uso la funcion filesize() porque no funciona correctamente
  // imap_mailboxmsginfo() ni con imap_open, ni con imap_reopen()
  $tam=filesize("$dir/$usuario");
  $tam+=filesize("$dir/$usuario.Enviados");
  $tam+=filesize("$dir/$usuario.Papelera");
  return round($tam/1024)."k (".round(($tam*100)/$cuota)."%";
```

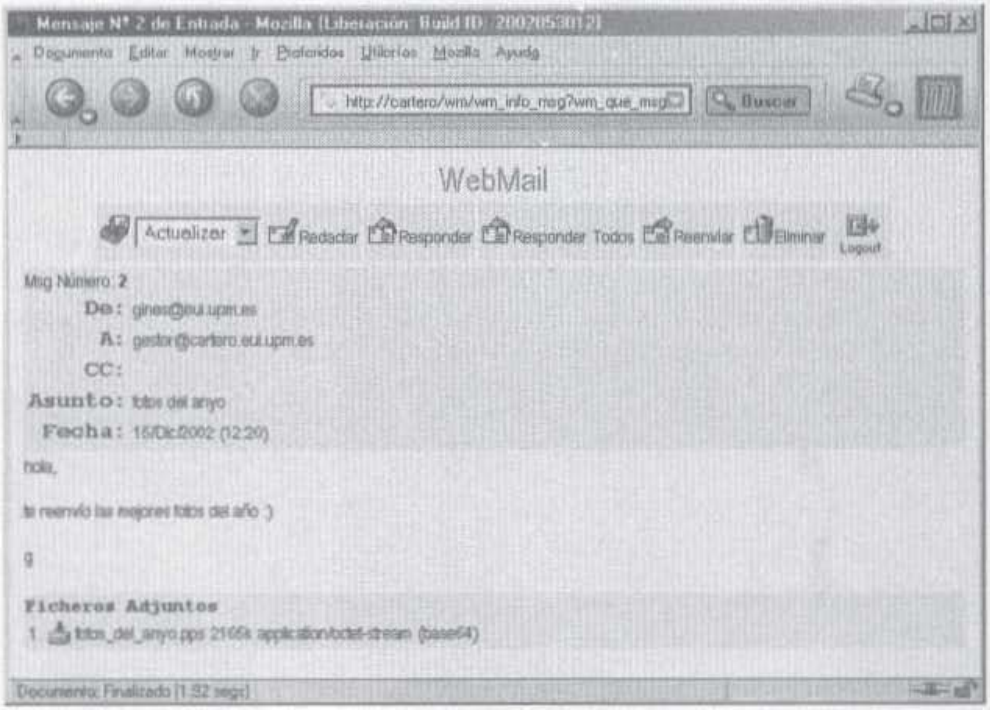
```
}

function es_msg_nuevo($struct) {
    return ($struct->Unseen == "U"|| $struct->Recent == "N");
}

function pinta_fila_operaciones() {
  global $WM_BUZON_ACTUAL, $i_arial, $f_arial;
  echo "
    <tr bgcolor='#FFFBAD'>
    <td colspan='4' valign='top'>
    <form name='fOperacs' method='post' action='kk'>
    {$i_arial}
    <select name='sLaOperac' onChange='javascript:quita_pon_buzon_destino()'>
    <option value='-1'>Operacion</option>
    <option value='Mover'>Mover</option>
    <option value='Eliminar'>Eliminar</option>
    </select>
    <b>Mensajes Seleccs</b>$f_arial
    </td>
    </form>";

  echo "
    <td colspan='1' valign='top'>
    <span id='cdestino'>
    <form name='fCarpetas' action='javascript:void(0)' method='post'>
    ($i_arial)<b> a la carpeta</b>$f_arial
    <select name='sLaCarpeta'>";

  if($WM_BUZON_ACTUAL == "Entrada" || $WM_BUZON_ACTUAL == "") {
      echo "<option value='Enviados'>Enviados</option>
            <option value='Papelera'>Papelera</option>";
  } elseif ($WM_BUZON_ACTUAL == "Enviados") {
    echo "<option value='Entrada'>Entrada</option>
          <option value='Papelera'>Papelera</option>";
  } elseif ($WM_BUZON_ACTUAL == "Papelera") {
    echo "<option value='Entrada'>Entrada</option>
          <option value='Enviados'>Enviados</option>";
  } else {
      die ("** ERROR en el buzon actual **");
  }

  echo "
    </select>
    </span>
    </td>
    </form>
    ";  // el orden de cierre de span, td y form no es caprichoso: es para
        // no generar una celda muy 'alta'.
        // Ademas, si pongo el form antes que el td no aparece el select
    /* el bueno:
     * </span>
     * </td>
     * </form>
    */
```

```
  echo "
    <td colspan='3' valign='top'>
    <form name='f_submit_operac' action='javascript:void(0)' method='get'>
          <input type='button' onclick='javascript:submit_con_params()'
            value='Ejecutar'>
    </td>
    </form>";
  echo "</tr>";

} // fin de pinta_fila_operaciones()

function pinta_cabecera_tabla() {
  global $WM_BUZON_ACTUAL, $naranja;

  echo "<form name='f_cabs' method='get' action='wm_selecs.php'>
        <tr bgcolor=\"$naranja\">
         <th align=left><font face=\"verdana\" color=white size=-
1> </th>
         <th align=left><font face=\"verdana\" color=white size=-
1> </th>
         <th align=left><font face=\"verdana\" color=white size=-
1> </th>";

  pinta_celda_con_flechas_ordenacion("De", $WM_BUZON_ACTUAL, "pos_de");
  pinta_celda_con_flechas_ordenacion("Asunto", $WM_BUZON_ACTUAL,
                                     "pos_asunto");
  pinta_celda_con_flechas_ordenacion("", $WM_BUZON_ACTUAL, "pos_tam");
  echo "<th align=left><font face=\"verdana\" color=white size=-
1> </th>";
  pinta_celda_con_flechas_ordenacion("Fecha", $WM_BUZON_ACTUAL, "pos_epoch");

  echo "</tr>";
} // fin de pinta_cabecera_tabla()

function pinta_celda_con_flechas_ordenacion($titulo, $buzon, $campo_ord)
{
  $ini_tira=' <a href="wm_ver_cabs?';
  $ini_tira.="wm_buzon_actual=$buzon&wm_campo_ord=$campo_ord";
  $tira_abajo=$ini_tira . "&wm_sentido=1\"><img src=images/up.gif
border=0></a>";
  $tira_arriba=$ini_tira . "&wm_sentido=0\"><img src=images/down.gif
border=0></a>";

  echo "<th align=left><font face=\"verdana\" color=white size=-1>
        $tira_abajo
        $titulo
        $tira_arriba
        </th>";
} // fin de pinta_celda_con_flechas_ordenacion()
?>
```

13.5 LECTURA DE UN MENSAJE

El programa encargado de revisar el contenido de los mensajes recibe en la variable `wm_que_mensaje` el número del mensaje a visualizar. La carpeta de donde extraerlo está almacenada en la variable de sesión `WM_BUZON_ACTUAL`.

Los ficheros adjuntos no se visualizan en el mensaje, sino que se muestran en el pie de la página indicando el nombre, el tipo de codificación usado para su almacenamiento en el buzón, el tamaño, etc.



Lógicamente, en esta pantalla sí que están activos los botones de responder y reenviar el mensaje que está siendo visualizado. Además, puesto que el programa de visualización de mensajes ha extraído ya la información de él (remitente, destinatarios, asunto y texto), y es desde esta página desde donde se llama al programa de componer mensajes, para ahorrar conexiones al servidor IMAP, todos estos datos se envían en variables de formulario tipo `hidden` al programa de composición de mensajes.

```
<input type=hidden name=wm_componer_oper>
<input type=hidden name=wm_que_msg     value='$wm_que_msg'>
<input type=hidden name=wm_de          value='$el_de'>
<input type=hidden name=wm_para        value='$el_to'>
<input type=hidden name=wm_concopia    value='$el_cc'>
<input type=hidden name=wm_asunto      value='$el_subject'>
```

```
<input type=hidden name=wm_texto      value='$txt'>
<input type=hidden name=wm_num_adjs   value='" . ($total_adjs-
1) . "'>
```

Por último, se debe comentar que, curiosamente, ha sido complicado extraer la fecha de los mensajes añadidos a un buzón mediante `imap_append()`, pues el objeto `date` devuelto por `imap_header()` estaba vacío; `imap_fetchbody()` no proporciona las cabeceras (*envelope*) donde está la fecha e `imap_fetch_overview()` tampoco da la fecha. Finalmente, ha sido del objeto `udate` devuelto por `imap_header()` de donde hemos podido extraerla.

A continuación mostramos el código completo del *script* que nos muestra un mensaje:

```
<?
session_start();
if (!session_is_registered(WM)) header("Location: index.php");
$amarillo='#fffbad';

/*
 * llamado desde: wm_ver_cabs.php, wm_info_msg.php
 * params entrada:
 *   - wm_que_mensaje (numero)
 */

include("./wm_cabeceras.php");

$el_script="
<script language='javascript'>
function reenvia(accion) {
   document.f_reenviar.wm_componer_oper.value=accion;
   document.f_reenviar.submit();
}
</script>
";

pinta_cabecera_html("Mensaje Nº $wm_que_msg de $WM_BUZON_ACTUAL",
"$el_script");

$botonera=array();
$botonera[0]="1"; // Actualizar
$botonera[1]="1"; // Redactar
$botonera[2]="1"; // Responder
$botonera[3]="1"; // Responder Todos
$botonera[4]="1"; // Reenviar

$params="wm_msgs=$wm_que_msg";
$params.="&wm_operacion=Eliminar";
$botonera[5]="$params"; // Eliminar
$botonera[7]="1"; // Logout
```

```
pinta_botonera($botonera);

/*
 *
 * body[0]: cabecera completa del mensaje
 * parameters (normal): attribute: CHARSET, value: US-ASCII, iso-
8859-1, etc
 * parameters (multipart): attribute: BOUNDARY, value: cadena de
separacion
 * dparameters de cada parte: attribute: FILENAME, value: nombre del
fichero
 * parameters de cada parte : attribute: NAME,     value: nombre del
fichero
 * disposition suele tener ATTACHMENT o INLINE (enviado desde
netscape)
 *
 */

include("./wm_lib_imap.php");
$buzon=dame_canal("$WM_BUZON_ACTUAL");

/*
 * Imprimo la cabecera
 */

$cab_aux=imap_header($buzon, $wm_que_msg);

$el_de=$cab_aux->from;
$el_de=sprintf("%s@%s", $el_de[0]->mailbox, $el_de[0]->host);

$el_to="";
for ($i=0; $i<count($cab_aux->to); $i++){
  $el_to=sprintf("%s %s@%s", $el_to,
                             $cab_aux->to[$i]->mailbox,
                             $cab_aux->to[$i]->host);
}

$el_cc="";
for ($i=0; $i<count($cab_aux->cc); $i++){
  $el_cc=sprintf("%s %s@%s", $el_cc,
                             $cab_aux->cc[$i]->mailbox,
                             $cab_aux->cc[$i]->host);
}

$el_subject= parsea_iso_8859($cab_aux->Subject);

/*
 * cuando es un mensaje anyadido con imap_append, no obtengo la
fecha: el
 * objeto 'date' devuelto por imap_header esta vacio.
 * imap_fetchbody($buzon, num_msg, 0): no da el envelope (donde está
la fecha)
 * imap_fetch_overview: tampoco da la fecha...
 * Tiene que ser con el objeto 'udate'
```

```
 *
 */

$la_fecha=dame_fecha($cab_aux->udate);

echo "
     <form name=f_reenviar action='wm_componer.php' method='post'>
     <table width=100%>
     <tr><td bgcolor=\"$amarillo\">
     Msg Número: <b>$wm_que_msg</b><br>
     <table>
     <tr><th align=right><tt>De:</tt></th><td>$el_de<br></td></tr>
     <tr><th
align=right><tt>A:</th><td></tt>$el_to</tt><br></td></tr>
     <tr><th
align=right><tt>CC:</th><td></tt>$el_cc</tt><br></td></tr>
     <tr><th
align=right><tt>Asunto:</th><td></tt>$el_subject<br></td></tr>
     <tr><th
align=right><tt>Fecha:</th><td></tt>$la_fecha<br></td></tr>
     </table>";

$estructura=imap_fetchstructure($buzon, $wm_que_msg);
if ($estructura) {
  echo "</td></tr>
        <tr><td>";

  // Cuerpo del Mensaje.....................................

  if (strtolower($estructura->subtype) == "html")
  {
    $txt = dame_porcion($buzon, $wm_que_msg, "TEXT/HTML");
    if ($txt) {
      echo $txt;
    }
  }
  else
{
    if ($estructura->parts[0]->subtype == "ALTERNATIVE")
    {
      $parte_msg=1.1;  // la parte 1.2 suele estar en html (ver
pr_dbody.php)
    }
    else
    {
      $parte_msg=1;
    }
    $txt=imap_fetchbody($buzon, $wm_que_msg, $parte_msg);

  // de la pagina de imap_qprint: This function seems to have a bug,
when the
  // quoted-printable string contains a "=" without the HEX code of a
Character.
  // I use the regular quoted_printable_decode instead.
  // (no es parte del modulo imap)
```

```
  // sin embargo, quoted_printable_decode me deja cortadas las lineas
> 76

    $txt= nl2br(imap_qprint($txt));
    echo $txt;
  }

  echo "<br>";
  // si es MULTIPART
  if ($estructura->parts) {
    if (is_array($estructura->parts)) {

      echo "</td></tr>
            <tr><td bgcolor=\"$amarillo\">
            <tt><b>Ficheros Adjuntos</b></tt><br>
            <table>";

      $total_adjs=count($estructura->parts);
      for ($pos=1; $pos<$total_adjs ; $pos++) {
        $subestr=$estructura->parts[$pos];

        // uso 'parameters' en lugar de 'dparameters' porque la
funcion attach
        //que uso no asigna el filename='' y si name='' (que esta en
parameters)
        if (is_array($subestr->parameters)) { // nombre del fichero
adjunto
          $nom_fich_adj=parsea_iso_8859($subestr->parameters[0]-
>value);
        } else
          $nom_fich_adj="(sin titulo)";

        if ($subestr->type)
          $tipo_mime=$lib_tipos[$subestr->type];
        else
          $tipo_mime="text";
        $mime=$tipo_mime."/". strtolower($subestr->subtype);

        if ($subestr->encoding) {
          $codif=$subestr->encoding;
          $la_codif="(".$lib_codifs[$codif].")";
        } else
          $la_codif=" ";

        $baja_fich="<a href=\"wm_bajar_fichero.php?";
        $baja_fich.="wm_que_msg=$wm_que_msg";
        $baja_fich.="&wm_que_parte=$pos\">";
        $baja_fich.="<img src=images/download.gif border=0>";
        $baja_fich.="</a>";

        echo "<tr><td>$pos.</td>";
        echo "<td>$baja_fich</td>";
        echo "<td>$nom_fich_adj</td>";
        echo "<td align=right>". dame_kas($subestr->bytes) .
"k</td>";
```

```
        echo "<td>$mime</td>";
        echo "<td>$la_codif</td>";
        echo "</tr>";
        echo "<input type=hidden name=wm_adj$pos
value='$nom_fich_adj'>";
      }
      echo "</tr></table>";

    }
  }

  echo "</table>";
  echo "
  <input type=hidden name=wm_componer_oper>
  <input type=hidden name=wm_que_msg      value='$wm_que_msg'>
  <input type=hidden name=wm_de           value='$el_de'>
  <input type=hidden name=wm_para         value='$el_to'>
  <input type=hidden name=wm_concopia     value='$el_cc'>
  <input type=hidden name=wm_asunto       value='$el_subject'>
  <input type=hidden name=wm_texto        value='$txt'>
  <input type=hidden name=wm_num_adjs     value='" . ($total_adjs-1)
. "'>

  </form>";

} else {
  echo "**ERROR: no me ha devuelto ninguna estructura!!!";
}
imap_close($buzon);

