Introduzione a PHP

Rielaborato con LaTeX 2ε da Andrea Rossetti <andy.ros@tin.it>

20 marzo 2002

Indice

Capitolo 1

PHP: Sintassi e comandi

Abstract

In questo tutorial spiegherò come usare il linguaggio PHP, il tool open-source usato da moltissimi siti web, per costruire semplici pagine dinamiche connettendosi ai database server mSQL e MySQL. Costruiremo passo passo una applicazione web basata su database completa: un database di bookmark.

La maggior parte dei tutorial su PHP non spiegano in modo esauriente il linguaggio e si limitano a presentare qualche semplice applicazione o qualche frammento di codice. Lo scopo qui è diverso: capire come usare il linguaggio e mettervi in grado di sviluppare le vostre applicazioni e avere delle buoni basi per approfondirne poi da soli le funzionalità più avanzate.

Il codice sorgente completo dell'applicazione sia in versione mSQL che per MySQL, più altri esempi di codice presentati durante il corso potete trovarlo nello zip allegato a questo tutorial.

Il tutorial è diviso in capitoli o lezioni. Potete leggere ad es. un capitolo al giorno. È consigliabile la lettura in ordine di tutti i capitoli. Il linguaggio utilizzato è simile a quello con cui normalmente ci si esprime a voce; personalmente ho in odio le riduzioni delle spiegazioni in forma scritta o l'utilizzo di un linguaggio troppo "accademico", o la pratica senza teoria e anche la teoria senza pratica. Spero quindi di essere il più possibile chiaro, anche se non esauriente. Se riscontrare degli errori che mi sono sfuggiti potete segnalarmelo via e-mail.

Per poter seguire il corso occorre installare un server web dotato di interprete PHP e un database server a scelta tra mSQL e MySQL (oppure entrambi). Occorre inoltre avere installato un qualsiasi browser web come Lynx o Netscape. Non occorre essere connessi permanentemente ad una rete o spendere molti soldi per procurarvi questo software o possedere un com-

puter particolarmente potente: alcune delle principali distribuzioni di Linux includono il web server Apache, MySQL e PHP, che sono tutti programmi gratuiti in questo ambiente.

Infatti se avete un computer dotato di poca memoria (es. soli 16MB), in Linux o altri Unix potete scegliere di lavorare in modo testo senza caricare alcuna interfaccia grafica, usando il browser testuale Lynx.

Spiegherò alcune cose riguardo l'uso di mSQL e MySQL, ma è impossibile per me (e comunque sarebbe fuori luogo) coprire tutti gli argomenti delle rispettive documentazioni, quindi fate riferimento a queste in caso di dubbi. Il linguaggio PHP invece sarà trattato con sufficiente dettaglio in tutti i suoi aspetti più generali. Non avrò invece la pretesa di trattare tutte le funzioni di libreria disponibili, che sono moltissime e che io stesso sto imparando a conoscere man mano con l'esperienza.

1.1 Prerequisiti

- conoscenze di base generali sulla programmazione (soprattutto in C)
- conoscenze di base sul web e l'html
- conoscenze di base sulla Common Gateway Interface (CGI)
- conoscenze di base di un sistema di basi di dati supportato da PHP

Sarebbe fuori luogo per me coprire qui tutti questi argomenti in modo esauriente. Cercherò comunque di essere piuttosto chiaro e preciso nelle spiegazioni per facilitare la comprensione anche ai principianti. Per quelli che partono da zero, ecco alcuni link consigliati.

Se non avete aluna conoscenza sulle CGI, ossia non avete mai scritto nessuna semplice applicazione CGI, consiglio di leggere il paragrafo *Wget e gli script* del mio manuale sul web spider Wget nel mio sito a:

http://king.rett.polimi.it/~ant/linux/wgettut[1]

In quel tutorial, interessante di per sé se non conoscete wget, c'è (nel paragrafo indicato) una breve introduzione al mondo del web scripting, con qualche esempio scritto nel linguaggio della shell bash.

Altro testo consigliato è il tutorial intitolato "Protocollo HTTP, interfaccia CGI e linguaggio Perl" nella homepage di Marco Liverani:

http://www.isinet.it/~marco

[1] Se trasferisco il sito, potete trovarlo sempre andando a: http://go.to/ninuzzo

Articoli interessanti si trovano anche nella homepage di Francesco Munaretto:

http://www.venis.it/francesco

Per quanto riguarda le conoscenze relative ad un DBMS (Data Base Management System, Sistema per la Gestione di Basi di Dati), esistono molti tutorial su internet. Eccone alcuni molto ben fatti:

1. questo sito include un tutorial e un interprete SQL utilizzabile online per fare pratica:
   http://www.sqlcourse.com

2. TOMO X, parte xlv DBMS e SQL di Appunti di Linux, di Daniele Giacomini. Consigliata almeno la lettura di tutto il capitolo 194 "Introduzione ai DBMS"
   http://www.pluto.linux.ildp/AppuntiLinux/

3. ottimo tutorial di James Hoffman, disponibile anche in .pdf:
   http://w3.one.net/~jhoffman

Occorre che conosciate almeno la sintassi di base delle istruzioni più semplici del linguaggio SQL (CREATE, INSERT, UPDATE, DELETE). Nel tutorial spiegherò solo gli aspetti pratici su come connettersi ad un database di mSQL o MySQL da PHP e manipolarlo con quelle query.

mSQL o Mini SQL è un DBMS molto leggero e semplice da usare, ma NON free per uso commerciale, prodotto da Hughes Tecnologies:

http://www.hughes.com.au

MySQL, prodotto da TcX DataKonsult AB, è distribuito secondo una licenza che vi permette di usarlo per scopi commerciali senza pagare una lira sotto Unix (ma sotto Windows invece occorre pagare, dopo 30 giorni di valutazione). Per maggiori informazioni:

http://www.mysql.com

Comunque niente paura, dirò anche come creare il DB vuoto con entrambi questi tool (cosa che di solito dovete fare manualmente prima di poter utilizzare il programma che creeremo, poiché questo non potrebbe averne il permesso).

Questo tutorial spiega la sintassi del linguaggio PHP e alcune funzioni. Non rappresenta in alcun modo un riferimento preciso e dettagliato (la documentazione che viene con PHP assolve invece questo compito), ma piuttosto un tutorial introduttivo.

Dopo aver spiegato qualcosa sul linguaggio, vi mostrerò come creare passo passo una applicazione web realistica. Per esigenza personale, ho scelto di creare un database di link, ossia di bookmark web accessibile e amministrabile completamente via web.

Solito disclaimer: non sono responsabile delle informazioni contenute in questo tutorial, nonostante vi ho posto la massima cura. Use at you own risk.

1.2 Installazione

Il sito di principale su cui trovare tutte le informazioni su PHP è:

http://www.php.net

PHP (Personal Home Page tools, così chiamato perché l'autore, Rasmus Lerdorf, lo aveva dapprima sviluppato per uso privato per la sua homepage) è un linguaggio di scripting dal lato server (server-side) integrabile direttamente nel codice HTML.