/*
 * Rutinas
 */

// dame_porcion() solo se llama en el caso en que el mensaje sea html
puro
function dame_porcion($buzon, $num_mensaje, $tipo_mime, $estr =
false,
                      $que_parte = false) {
  if(!$estr) {
    $estr = imap_fetchstructure($buzon, $num_mensaje);
  }
  if($estr) {
    if($tipo_mime == dime_tipo_mime($estr)) {
      if(!$que_parte) {
        $que_parte = "1";
      }
    $text = imap_fetchbody($buzon, $num_mensaje, $que_parte);
    if($estr->encoding == 3) {
      return imap_base64($text);
    } else
      if($estr->encoding == 4) {
        return imap_qprint($text);
      } else {
       return $text;
```

```
      }
    }

    if($estr->type == 1) /* multipart */ {
      while(list($indice, $sub_estruct) = each($estr->parts)) {
        if($que_parte) {
          $prefijo = $que_parte . '.';
        }
        $data = dame_porcion($buzon, $num_mensaje, $tipo_mime,
$sub_estruct,
                             $prefijo . ($indice + 1));
        if($data) {
          return $data;
        }
      }
    }
  }
  return false;
}

function dime_tipo_mime(&$estr) {
  $primary_mime_type = array("TEXT", "MULTIPART", "MESSAGE",
"APPLICATION",
                             "AUDIO", "IMAGE", "VIDEO", "OTHER");
  if($estr->subtype) {
    return $primary_mime_type[(int) $estr->type] . '/' . $estr-
>subtype;
  }
  return "TEXT/PLAIN";
}

function parsea_iso_8859($str)
{
  $str_ok="";
  $elems=imap_mime_header_decode($str);
  for($i=0;$i<count($elems);$i++)
    $str_ok.="{$elems[$i]->text}";
  return "$str_ok";
}

function dame_fecha($el_timestamp)
{
  $meses[0] = 'Ene'; $meses[1] = 'Feb'; $meses[2]  = 'Mar'; $meses[3]
= 'Abr';
  $meses[4] = 'May'; $meses[5] = 'Jun'; $meses[6]  = 'Jul'; $meses[7]
= 'Ago';
  $meses[8] = 'Sep'; $meses[9] = 'Oct'; $meses[10] = 'Nov';
$meses[11] = 'Dic';

  $array_fecha=localtime($el_timestamp,1);
  $anyo=1900 + $array_fecha['tm_year'];

$fecha="{$array_fecha['tm_mday']}/{$meses[$array_fecha['tm_mon']]}/$a
nyo
  ({$array_fecha['tm_hour']}:{$array_fecha['tm_min']})";
```

```
  return $fecha;
}

function dame_kas ($tamanyo) {
  return round ($tamanyo / 1024);
}

?>
```

13.6 DESCARGAS DE FICHEROS ADJUNTOS

Las descargas de ficheros adjuntos ocurren desde la pantalla de revisión de mensajes, donde éstos aparecen al pie. La llamada a este *script* siempre tendrá la forma:

```
wm_bajar_fichero?wm_que_msg=X&wm_que_parte=Y
```

Donde se indica cuál de los ficheros adjuntos (`Y`) hay que extraer del mensaje `X`. La carpeta está indicada "implícitamente" en la variable de sesión `WM_BUZON_ACTUAL`.

Usamos la función `imap_fetchbody()` para extraer selectivamente partes de un mensaje y, dado que el primer "adjunto" del mensaje es el cuerpo de él, si queremos sacar el adjunto número "n", habrá que incrementar en uno el parámetro que indica el adjunto a extraer:

```
$datos=imap_fetchbody($buzon, $wm_que_msg, ($wm_que_parte+1));
```

Además, puesto que los ficheros adjuntos se almacenan en la carpeta con una determinada codificación (*base64, quoted printable*, etc.), hay que averiguar con cuál de ellas lo está para efectuarle su correspondiente descodificación antes de transferir el contenido al cliente:

```
switch($subestr->encoding) {
case 1:
      //$datos = imap_8bit($datos); //me devuelve un
quoted_printable
      break;
case 2:
      $datos = imap_8bit(imap_qprint($datos));
      break;
case 3:
      $datos = imap_base64($datos);
      break;
case 4:
```

```
        $datos = imap_qprint($datos);
        break;
default:
}
```

Por último, para provocar que el navegador guarde el fichero adjunto en lugar de intentar visualizarlo, se envían al navegador las cabeceras apropiadas del protocolo HTTP[4]:

```
header("Content-Disposition: attachment;
filename=\"$nom_fich_adj\"");
header("Content-type: application/octet-stream;
        name=\"$nom_fich_adj\"");
```

El listado completo de este *script* es:

```
<?
session_start();
if (!session_is_registered(WM)) header("Location: index.php");

/*
 * ej de llamada: saca_attach.php?wm_que_msg=1&wm_que_parte=3
 */

include("wm_lib_imap.php");
$buzon=dame_canal("$WM_BUZON_ACTUAL");
$estructura=imap_fetchstructure($buzon, $wm_que_msg);
$subestr=$estructura->parts[$wm_que_parte];

// uso 'parameters' en lugar de 'dparameters' porque la
funcion attach
//que uso no asigna el filename='' y si name='' (que esta en
parameters)
if (is_array($subestr->parameters))  // nombre del fichero
adjunto
```

[4] La versión 5.5 de Internet Explorer no realiza adecuadamente las descargas de los ficheros adjuntos. Microsoft recomienda aplicar los parches correspondientes en:

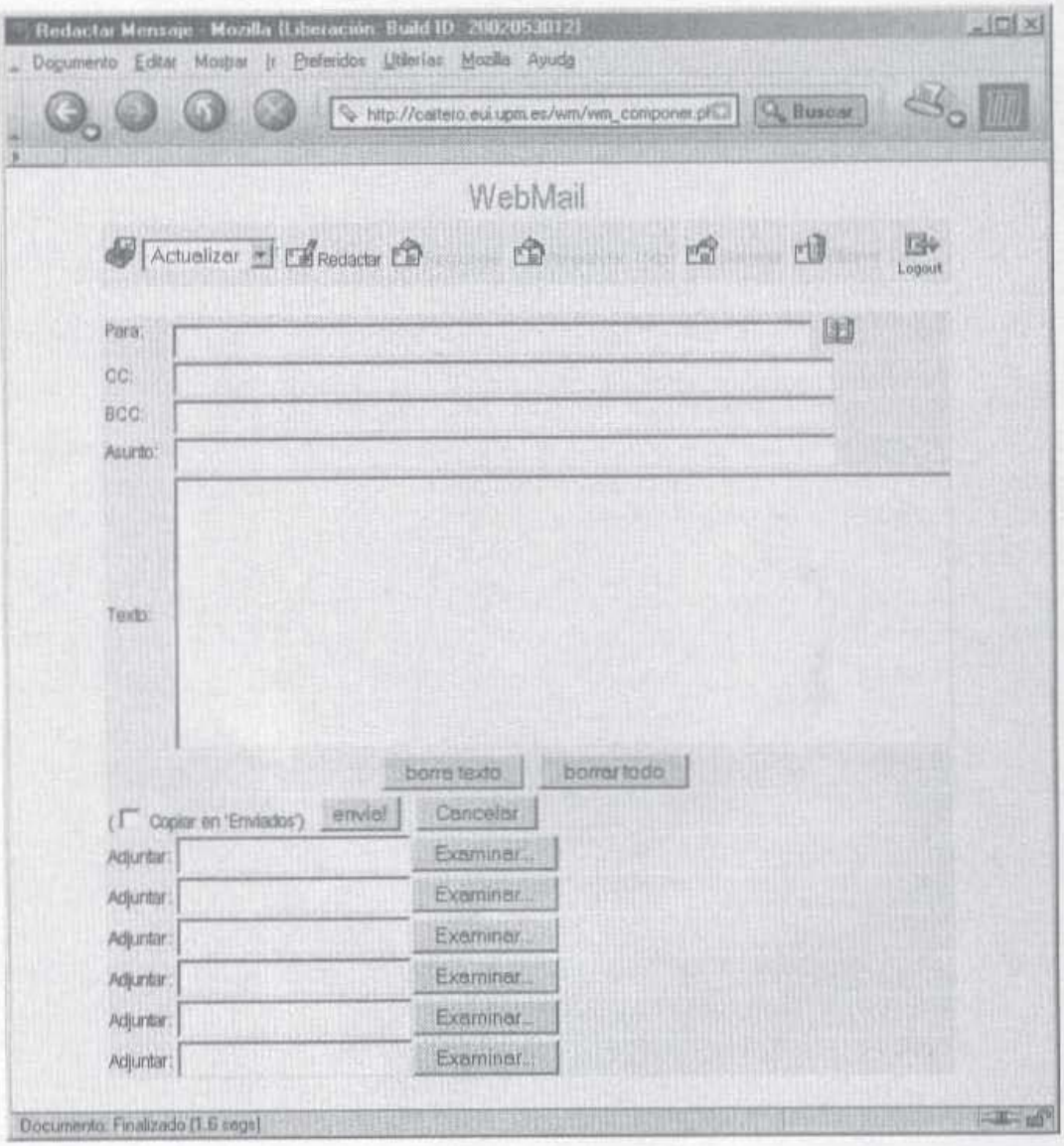
http://support.microsoft.com/support/kb/articles/Q279/6/67.ASP

http://support.microsoft.com/support/kb/articles/Q281/1/19.ASP

```
  $nom_fich_adj=$subestr->parameters[0]->value;
else
  $nom_fich_adj="SINTITULO";

if ($subestr->type)
  $tipo_mime=$lib_tipos[$subestr->type];
else
  $tipo_mime="text";

$mime=$tipo_mime."/". strtolower($subestr->subtype);

header("Content-Disposition: attachment;
filename=\"$nom_fich_adj\"");
header("Content-type: application/octet-
stream;name=\"$nom_fich_adj\"");

// OJO!, hay que sumar 1 a la parte que se quiera
wm_que_parte+1
$datos=imap_fetchbody($buzon, $wm_que_msg, ($wm_que_parte+1));
switch($subestr->encoding) {
case 1:
      //$datos = imap_8bit($datos); // me devuelve un
quoted_printable
      break;
case 2:
      $datos = imap_8bit(imap_qprint($datos));
      break;
case 3:
      $datos = imap_base64($datos);
      break;
case 4:
      $datos = imap_qprint($datos);
      break;
default:
}

echo $datos;
imap_close($buzon);
?>
```

13.7 COMPOSICIÓN DE MENSAJES: ENVIAR, RESPONDER, REENVIAR

Todas las acciones que suponen la redacción de un mensaje, *enviar*, *responder* o *reenviar*, comparten la misma pantalla; por lo tanto, se usará el mismo *script* para todas estas acciones.



Por otra parte, puesto que cada una de ellas tiene sus propias peculiaridades que se manifiestan en un comportamiento diferente (por ejemplo, la operación de responder necesita extraer el texto del mensaje a responder, insertarle el carácter ">" al principio de cada línea, incorporarlo al mensaje a enviar y, además, prefijar el campo "Asunto" [*Subject*] con la cadena `Re:`), este *script* necesitará saber cuál es la acción a realizar, esto es, necesitará un parámetro o variable que le indique qué

operación realizar: componer un mensaje nuevo, reenviar o responder. Esta variable es `wm_componer_oper` y los valores que puede tomar son:

- `0`: redactar un mensaje nuevo,
- `1`: responder a un mensaje,
- `2`: responder a todos los destinatarios que aparecen en el mensaje, y
- `3`: reenviar un mensaje recibido a otra dirección

Dado que las llamadas a este programa se hacen desde los *botones* situados al principio de cada pantalla, les asociamos los valores correspondientes en la función `pinta_botonera()`:

```
$botones=array (
          array("wm_ver_cabs.php",        "Actualizar",
                "images/checkmail.gif"),
          array("wm_componer.php",        "Redactar",
                "images/msg_new.gif"),
          array("javascript:reenvia(1)", "Responder",
                "images/msg_reply.gif"),
          array("javascript:reenvia(2)", "Responder Todos",
                "images/msg_reply.gif"),
          array("javascript:reenvia(3)", "Reenviar",
                "images/msg_forward.gif"),
          array("wm_selecs.php",          "Eliminar",
                "images/msg_delete.gif"),
          array("wm_salida.php",          "", "images/logout.gif"),
          );
```

Y ahora lo que necesitamos es construir la función JavaScript `reenvia()` para que llame al *script* `wm_componer.php` con el parámetro adecuado:

```
$el_script="
<script language='javascript'>
function reenvia(accion) {
   document.f_reenviar.wm_componer_oper.value=accion;
   document.f_reenviar.submit();
}
</script>
";
...
echo "<form name=f_reenviar action='wm_componer.php' method='post'>";
...
```

Los ficheros adjuntos que el usuario quiera enviar se suben (*upload*) al servidor gracias a las líneas:

```
echo '<form name="wm_form_redactar" method="post"
       action="wm_enviar.php"
       enctype="multipart/form-data">';
```

```
echo "<input type='hidden'
            name='MAX_FILE_SIZE'
            value='$WM_MAX_TAM_FICH_ADJUNTO'>";
```

En las que se indica, por una parte, cómo se codifican los datos del formulario al ser enviados al servidor y, por otra, una variable oculta de nombre WM_MAX_FILE_SIZE, que PHP usa para poner un límite (el valor WM_MAX_TAM_FICH_ADJUNTO) de tamaño máximo de fichero al ser subido al servidor.

Como observamos de la figura anterior, una característica añadida es que los mensajes enviados, por defecto, no se copian a la carpeta *Enviados*; se ha de indicar expresamente esta acción.

El código completo de este programa es:

```
<?
session_start();
if (!session_is_registered(WM)) header("Location: index.php");

/*
 * Usado para: redactar, responder, responder a todos
 *
 * Llamado desde:
 *    - wm_cabeceras.php
 *    - wm_info_msg al rellenar los botones de redactar, responder,
etc
 *
 * Rutinas que necesita
 *  - wm_dame_canal()
 *
 * parametros que necesita:
 *    - wm_componer_oper: 0/vacio (redactar), 1 (remitente), 2 (a
todos)
 *    - wm_que_msg: numero de mensaje (solo necesario cuando
wm_componer_oper = (1,2))
 *    - al ser llamado desde wm_info_msg.php  para responder o
reenviar, este
 *      le envía via hiddens los datos de wm_para, wm_concopia,
wm_texto, etc
 *
 */

$el_script="
    <script language='javascript'>
    var WM_MAX_NUM_FICH_ADJUNTOS=$WM_MAX_NUM_FICH_ADJUNTOS";

$el_script.='
   function envia()
```

```
    {
      if (document.wm_form_redactar.wm_para.value) {
        for (i=1;i<=WM_MAX_NUM_FICH_ADJUNTOS;i++)
        {
          v1="document.wm_form_redactar.wm_adj"+i+".value"
          v2="document.wm_form_redactar.wm_adj_oculto"+i+".value"
          v3=v2+"="+v1
          eval (v3);
         }
        document.wm_form_redactar.submit()
      }
      else
      {
        alert(\'Debe rellenar el campo "Para"\')
      }
    }
    function cancela()
    {
      var str;
      str  = "wm_ver_cabs.php";
      document.location=str;
    }
     function incluye_texto() {
      with (document.wm_form_redactar)
        wm_texto.value += wm_texto_escondido.value;
    }
     function limpia_campos_form() {
      with (document.wm_form_redactar) {
        wm_para.value="";
        wm_concopia.value="";
        wm_concopia_ciega.value="";
        wm_asunto.value="";
        wm_texto.value="";
        if (wm_adj1) wm_adj1.value="";
        if (wm_adj2) wm_adj2.value="";
        if (wm_adj3) wm_adj3.value="";
        if (wm_adj4) wm_adj4.value="";
        if (wm_adj5) wm_adj5.value="";
        if (wm_adj6) wm_adj6.value="";
      }
    }
     function limpia_campo_texto() {
      document.wm_form_redactar.wm_texto.value=""
    }
</script>';

include("./wm_cabeceras.php");

switch ($wm_componer_oper) {
case 0: $msg_cab="Redactar Mensaje";
        break;
case 1: $msg_cab="Responder Mensaje Nº $wm_que_msg de Buzón
$WM_BUZON_ACTUAL";
        break;
```

```
case 2: $msg_cab="Responder a Todos Mensaje Nº $wm_que_msg de Buzón
$WM_BUZON_ACTUAL";
        break;
case 3: $msg_cab="Reenviar Mensaje Nº $wm_que_msg de Buzón
$WM_BUZON_ACTUAL";
        break;
default: $msg_cab="Componer";
}

pinta_cabecera_html("$msg_cab", $el_script);

$botonera=array();
$botonera[0]="1"; // Actualizar
$botonera[1]="1"; // Redactar
$botonera[2]="";  // Responder
$botonera[3]="";  // Responder Todos
$botonera[4]="";  // Reenviar
$botonera[5]="";  // Eliminar
$botonera[6]="1"; // Logout

pinta_botonera($botonera);

$col_filas[0]="bgcolor=#FFFBAD";
$col_filas[1]="bgcolor=#FFFFFF";
$ind_col_filas=0;

$wm_concopia_ciega="";
if ($wm_componer_oper == 0) {

  $boton_incluir_texto="";

} elseif ($wm_componer_oper == 1 || $wm_componer_oper == 2) {
  if ($wm_componer_oper == 1) {
    $wm_para=$wm_de;
    $wm_concopia="";
  } else {
    $wm_para="$wm_de $wm_para";
  }

  $wm_asunto="Re: $wm_asunto";
  $boton_incluir_texto ="<input type=\"button\"
onclick=\"incluye_texto()\"";
  $boton_incluir_texto.=" value=\"Incluir Original\"> ";

  $wm_texto =  ereg_replace("<br />", '', $wm_texto);
  $car= "^";
  $wm_texto =  ereg_replace("$car", '&gt; ', $wm_texto);
  $car= "\n";
  $wm_texto = ereg_replace("$car", '&gt; ', $wm_texto);

} elseif ($wm_componer_oper == 3) {

  $wm_asunto="FW: $wm_asunto";
  $wm_boton_incluir_texto="";
  $wm_para="";
```

```
  $wm_concopia="";
}

pinta_plantilla_redactar($wm_para, $wm_concopia, $wm_concopia_ciega,
$wm_asunto,
                         $wm_texto, $boton_incluir_texto);

/*
 * Rutinas
 */


function dame_fichs_adjs($que_usr, $que_buzon, $que_msg)
{
  $info_adjs = array();

  include_once ("./wm_lib_imap.php");
  $df_buzon = dame_canal($que_buzon);
  $estructura=imap_fetchstructure($df_buzon, $que_msg);
  imap_close($df_buzon);

  if ($estructura->parts) {
    if (is_array($estructura->parts)) {
      $total_adjs=count($estructura->parts);
      for ($pos=1; $pos<$total_adjs ; $pos++) {
        $subestr=$estructura->parts[$pos];
        if (is_array($subestr->dparameters)) { // nombre del fichero
adjunto
          $nom_fich_adj=$subestr->dparameters[0]->value;
        } else
            $nom_fich_adj="SINTITULO";

        if ($subestr->type)
          $tipo_mime=$lib_tipos[$subestr->type];
        else
          $tipo_mime="text";

        $mime=$tipo_mime."/". strtolower($subestr->subtype);

        $info_adjs[$pos] = array("$nom_fich_adj",
                                 round($subestr->bytes/1024) . "k");
      }
    }
  }
  return $info_adjs;
}  // fin de dame_fichs_adjs()

function dame_fulano($array_obj, $cuantos) {
  $str="";
  if ($cuantos == "1") {
     if ($array_obj[0]->mailbox) {
       $str=$array_obj[0]->mailbox."@".$array_obj[0]->host;
     }
  } else  {
    for ($i=0;$i<count($array_obj);$i++) {
```

```
      $una_dir="";
      $una_dir.=" ".$array_obj[$i]->mailbox."@".$array_obj[$i]->host;
      $str .= " $una_dir";
    }
  }
  return ltrim($str);
}


function pinta_plantilla_redactar($un_to, $un_cc, $un_bcc,
                                  $un_asunto, $un_texto,
$b_incluir_texto)
{

  global $WM_MAX_NUM_FICH_ADJUNTOS, $WM_MAX_TAM_FICH_ADJUNTO,
$WM_BUZON_ACTUAL,
         $wm_componer_oper, $wm_num_adjs, $wm_que_msg;

  echo "<form name='wm_form_redactar' method='post'
action='wm_enviar.php'
        enctype='multipart/form-data'>
        <input type='hidden' name='MAX_FILE_SIZE'
                             value='$WM_MAX_TAM_FICH_ADJUNTO'>
        <center>
        <table BORDER=0 CELLPADDING=1 CELLSPACING=1>
        <tr bgcolor='#FFFBAD' valign='top'>
        <td>Para:</td>
        <td><input type='text' size=58 name='wm_para'
        value='".$un_to."'>
        </td>
        </tr>";

  echo "<tr bgcolor='#FFFBAD' valign='top'>
        <td>CC:</td>
        <td><input type='text' size=60 name='wm_concopia'
value='".$un_cc."' >
        </td>
        </tr>

        <tr bgcolor='#FFFBAD' valign='top'>
        <td>BCC:</td>
        <td><input type='text' size=60 name='wm_concopia_ciega'
value='".$un_bcc."' ></td>
        </tr>

        <tr bgcolor='#FFFBAD' valign='top'>
        <td>Asunto:</td>
        <td><input type='text' size=60 name='wm_asunto'
value='".$un_asunto."'></td>
        </tr>

        <tr bgcolor='#FFFBAD'>
        <td>Texto:</td>
        <td><textarea cols=60 rows=10
name='wm_texto'>".$un_texto."</textarea>
```

```
        <input type='hidden' name='wm_texto_escondido'
value='".$un_texto."'>
        </td>
        </tr>

        <tr bgcolor='#FFFBAD'>
        <td colspan=2 align='center' align='left'>
         
        ";
  echo "$b_incluir_texto";
  echo '<input type="button" onclick="limpia_campo_texto()"
value="borra texto">
         
        <input type="button" onclick="limpia_campos_form()"
value="borrar todo">
        </td>
        </tr>';

  echo "<tr bgcolor='#FFFBAD'><td colspan=2>
        <font face='arial narrow' size=-1>
        (<input type='checkbox' value='SI' name='wm_copiar_enviados'>
        Copiar en 'Enviados')
               
              <input type='button' value='envía!' onclick='envia()'>
               
              <input type='button' value='Cancelar'
onclick='cancela()'>
        </td></tr>";

  if ($wm_componer_oper == 3)
  {
    $adjs = dame_fichs_adjs($WM_USR, $WM_BUZON_ACTUAL, $wm_que_msg);

    for ($i=1; $i<=$wm_num_adjs; $i++)
    {
      echo"<tr bgcolor='#FFFBAD'>
           <td>Adjuntar:</td>
           <td>
           <input type='checkbox' name='wm_adj_ck$i' checked>
           <u>{$adjs[$i][0]}</u> ({$adjs[$i][1]})
           </td>
           </tr>";
    }

    $resto_adjs=$WM_MAX_NUM_FICH_ADJUNTOS - $wm_num_adjs;

  } else
    $resto_adjs=$WM_MAX_NUM_FICH_ADJUNTOS;

  for ($i=1; $i<=$resto_adjs; $i++)
  {
    echo"<tr bgcolor='#FFFBAD'>
         <td>Adjuntar:</td>
         <td>
         <input type='file' name='wm_adj$i' value='wm_adj1'>
```

```
    <input type='hidden' name='wm_adj_oculto$i' value='wm_adj1'>
          </td>
          </tr>";
  }
  echo '</table>
        </form>';
} // fin de pinta_plantilla_redactar()

?>
```

13.8 ENVIAR MENSAJES

Es desde la página generada por `wm_componer.php` desde donde se pulsa el botón de enviar mensaje, esto es, desde donde se llama al *script* `wm_enviar.php` para que haga efectiva la entrega del mensaje redactado al MTA (*Mail Transfer Agent*) y que éste lo retransmita al destinatario.

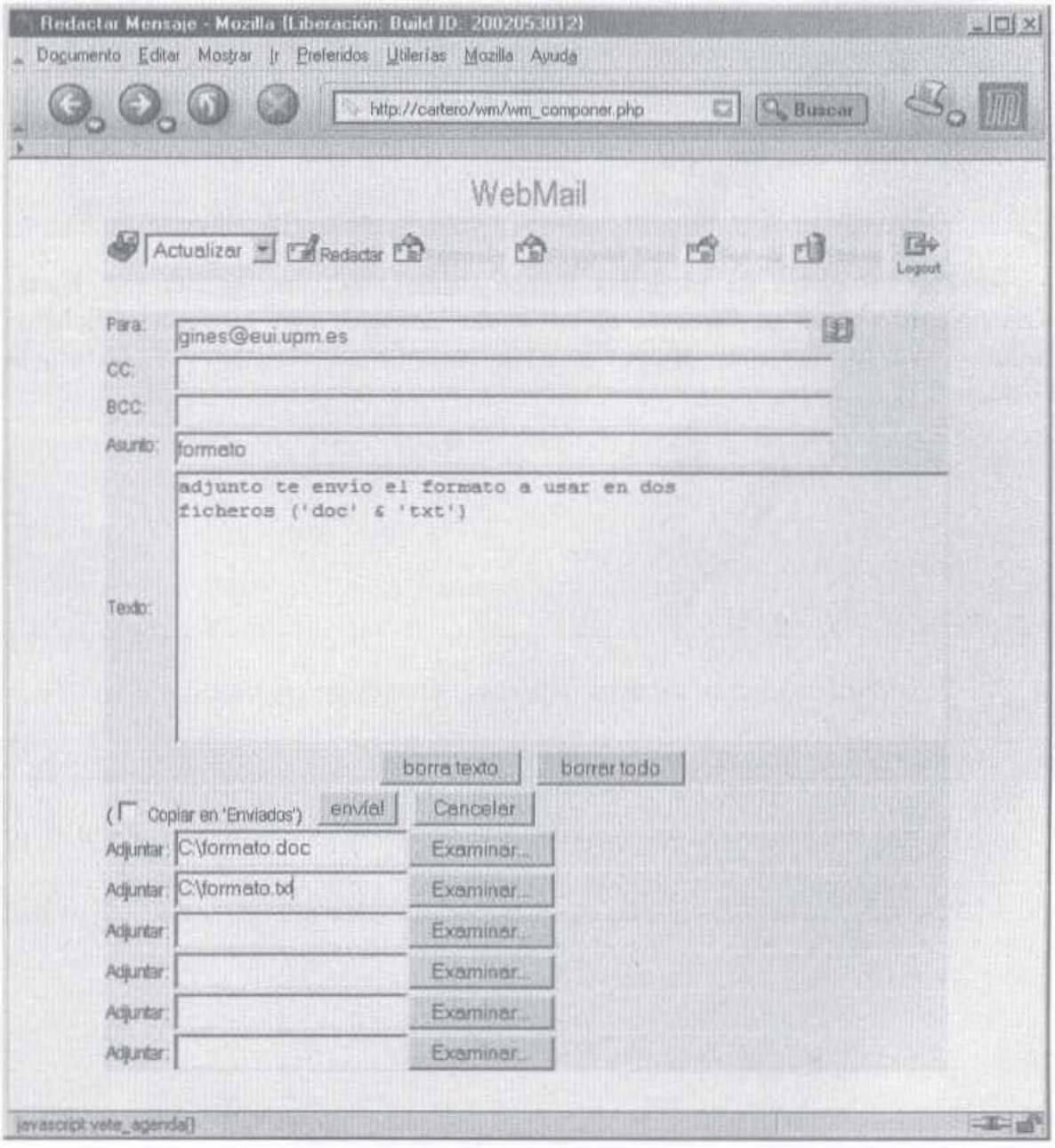
Los ficheros adjuntos que se suben al servidor para ser enviados se recuperan gracias al *array* `$_FILES` de PHP. Este *array* es bidimensional, de forma que en la primera columna se encuentra el nombre del fichero y ésta, a su vez, aloja a otro *array* asociativo con las claves `name`, `type`, `tmp_name` y `size`, que contienen el nombre del fichero, el tipo, el nombre del fichero temporal donde está guardado y el tamaño, respectivamente. Por esta razón, se genera el siguiente bucle que tratará cada uno de los ficheros adjuntos:

```
for ($i=1; $i<=$WM_MAX_NUM_FICH_ADJUNTOS ; $i++)
{
  $fich="wm_adj$i";
  if (is_uploaded_file($_FILES[$fich]['tmp_name'])) {
    $f_nombre  = "{$_FILES[$fich]['name']}";
    $f_tipo    = "{$_FILES[$fich]['type']}";
    $f_nom_tmp = "{$_FILES[$fich]['tmp_name']}";
    $f_tamanyo = "{$_FILES[$fich]['size']}";
    ....
    ....
  }
}
```

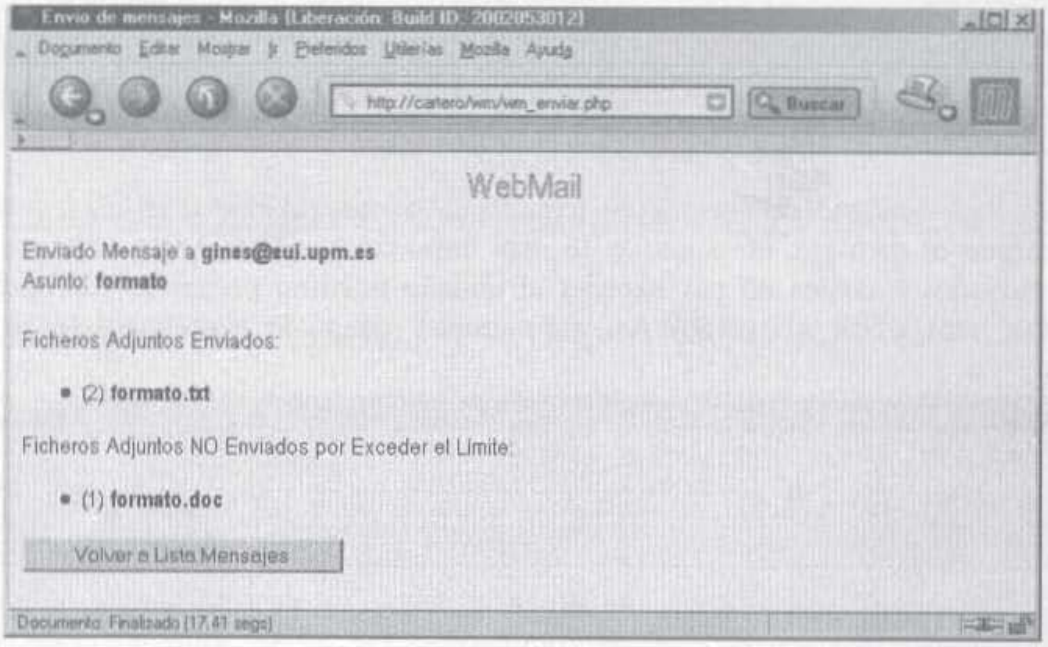
Resaltamos que el fichero que se ha *subido* al servidor, físicamente, tiene un nombre temporal aleatorio; por lo tanto, si se adjunta sin más, el nombre que tomará será el creado de forma ocasional. Para evitar este efecto no deseado, creamos un enlace duro (*hard link*) del fichero temporal (en Unix no ocupa disco), adjuntamos este último y, por supuesto, borramos estos ficheros en el servidor:

```
$f_temp = '/tmp/'.$f_nombre;  // fichero real
link($f_nom_tmp, $f_temp);
$msg->fattach($f_temp, "$f_nombre", $el_tipo);
unlink($f_temp);
unlink($f_nom_tmp);
```

Para finalizar, se muestra una pantalla de resumen donde se indica a quién se ha enviado el mensaje, el asunto y, lo más importante, qué ficheros adjuntos han sido enviados y cuáles no por exceder el tamaño máximo permitido (definido en `WM_MAX_TAM_FICH_ADJUNTO`). Así, por ejemplo, para el siguiente mensaje:



Se generaría la siguiente pantalla:



Para el envío de mensajes hemos usado una clase escrita por Kartic Krishnamurthy[5] que implementa el envío de mensajes que incorporan ficheros adjuntos. Su uso es muy simple: tras la creación de un ejemplar de objeto `MIME_mail`:

```
$msg = new MIME_mail($el_from,
                     $wm_para,
                     "$wm_asunto",
                     "$wm_texto",
                     "Cc: $wm_concopia\nBcc:
$wm_concopia_ciega");
```

Se le pueden adjuntar cuantos ficheros queramos con el método:

```
$msg->fattach($f_temp, "$f_nombre", $el_tipo);
```

Y, una vez terminado de componer el mensaje, se envía con el método:

```
$msg->send_mail();
```

[5] http://www.phpbuilder.net/columns/kartic20000807.php3

Hay que examinar si el usuario, en la pantalla anterior, ha marcado la opción de copiar el mensaje a enviar en la carpeta de `Enviados`. En su caso, hacemos uso de la función `imap_append()`:

```
if ($wm_copiar_enviados == "SI")
{
  $buz_enviados = dame_canal("Enviados");
  $nom_fich_buz_enviados=dame_fichero_buzon($WM_USR, 'Enviados');
  if (!imap_append($buz_enviados,
"\{$WM_SRV_IMAP}$nom_fich_buz_enviados",
              "From: $el_from\n"
              . "To: $wm_para\n"
              . "Cc: $wm_concopia\n"
              . "Bcc: $wm_concopia_ciega\n"
              . "Subject: $wm_asunto\n\n"
              . "$wm_texto\n\n"
              . "\n"
              ))
    die("*** ERROR al añadir a Enviados: ".imap_last_error());

  imap_close($buz_enviados);
  echo "(Copiado el mensaje en la carpeta 'Enviados')<br>";
}
```

El texto completo del *script* `wm_enviar.php` es:

```
<?
session_start();
if (!session_is_registered(WM)) header("Location: index.php");

include("./wm_cabeceras.php");

// para volver automáticamente:
// $script="<META HTTP-EQUIV=\"Refresh\" CONTENT=\"10;
URL=wm_ver_cabs.php\">";

pinta_cabecera_html("Envio de mensajes", "$script");

include "./wm_mime.class";
include "./wm_lib_imap.php";

if ($wm_copiar_enviados == "SI")
{
  $buz_enviados = dame_canal("Enviados");
  $nom_fich_buz_enviados=dame_fichero_buzon($WM_USR, 'Enviados');
  if (!imap_append($buz_enviados,
"\{$WM_SRV_IMAP}$nom_fich_buz_enviados",
              "From: $el_from\n"
              . "To: $wm_para\n"
              . "Cc: $wm_concopia\n"
              . "Bcc: $wm_concopia_ciega\n"
              . "Subject: $wm_asunto\n\n"
              . "$wm_texto\n\n"
              . "\n"
```

```
            ))
    die("*** ERROR al añadir a Enviados: ".imap_last_error());

  imap_close($buz_enviados);
  echo "(Copiado el mensaje en la carpeta 'Enviados')<br>";
}

$dominio = posix_uname();
$dir_usuario="$WM_USR@". $dominio['nodename'];
$el_from = sprintf("%s <%s>", $WM_USR, $dir_usuario);

$msg = new MIME_mail($el_from,
                     $wm_para,
                     "$wm_asunto", "$wm_texto",
                     "Cc: $wm_concopia\nBcc: $wm_concopia_ciega");

$i_arial="<font face='arial narrow'>";
print "$(i_arial)<br>Enviado Mensaje a <b>$wm_para</b><br>
       Asunto: <b>$wm_asunto</b><br>";

$msg_adjuntos="";
$msg_adj_no_enviados="";

for ($i=1; $i<=$WM_MAX_NUM_FICH_ADJUNTOS ; $i++)
{
  $fich="wm_adj$i";
  if (is_uploaded_file($_FILES[$fich]['tmp_name'])) {
    $f_nombre  = "($_FILES[$fich]['name'])";
    $f_tipo    = "($_FILES[$fich]['type'])";
    $f_nom_tmp = "($_FILES[$fich]['tmp_name'])";
    $f_tamanyo = "($_FILES[$fich]['size'])";

    if ($f_tamanyo > $WM_MAX_TAM_FICH_ADJUNTO) {
    // aqui no llega porque no se envia nada sobre el fichero 'gordo'
      echo "<h3>*** AVISO: el fichero <u>$f_nombre</u> NO se ha
enviado por ser
      mayor de <u>$WM_MAX_TAM_FICH_ADJUNTO</u>: $f_tamanyo
bytes</h3>";
    }

    $msg_adjuntos .="<li>($i) <b>$f_nombre</b>";
    if (preg_match("!/x\-.+!i", $f_tipo)) // obtengo el content-type
        $el_tipo = 'OCTET';
    else
        $el_tipo = $f_tipo;

    // si no hago la copia/enlace toma el nombre del fichero temporal
de php...
    $f_temp = '/tmp/'.$f_nombre;  // fichero real
    link($f_nom_tmp, $f_temp);

    /* ------------------------------------------------------------
------
     * puede ocurrir que el fichero ya exista y que además... no sea
nuestro...
```

```
     * solución: crear un directorio bajo tmp con el nombre del
usuario y allí
     * crear el enlace
     * ------------------------------------------------------------
------ */
    $msg->fattach($f_temp, "$f_nombre", $el_tipo);
    unlink($f_temp);
    unlink($f_nom_tmp);
  }
  else
  {
    $fich_oculto="\$nom_oculto=\$wm_adj_oculto$i;";
    eval("$fich_oculto");
    if ($nom_oculto)
    {
      $msg_adj_no_enviados .="<li>($i)
<b>".wbasename($nom_oculto)."</b>";
    }
  }
}

$msg->send_mail();

if ($msg_adjuntos)
{
  echo "<br>Ficheros Adjuntos Enviados:
  <ul>
  $msg_adjuntos
  </ul>";
}
if ($msg_adj_no_enviados)
{
  echo "Ficheros Adjuntos NO Enviados por Exceder el Límite:
  <ul>
  $msg_adj_no_enviados
  </ul>";
}

print '<form method="get" action="wm_ver_cabs.php">
       <input type=submit value="Volver a Lista Mensajes">
       </form>';

/*
 * Rutinas
 */

function wbasename($un_str)
{
  for($i=0; $i < strlen($un_str); $i++ )
  {
    $pos = strpos($un_str, '\\', $i);
    if ($pos != false)
      $ult_pos = $pos;
  }
  return substr($un_str, $ult_pos + 1, strlen($un_str));
}
?>
```

13.9 BORRAR O MOVER MENSAJES

Este *script* recibe la variable `wm_operacion` donde se indica la operación a realizar con la lista de mensajes que hay contenida en `wm_msgs`.

- Operación **Borrar**: Si la carpeta actual es `Entrada` o `Enviados`, los mensajes se mueven a la carpeta `Papelera`, pero, si la carpeta actual es `Papelera` y la operación que se quiere realizar es la de borrar, entonces los mensajes se borran de forma definitiva.

  Hay que tener en cuenta que la función `imap_delete()` no borra físicamente los mensajes, sino que los deja marcados para ser borrados. Se borrarán físicamente de la carpeta siempre que a `imap_delete()` le pasemos como parámetro `CL_EXPUNGE` o llamemos a la función `imap_expunge()`.

- Operación **Mover**: Si hay que mover mensajes de una carpeta a otra, se comprueba que ambas carpetas (origen y destino) no sean la misma (aspecto que se garantiza en `wm_ver_cabs`).

El código completo del *script* (`wm_selecs.php`) que ejecuta las operaciones de mover o borrar mensajes es:

```
<?
session_start();
if (!session_is_registered(WM)) header("Location: index.php");

/*
 * llamado desde wm_ver_cabs.php. Parametros:
 *    - wm_buzon_destino (si es mover)
 *    - wm_msgs: que tiene una lista de numeros de mensajes
 *    - wm_operacion (Mover/Eliminar),
 */

include("./wm_cabeceras.php");
pinta_cabecera_html("$wm_operacion msgs seleccionados", "");

$botonera=array();
$botonera[0]=""; // Actualizar
$botonera[1]=""; // Redactar
$botonera[2]=""; // Responder
$botonera[3]=""; // Responder Todos
$botonera[4]=""; // Reenviar
$botonera[5]=""; // Eliminar
$botonera[6]="1"; // Logout
```

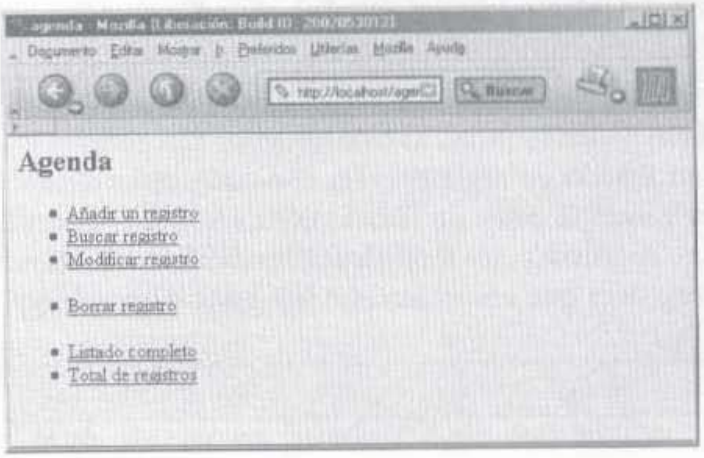
```
pinta_botonera($botonera);

$wm_lista_num_mensajes = explode (",", $wm_msgs);
// comprobar que origen y destino no es el mismo...
if ($wm_operacion == "Mover" && $WM_BUZON_ACTUAL ==
$wm_buzon_destino)
{
  die("**** ERROR: NO se puede mover un mensaje dentro del mismo
buzón");
}
// comprobar que al menos hay mas de un mensaje
if (strlen($wm_msgs)==0)
{
  die("**** ERROR: NINGUN mensaje para procesar ");
}
sort($wm_lista_num_mensajes, SORT_NUMERIC);
$lista_victimas=implode(",",$wm_lista_num_mensajes);

// otra comprobacion es que los numeros de mensaje pasados sean
reales
// habria que hacer una conexion previa y ver cuantos msgs hay en el
buzon
include("./wm_lib_imap.php");
if ($wm_operacion == "Eliminar")
{
  if ($WM_BUZON_ACTUAL == "Papelera")
  {
    echo "<h2>Borrado(s) de Papelera";
    $buzon=dame_canal("$WM_BUZON_ACTUAL");
    imap_delete($buzon, $lista_victimas) ||
       die("ERROR: al borrar: ". imap_last_error());
    imap_expunge($buzon);
    imap_close($buzon);
  }
  else
  {
    echo "<h2>Movido/s a Papelera ";
    $buzon=dame_canal("$WM_BUZON_ACTUAL");
    $fich_papelera=dame_fichero_buzon($WM_USR, "Papelera");
    imap_mail_move($buzon,$lista_victimas,$fich_papelera) ||
       die("ERROR: al mover a la papelera: ". imap_last_error());
    imap_expunge($buzon);
    imap_close($buzon);
  }
}
else
{
  echo "<h2>Movidos desde $WM_BUZON_ACTUAL a $wm_buzon_destino";
  $buzon=dame_canal("$WM_BUZON_ACTUAL");
  $fich_destino=dame_fichero_buzon($WM_USR, "$wm_buzon_destino");
  imap_mail_move($buzon,$lista_victimas,$fich_destino);
  imap_expunge($buzon);
  imap_close($buzon);
}
echo " el/los mensaje/s: $wm_msgs</h2>";
?>
```

APÉNDICE A

EJEMPLO DE USO DE FUNCIONES DE FICHEROS: AGENDA

Como ejemplo de aplicación de las funciones relacionadas con los ficheros, vamos a implementar una pequeña agenda electrónica que almacenará datos tales como el nombre, la dirección de correo y los números de teléfono fijo y móvil. Con esta agenda seremos capaces de añadir nuevos contactos, modificar datos existentes (por ejemplo, alguien cambia su dirección de correo), buscar, borrar y listar todas las entradas.

La entrada a la aplicación está centralizada en una página HTML (`ag_opers.html`) que nos muestra el acceso a las operaciones comentadas anteriormente:



A.1 DISEÑO DE LA APLICACIÓN

La implementación de la agenda se hará usando un fichero de texto donde iremos almacenando las entradas a la agenda. Los registros de la agenda los haremos coincidir con líneas, es decir, cada línea del fichero será un registro, una entrada de la base de datos. Dentro de cada registro distinguiremos los distintos campos (nombre, correo-e, teléfonos) a través de un carácter delimitador.

Dado que la aplicación es muy sencilla y las operaciones están bien delimitadas, el diseño también lo es: cada operación citada será tratada de manera independiente mediante un programa específico. Sin embargo, cada uno de estos programas tiene que ponerse de acuerdo en una serie de datos comunes, como, por ejemplo, el nombre y la ruta del fichero que hará las veces de base de datos. Por tanto, usaremos un fichero *externo* que será incorporado o importado por todos los programas y que contendrá las definiciones y rutinas que vayan a ser usadas por todos ellos.

En este fichero común se definen constantes con el nombre del fichero que nos servirá como base de datos (`FICH_BD`), el carácter que usaremos como separador de campo (`SEP_CAMPO`) y el carácter que usaremos como separador de registro (`SEP_REG`) que va a ser un cambio de línea.

Para hacer la agenda lo más flexible posible, en lo que a número de campos se refiere, definimos un *array* con todos ellos (`$datos`). Los valores de los elementos del *array* son cadenas de caracteres que los hacemos coincidir con los identificadores de los elementos del formulario. Si en algún momento hiciera falta cambiar algún elemento del formulario, modificaríamos convenientemente este *array*.

También se definirá la función `pinta_listado()` que será útil para sacar por pantalla un listado de registros. En concreto, estos listados se generarán en las siguientes ocasiones: al pedir un listado completo de la agenda, cuando necesitemos seleccionar un registro para modificarlo o borrarlo y, finalmente, a la hora de imprimir los registros que coinciden con los datos de una búsqueda.