I principali vantaggi di PHP sono:

- l'interprete PHP (o processore di hypertesto PHP) può essere integrato facilmente nel web server Apache come modulo ed è piuttosto leggero, in confronto ad altri tool;

- è disponibile gratuitamente (i dettagli della licenza li trovate sul sito, oppure nel file LICENSE della distribuzione in sorgente);

- funziona sui sistemi UNIX (Linux compreso) ed anche sui sistemi Windows, permettendovi così di portare facilmente gli script da un ambiente all'altro;

- è già fornito di tutte le librerie per accesso ai database più comuni e di tutte le funzioni per la gestione dei cookies, delle sessioni di navigazione, per l'utilizzo di protocolli di rete, per la generazione dinamica delle immagini e di tutto quello di cui avete bisogno per programmare pagine web dinamiche anche complesse con poca fatica;

- facilita il debugging: fa uscire gli errori direttamente sulle pagine web e/o nell'errorlog, indicando chiaramente il tipo di errore, il percorso completo dello script che ha generato l'errore e il numero di riga su cui si è fermato il parser;
- può essere utilizzato anche come linguaggio di scripting al di fuori del web, ad es. per produrre script destinati ad essere chiamati dalla shell. Basta compilarne la versione CGI.

Per queste ragioni, sempre più siti scelgono PHP come tecnologia di scripting. PHP è la tecnologia alternativa ad ASP (Active Server Pages) di Microsoft con la differenza che mettere su un server web con Linux, Apache, MySQL (o PostgreSQL) e PHP non costa nulla in termini di software.

Con PHP si possono fare non solo manipolazione di database tramite interfaccia web e report (anche grafici) di dati di un database, ma anche cose divertenti, come sondaggi online, forum, chat, giochi, contatori.

L'installazione e la configurazione di PHP, di Apache, di mSQL o MySQL è un argomento complesso ed al di fuori degli scopi di questo tutorial. Ogni pacchetto contiene la documentazione e le istruzioni per l'installazione e consiglio caldamente di prendersi il tempo e la pazienza di consultarla. Tuttavia per darvi un'idea di quello che occorre fare, e per aiutarvi ad iniziare ad usare questi programmi con poca fatica, ho preparato una lista di tutti i comandi che ho usato per installare questi tool sulla mia linuxbox casalinga, arricchita di alcune note. Esistono port di tutti questi tool (PHP, Apache, mSQL e MySQL) anche per i sistemi Windows 95 o NT, che funzionano abbastanza bene; in questo tutorial tuttavia faremo riferimento soprattutto a sistemi Unix, dove PHP è maggiormente diffuso e non possiede alcuna limitazione delle sue funzionalità[2].

Se avete problemi di installazione, consiglio di leggere le FAQ:
http://www.php.net/FAQ.php

per vedere se è già stata data una risposta al vostro problema. C'è anche una mailing list con archivi nel sito
http://www.php.net.

Se invece avete un amministratore di sistema che fa tutto per voi (se siete così fortunati), allora potete passare subito al paragrafo successivo.

1.3 Primo programma

Come tradizione universale, il primo programma che un principiante deve scrivere è Hello World. Questo banale programma vi permetterà di verificare

[2]V. file install.txt

la correttezza dell'installazione di Apache e PHP:

```
<HTML>
 <HEAD>
  <TITLE>Hello World!</TITLE>
 </HEAD>
<BODY>
 <?php echo "Hello World!"; ?>
</BODY>
</HTML>
```

Salvate il codice nel file `hello.php` da qualche parte accessibile via web e poi con un qualunque browser (lynx o netscape sono popolari in Unix), andate all'URL corrispondente, che sarà ovviamente del tipo:

http://hostname/percorso/prova.php

Nel mio caso, la Document Root di Apache è impostata a:

/usr/local/apache/htdocs/

ed ho messo `prova.php` in quella directory. Quindi da web si vede come:

http://localhost/prova.php

Se il vostro computer non è connesso a nessuna rete, oppure non avete ancora impostato un nome di host, allora mettere localhost al posto di hostname.

In caso di errore: ammesso che l'installazione di php sia corretta, ricordatevi anche che nei sistemi Unix ci sono le permission sui file e quindi occorre assicurarsi che l'utente sotto cui gira il server web possa leggere il file `prova.php`. Ricordo che il comando per cambiare i permessi è `chmod(1)`, quello per cambiare possessore e gruppo di un file è `chown(1)`.

Dovreste vedere una pagina di questo tipo (con lynx):

```
                                                   Hello World!

Hello World!
```

In alto a destra c'è il titolo della pagina, mentre la prima riga consiste in quello che ha prodotto lo script: la stampa della stringa "Hello World!".

Il vantaggio sintattico di usare PHP per la programmazione CGI, rispetto a linguaggi classici come il C/C++ o il Perl è che non bisogna utilizzare una miriade di istruzioni di stampa (`print`, `printf`, `puts`, `cout«` e simili) per fare l'output di codice HTML. Il processore PHP riporta al server web Apache tutto il codice al di fuori dei tag `<?php, ?>` esattamente come lo scrivete, mentre le parti che si trovano tra i tag `<?php, ?>` vengono interpretate e poi sostituite con l'eventuale output che generano, se ci sono delle istruzioni di stampa.

In questo senso si dice che PHP è "embedded in HTML", proprio come JavaScript, con la differenza però che JavaScript è un linguaggio lato client, mentre PHP viene elaborato dal lato server. Un'altra differenza è che, essendo embedded e dovendo essere interpretato dal client, JavaScript è visibile all'utente (basta guardare i sorgenti della pagina html), mentre PHP no: se visualizzate il sorgente della pagina `prova.php` ecco cosa vedrete[3]:

```
<HTML>
 <HEAD>
  <TITLE>Hello World!</TITLE>
 </HEAD>
<BODY>
 Hello World!</BODY>
</HTML>
```

Come vedete l'utente non ha modo di conoscere il vostro codice PHP, poiché non ha modo di impedire ad Apache di non processare un file `.php` col processore PHP.

Potete configurare Apache in modo che tutti i file html siano processati con PHP e non solo quelli con estensione PHP. Se fate questo tuttavia, probabilmente introducete un carico maggiore sul server, a meno che non servite tutte pagine PHP; generalmente conviene far sì che solo le pagine con estensione `.php` siano passate tramite il processore PHP, mentre le pagine statiche no; il semplice passaggio di una pagina statica per il processore PHP è innocuo: esso non trova alcuna sezione PHP da interpretare e ritorna la pagina esattamente come gli viene trasmessa, con un perfetto spreco di cicli di CPU.

Naturalmente al posto di `.php` potete scegliere una estensione qualunque (`.cgi`, `.script` o quello che vi pare!): configurando il server web in questo modo l'utente non ha modo di sapere se state usando php o qualche altro tool. Anzi il solo fatto che un sito abbia pagine che finiscono in `.php` non implica necessariamejnte che sta usando php! Meglio comunque usare `.php`

[3]Per visualizzare il sorgente di una pagina, con Netscape per Unix andate nel menù View e scegliete Page Source oppure premete <Alt>U e si aprirà una nuova finestra, con lynx il tasto \ fa passare al sorgente e viceversa.

come estensione per le pagine `.php`, non c'è ragione di nascondere l'uso di php, visto che è gratuito, no? ;)

È opportuno indentare opportunamente il codice html per migliorarne la leggibilità e questo vale ancor di più quando ci sono degli script dentro. Nel caso di Hello Word, è meglio aggiungere un accapo:

```
<?php echo "Hello World!\n"; ?>
```

Notate quindi che ora viene prodotto un accapo nell'html generato:

```
<HTML>
 <HEAD>
  <TITLE>Hello World!</TITLE>
 </HEAD>
<BODY>
 Hello World!
</BODY>
</HTML>
```

Se non volete usare la sequenza di escape \n, in questo caso, basta aggiungere una riga vuota dopo la sezione PHP per ottenere lo stesso effetto:

```
<BODY>
 <?php echo "Hello World!"; ?>

</BODY>
```

È comune adottare anche questi stili di indentazione, per distinguere meglio il codice PHP da quello HTML:

```
<BODY>

 <?php echo "Hello World!"; ?>

</BODY>
```

```
<BODY>

 <?php
echo "Hello World!";
 ?>

</BODY>
```

```
<BODY>
 <?php

echo "Hello World!";

 ?>

</BODY>
```

Quest'ultima forma è per me preferibile poiché non introduce nel codice HTML finelinea in più, ed evidenzia bene l'inizio e la fine del codice PHP rispetto all'HTML, ma in questo campo non ci sono regole, solo preferenze.