```
pinta_listado($lineas, $tipo_elem_form, $leyenda_submit);
```

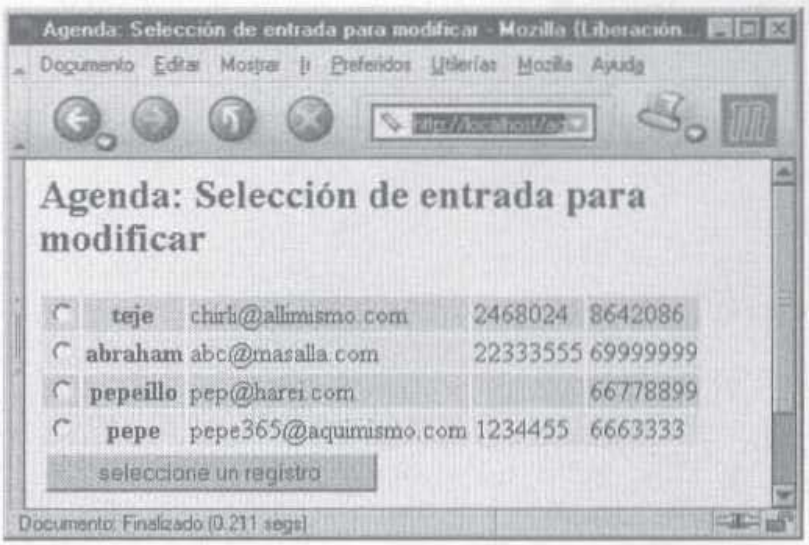
Los parámetros pasados serán las líneas que van a ser listadas, el tipo de elemento que aparecerá en la primera columna de la tabla (número, radio o casilla de verificación) y el texto que aparecerá en el botón utilizado para ejecutar la operación. Veamos tres ejemplos:

Para generar un listado con *checkboxes*, emplearíamos la siguiente línea:

```
pinta_listado($lineas_bd, 'c', 'borrar!');
```

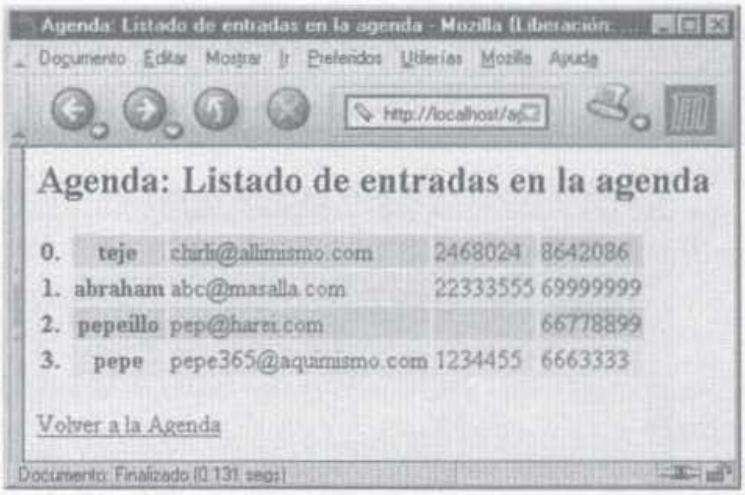
Para generar un listado con elementos de formulario *radio*, emplearíamos la siguiente línea:

```
pinta_listado($lineas, 'r', 'seleccione un registro');
```



Y, finalmente, para generar un listado simple que enumere las líneas a mostrar, emplearíamos la siguiente sentencia:

```
pinta_listado($arr_encontrados, 'n', '');
```



El listado completo del *script* (`ag_datos.inc`) que se encarga de implementar esta función junto con la definición de las constantes comunes indicadas anteriormente es el siguiente:

```
<?
$datos=array('nombre', 'correoe', 'tlf_fijo', 'tlf_movil');
$pos_datos=array();
for ($i=0; $i<count($datos); $i++)
  $pos_datos[$datos[$i]]=$i;

define("FICH_BD", "datos/agenda.dat");
define("SEP_CAMPO", ":");
define("SEP_REG", "\n");
define("color1", "#cccccc");
define("color2", "lightblue");

function pinta_listado($lineas, $tipo_elem_form, $leyenda_submit)
// $lineas es un array de registros
// $tipo_elem_form podrá tener 'r' (radio), 'c' (checkbox) o 'n' (numérico)
{
  $colores_filas = array('#FFFBAD','lightblue');
  $ind_col_filas=0;
  $total_lineas=count($lineas);
  $cont_lineas=0;
  echo "<table>";
  while ($cont_lineas<$total_lineas) {
    switch($tipo_elem_form){
    case 'c': $elem_form="<td>
       <input type='checkbox' name='victima$cont_lineas'
value='$cont_lineas'>
       </td>";
             break;
    case 'r': $elem_form="<td>
    <input type='radio' name='victima' value='$cont_lineas'>
                         </td>";
             break;
    case 'n':$elem_form="<th align=right
bgcolor='white'>$cont_lineas. </th>";
             break;
    default: $elem_form="";
    }
```

```
    $ind_col_filas++;
    $ind_col_filas%=2;

    echo "<tr bgcolor='$colores_filas[$ind_col_filas]'>";
    list($nombre, $correoe, $tlf_fijo, $tlf_movil) = explode(SEP_CAMPO,
$lineas[$cont_lineas]);
    echo "$elem_form";
    echo "<th>$nombre</th>
          <td>$correoe</td>
          <td>$tlf_fijo</td>
          <td>$tlf_movil</td>";
    echo "</tr>";
    $cont_lineas++;

  }
  if (strlen($leyenda_submit))
    echo "<tr>
          <td colspan='5'><input type='submit' value='$leyenda_submit'></td>
          </tr>";
  echo "</table>";
}
?>
```

A.2 INSERCIÓN DE NUEVOS REGISTROS

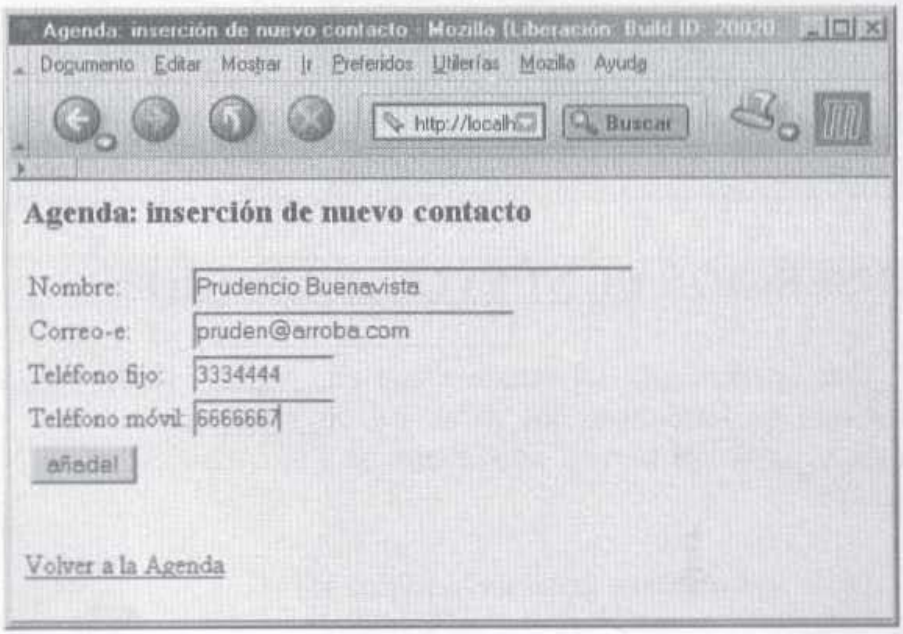
Esta operación la solventamos con una página HTML (`ag_suma1.html`), desde donde se introducen los datos de un nuevo contacto, y con un *script* (`ag_suma1.php`) que tomará estos datos para almacenarlos en el correspondiente fichero:

El listado siguiente generará la página HTML:

```
<html>
<head>
<title>Agenda: inserción de nuevo contacto</title>
</head>
<body>
<h3>Agenda: inserción de nuevo contacto</h3>
<form action='ag_suma1.php' method='get'>
<table border='0'>
<tr>
  <td>Nombre:
  <td><input type='text' name='nombre' size='35'>
</tr>
<tr>
  <td>Correo-e:
  <td><input type='text' name='correoe' size='25'>
</tr>
<tr>
  <td>Teléfono fijo:
  <td><input type='text' size='10' name='tlf_fijo'>
</tr>
<tr>
  <td>Teléfono móvil:
  <td><input type='text' size='10' name='tlf_movil'>
</tr>
```

```
<tr>
      <td colspan='2'><input type='submit' value='añade!'>
</tr>

</table>
</form>

<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
</body>
</html>
```

El resultado de visualizar esta página es el siguiente:



El programa que invoca la página anterior antes de insertar el registro hace un par de comprobaciones básicas: la primera es que, para prevenir que la base de datos se corrompa, evitamos que el carácter usado como delimitador de campos (`SEP_CAMPOS`) pueda estar contenido en algún elemento del formulario (esta comprobación la hacemos en la función `construye_reg()`). La segunda verificación consiste en evitar entradas duplicadas, esto es, que no haya dos entradas con el mismo nombre. Por esta razón, recorremos el fichero entero para contrastar el campo nombre introducido en la página HTML con todos los campos que almacenan el nombre de todos los registros (función `esta_repe()`). Y, ya por último, pasados los *controles*, los registros nuevos siempre se añaden al final del fichero (lo abrimos con la opción "a"). El listado completo de este *script* lo mostramos a continuación:

```
<html>
<head> <title>suma1.php</title> </head>
<body>
<?php
```

```
include("ag_datos.inc");

function construye_reg()
{
  $reg="";
  foreach ($_GET as $valor) {
    if (strstr($valor, SEP_CAMPO)) {
      echo "*** ERROR: NO está permitido el caracter '".SEP_CAMPO."' en
'$valor'";
      echo "<br><br><a href='javascript:history.back()'>&lt;&lt; Atrás</a>";
      exit;
    } else
      $reg .= SEP_CAMPO . "$valor";
  }
  return substr($reg,1);
}

function esta_repe($un_nombre)
{
  // para todos los registros del fichero separo el registro en
  // sus campos y compruebo que no coincide el nombre

  $lineas=file(FICH_BD);
  $total_regs=count($lineas);

  $i=0;
  while ($i<$total_regs)
  {
    $campos = explode(SEP_CAMPO,$lineas[$i]);
    if ($campos[0] == $un_nombre)
      return true;
    $i++;
  }
  return false;
}

function anyade_reg($reg)
{
  $df=fopen(FICH_BD, "a")
     or die("*** ERROR al abrir la BD");
  $reg .= SEP_REG;
  fputs($df, "$reg");
  fclose($df);
}

function dame_nombre($un_registro)
{
    $nom = explode(SEP_CAMPO,$un_registro);
    return $nom[0];
}

$registro=construye_reg();
$el_nombre=dame_nombre($registro);

if (esta_repe($el_nombre))
  echo "<h2>** ERROR el nombre '$el_nombre' ya está repetido</h2>";
else {
  anyade_reg($registro);
  echo "<h3>Registro anyadido! :^)</h3>";
}

?>
```

```
<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
```

A.3 BUSCAR UN REGISTRO

En las especificaciones iniciales comentamos que nuestra agenda sería capaz de hacer búsquedas. Las pesquisas las haremos sobre el campo del nombre, de manera que, dada una cadena, se mostrarán todos los registros que coinciden con el patrón introducido.

El procedimiento que empleamos para resolver las búsquedas es similar al utilizado en la anterior operación. Mostramos una página HTML (`ag_busca.html`) en la que se le solicita al usuario la introducción del nombre a buscar. Esta página invocará a un programa PHP (`ag_buscar.php`), al que le pasará la cadena a buscar, que llevará a cabo la operación mostrando los resultados obtenidos. La página HTML primera donde introducimos la cadena a buscar es:

```
<html>
<head>
<title>Agenda: Búsqueda de registros</title>
</head>
<body>
<h2>Agenda: Búsqueda de registros</h2>

<form action='ag_buscar.php' method='get'>
  Nombre: <input type='text' name='a_buscar'>
  <br><br>
  <input type='submit' value='buscar!'>
</form>

<br>
<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
```

La siguiente imagen nos muestra el resultado de visualizar la página anterior:

El programa que hace la búsqueda es:

```
<html>
<head><title>Agenda: Busqueda de registros</title></head>
<body>
<h2>Agenda: búsqueda de registros</h2>
<?php
include("ag_datos.inc");

function existe($un_nombre)
{
  // para todos los registros del fichero separo el registro
  // en sus campos compruebo que no coincide el nombre

  $lineas=file(FICH_BD);
  $total_regs=count($lineas);

  $i=0;
  $arr_encontrados=array();
  $tot_exitos=0;
  while ($i<$total_regs) {
    $campos = explode(SEP_CAMPO,$lineas[$i]);
    if (preg_match("/$un_nombre/i", $campos[0])) {
       $arr_encontrados[$tot_exitos]=$lineas[$i];
       $tot_exitos++;
    }
    $i++;
  }
  if ($tot_exitos > 0) {
    echo "<u>$tot_exitos</u> registro/s encontrado/s";
    pinta_listado($arr_encontrados, 'n', '');
  }
  else
    echo "NO se ha encontrado ninguna coincidencia :(<br>";
}

$se_busca=$_GET['a_buscar'];
echo "<h4>Cadena de búsqueda: <u>$se_busca</u></h4>";
existe($se_busca);

?>

<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
```

Suponiendo que partimos de los siguientes registros:

Si buscamos por la cadena `pepe`, obtendremos:

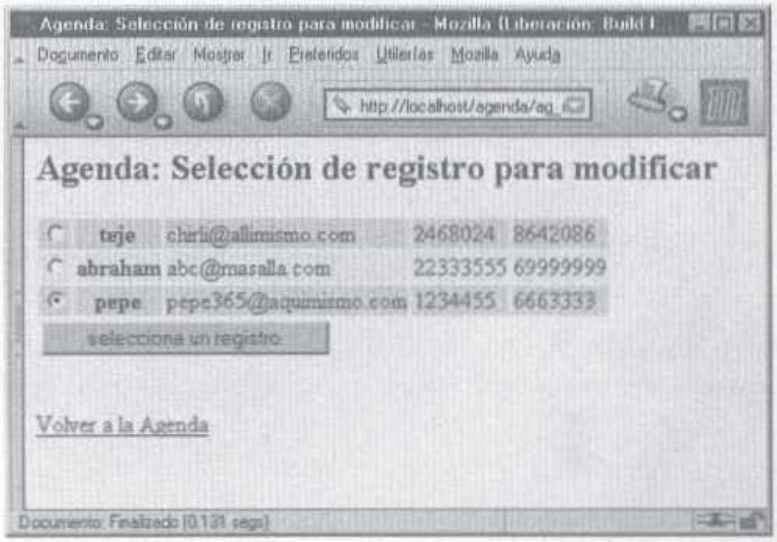
A.4 MODIFICACIÓN DE UN REGISTRO

Esta operación es algo más compleja en términos de número de *scripts* intervinientes. En total, necesitamos tres: uno que muestre al usuario una lista de todos los registros para que decida cuál elegir (`ag_modif_selec.php`), otro que muestre los datos del usuario en un formulario para poder ser modificados (`ag_modif_plantilla.php`) y el que por fin realiza los cambios en el fichero (`ag_modificar.php`).

Éste es el primero de ellos y su resultado:

```
<html>
<head>
<title>Agenda: Selección de registro para modificacar</title>
</head>
<body>
<h2>Agenda: Selección de registro para modificacar</h2>
<?php
include("ag_datos.inc");
```

```
$cont_lineas=0;
$lineas=array();
$df=fopen(FICH_BD, "r");
while ($reg=fgets($df)){
  $lineas[$cont_lineas]=$reg;
  $cont_lineas++;
}
fclose($df);


echo "<form action='ag_modif_plantilla.php' method='get'>";
pinta_listado($lineas, 'r', 'seleccione un registro');
echo "</form>";
?>
<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
```



El *script* que entra en acción en segundo lugar y su resultado es:

```
<html>
<head>
<title>Agenda: modificación de registro</title>
</head>
<body>
<h2>Agenda: modificación de registro</h2>
<?php
include("ag_datos.inc");

function dame_reg_num($elegido)
{
  $df=fopen(FICH_BD, "r");
  $cont=0;
  while ($cont<=$elegido) {
    $reg=fgets($df);
    $cont++;
  }
  fclose($df);
  return $reg;
```

```
}
if (!isset($_GET['victima']))
  die("<h4>NO ha seleccionado ningún registro para borrar :(</h4>");

$victima=$_GET['victima'];

$la_victima=dame_reg_num($victima);

$campos = explode(SEP_CAMPO, $la_victima);

$nombre    = $campos[$pos_datos['nombre']];
$correoe   = $campos[$pos_datos['correoe']];
$tlf_fijo  = $campos[$pos_datos['tlf_fijo']];
$tlf_movil = $campos[$pos_datos['tlf_movil']];

echo "
<form action='ag_modificar.php' method=get'>
<input type='hidden' name='victima_modificacion' value='$victima'>
<table border='0'>
<tr>
  <td>Nombre:
  <td><input type='text' size='35' name='nombre' value='$nombre'>
</tr>
<tr>
  <td>Dirección de correo-e:
  <td><input type='text' size='25' name='correoe' value='$correoe'>
</tr>
<tr>
  <td>teléfono fijo:
  <td><input type='text' size='10' name='tlf_fijo' value='$tlf_fijo'>
</tr>
<tr>
  <td>teléfono móvil:
  <td><input type='text' size='10' name='tlf_movil' value='$tlf_movil'>
</tr>
<tr>
      <td colspan='2'><input type='submit' value='modifica!'>
</tr>
</table>
</form>
<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
";
?>
```

Al igual que en el programa que inserta registros, antes de hacer efectiva la modificación, hay que comprobar que no se introduce el carácter que usamos como delimitador de campos y que el nombre no está siendo usado ya:

```
<html>
<head> <title>Agenda: Modificación de entradas</title> </head>
<body>
<h2>Agenda: Modificación de entradas</h2>
<?php
include("ag_datos.inc");

function esta_repe($un_nombre)
{
  // para todos los registros del fichero separo el registro en
  // sus campos y compruebo que no coincide el nombre

  $lineas=file(FICH_BD);
  $total_regs=count($lineas);

  $i=0;
  while ($i<$total_regs)
  {
    $campos = explode(SEP_CAMPO,$lineas[$i]);
    if ($campos[0] == $un_nombre)
       return true;
    $i++;
  }
  return false;
}

function construye_reg()
{
  global $datos;  // definido en 'ag_datos.inc'
  $un_reg="";
  // importante que $datos este ordenado
  for ($i=0; $i<count($datos); $i++) {
    if (strstr($_GET[$datos[$i]], SEP_CAMPO)) {
      echo "** ERROR: NO está permitido el car '".SEP_CAMPO. "' en
'${_GET[$datos[$i]]}'";
```

```
      echo "<br><br><a href='javascript:history.back()'>&lt;&lt; Atrás</a>";
      exit;
    } else
      $un_reg.= SEP_CAMPO . $_GET[$datos[$i]];
  }
  $un_reg = substr($un_reg,1);  // caracter ':' primero
  return $un_reg . SEP_REG;
}


$el_nombre=$_GET['nombre'];
if (esta_repe($el_nombre)) {
  echo "<h2>** ERROR el nombre '$el_nombre' ya está usado</h2>";
  echo "<br><br><a href='javascript:history.back()'>&lt;&lt; Atrás</a>";
  exit;
} else {
  $regs=file(FICH_BD);
  $regs[$_GET['victima_modificacion']]=construye_reg();
  $df=fopen(FICH_BD, "w");
  for ($i=0;$i<count($regs);$i++)
    fwrite($df, $regs[$i]);
  fclose($df);
}
?>

<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
```

A.5 BORRADO DE UN REGISTRO

Evidentemente, lo primero que hay que hacer para borrar un registro es saber cuál. Por tanto, para esta operación necesitaremos dos programas. El primero (`ag_borra_selec.php`) nos mostrará todas las líneas de la agenda junto con casillas de verificación (*checkboxes*) para que seleccionemos todos los registros que queramos:

```
<html>
<head>
<title>Agenda: Selección de Entradas a Borrar</title> </head>
<body>
<h2>Agenda: Selección de Entradas a Borrar</h2>
<?php
include("ag_datos.inc");

$cont_lineas=0;
echo " <table border='0' cellspacing=5>";
$lineas_bd=array();
$df=fopen(FICH_BD, "r");
while ($reg=fgets($df))
{
  $lineas_bd[$cont_lineas]=$reg;
  $cont_lineas++;
}
fclose($df);

echo "<form action='ag_borrar.php' method='get'>";
pinta_listado($lineas_bd, 'c', 'borrar!');
echo "</form>";
?>

<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
```

El segundo programa (`ag_borrar.php`), el que hace efectivo el borrado de los registros, sigue la siguiente estrategia: primero, confecciona un *array* cuyos elementos son los números de línea/registro que van a ser borrados (`$lista_victimas`); luego, lee el fichero entero sobre otro *array* (`$regs`) y, por último, recorre este *array* entero para aplicarles la función `in_array()` y ver así qué líneas han de eliminarse.

```
<html>
<head> <title>Agenda: Borrar entradas</title> </head>
<body>
<h2>Agenda: Borrar entradas</h2>
<?php
include("ag_datos.inc");

$lista_victimas=array();

$total_victimas=0;
foreach ($_GET as $clave=>$valor) {
  $lista_victimas[$total_victimas]=$valor;
  $total_victimas++;
}
echo "<h3>Número de registros a borrar: <u>",
count($lista_victimas),"</u></h3>";
if ($total_victimas > 0) {
  $regs=file(FICH_BD);
  $df=fopen(FICH_BD, "w");
  for ($i=0;$i<count($regs);$i++)
    if (!in_array($i, $lista_victimas))
```

```
      fwrite($df, $regs[$i]);
  fclose($df);
}
else
   echo "<h4>NO ha seleccionado ningún registro para borrar :(</h4>";
?>

<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
```

A.6 LISTADO DE TODOS LOS REGISTROS

En este programa (`ag_listado.php`) leemos todas las líneas/registros del fichero y las insertamos en un *array* que posteriormente pasaremos como argumento a la función `pinta_listado()` para que las imprima. Su contenido es:

```
<html>
<head> <title>Agenda: Listado de entradas en la agenda</title> </head>
<body>
<h2>Agenda: Listado de entradas en la agenda</h2>
<?php
include("ag_datos.inc");

$df=fopen(FICH_BD, "r")
    or die("**** ERROR al abrir la BD");
$cont_lineas=0;
$kk=array();
while ($reg=fgets($df))
  $kk[$cont_lineas++]=$reg;
fclose($df);

pinta_listado($kk, 'n', '');
?>

<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
```

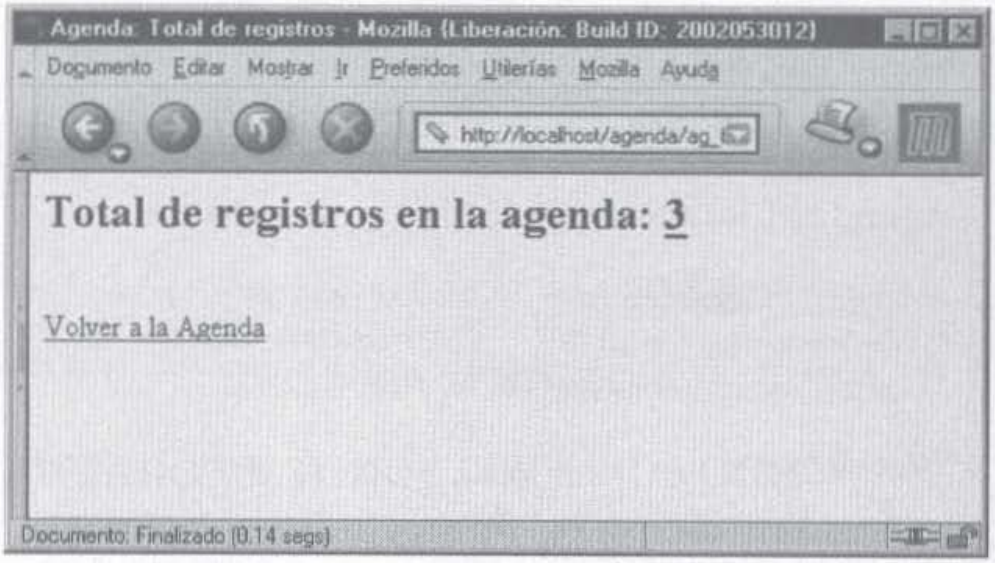
El resultado se muestra en la siguiente imagen:

A.7 TOTAL DE REGISTROS

Éste es el *script* (`ag_total.php`) más sencillo de todos, puesto que cada línea de nuestro fichero constituye un registro; lo único que hay que hacer es leer el fichero entero sobre un `array` con la función `file()` e imprimir el total de elementos del *array*:

```
<html>
<head> <title>Agenda: Total de registros</title> </head>
<body>
<h2>Total de registros en la agenda:
<?php
include("ag_datos.inc");

$lineas=file(FICH_BD);
echo "<u>", count($lineas), "</u>";
?>
</h2>
<br>
<a href='ag_opers.php'>Volver a la Agenda</a>
```

El resultado de su ejecución sería:


Agenda: Total de registros - Mozilla {Liberación: Build ID: 2002053012}
Documento Editar Mostrar Ir Preferidos Utilerías Mozilla Ayuda
http://localhost/agenda/ag_t
Total de registros en la agenda: 3
Volver a la Agenda
Documento: Finalizado (0.14 segs)

APÉNDICE B

FICHERO DE CONFIGURACIÓN PHP.INI

El fichero de configuración `php.ini` determina el comportamiento, la seguridad, la fiabilidad y estabilidad del intérprete PHP. Dependiendo de la instalación, en el caso de máquinas U*ix suele encontrarse en el directorio `/usr/local/lib` o bajo `/etc` y, en las máquinas Windows, en los directorios `c:\windows` o `c:\winnt`.

En este fichero usaremos el carácter `;` como inicio de comentario y podremos usar las marcas de sección del tipo `[una_seccion]`, pues también se ignoran. El formato de este fichero es muy sencillo, se basa en líneas del estilo:

```
nombre_de_directiva=valor
```

Los valores pueden ser, o bien booleanos (en cuyo caso los valores que podrán tomar serán `true/on/yes/1` o `false/off/no/0`), o cadenas de caracteres que habrán de encerrarse entre comillas dobles.

B.1 DIRECTIVAS GENERALES

`asp_tags=Off`

Además de las usuales etiquetas `<?php` y `?>`, permite el uso de las etiquetas tipo ASP `<%` y `%>`.

`short_open_tag=On`

Permite o no las etiquetas `<?` y `?>` para abrir y cerrar PHP. Si se desea utilizar PHP junto con XML, se deberá desactivar esta directiva para poder usar las etiquetas `<?xml` y `?>`.

`magic_quotes_gpc=On`
: Activa la conversión automática de las comillas para las operaciones GPC (`Get/Post/Cookie`): todas las comillas simples ('), comillas dobles ("), barras invertidas (\) y los valores `NULL` son automáticamente prefijados con una barra invertida. Si, además, `magic_quotes_sybase` tiene asignado el valor *on*, la comilla sencilla es marcada con otra comilla sencilla en lugar de la barra invertida.

`magic_quotes_runtime=Off`
: Se activa el prefijado con la barra inversa de los caracteres anteriores para los datos generados a partir de cualquier fuente externa (bases de datos, archivos, etc.).

`variables_order=`
: Establece el orden en que se evalúan las variables EGPCS (`Environment, GET, POST, Cookie, Server`). El valor por defecto para esta directiva es `EGPCS`. Si le asignáramos, por ejemplo, `PG`, PHP entonces ignoraría las variables de entorno, las *cookies* y las variables de servidor y, además, sobrescribiría las variables recibidas mediante POST con las recibidas mediante GET que tuvieran el mismo nombre.

`register_globals=Off`
: Implica que se puede acceder a las variables EGPCS (`Environment, GET, POST, Cookie, Server`) como variables globales.

`register_argc_argv=Off`
: Con esta directiva se declaran (o no) las variables `argv` y `argc` (las que dan información sobre las variables del método `GET`). Si no se usan, es preferible deshabilitarlas en aras del rendimiento.

`precision=12`
: Número de dígitos significativos mostrados en números de coma flotante.

B.2 ERRORES

`display_errors=Off`
: Determina que los errores producidos en la ejecución de un *script* se muestren en pantalla como parte de la salida generada HTML.

`error_log=`
: Especifica el fichero donde se registrarán los errores generados por los *scripts*. Si asignamos el valor especial `syslog`, los errores se envían al registro de errores del sistema: en U*IX los recogerá el demonio `syslogd` y, en Windows NT/2000/XP, será el registro de eventos.

`error_reporting=`

Indica el nivel de detalle de errores que queremos obtener. El valor que recibe esta directiva se conforma con las siguientes constantes:

Constante	Significado
E_ALL	todos los errores y avisos
E_ERROR	errores graves en tiempo de ejecución
E_WARNING	avisos en tiempo de ejecución
E_PARSE	errores de compilación
E_NOTICE	advertencias en tiempo de ejecución (suelen provenir de *errores semánticos* del tipo, por ejemplo, de usar una variable directamente sin haber sido previamente inicializada)
E_CORE_ERROR	errores graves que ocurren en el arranque de PHP
E_CORE_WARNING	avisos que ocurren en el arranque de PHP
E_COMPILE_ERROR	errores graves de compilación
E_COMPILE_WARNING	avisos de errores de compilación
E_USER_ERROR	mensajes de error generados por el usuario
E_USER_WARNING	mensajes de aviso generados por el usuario
E_USER_NOTICE	mensajes de advertencia generados por el usuario

Así, si queremos mostrar solamente los errores, escribiremos:

```
error_reporting=E_COMPILE_ERROR|E_ERROR|E_CORE_ERROR
```

Y para mostrar todos los errores excepto, las advertencias:

```
error_reporting=E_ALL & ~E_NOTICE
```

`html_errors=Off`

Incluye (o no) etiquetas HTML en los mensajes de error. En el nuevo formato de los errores se puede pinchar en ellos para ser reconducido a una página donde se describa dicho error.

`docref_root=`

El nuevo formato de errores contiene una referencia a una página que describe el error cometido. En esta directiva se indica el lugar donde podemos conseguir esta información (el nombre debe acabar con el carácter "`/`"). Valores válidos pueden ser: `/manual/` en caso de tengamos una copia de él en dicho directorio, o `http://unsitio.com/manual/es/`, si es que estuviera en dicho URL.

`docref_ext=`
: Indica la extensión de los ficheros. El valor introducido debe comenzar con el carácter "." (por ejemplo, `.html`).

`log_errors=Off`
: Indica si los errores generados por los *scripts* deben ser registrados en el registro del servidor (esta opción es dependiente del servidor).

`track_errors=Off`
: Si está habilitada, el último mensaje de error lo almacenará en la variable global `$php_errormsg`.

B.3 FICHEROS

`doc_root=`
: "Directorio raíz" de PHP en el servidor. Esta directiva sólo se usa si tiene un valor asignado. Si PHP está configurado en *safe mode*, no se sirven archivos fuera de este directorio.

`auto_append_file=`
: Si se indica un nombre de fichero, éste se ejecutará al final de cada *script* tal como si se hubiese ejecutado un `include()` de él. Si el *script* acaba con `exit()`, esta directiva no tiene efecto.

`auto_prepend_file=`
: Si se indica un nombre de fichero, éste se ejecutará antes de cada *script* tal como si se hubiese ejecutado un `include()` de él.

`open_basedir=`
: Si se indica un nombre de directorio (acabado en "/", por ejemplo, `/dir1/dir2/`), los *scripts* PHP sólo podrán abrir ficheros que estén bajo dicho directorio. Si se indica el valor ".", entonces un *script* sólo podrá abrir aquellos ficheros que estén en su mismo directorio.

  Se puede especificar una lista de directorios: se separan con el carácter ":", excepto en Windows que hay que usar un ";". Por defecto, se permite abrir cualquier fichero.

`include_path=`
: Esta directiva se comporta de manera similar a la variable de entorno PATH de Unix o Windows: especifica una lista de directorios en los que las funciones `require()`, `include()` y `fopen_with_path()` buscan los archivos. Los distintos directorios se separan con el carácter ":", excepto en Windows que hay que usar un ';'. El valor por defecto para esta directiva es "." (sólo el directorio actual).

`allow_url_fopen=On`
: Esta directiva es la responsable de que se puedan abrir ficheros remotos (con formato URL).

`file_uploads=On`
: Si se permite o no la *subida* de ficheros al servidor.

`upload_tmp_dir=`
: En esta directiva se especificará el directorio temporal donde se almacenarán los archivos que se suban (*upload*) desde el cliente al servidor. Lógicamente, este directorio debe tener permiso de escritura para el usuario bajo el que se ejecute PHP.

`upload_max_filesize=2M`
: Tamaño máximo que puede tener un fichero subido al servidor.

`post_max_size=8M`
: Establece el tamaño máximo de los datos que pueden enviarse vía `POST`. Esta directiva afecta a la *subida* de ficheros: para cargar ficheros grandes, este valor debe ser mayor que `upload_max_filesize`. Además, la directiva `memory_limit` también afecta a esta operación, por lo que debe ser mayor que `post_max_size`.

`user_dir=`
: Nombre del directorio que está debajo del directorio de conexión (o casa) del usuario (por ejemplo, `public_html`) donde están los *scripts* PHP.

`implicit_flush=Off`
: Si está activada, los datos generados como salida (funciones `echo`, `print` o texto HTML), se envían directamente sin almacenarse en un *buffer*. Sólo es recomendable su uso si se necesita depurar un *script* dado que puede repercutir negativamente en el rendimiento del servidor.

B.4 RECURSOS

`max_execution_time=30`
: Establece el tiempo máximo (en segundos) que un *script* puede estar ejecutándose.

`memory_limit=8M`
: Establece la cantidad máxima de memoria que un *script* puede usar.

B.5 SEGURIDAD

`engine=On`

Sólo tiene efecto cuando se usa PHP como módulo de Apache. Se emplea para permitir la ejecución de PHP sólo en determinados directorios o servidores virtuales (poniendo `engine off` en el lugar adecuado en el fichero `httpd.conf`, se evita la ejecución de PHP).

`safe_mode=Off`

Activa el modo seguro.

`safe_mode_exec_dir=`

Si PHP se utiliza en modo seguro, la función `system()` y el resto de funciones que ejecutan programas del sistema no ejecutarán programas que no estén en dicho directorio directorio.

`safe_mode_include_dir=`

Las comprobaciones de UID/GID se obvian cuando se incluyen ficheros que están a partir de este directorio.

`safe_mode_gid=Off`

Si se asigna `On`, al abrir un fichero no se hace la comprobación del UID (activado por defecto en modo seguro), sino la del GID.

`disable_functions=`

Se impedirá la ejecución de las funciones que aparezcan listadas (separadas por comas) en esta directiva. Esta directiva es independiente del modo seguro.

B.6 EXTENSIONES DINÁMICAS

`enable_dl=On`

Sólo aplicable a PHP como módulo de Apache. Permite la carga dinámica de extensiones PHP mediante la función `dl()`. En modo seguro, no se puede usar `dl()`.

`extension_dir=`

Directorio donde se encuentran las extensiones que se pueden cargar de manera dinámica.

`extension=`

Indica qué extensiones dinámicas debe cargar el PHP cuando arranca.

B.7 SESIONES

`session.use_cookies=1`
Indica si el módulo puede usar *cookies* para guardar el `session id` en el lado del cliente. El valor 1 indica que está activado.

`session.cookie_path=/`
Camino para el cual es válida la *cookie*.

`session.cookie_domain=`
Dominio para el cual es válida la *cookie*.

`session.save_handler=files`
Especifica el nombre del controlador empleado para gestionar la información relativa a las sesiones.

`session.save_path=/tmp`
Argumentos pasados al controlador de almacenamiento. Si se ha especificado `files`, entonces aquí indicaremos el directorio donde se crean los ficheros de las sesiones.

`session.name=PHPSESSID`
Nombre de la sesión que se usa como nombre de la *cookie*. Debe contener caracteres alfanuméricos.

`session.auto_start=0`
Especifica si el módulo de las sesión inicia una sesión automáticamente al hacer la petición (el valor 0 indica que está desactivado).

`session.cookie_lifetime=0`
Especifica la duración de la *cookie* en segundos que se manda al navegador. El valor 0 significa 'hasta que se cierra el navegador'.

`session.serialize_handler=php`
Controlador utilizado para guardar y restaurar los datos.

`session.gc_probability=1`
Especifica, en porcentaje, la probabilidad de que se inicie el proceso de recolección de basura (*garbage collection*) en cada inicialización de una sesión.

`session.gc_maxlifetime=1440`
Indica el número de segundos tras los cuales los datos se considerarán como "basura" y serán eliminados por el proceso de recolección de basura.

`session.referer_check=`
: El valor que se asigne a esta directiva se comprobará en cada HTTP Referer, esto es, la página desde la que hemos llegado a la actual. HTTP_REFERER debe contener la cadena aquí indicada para ser considerado válido.

`session.entropy_file=`
: Establece la ruta a un recurso externo (normalmente un archivo) que se usará como fuente adicional de entropía en el proceso de creación de session id's, por ejemplo, `/dev/random` o `/dev/urandom`, disponibles en muchos sistemas Unix.

`session.entropy_length=16`
: Especifica el número de *bytes* que serán leidos del archivo indicado en `session.entropy_file`. El valor 0 indica desactivado.

`session.cache_limiter=nocache`
: Especifica el método de control de la caché a usar en las páginas de la sesión. Puede tomar los valores: `none` / `nocache` / `private` / `private_no_expire` / `public`.

`session.cache_expire=180`
: Especifica, en minutos, la caducidad o tiempo de vida de las páginas de la sesión que se encuentran en la caché. No tiene efecto para el limitador `nocache`.

`session.use_trans_sid=0`
: Automatiza la incorporación del identificador de sesión en todas las peticiones. Activar esta directiva puede ser peligroso puesto que alguien con una sesión activada puede enviar a otra persona un URLs que contenga un identificador de sesión activo.

`url_rewriter.tags="a=href,area=href,frame=src,input=src,form=fakeentry"`
: Especifica qué etiquetas HTML serán reescritas para incluir el identificador de sesión si su inclusión transparente está activada.

B.8 CORREO ELECTRÓNICO

`SMTP=`
: Nombre DNS o dirección IP del servidor de correo saliente que PHP deberá usar para enviar correo con la función `mail()`.

NOTA: Sólo para Windows.

`sendmail_from=`
: La dirección del remitente (campo `De:`) que aparecerá en los mensajes enviados desde PHP (por ejemplo, `usr@ningunsitio.com`).

> **NOTA:** Sólo para Windows.

`sendmail_path=`
: Ruta del demonio de correo SMTP. Tiene posibles valores: `/usr/sbin/sendmail`, `/usr/lib/sendmail` o `/var/qmail/bin/sendmail`.

> **NOTA:** Sólo para Unix.

B.9 MySQL

`mysql.allow_persistent=On`
: Permitir conexiones MySQL persistentes.

`mysql.max_persistent=-1`
: Número máximo de conexiones MySQL persistentes por proceso (`-1` significa indefinidas).

`mysql.max_links=-1`
: Número máximo de conexiones MySQL por proceso, incluyendo las persistentes (`-1` significa indefinidas).

`mysql.default_host=`
: Servidor por defecto cuando no se especifica ninguno en `mysql_open()`. No funciona en modo seguro.

`mysql.default_user=`
: Nombre de usuario por defecto para las conexiones al de base de datos si no se especifica otro. No funciona en modo seguro.

`mysql.default_password=`
: Clave por defecto para las conexiones al servidor de base de datos si no se especifica otro.

> **NOTA:** No es aconsejable usar esta directiva puesto que cualquier usuario con acceso PHP puede ver dicha clave simplemente ejecutando: *echo cfg_get_var("mysql.default_password")*; además, por supuesto, de todos aquellos usuarios que tengan permiso de lectura en PHP.INI.

`mysql.default_port=`
: Número de puerto TCP al que se intentará establecer la conexión con el servidor MySQL mediante la función `mysql_connect()`.

`mysql.default_socket=`
: Nombre por defecto del *socket* usado en conexiones locales al servidor MySQL.

B.10 ODBC

`uodbc.default_db=`
: Fuentes de datos ODBC a utilizar si no se especifica una en `odbc_connect()` o en `odbc_pconnect()`.

`uodbc.default_user=`
: Nombre de usuario si no se especifica uno en `odbc_connect()` o en `odbc_pconnect()`.

`uodbc.default_pw=`
: Clave para usar si no se especifica una en `odbc_connect()` o en `odbc_pconnect()`.

`uodbc.allow_persistent=On`
: Indica si se permiten o no conexiones persistentes de ODBC.

`uodbc.max_persistent=-1`
: Número máximo de conexiones persistentes de ODBC por proceso (`-1` significa indefinidas).

`uodbc.max_links=-1`
: El número máximo de conexiones ODBC por proceso, incluyendo las persistentes (`-1` significa indefinidas).

`odbc.defaultlrl = 4096`
: Tamaño máximo de caracteres que devuelve de un campo (por ejemplo, de tipo `TEXT`).

B.11 MATEMATICABC

`bcmath.scale =0`
: Número de dígitos decimales de precisión para todas las funciones de `bcmath`.

B.12 DIRECTIVAS RELACIONADAS CON LOS NAVEGADORES

`browscap=`
: Nombre del archivo que contiene la descripción de las capacidades de los distintos navegadores.

APÉNDICE C

RESUMEN DE FUNCIONES DE MySQL

mysql_connect(servidor,usr,clave[,nuevo_enlace[,opcs]])

Establece una conexión con el servidor MySQL. Devuelve un descriptor de conexión con el que habrá que realizar las subsecuentes operaciones con el servidor.

mysql_pconnect(servidor, usr, clave [, opcs])

Establece una conexión persistente con el servidor MySQL.

mysql_close(descr_conexion)

Cierra la conexión MySQL.

mysql_select_db(nombre_bd, descr_conexion)

Selecciona un base de datos y la deja como activa.

mysql_list_dbs(descr_conexion)

Devuelve en un cursor todas las bases de datos que están disponibles en el servidor.

mysql_list_tables(nombre_bd, descr_conexion)

Devuelve un cursor con las tablas que hay en una base de datos.

```
$cursor = mysql_list_tables("agenda", $dbd);
while ($fila = mysql_fetch_row($cursor)) {
  print "tabla: $fila[0]<br>";
}
```

mysql_tablename(cursor, posicion)

A partir del cursor devuelto por `mysql_list_tables()` devuelve el nombre de las tablas.

```
$cursor = mysql_list_tables("agenda");
for ($i = 0; $i < mysql_num_rows($cursor); $i++)
  echo "tabla:" , mysql_tablename($cursor, $i), "<br>";
```

mysql_list_fields(nombre_bd, nombre_tabla, descr_conexion)

Devuelve un cursor con información sobre los campos de la tabla indicada. Esta información habrá de ser recorrida con cualquiera de las funciones `mysql_field_flags()`, `mysql_field_len()`, `mysql_field_name()` y `mysql_field_type()`.

```
$campos  = mysql_list_fields("agenda", "la_agenda", $dbd);
for ($i = 0; $i < mysql_num_fields($campos); $i++) {
   echo "campo($i): ", mysql_field_name($campos, $i) . "<br>";
}
```

mysql_errno(descr_conexion)

Devuelve el número del mensaje de error ocurrido en la última operación SQL.

mysql_error(descr_conexion)

Devuelve el mensaje de error ocurrido en la última operación SQL.

mysql_create_db(nombre_nueva_bd, descr_conexion)

Crea una base de datos en el servidor MySQL..

mysql_drop_db(nombre_bd, descr_conexion)

Borra una base de datos completa en el servidor.

mysql_db_query(nombre_bd, consulta_sql, descr_conexion)

Envía y ejecuta una consulta SQL en la base de datos indicada como parámetro.

mysql_query(consulta_sql, descr_conexion [, modo])

Envía y ejecuta una consulta SQL en la base de datos activa.

mysql_unbuffered_query(consulta [descr_conexion [, modo]])

Envía y ejecuta una consulta SQL en la base de datos activa. A diferencia de `mysql_query()`, el resultado lo devuelve directamente, sin pasar por ninguna memoria interna intermedia (*buffer*), lo cual redunda en una mayor rapidez y ahorro de memoria. La operación que no puede realizarse sobre el cursor devuelto por esta función es `mysql_num_rows()`.

mysql_insert_id(descr_conexion)

Devuelve el valor que obtiene una columna de tipo `AUTO_INCREMENT` en la última operación `INSERT`.

mysql_affected_rows(descr_conexion)

Devuelve el número de filas que se han visto involucradas o afectadas por la última operación SQL.

mysql_num_fields (descr_conexion)

Devuelve el número de campos de un cursor.

mysql_num_rows(descr_conexion)

Devuelve el número de filas de un cursor.

mysql_fetch_array(cursor)

Extrae del cursor una fila en forma de un *array* que puede ser indistintamente referenciado de forma numérica y/o asociativa (con nombres de campos).

mysql_fetch_assoc(cursor)

Extrae del cursor una fila en forma de un *array* asociativo.

mysql_fetch_field(cursor [, num_campo])

Obtiene de un cursor la información relativa a una columna para devolverla como un objeto con información tal como el nombre de la columna, nombre de la tabla a la que pertenece, el tipo, etc.

mysql_fetch_lengths(cursor)

A partir de un cursor, devuelve un *array* numérico que contiene las longitudes de cada campo de la última fila extraída por `mysql_fetch_row()`.

mysql_fetch_object(cursor)

Obtiene del cursor una fila y la devuelve en forma de objeto.

```
$sql="select * from la_agenda";
$cursor=mysql_query($sql, $dbd);
while ($un_obj=mysql_fetch_object($cursor))
{
  echo "$un_obj->nombre - $un_obj->correoe<br>";
}
```

mysql_fetch_row(cursor)

Obtiene una fila del cursor y la devuelve en forma de *array* numérico.

mysql_result(cursor, num_fila, campo)

A partir del cursor, y de la fila indicada, se obtiene el campo especificado mediante la posición numérica, el nombre o el formato de puntos: `nombre_tabla.nombre_campo`

```
$cursor=mysql_query("Select * from la_agenda", $dbd);
// obtengo el cuarto campo de la tercera fila
$campo=mysql_result($cursor, 2, 3);
```

mysql_data_seek(cursor, num_fila)

Mueve el puntero dentro del cursor a la fila indicada.

mysql_field_seek(cursor, num_campo)

Mueve el puntero del cursor a la posición del campo especificado.

mysql_free_result(cursor)

Libera la memoria ocupada por el cursor (típicamente, el resultado de cualquier operación SQL hecha con `mysql_query()`).

mysql_db_name(cursor, fila)

Devuelve el nombre de una base de datos a partir del cursor devuelto por `mysql_list_dbs()`. Un ejemplo que imprime el nombre de todas las bases de datos existentes en nuestro servidor:

```
$lista_bds = mysql_list_dbs($dbd);
$i = 0;
$cont = mysql_num_rows($lista_dbs);
while ($i < $cont) {
  echo mysql_db_name($lista_dbs, $i) . "<br>";
  $i++;
}
```

mysql_field_flags(cursor, num_campo)

A partir de un cursor y una indicación del campo dentro de aquél, obtiene los *flags* asociados al campo. Los valores que se podrán obtener son: `not_null`, `primary_key`, `unique_key`, `multiple_key`, `blob`, `unsigned`, `zerofill`, `binary`, `enum`, `auto_increment`, `timestamp`.

mysql_field_len(cursor, num_campo)

A partir de un cursor y una indicación del campo dentro de aquél, obtiene el tamaño de éste.

```
$cursor = mysql_query("select * from la_agenda", $dbd);
echo "tamaño 1er campo:", mysql_field_len($cursor, 0), "<br>";
echo "tamaño 2o campo:", mysql_field_len($cursor, 1), "<br>";
```

mysql_field_name(cursor, num_campo)

Devuelve el nombre del campo especificado en un resultado.