Avrete quindi capito che per programmare in PHP basta un browser web e un normale editor di testo, quale `vi(m)` o `emacs`, non è necessario acquistare costosi e pesanti sistemi di composizione visuale del codice che promettono quello che poi non possono essere in grado di mantenere. Nei casi più difficili infatti occorre sempre conoscere il linguaggio e mettersi a scrivere il codice e codificare algoritmi. Non esistono scappatoie: nessun sistema visuale può generare tutto il codice per voi. Se così fosse tutti i programmatori perderebbero il lavoro. Gli ambienti integrati possono essere utili per riunire in un unico posto tutti gli strumenti che il programmatore deve usare; ma sono solo un'interfaccia: non c'è niente di magico. A maggior ragione occorre sapere bene quello che si sta facendo.

Il sorgente php, oltre che essere racchiuso tra i tag:

```
<?php, ?>
```

può anche essere racchiuso tra i tag più brevi:

```
<?, ?>
```

oppure tramite il tag `<script>` standard di html:

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

Poiché però i tag `<?, ?>` sono usati da XML, non potete usarli se volete usare XML in combinazione con PHP pena ambiguità ed errore e dovete quindi editare il file di configurazione

```
/usr/local/lib/php.ini
```

inserendo Off al posto di On nella riga:

```
; allow the <? tag.  otherwise, only <?php and <script> tags are
; recognized.
short_open_tag  =       On
```

oppure, passate l'opzione `-disable-short-tags` al configure nella fase di installazione. Personalmente preferisco usare `<?php, ?>` oppure `<%, %>` che non danno problemi nel caso di futura conversione di un documento HTML in XML (non si sa mai).

Dalla versione 3.0.4, PHP supporta anche i tag in stile ASP, che sono:

```
<%, %>
<%=, %>
```

quindi avremmo potuto scrivere:

```
<% echo "Hello World!"; %>
```

oppure equivalentemente:

```
<%= "Hello World"; %>
```

molto comodo il secondo tipo. Notare che le virgolette doppie sono necessarie per quasi tutte le stringhe. Potrebbero non esserlo nel caso di stringa composta da una sola parola, ma, se questa corrisponde ad una parola chiave del php, il parser si inganna, quindi meglio usarle SEMPRE.

Il punto e virgola finale, poiché c'è una sola istruzione, non è necessaria, ma io preferisco usarlo sempre. Più precisamente il ; finale è opzionale dopo l'ultima istruzione, ossia prima di `?>` e simili. Non è obbligatorio lo spazio dopo `<?`, `<%`, `<%=` o prima di `%>`, `?>`, ma io preferisco usarlo sempre, perché penso migliori la leggibilità.

Comunque per poter usare i tag stile asp occorre abilitarne il riconoscimento in `php.ini`, mettendo un `On` al posto di `Off` nella seguente riga:

```
asp_tags                =       Off ; allow ASP-style <% %> tags
```

Se avete compilato PHP come modulo (come indicato dalle mie note di installazione), dopo una modifica al file di configurazione di php, dovete riavviare apache per renderla effettiva, poiché per ragioni di efficienza il file di configurazione di php viene letto solo una volta quando apache si avvia e i suoi settings vengono mantenuti in memoria, anziché essere riletto ogni volta che si accede ad una pagina php come con la versione CGI di PHP. Fate quindi:

```
/usr/local/apache/bin/apachectl graceful
```

oppure:

```
/usr/local/apache/bin/apachectl restart
```

Consiglio di scegliere in uno script uno stile di delimitatori ed usare sempre quello. Usare una volta `<?`, un'altra `<%` e un'altra `<script language="php">` in uno stesso script ne rende meno agevole la lettura.

La funzione echo è l'istruzione di output più semplice e più usata. Attenzione però che in realtà non è una vera e propria funzione come quelle che può definire l'utente, ma piuttosto un costrutto cablato nel linguaggio, e per ciò non si deve racchiudere la lista dei suoi argomenti tra `( )`, nel caso fossero più d'uno:

```
<% echo("Hello", " World!"); %>
```

è un errore, scrivete invece:

```
<% echo "Hello", " World!"; %>
```

Andando a capo nel mezzo di una stringa produce lo stesso effetto di inserire il carattere newline `\n` nella stringa, quindi

```
<PRE>
<%
echo "Is There
Anybody
Out There?"
%>
</PRE>
```

è lo stesso di:

```
<PRE>
<% echo "Is There\nAnybody\nOut There?" %>
</PRE>
```

Quando usate i tag `<%=, %>`, potete inserire più espressioni-istruzioni, separandole tramite `";"`, ma solo la prima espressione valutata sarà implicitamente stampata:

```
<%= php; " is great"; %>
```

produce in output solo:

```
php
```

invece

```
<%= php; echo " is great"; %>
```

produce:

```
php is great
```

mentre

```
<%= "alpha", 77 %>
```

produce:

```
alpha77
```

come osservato prima qui l'uso delle " attorno alla stringa "php" sarebbe consigliabile.

L'ultimo esempio mette in evidenza che:

```
<%= ... %>
```

è una forma sintattica abbreviata equivalente a:

```
<%echo ... %>
```

e quindi potete avere più espressioni argomento separate da virgola che vengono tutte valutate e stampate in sequenza, esattamente come quando fornite parametri al comando echo. Anche qui è un errore racchiudere i parametri tra `( )`.

1.4 Commenti

Si possono inserire commenti multilinea nel codice racchiudendoli tra `/*` e `*/` come in C, oppure commenti fino alla fine della linea (o del codice PHP, a seconda di quello che viene prima), iniziandoli con `//` in stile C++, oppure con `#` come nei linguaggi delle shell di Unix (o come in Perl). Ecco degli esempi dei tre stili di commento possibili:

```
<%
   /*
      commento multilinea;
      i commenti multilinea non si possono annidare
   */

  echo "linea di codice con commento";  // commento
  echo "commento in stile Perl";        # commento
%>
```

Queste quattro sezioni di codice php che presentano un commento di linea che termina con la sezione stessa sono tutte corrette:

```
<PRE>
<?php echo "commento1"; # commento1 ?>

<?php echo "commento2"; // commento2 ?>

<? echo "commento3"; # commento3 ?>

<? echo "commento4"; // commento4 ?>
</PRE>
```

Di queste tre invece solo la prima è vista correttamente dal parser:

```
<PRE>
<% echo "commento1"; /* commento1 */ %>

<% echo "commento1"; # commento1 %>

<% echo "commento2"; // commento2 %>
</PRE>
```

Se si va accapo prima di `%>` anche le ultime due sono corrette. Non c'è motivo per cui le ultime due non dovrebbero essere accettate. Si tratta di una limitazione (dovuta a dimenticanza) nel codice del parser che gestisce i commenti fino alla fine delle linee (che non tiene conto di `%>` e lo considera parte del commento). Probabilmente sarà corretta nelle prossime versioni, anzi sicuramente sarà corretta, perché il sottoscritto l'ha segnalata agli sviluppatori di PHP che l'hanno corretta dopo una settimana nella versione CVS attualmente in sviluppo[4].