```
$cursor = mysql_query("select * from la_agenda", $dbd);
echo "primer campo: ", mysql_field_name($cursor, 0), "<br>";
echo "segundo campo:", mysql_field_name($cursor, 1), "<br>";
```

mysql_field_table(cursor, num_campo)

A partir de un cursor y una indicación del campo dentro de aquél, devuelve el nombre de la tabla donde está el campo especificado.

mysql_field_type(cursor, num_campo)

A partir de un cursor y una indicación del campo dentro de aquél, devuelve el tipo del campo especificado. El siguiente código nos muestra un ejemplo combinado de las dos últimas funciones:

```
$cursor     = mysql_query("SELECT * FROM la_agenda");
$num_campos = mysql_num_fields($cursor);
$regs       = mysql_num_rows($cursor);


$tabla      = mysql_field_table($cursor, 0);
echo "'",$tabla,"' tiene ",$num_campos," campos y ",$regs." registros <br>";
echo "además, tiene los siguientes campos:<BR>";
$i = 0;
while ($i < $num_campos)
{
  $tipo   = mysql_field_type  ($cursor, $i);
  $nombre = mysql_field_name  ($cursor, $i);
  $tam    = mysql_field_len   ($cursor, $i);
  $flags  = mysql_field_flags ($cursor, $i);
  echo $nombre," de tipo ",$tipo,", y ",$tam," de tamaño y flags:
",$flags."<br>";
  $i++;
}
```

mysql_change_user(usr, clave [,base_datos [,descr_conexion]])

Cambia el usuario activo para la conexión actual.

mysql_get_client_info()

Devuelve la versión de MySQL.

mysql_get_host_info(descr_conexion)

A partir de un descriptor de conexión al servidor, devuelve el tipo de conexión establecida (por ejemplo, remota, vía TCP/IP, o local, vía *sockets* de Unix).

mysql_get_proto_info(descr_conexion)

Devuelve el número de versión del protocolo de conexión.

mysql_get_server_info(descr_conexion)

Devuelve la versión del servidor MySQL.

mysql_info(descr_conexion)

Devuelve información de la última consulta SQL.

mysql_client_encoding(descr_conexion)

Devuelve el nombre del conjunto de caracteres (*character set*) activo para esta conexión.

mysql_real_escape_string(cadena_caracteres)

Enmascara los caracteres especiales de una cadena de caracteres teniendo en cuenta el conjunto de caracteres activo (nota: no se enmascaran los caracteres "%", ni "_"), comillas de una cadena de caracteres anteponiendo el carácter de barra invertida (\).

mysql_escape_string(cadena_caracteres)

Enmascara las comillas de una cadena de caracteres anteponiendo el carácter de barra invertida (\).

APÉNDICE D

INTERFACES DOM, DEFINICIONES EN IDL

Para poder proporcionar una especificación independiente del lenguaje, el W3C optó por definir las interfaces de DOM utilizando el IDL (*Interface Definition Language*) definido por el OMG (*Object Management Group*).

En este anexo se muestran las definiciones en IDL de las interfaces utilizadas en el capítulo dedicado a XML.

> **NOTA:** Para obtener más información sobre DOM y tecnologías afines, se puede recurrir a la dirección http://www.w3.org/TR/DOM-Level-3-Core .

D.1 Interface Node

```
interface Node {

  // NodeType
  const unsigned short      ELEMENT_NODE                = 1;
  const unsigned short      ATTRIBUTE_NODE              = 2;
  const unsigned short      TEXT_NODE                   = 3;
  const unsigned short      CDATA_SECTION_NODE          = 4;
  const unsigned short      ENTITY_REFERENCE_NODE       = 5;
  const unsigned short      ENTITY_NODE                 = 6;
  const unsigned short      PROCESSING_INSTRUCTION_NODE = 7;
  const unsigned short      COMMENT_NODE                = 8;
  const unsigned short      DOCUMENT_NODE               = 9;
  const unsigned short      DOCUMENT_TYPE_NODE          = 10;
  const unsigned short      DOCUMENT_FRAGMENT_NODE      = 11;
  const unsigned short      NOTATION_NODE               = 12;

  readonly attribute DOMString       nodeName;
           attribute DOMString       nodeValue;
                                        // raises(DOMException) on setting
                                        // raises(DOMException) on retrieval
```

```
readonly attribute unsigned short  nodeType;
readonly attribute Node            parentNode;
readonly attribute NodeList        childNodes;
readonly attribute Node            firstChild;
readonly attribute Node            lastChild;
readonly attribute Node            previousSibling;
readonly attribute Node            nextSibling;
readonly attribute NamedNodeMap    attributes;
// Modified in DOM Level 2:
readonly attribute Document        ownerDocument;
// Modified in DOM Level 3:
Node               insertBefore(in Node newChild,
                                in Node refChild)
                                   raises(DOMException);
// Modified in DOM Level 3:
Node               replaceChild(in Node newChild,
                                in Node oldChild)
                                   raises(DOMException);
// Modified in DOM Level 3:
Node               removeChild(in Node oldChild)
                                   raises(DOMException);
Node               appendChild(in Node newChild)
                                   raises(DOMException);
boolean            hasChildNodes();
Node               cloneNode(in boolean deep);
// Modified in DOM Level 2:
void               normalize();
// Introduced in DOM Level 2:
boolean            isSupported(in DOMString feature,
                               in DOMString version);
// Introduced in DOM Level 2:
readonly attribute DOMString       namespaceURI;
// Introduced in DOM Level 2:
         attribute DOMString       prefix;
                                      // raises(DOMException) on setting

// Introduced in DOM Level 2:
readonly attribute DOMString       localName;
// Introduced in DOM Level 2:
boolean            hasAttributes();
// Introduced in DOM Level 3:
readonly attribute DOMString       baseURI;

// DocumentPosition
const unsigned short       DOCUMENT_POSITION_DISCONNECTED = 0x01;
const unsigned short       DOCUMENT_POSITION_PRECEDING    = 0x02;
const unsigned short       DOCUMENT_POSITION_FOLLOWING    = 0x04;
const unsigned short       DOCUMENT_POSITION_CONTAINS     = 0x08;
const unsigned short       DOCUMENT_POSITION_IS_CONTAINED = 0x10;
const unsigned short       DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC = 0x20;

// Introduced in DOM Level 3:
unsigned short     compareDocumentPosition(in Node other)
                                      raises(DOMException);
// Introduced in DOM Level 3:
         attribute DOMString       textContent;
                                      // raises(DOMException) on setting
                                      // raises(DOMException) on retrieval

// Introduced in DOM Level 3:
boolean            isSameNode(in Node other);
```

```
  // Introduced in DOM Level 3:
  DOMString           lookupPrefix(in DOMString namespaceURI);
  // Introduced in DOM Level 3:
  boolean             isDefaultNamespace(in DOMString namespaceURI);
  // Introduced in DOM Level 3:
  DOMString           lookupNamespaceURI(in DOMString prefix);
  // Introduced in DOM Level 3:
  boolean             isEqualNode(in Node arg);
  // Introduced in DOM Level 3:
  Node                getFeature(in DOMString feature,
                                 in DOMString version);
  // Introduced in DOM Level 3:
  DOMUserData         setUserData(in DOMString key,
                                  in DOMUserData data,
                                  in UserDataHandler handler);
  // Introduced in DOM Level 3:
  DOMUserData         getUserData(in DOMString key);
};
```

D.2 Interface Document

```
interface Document : Node {
  // Modified in DOM Level 3:
  readonly attribute DocumentType    doctype;
  readonly attribute DOMImplementation implementation;
  readonly attribute Element         documentElement;
  Element             createElement(in DOMString tagName)
                                    raises(DOMException);
  DocumentFragment    createDocumentFragment();
  Text                createTextNode(in DOMString data);
  Comment             createComment(in DOMString data);
  CDATASection        createCDATASection(in DOMString data)
                                    raises(DOMException);
  ProcessingInstruction createProcessingInstruction(in DOMString target,
                                                     in DOMString data)
                                    raises(DOMException);
  Attr                createAttribute(in DOMString name)
                                    raises(DOMException);
  EntityReference     createEntityReference(in DOMString name)
                                    raises(DOMException);
  NodeList            getElementsByTagName(in DOMString tagname);
  // Introduced in DOM Level 2:
  Node                importNode(in Node importedNode,
                                 in boolean deep)
                                    raises(DOMException);
  // Introduced in DOM Level 2:
  Element             createElementNS(in DOMString namespaceURI,
                                      in DOMString qualifiedName)
                                    raises(DOMException);
  // Introduced in DOM Level 2:
  Attr                createAttributeNS(in DOMString namespaceURI,
                                        in DOMString qualifiedName)
                                    raises(DOMException);
  // Introduced in DOM Level 2:
  NodeList            getElementsByTagNameNS(in DOMString namespaceURI,
                                             in DOMString localName);
  // Introduced in DOM Level 2:
  Element             getElementById(in DOMString elementId);
  // Introduced in DOM Level 3:
        attribute DOMString       actualEncoding;
```

```
  // Introduced in DOM Level 3:
           attribute DOMString       encoding;
  // Introduced in DOM Level 3:
           attribute boolean         standalone;
  // Introduced in DOM Level 3:
           attribute DOMString       version;
                                       // raises(DOMException) on setting

  // Introduced in DOM Level 3:
           attribute boolean         strictErrorChecking;
  // Introduced in DOM Level 3:
           attribute DOMString       documentURI;
  // Introduced in DOM Level 3:
  Node                adoptNode(in Node source)
                                       raises(DOMException);
  // Introduced in DOM Level 3:
  readonly attribute DOMConfiguration config;
  // Introduced in DOM Level 3:
  void                normalizeDocument();
  // Introduced in DOM Level 3:
  Node                renameNode(in Node n,
                                 in DOMString namespaceURI,
                                 in DOMString qualifiedName)
                                       raises(DOMException);
};
```

D.3 Interface Element

```
interface Element : Node {
  readonly attribute DOMString       tagName;
  DOMString          getAttribute(in DOMString name);
  void               setAttribute(in DOMString name,
                                  in DOMString value)
                                       raises(DOMException);
  void               removeAttribute(in DOMString name)
                                       raises(DOMException);
  Attr               getAttributeNode(in DOMString name);
  Attr               setAttributeNode(in Attr newAttr)
                                       raises(DOMException);
  Attr               removeAttributeNode(in Attr oldAttr)
                                       raises(DOMException);
  NodeList           getElementsByTagName(in DOMString name);
  // Introduced in DOM Level 2:
  DOMString          getAttributeNS(in DOMString namespaceURI,
                                    in DOMString localName)
                                       raises(DOMException);
  // Introduced in DOM Level 2:
  void               setAttributeNS(in DOMString namespaceURI,
                                    in DOMString qualifiedName,
                                    in DOMString value)
                                       raises(DOMException);
  // Introduced in DOM Level 2:
  void               removeAttributeNS(in DOMString namespaceURI,
                                       in DOMString localName)
                                       raises(DOMException);
  // Introduced in DOM Level 2:
  Attr               getAttributeNodeNS(in DOMString namespaceURI,
                                        in DOMString localName)
                                       raises(DOMException);
  // Introduced in DOM Level 2:
  Attr               setAttributeNodeNS(in Attr newAttr)
```

```
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  NodeList          getElementsByTagNameNS(in DOMString namespaceURI,
                                           in DOMString localName)
                                        raises(DOMException);
  // Introduced in DOM Level 2:
  boolean           hasAttribute(in DOMString name);
  // Introduced in DOM Level 2:
  boolean           hasAttributeNS(in DOMString namespaceURI,
                                   in DOMString localName)
                                        raises(DOMException);
  // Introduced in DOM Level 3:
  readonly attribute TypeInfo         schemaTypeInfo;
  // Introduced in DOM Level 3:
  void              setIdAttribute(in DOMString name,
                                   in boolean isId)
                                        raises(DOMException);
  // Introduced in DOM Level 3:
  void              setIdAttributeNS(in DOMString namespaceURI,
                                     in DOMString localName,
                                     in boolean isId)
                                        raises(DOMException);
  // Introduced in DOM Level 3:
  void              setIdAttributeNode(in Attr idAttr,
                                        in boolean isId)
                                        raises(DOMException);
};
```

D.4 Interface Attr

```
interface Attr : Node {
  readonly attribute DOMString        name;
  readonly attribute boolean          specified;
           attribute DOMString        value;
                                        // raises(DOMException) on setting

  // Introduced in DOM Level 2:
  readonly attribute Element          ownerElement;
  // Introduced in DOM Level 3:
  readonly attribute TypeInfo         schemaTypeInfo;
  // Introduced in DOM Level 3:
  boolean           isId();
};
```

D.5 Interface ProcessingInstruction

```
interface ProcessingInstruction : Node {
  readonly attribute DOMString        target;
           attribute DOMString        data;
                                        // raises(DOMException) on setting

};
```

D.6 Interface characterData

```
interface CharacterData : Node {
           attribute DOMString        data;
                                        // raises(DOMException) on setting
                                        // raises(DOMException) on retrieval
```

```
  readonly attribute unsigned long   length;
  DOMString          substringData(in unsigned long offset,
                                   in unsigned long count)
                                        raises(DOMException);
  void               appendData(in DOMString arg)
                                        raises(DOMException);
  void               insertData(in unsigned long offset,
                                in DOMString arg)
                                        raises(DOMException);
  void               deleteData(in unsigned long offset,
                                in unsigned long count)
                                        raises(DOMException);
  void               replaceData(in unsigned long offset,
                                 in unsigned long count,
                                 in DOMString arg)
                                        raises(DOMException);
};
```

D.7 Interface Text

```
interface Text : CharacterData {
  Text               splitText(in unsigned long offset)
                                        raises(DOMException);
  // Introduced in DOM Level 3:
  boolean            isWhitespaceInElementContent();
  // Introduced in DOM Level 3:
  readonly attribute DOMString       wholeText;
  // Introduced in DOM Level 3:
  Text               replaceWholeText(in DOMString content)
                                        raises(DOMException);
};
```

D.8 Interface CDATASection

```
interface CDATASection : Text {
};
```

D.9 Interface Commnet

```
interface Comment : CharacterData {
};
```

D.10 Interface Entity

```
interface Entity : Node {
  readonly attribute DOMString       publicId;
  readonly attribute DOMString       systemId;
  readonly attribute DOMString       notationName;
  // Introduced in DOM Level 3:
           attribute DOMString       actualEncoding;
  // Introduced in DOM Level 3:
           attribute DOMString       encoding;
  // Introduced in DOM Level 3:
           attribute DOMString       version;
};
```

D.11 Interface EntityReference

```
interface EntityReference : Node {
};
```

D.12 Interface Notation

```
interface Notation : Node {
  readonly attribute DOMString       publicId;
  readonly attribute DOMString       systemId;
};
```

D.13 Interface DocumentType

```
interface DocumentType : Node {
  readonly attribute DOMString       name;
  readonly attribute NamedNodeMap    entities;
  readonly attribute NamedNodeMap    notations;
  // Introduced in DOM Level 2:
  readonly attribute DOMString       publicId;
  // Introduced in DOM Level 2:
  readonly attribute DOMString       systemId;
  // Introduced in DOM Level 2:
  readonly attribute DOMString       internalSubset;
};
```

D.14 Interface DocumentFragment

```
interface DocumentFragment : Node {
};
```

D.15 Interface nodeList

```
interface NodeList {
  Node               item(in unsigned long index);
  readonly attribute unsigned long   length;
};
```

D.16 Interface NamedNodeMap

```
interface NamedNodeMap {
  Node               getNamedItem(in DOMString name);
  Node               setNamedItem(in Node arg)
                                    raises(DOMException);
  Node               removeNamedItem(in DOMString name)
                                    raises(DOMException);
  Node               item(in unsigned long index);
  readonly attribute unsigned long   length;
  // Introduced in DOM Level 2:
  Node               getNamedItemNS(in DOMString namespaceURI,
                                    in DOMString localName)
                                     raises(DOMException);
  // Introduced in DOM Level 2:
  Node               setNamedItemNS(in Node arg)
                                     raises(DOMException);
  // Introduced in DOM Level 2:
  Node               removeNamedItemNS(in DOMString namespaceURI,
                                     in DOMString localName)
                                     raises(DOMException);
};
```

ÍNDICE ALFABÉTICO

D

E

F

G

H

I

K

L

M

N

S

PHP5
a través de ejemplos

Usuarios de:	Nivel del libro:
Lenguajes de programación	✓ Iniciación ✓ Medio ✓ Avanzado

Las páginas dinámicas de servidor, que son capaces de responder de manera inteligente a las demandas del cliente y permiten la automatización de gran cantidad de tareas, son la base del actual desarrollo de sitios Web. Este libro nos descubre las posibilidades del lenguaje PHP a la hora de diseñar dichas páginas.

Aunque existen multitud de lenguajes y entornos de desarrollo concebidos para Internet, PHP es uno de los más empleados. De hecho, los datos de Netcraft de agosto de 2004 (http://www.netcraf.com) estiman que son casi 17 millones los dominios que emplean este lenguaje. Son muchas las características que contribuyen a este éxito: entre otras, que es un software de libre distribución y multiplataforma (existen versiones para U*ix, Win32, Mac OS X, etc.) que sigue la filosofía Open Source. Además se ha convertido en un complemento ideal para el tándem Linux-Apache en el desarrollo de sitios Web. Pero son sin duda la sencillez de su uso y las posibilidades avanzadas que proporciona (comunicación con bases de datos, comunicación vía sockets, generación de gráficos, utilización de XML, etc.) las claves de su popularidad.

La versión 5 de PHP (basada en el nuevo motor Zend 2) destaca por el completo soporte que presenta para la programación orientada a objetos, la nueva librería de XML (libxml2), el soporte nativo para el sistema gestor de base de datos SQLite, la ampliación de MySQL y las mejoras en la gestión de la cadena de caracteres.

Información proporcionada en el libro:

- Instalación y configuración (en sistemas U*ix y Windows) del servidor Web Apache, de las bases de datos MySQL y SQLite y del intérprete de PHP.
- Fundamentos y estructuras básicas del lenguaje.
- Modelo de objetos en PHP.
- Utilización de formularios, *cookies* y sesiones.
- Funciones asociadas al trabajo con archivos y directorios.
- Funciones asociadas al trabajo con bases de datos (MySQL y SQLite)
- Trabajo con procesadores XML: SimpleXML, SAX y DOM.
- Desarrollo de una aplicación de *Webmail*.

ISBN 970-15-1083-6

Incluye CD-ROM

Alfaomega Grupo Editor