Per ora il workaround consiste nell'andare accapo (oppure mettere mano al parser PHP, generato tramite bison, e ricompilare, ma non è consigliabile, poiché rende il codice non portabile – ovviamente su altri interpreti PHP non patchati il baco permane). Questo per dire che se trovate bachi in PHP non dovete attendere anni finché vengano corretti. Vantaggi evidenti dell'open source, che è in grado di produrre software maggiormente stabile rispetto a quello commerciale.

1.5 Variabili e tipi numerici

Come accade in altri linguaggi di scripting e interpretativi e al contrario del C (che è prevalentemente compilato), in PHP non si dichiarano le variabili (per lo meno quelle "normali") e per rendere la vita facile al parser tutti i nomi di variabile vanno prefissati con un simbolo, che è `$`. Poiché non esiste una dichiarazione di variabile, come si fa allora a stabilire il tipo di una variabile? Alcuni linguaggi usano simboli diversi davanti al nome della variabile per indicarne ogni volta il tipo. In PHP invece si usa sempre solo small`$` davanti ai nomi delle variabili e il tipo di una variabile viene determinato automaticamente dall'interprete a seconda del contesto in cui la variabile viene usata.

Questo in particolare significa che se assegnate un intero ad una variabile essa sarà intera, se gli assegnate una stringa sarà una stringa, ecc...:

```
<%
   /* output:

integer<br>
string<br>
string
   */

   $num = 56;
   echo gettype($num), "<br>\n";
```

[4] Informazioni all'url http://bugs.php.net/?id=8203).

```
    $str = "cinquantasei";
    echo gettype($str), "<br>\n";

    $num = $str;
    echo gettype($num);
%>
```

Se avevate assegnato un intero e poi assegnate una stringa, il tipo della variabile cambierà e diventerà una stringa e viceversa, e così anche per tutti gli altri tipi. Questa flessibilità è tipica dei linguaggi interpretativi.

PHP è case sensitive nei nomi di variabile, come il C e quindi la variabile `$FOO` è diversa da `$Foo` o da `$FoO`. Tuttavia PHP non fa differenza di case tra le parole riservate e i nomi di funzione: `while`, `WHILE`, `While`, `WhIlE`, ecc. sono tutte la stessa parola riservata, il che vi permette di scegliere liberamente lo stile che preferite. Qualunque sia lo stile che scegliate, consiglio tuttavia di essere consistenti.

Un nome di variabile valido deve iniziare con una lettera o con il carattere di underscore e può comprendere altre lettere e cifre, il carattere di underscore. I caratteri ascii dal 127 al 255 sono considerati lettere, quindi anche le lettere accentate dell'italiano possono fare parte di nomi di variabile. Comunque questi codici di caratteri non sono standard.

L'interprete mantiene una tabella con nomi, valori e tipi delle variabili e potete chiedergli il tipo di una variabile usando la funzione:

string **gettype** (mixed *var*)

questa ritorna una stringa identificativa del tipo come nell'esempio precedente ed accetta per argomento una variabile di tipo qualsiasi.

Le costanti intere possono essere indicate, oltre che nella normale base 10, anche in ottale o in esadecimale, secondo una sintassi identica a quella del C. Ecco degli esempi tratti dal manuale:

```
$a = 1234; # numero decimale
$a = -123; # un numero negativo
$a = 0123; # numero ottale (equivalente al decimale 83)
$a = 0x12; # numero esadecimale (equivalente a 18 decimale)
```

Un intero ha sempre il segno; la dimensione è dipendente dal sistema utilizzato; di solito il range per gli interi è quello che si può rappresentare in complemento a 2 con registri di macchina a 32 bit:

[-2^31=-2147483648, 2^31-1=2147483647]

Le variabili in virgola mobile hanno un tipo detto "double" (così come ritornato da gettype) e si possono indicare nelle forme usuali (notazione a punto fisso ed esponenziale):

```
$a = 1.234;
$a = 1.2e3;
$a = 1.; // $a vale 1.0
$a = .5; // $a vale 0.5
```

nell'ultimo esempio ho scritto "`1.`" invece di "`1`" per forzare il tipo a double (altrimenti l'interprete avrebbe stabilito il tipo di `$a` come integer). Avrei anche potuto scrivere `1.0` o `1.00`, ecc.

Un modo più generale di forzare il tipo di una espressione effettuando una conversione al tipo voluto, se necessario, consiste nell'usare l'operatore di casting, che come in C ha la forma:

`(`*tipo_ voluto*`)` `(`*espressione di cui forzare il tipo*`)`

ad es. per ottenere la parte intera di un numero double `$a`, scriveremo:

```
$int_a = (integer)$a;
```

la keyword "integer" può essere abbreviata in "int":

```
$int_a = (int)$a;
```

Supponendo sia stato fatto `$int_a = 10;`, le istruzioni:

```
$a = $int\_a;
$a = (double)$int\_a;
```

sono leggermente diverse: la prima assegna il tipo integer e il valore 10 ad `$a`. La seconda gli assegna ancora il valore 10, ma il tipo è double. Sinonimi di (double) sono (real), (float). Personalmente preferisco usare sempre solo (double).

Esiste anche la funzione inversa di gettype:

int **settype** (string *var*, string *type*)

che converte la variabile di nome "var" nel tipo "type" che può essere una delle seguenti stringhe:

"boolean"
"integer"
"double"
"string"
"array"
"object"

```
$a = 10.6;
settype($a, integer); # $a adesso è un intero e vale 10
```

È possibile non racchiudere la stringa che rappresenta il nome del tipo tra `""` con la funzione built-in settype, tuttavia non sempre (non nel caso di array, poiché array è anche il nome di un altro costrutto che vedremo). La funzione settype svolge un lavoro identico alla seguente assegnazione:

```
$a = (type)$a;
```

Una variabile che viene usata senza prima essere inizializzata non contiene alcun valore, non essendo presente nella tabella dei simboli dell'interprete e, ad es., stamparla su output non produce nulla. Il suo tipo è nessuno, infatti la funzione gettype ritorna la stringa "NULL".

```
<%
   echo $var; // non stampa niente
   echo (int)$var; // stampa 0
   echo gettype($var)  // stampa NULL
%>
```

Consiglio di inizializzare le vostre variabili prima di usarle con un valore del tipo che volete assegnare a quella variabile, oppure almeno stabilitene il tipo tramite settype. settype inizializza integer e double non definiti a 0 e le stringhe a stringhe vuote:

```
<%
   settype($var, integer);
   echo $var; // stampa 0;
%>
```

Per sapere se una variabile è presente nella tabella dei simboli dell'interprete potete utilizzare la funzione:

int **isset** (mixed *var*)

la quale restituisce true se la variabile esiste, false altrimenti. La funzione:

int **unset** (mixed *var*)

libera tutta la memoria associata alla variabile indicata (se non vi sono altri riferimenti alla variabile - vedi capitolo "Riferimenti e Variabili") e la toglie dalla tabella dei simboli dell'interprete. Dopo aver fatto l'unset di una variabile, finché non la si inizializza o non se ne setta il tipo, "isset" su quella variabile varrà sempre "false". La funzione "unset" ritorna sempre "true" in questo caso.

In PHP esiste il tipo boolean e l'insieme dei suoi valori sono le costanti predefinite "true" e "false":

```
<%
$a=true;
echo gettype($a); // stampa boolean
echo gettype(false); // stampa boolean
echo true; // stampa 1
echo false; // non stampa niente
// stampa 1: ogni intero !=0 usato come boolean è
// considerato vero echo (boolean)-1;
%>
```

Potete scrivere queste costanti col case che volete: `TRUE`, `true`, `True`, `tRUe`, ecc.. sono tutte la stessa costante. Tuttavia PHP non ha un tipo booleano dedicato, similmente al C e al Perl e a differenza del Pascal: questo è conveniente poiché ogni valore (numerico e anche non numerico) può essere usato direttamente come valore di verità: è considerato vero se è diverso da zero (anche se negativo), falso se uguale a zero. La stringa vuota (`""`) e la stringa con la cifra zero (`"0"`) valgono false in termini booleani; tutte le altre stringhe valgono true. Un array vale false solo se vuoto, true altrimenti.

Gli usuali operatori aritmetici `+`, `-`, `*`, `/` si applicano sia agli integer che ai double. Se uno degli operandi è un double, il risultato sarà double; se entrambi sono integer il risultato sarà intero.

Il manuale dice che la divisione `/` è una divisione intera e ritorna un intero se i due operandi sono interi (o stringhe che vengono convertite ad interi), mentre se uno degli operandi è un numero a virgola mobile, viene effettuata la divisione tra double.

Ho notato che le cose non stanno proprio così e che viene effettuata la divisione decimale anche quando ci sono due operandi interi non divisibili. Se volete sempre la divisione intera, è consigliabile fare un cast:

```
$a=10; $b=3;
$c = $a / $b;
echo $c; // stampa 3.3333333333333
$c = (int)($a / $b);
echo $c; // stampa 3
```

L'operatore modulo `%` restituisce il resto della divisione intera tra gli operandi e restituisce quindi sempre un integer. Eventuali argomenti double vengono troncati e convertiti ad intero.

Tutti gli operatori aritmetici possono essere combinati con l'operatore di assegnamento esattamente come in C:

```
$a += $b <=>$a = $a + $b
$a -= $b <=>$a = $a - $b
$a *= $b <=>$a = $a * $b
$a /= $b <=>$a = $a / $b
$a %= $b <=>$a = $a % $b
```

integer e double possono essere confrontati con gli usuali operatori relazionali <, >, <=, >=, ==, `!=`, la cui sintassi è identica a quella del C. Dal C, PHP prende a prestito anche l'operatore ternario `?:`, la cui sintassi consigliata è:

$$(\mathit{expr1})\ \texttt{?}\ (\mathit{expr2})\ \texttt{:}\ (\mathit{expr3});$$

questo operatore permette di effettuare una selezione di due espressioni (expr2 ed expr3), in base al valore di verità (rispettivamente true e false) della prima. L'operatore valuta e ritorna il risultato dell'espressione selezionata, che può a sua volta far parte di una espressione. Quando avete dubbi sulla precenza di `?:`, potete racchiudere tutta l'espressione ternaria tra parentesi (in ogni caso, vedi in seguito per la tabella delle precedenze degli operatori).

```
assegna a $max il valore massimo tra $a e $b:
$max = ($a>$b) ? ($a) : ($b);
```

PHP mette disposizione la funzione matematica `max` che restituisce il massimo di una serie di valori di lunghezza arbitraria e la sua controparte `min` per il minimo. Entrambe le funzioni possono confrontare interi, double e stringhe.

Come in C, un assegnamento è una espressione che ritorna il valore assegnato e l'operatore `=` (da non confondersi con `==`), è associativo da destra verso sinistra. Quindi as es.:

```
$a = $b = 3 <=> $b=3; $a=$b
```

Anche gli operatori unari di incremento e decremento seguono la stessa sintassi e semantica del C. Ad. es. un operatore ++ postfisso incrementa di 1 l'integer a cui si applica, ma restituisce il valore prima dell'incremento, mentre il ++ prefisso incrementa di 1 e restituisce il valore incrementato. Questi operatori permettono di incrementare o decrementare dell'unità delle variabili all'interno della valutazione di una espressione.

```
$a=6;
echo $a++ . "<br>\n"; // stampa 6
echo $a; // stampa 7
```

++ e -- incrementano/decrementano di 1 anche i double in PHP. Naturalmente si devono applicare a delle variabili. Non ha senso scrivere cose del tipo ++5.

Associatività e precedenza degli operatori è espressa da questa tabella, tratta dal manuale di PHP, in cui gli operatori più in basso hanno sempre precedenza MAGGIORE rispetto a quelli listati primi. Operatori con lo stesso livello di precedenza, ossia listati sulla stessa riga, in mancanza di parentesi che forzino un ordine diverso, si associano come indicato nella prima colonna (ammesso che si possono associare, altrimenti è indicato "non-associative")[5]:

1.6 Stringhe

Finora abbiamo racchiuso tutte le stringhe tra apici doppi, ma esiste anche un altro tipi di apici, quelli singoli. La differenza tra apici doppi e singoli consiste nel fatto che le variabili e le sequenze di escape sono interpretati ed espansi negli apici doppi, ma non in quelli singoli (con l'unica eccezione delle sequenze di escape \\ e \').

```
$a="contenuto espanso";
echo "ecco il $a"; // stampa "ecco il contenuto espanso"
echo 'ecco il $a'; // stampa "ecco il $a"

# stampa "riga1" su una riga e "riga2" sulla successiva
<PRE>
<%= "riga1\nriga2" %>
</PRE>

# stampa "riga\nriga2" su una sola riga
<PRE>
```

[5] Alcuni di questi operatori saranno trattati in seguito

<table>
<tr><th>Associativity</th><th>Operators</th></tr>
<tr><td>left</td><td>,</td></tr>
<tr><td>left</td><td>or</td></tr>
<tr><td>left</td><td>xor</td></tr>
<tr><td>left</td><td>and</td></tr>
<tr><td>right</td><td>print</td></tr>
<tr><td>left</td><td>= += -= *= /= .= %= &= |= ^= ~= <<= >>=</td></tr>
<tr><td>left</td><td>? :</td></tr>
<tr><td>left</td><td>||</td></tr>
<tr><td>left</td><td>&&</td></tr>
<tr><td>left</td><td>|</td></tr>
<tr><td>left</td><td>^</td></tr>
<tr><td>left</td><td>&</td></tr>
<tr><td>non-associative</td><td>== != === !==</td></tr>
<tr><td>non-associative</td><td>< <= > >=</td></tr>
<tr><td>left</td><td><< >></td></tr>
<tr><td>left</td><td>+ - .</td></tr>
<tr><td>left</td><td>* / %</td></tr>
<tr><td>right</td><td>! ~ ++ -- (int) (double) (string) (array) (object)</td></tr>
<tr><td>right</td><td>[</td></tr>
<tr><td>non-associative</td><td>new</td></tr>
</table>

Tabella 1.1: Operator precedence.

```
<%= 'riga1\nriga2' %>
</PRE>
```

Potete includere un apice doppio in una stringa quotata con apici singoli senza problemi, mentre se volete includerlo in una stringa con apici doppi dovete escaparlo (inglesismo che sta per "farlo precedere con \""), altrimenti il parser crederà che la stringa finisce lì.

```
echo 'Un " in una stringa tra apici singoli non dà problemi';
echo "Ecco un \" in una stringa tra apici doppi";
```

Per la stessa ragione, se volete includere un $ in una stringa tra ", dovete escaparlo, altrimenti il parser penserà che quello che segue è un nome di variabile, che non è quello che volete.

Inoltre se volete includere il backslash, dovete escapare un backslash stesso (\\). Come accennato poco fa, questa sequenza di escape è interpretata anche dentro una stringa con gli apici singoli, così come lo è `\'`, che permette di inserire un apice singolo in una stringa con apici singoli. Come detto prima queste sono le uniche sequenze interpretate in una stringa con apici singoli.

Oltre ad escapare singoli caratteri per evitare di mandare in confusione il parser, il carattere \ viene usato anche per rappresentare alcuni caratteri speciali e non altrimenti stampabili. Eccoli:

sequenza	**significato**
`\n`	`linefeed (LF o 0x0A in ASCII)`
`\r`	`carriage return (CR o 0x0D in ASCII)`
`\t`	`horizontal tab (HT o 0x09 in ASCII)`

Tabella 1.2: Caratteri di escape.

È poi possibile specificare un qualsiasi carattere ASCII (anche non standard, da 128 a 255) facendo seguire il \ col codice in ottale o esadecimale, come in C. Ad es. `\0x0A` oppure `\012` equivalgono a `\n`.

L'operatore di concatenamento delle stringhe è il punto (`.`). Al contrario del C, non dovete preoccuparvi di allocare la memoria necessaria e poi copiare i caratteri di una stringa alla fine di un'altra con `strcat()`. Potete concatenare stringhe con la stessa facilità con cui sommate numeri, ma ricordatevi di usare l'operatore `.` e non l'operatore `+`, che ha effetti ben diversi.
supponendo che la variabile `$nome` contenga il nome di una persona, la costruzione di una stringa di saluto si può effettuare così:

```
$saluto = 'Ciao ' . $nome . ', come stai?';
```

se ad. es. `$nome` è `"Antonio"`, `$saluto` risulterà `", come stai?"`. Questo lavoro avrebbe potuto essere ottenuto anche grazie all'espansione delle variabili negli apici doppi:

```
$saluto = "Ciao $nome, come stai?";
```

L'operatore `.` è comunque utile se si vuole concatenare direttamente una stringa ottenuta come valore di ritorno da una funzione, senza usare variabili intermedie:

```
$saluto = 'Ciao ' . getnome() . ', come stai?';
```

Se utilizzate l'operatore `+` per tentare di concatenare due stringhe, poiché `+` è un operatore aritmetico, PHP cercherà di convertire le stringhe in numeri e poi di sommarle. Una stringa che inizia con un carattere non numerico, viene considerata valere `0` (a meno che non si tratti di un segno `+` o `-` seguito subito dopo da una cifra), quindi, ad es.:

```
$saluto = 'Ciao ' + $nome + ', come stai?';
```

nell'ipotesi che nome è `"Antonio"`, assegna a `$saluto` il valore integer `0`. Se faccio:

```
$num = "44 gatti" + '1cane' + 1 + "ippopotamo";
```

`$num` diventa una variabile intera col valore 46. A volte la conversione implicita di stringhe in numeri può essere utile, come quando usate la funzione `atoi` in C.

Se desiderate concatenare una stringa con un'altra o più e poi salvare il risultato nella prima stringa è più immediato utilizzare l'operatore di assegnamento composto con la concatenazione `.=`:

```
$str1 = $str1 . $str2;
equivale a:
$str1 .= $str2;
```

Si possono accedere i singoli caratteri di una stringa indicizzandola come se fosse un array di caratteri (come in C). Tuttavia in PHP non esiste il tipo "char" del C e il tipo risultante da questa indicizzazione è ancora una stringa, contenente però un solo carattere. Quindi in PHP un "carattere" è una stringa costituita da un solo carattere.

```
<%
$s = "php";
echo $s[1], ' ', gettype($s[1]); // stampa "h string"
// come sopra, ma con un solo argomento alla funzione echo
echo $s[1] . ' ' . gettype($s[1]);
%>
```

Come in C, in PHP l'indicizzazione dei vettori è a base zero, quindi il primo carattere ha l'indice 0 e non 1 e l'ultimo carattere non ha indice pari alla lunghezza della stringa, bensì pari alla lunghezza della stringa meno 1. Indicizzare fuori dai limiti non è un errore come in C, ma produce una stringa vuota.

La gestione delle stringhe è un aspetto molto richiesto dalla programmazione CGI e PHP provvede notevole potenza in questo campo. Difatti contiene molte funzioni di manipolazione avanzata già pronte, che altrimenti impieghereste giorni per programmarvele in modo efficiente e per testarle; per approfondire consultate la sottosezione *"LX. String functions"* della sezione *"IV. Function Reference"* del manuale.

Qui mi limito a considerare alcune funzioni tra cui quella che restituisce la lunghezza di una stringa:

int **strlen** (string *str*)

e quella che confronta due stringhe:

int **strcmp** (string *str1*, string *str2*)

quest'ultima ritorna `<0` se *str1* è minore di *str2*; `>0` se *str1* è maggiore di *str2*; ritorna `=0` se sono uguali.

L'ultimo carattere di una stringa `$s` si ottiene con l'espressione:

```
$s[strlen($s)-1]
```

Entrambe queste funzioni sono del tutto analoghe alle loro controparti in C. Esistono anche delle varianti di **strcmp** che effettuano un confronto case insensitive (ossia ignorano la differenza maiuscolo-minuscolo) o che si limitano a confrontare i primi n caratteri, rispettivamente

int **strcasecmp** (string *str1*, string *str2*)

int **strncmp** (string *str1*, string *str2*, int *len*)

Potete evitare di usare la funzione **strcmp**, poiché gli operatori relazionali `<`, `>`, `<=`, `>=`, `==`, `!=` funzionano allo stesso modo quando si confrontano stringhe. È bene però stare attenti quando si confrontano tramite gli operatori relazionali stringhe con interi o con double:

```
echo "ant" < "1";         // non stampa niente
echo "ant" < 1;           // stampa 1
```

Nel primo caso c'è un confronto tra stringhe e "ant" è lessicograficamente maggiore di "1" (il carattere ASCII `"a"` ha codice maggiore di `"1"`, rispettivamente 97 e 49), non viene stampato niente, cioè confronto falso.

Nel secondo caso c'è una conversione implicita di "ant" in intero, che dà zero e siccome $0 < 1$, viene stampato 1 (cioè confronto vero).

Se ad es. `$a` è una variabile intera con valore 1 che si vuole confrontare con la stringa "ant", il primo tipo di confronto (confronto tra stringhe) si può forzare con un cast:

```
$a=1;
echo "ant" < (string)$a; // non stampa niente
```

Infine spesso avete la necessità di rimuovere finelinea e spazi dall'inizio e/o dalla fine di una stringa (specialmente quelle passate al vostro programma da una form HTML). PHP offre tre funzioni, le quali non modificano la stringa argomento, ma piuttosto ritornano la stringa modificata:

string **chop** (string *str*)

elimina un solo carattere di spaziatura alla fine, se c'è

string **ltrim** (string *str*)

elimina tutti i caratteri di spaziatura all'inizio di una stringa

string **trim** (string *str*)

elimina tutti i caratteri di spaziatura tanto all'inizio che alla fine di una stringa.

Per tutte queste funzioni è considerato carattere di spaziatura uno qualsiasi tra `"\n"`, `"\r"`, `"\t"`, `"\0"`, e il normale spazio `" "`.

```
$s=chop($s);
$s=ltrim($s);
$s=trim($s);
```

In alcuni algoritmi potreste aver bisogno di conoscere il codice ascii di un carattere oppure di ottenere il carattere dato il corrispondente codice ascii. Per questo scopo in PHP occorre utilizzare rispettivamente:

int **ord** (string *string*)
string **chr** (int *ascii*)

Ricordo ancora che "un carattere" in PHP è una stringa di lunghezza 1. La funzione **ord** può ricevere anche una stringa più lunga di 1 carattere: prende in considerazione solo il primo carattere e ne restituisce il codice.

Per una ricerca semplice di una sottostringa in una stringa usate la funzione:

string **strstr** (string *str*, string *substr*)

questa è simile alla funzione **strstr** del C, ma non ritorna puntatori a carattere (char *), bensì ritorna una stringa formata dall'intera parte finale di *str*, a partire dalla posizione in cui si trova la sottostringa *substr*. Se la sottostringa *substr* non si trova a partire da nessuna posizione di *str*, allora **strstr** restituisce un valore boolean false anziché string. **strstr** effettua un confronto case-sensitive. La funzione:

string **stristr** (string *str*, string *substr*)

agisce in modo identico solo che effettua un confronto case-insensitive tra le due stringhe. Per effettuare una ricerca con sostituzione utilizzate la funzione:

string **str_replace** (string *substr*, string *newstr*, string *str*)

Questa sostituisce tutte le occorrenze di substr in *str* con *newstr*. Più avanti ne vedremo un utilizzo tipico nella programmazione CGI.

Altre due funzioni molto semplici convertono i caratteri maiuscoli di un'intera stringa in minuscolo o viceversa e ritornano la stringa convertita (non alterano la stringa argomento e i caratteri senza case, ossia quelli né maiuscoli né minuscoli):

string **strtolower** (string *str*)
string **strtoupper** (string *string*)

Ci sono molte altre potenti funzioni che qui per brevità non descriveremo. Ogni volta che sentite il bisogno di fare qualche lavoro con una stringa, consultate l'indice nel capitolo "LX. String functions" del manuale PHP; quasi sicuramente c'è già una funzione pronta che vi farà risparmiare il tempo necessario a dovervi codificare le cose da soli e renderà anche il programma più leggibile, breve e veloce.

1.7 Array

Abbiamo visto l'utilizzo dei tipi integer, double e string. Esiste un'altro tipo predefinito in PHP piuttosto ricco di funzionalità: il tipo array che discuteremo adesso. È possibile poi crearsi dei propri tipi di dati strutturati, che includono anche le funzioni per manipolarli, ricorrendo al paradigma delle classi, come vedremo in seguito.

In PHP gli array sono molto più potenti che in linguaggi come il C: PHP unifica il concetto di array scalare (o vettore) con quello di tabella hash (o array associativo). Di fatto l'array scalare in PHP non esiste ed esistono solo hash, che sono più potenti e possono essere usati anche come semplici vettori. Di seguito quando si parla semplicemente di array ci si vuole riferire ad hash, ossia ad array associativo; se ci si vuole riferire ad array scalari, si userà esplicitamente l'aggettivo scalare o si chiameranno "vettori".

Cos'è un hash? È un concetto più generale di un array scalare, implementabile in memoria dinamica anche in linguaggio compilato come il C (si veda a proposito *"Linguaggio C"* di Kernighan e Ritchie, *"Analisi delle tabelle"*, pag. 184 per un esempio di semplice implementazione). Mentre in un vettore si indicizzano gli elementi con degli interi, in un hash si possono indicizzare con stringhe qualsiasi, quindi è una specie di memoria associativa.

In altri termini un hash è una struttura dati che mette in relazione univoca un certo numero di stringhe (dette chiavi) con lo stesso numero di altri dati (detti valori). Ad ogni chiave corrisponde un valore.

Quando le chiavi sono tutte "numeriche", il concetto di hash si riduce a quello più semplice di vettore. Ho messo la parola numeriche tra virgolette, poiché in realtà le chiavi di un hash sono tutte di tipo stringa: quando utilizzate un numero come indice, esso viene convertito in stringa internamente. La stringa viene poi passata ad una funzione di hashing, ma questi sono dettagli di implementazione che preferisco non affrontare in questa sede. Fate girare questi esempi per convinceverne:

```
<%
$a["56"]=56;
echo $a[56]; // stampa 56
%>

<%
$a[56]=56;
echo $a["56"]; // stampa 56
%>
```

Abbiamo detto che in generale in PHP non occorre dichiarare le variabili, quindi non occorre dichiarare gli array (o hash). Basta indicizzarli e settare gli elementi: PHP alloca automaticamente la memoria necessaria e gestisce le strutture dati dinamiche che li implementano.

```
# setta un array scalare con i primi 5 numeri primi
$n[0]=1;
$n[1]=3;
$n[2]=5;
$n[3]=7;
$n[4]=11;

# setta solo gli elementi dispari con numeri pari (array scalare "bucato")
$d[1]=2;
$d[3]=4;
$d[5]=6;
```

Nell'ultimo esempio ci si può chiedere: qual'è il valore e il tipo di `$d[0]`, `$d[2]`, `$d[4]`, `$d[6]`, ecc...? La risposta è nessun valore e il tipo è NULL, poiché questi valori non sono definiti nel mucchio (hash in inglese significa mucchio). Quindi si possono avere questa specie di array bucati, che non sono altro che un caso particolare di hash.

La flessibilità di un hash permette quindi di avere alcuni indici "numerici" ed altri di tipo stringa:

```
$f[0]=1000000;
$f["zero"]=1000000;
```

è importante realizzare che se scrivessi

```
$f[0]=1000000;
$f["zero"]=$f[0];
```

il valore di `$f[0]` viene copiato in `$f["zero"]`, non condiviso tra i due e quindi la modifica di `$f[0]` non influisce su `$f["zero"]`. Gli " che racchiudono la chiave si possono omettere, essendo formata da una sola parola, tuttavia consiglio di usarli sempre, per evitare problemi. Se infatti scegliete una chiave identica ad una parola riservata e non la racchiudete tra apici il parser si confonde e dà errore:

```
$a[WHILE]; // errore
$a["WHILE"]; // giusto
$a[While]; // errore
$a[while]; // errore
$a["while"]; // giusto
$a[zuz]; // giusto
$a["zuz"]; // giusto e preferibile
```

$a["zuz"] è preferibile rispetto a `$a[zuz]`, che pure è giusta perché `zuz` non è una parola chiave, ma se in futuro si decide di fare di `zuz` una parola chiave, il vostro programma non funzionerà più se non vengono messe le " attorno a `zuz`. Non si sa mai, quindi usate sempre le " che racchiudono i nomi delle chiavi, e in generale tutte le stringhe, anche se composte da una sola parola non chiave e iniziante per lettera[6].

Un hash può contenere valori di tipi diversi, osservate il seguente

```
$a[5]=44; // $a[5] è un integer
$a["a"]="quarantaquattro"; // $a["a"] è una string
echo '$a[5] '.gettype($a[5]), '<br>$["a"] '.gettype($a["a"]);
```

[6]Una stringa non quotata che non fa match con nessuna parola riservata né costante è detta in gergo "barestring" (letteralmente "stringa nuda"), oppure "bareword". Meglio non lasciare nude le stringhe.

l'output in html risulta:

```
$a[5] integer
$["a"] string
```

PHP permette molte volte di sottoindere l'indice di un array, snellendo la sintassi:

```
$a[]="php";
$a[]="is";
$a[]="great";
echo "$a[0] $a[1] $a[2]"; // stampa "php is great"
```

equivale a:

```
$a[0]="php";
$a[1]="is";
$a[2]="great";
echo "$a[0] $a[1] $a[2]"; // stampa "php is great"
```

Questo è un modo conveniente di aggiungere elementi ALLA FINE di un array. Ritengo che sia implementato pressapoco in questo modo: PHP mantiene per ogni hash un valore pari alla massima chiave numerica che fa parte dell'insieme delle chiavi di quell'hash. Ogni volta che si assegna un elemento di un hash con una chiave numerica il valore della chiave numerica viene confrontato con quello massimo finora registrato e se risulta maggiore, il valore massimo viene aggiornato col nuovo valore. L'operatore di subscript vuoto `[]` semplicemente prima incrementa questo valore massimo e poi lo usa implicitamente, e nel caso non sia mai stata usata una chiave "numerica" finora assegna all'elemento con chiave 0 (il valore della massima chiave numerica è infatti inizializzato con 0).

Se una sintassi del tipo:

```
$a[]="php";
$a[]="is";
$a[]="great";
```

vi sembra troppo prolissa (i token `$a[]=` sono ripetuti tre volte), php offre la funzione built-in array, che formalmente restituisce un array dei suoi argomenti:

array **array** ([mixed ...])

quindi potete riscrivere così:

```
$a = array("php","is","great");
```

La funzione array è sufficientemente potente da permettevervi di abbreviare tutte le definizioni di hash che si possono sintetizzare:

```
$a["banane"] = "fuori stagione";
$a["pere"] = "2500 £/Kg";
$a["mele"] = "2000 £/Kg";
```

si può abbreviare in:

```
$a = array("banane"=>"fuori stagione",
"pere"=>"2500 £/Kg", "mele"=>"2000 £/Kg");
```

Anche la definizione di hash "bucati" si può abbreviare:

```
$d = array(1=>2, 3=>4, 5=>6);
```

equivale a:

```
$d[1]=2;
$d[3]=4;
$d[5]=6;
```

e non è necessario che tutte le chiavi numeriche siano specificate esplicitamente. Questo esempio tratto dalla documentazione ufficiale di PHP:

```
$holes = array ("first", 5 => "second", "third");}\end{center}
```

equivale a:

```
$holes[0] = "first";
$holes[5] = "second";
$holes[6] = "third";
```

Inoltre se scrivete:

```
$a = array();
```

assegnate ad $a un array vuoto, che equivale a settare il tipo di $a come un tipo array scrivendo:

```
settype($a, 'array');
```

Come nel caso delle stringhe, PHP possiede molte potenti funzioni predefinite che operano sugli hash. Vediamo le più importanti (come al solito vi rimando alla documentazione per approfondire).

La funzione:

int **count** (mixed *var*)

ritorna il numero di elementi di un array. Un array vuoto ha un numero di elementi pari a 0. Se invece di un array passate un altro tipo di variabile, se la variabile non è settata (isset è false) ritorna pure 0, altrimenti ritorna 1.

Provate indipendentemente questi miniprogrammi per capire la logica dei valori restituiti da **count**.

```
<%
echo count($a); // scrive 0, $a è non settata
%>

<%
$a = array(1,2,3);
echo count($a); // scrive 3, $a è un array con 3 elementi
%>

<%
settype($a, "array");
echo count($a); // scrive 0, $a è un array vuoto
%>

<%
$a=3;
echo count($a); // scrive 1, $a è uno scalare
%>

<%
settype($a, integer);
echo count($a); // scrive 1, $a è uno scalare (vale 0)
%>
```

voglio far notare che nell'esempio dell'array vuoto la stringa "array", secondo argomento di settype, deve essere quotata, altrimenti il parser non la vede come stringa, ma come la parola riservata array che abbiamo introdotto prima. Si comporta allo stesso modo della funzione count anche la funzione:

int **sizeof** (array *array*)

Mentre nel caso di un array indicizzato con soli numeri viene semplice percorrere tutti gli elementi dell'array, basta semplicemente percorrere un intervallo di valori interi usati come indici, nel caso di un array indicizzato anche con stringhe ci deve essere un meccanismo per ottenere la lista delle chiavi.

PHP mette a disposizione delle funzioni apposite. PHP memorizza insieme ad ogni array un puntatore all'elemento corrente, che viene inizializzato con il riferimento al primo elemento inserito nell'array. La funzione:

mixed **key** (array *array*)

ritorna la chiave dell'elemento corrispondente alla posizione corrente nell'array specificato come argomento. Essa non altera il puntatore alla posizione corrente, mentre la funzione:

mixed **next** (array *array*)

fa avanzare il puntatore alla posizione corrente e ritorna il nuovo elemento puntato, oppure false se sono stati percorsi tutti gli elementi nell'hash. La funzione:

mixed **reset** (array *array*)

riporta il puntatore interno all'elemento corrente di array a puntare al primo elemento e ritorna proprio il valore di questo primo elemento.

```
<%
/* stampa
$a[banane]=fuori stagione
$a[pere]=2500 £/Kg
$a[mele]=2000 £/Kg
*/

$a = array("banane"=>"fuori stagione",
        "pere"=>"2500 £/Kg", "mele"=>"2000 £/Kg");

// stampa dell'array $a
for (reset($a); $frutta = key($a); next($a))
   echo "\$a[$frutta]=$a[$frutta]<br>";
%>
```

Va notato che il ciclo si basa sul fatto che la funzione key restituisce un valore logico false dopo che next rileva che tutti gli elementi sono stati percorsi. Poiché il valore ritornato da key ed assegnato a $frutta è anche usato come condizione di terminazione del ciclo for, il ciclo termina quando key restituisce false. La chiamata iniziale di reset in questo programma è opzionale e può essere tralasciata.

La funzione:

mixed **current** (array *array*)

è la controparte di key: essa ritorna l'elemento dell'array riferito dal puntatore all'elemento corrente, anziché la sua chiave come fa key. Come key essa non altera il puntatore e ritorna false se il puntatore punta oltre la fine dell'array. Avremmo potuto scrivere il programma precedente anche in questo modo:

```
<%
/* stampa
$a[banane]=fuori stagione
$a[pere]=2500 £/Kg
$a[mele]=2000 £/Kg
*/

$a = array("banane"=>"fuori stagione",
        "pere"=>"2500 £/Kg", "mele"=>"2000 £/Kg");

// stampa dell'array $a
for (reset($a); $frutta = key($a); next($a))
   echo "\$a[$frutta]=".current($a)."<br>";
%>
```

oppure:

```
<%
/* stampa
$a[banane]=fuori stagione
$a[pere]=2500 £/Kg
$a[mele]=2000 £/Kg
*/

$a = array("banane"=>"fuori stagione",
        "pere"=>"2500 £/Kg", "mele"=>"2000 £/Kg");

// stampa dell'array $a
for (reset($a); key($a); next($a)) // oppure current($a) come condizione
   echo '$a['. key($a) .']='. current($a) .'<br>';
%>
```

La terna reset,key,next ha il pregio di traversare l'array nell'ordine esatto in cui gli elementi sono inseriti, ma purtroppo non può essere convenientemente utilizzata per attraversare ogni tipo di array. Il ciclo precedente, sicuramente comodo in molti casi, presenta infatti una limitazione: si ferma non appena viene incontrata una chiave pari a 0, poiché la condizione del for risulta falsa. Difatti un ciclo del tipo:

```
for(...;"0";...) // oppure while (0), for (;"";) ecc...
  {
     echo "while eseguito?!";
  }
```

non viene eseguito nemmeno una volta, poiché "0" (o 0) si valuta a false. Allo stesso modo un ciclo basato su current() si fermerebbe al primo elemento contenente il numero 0, la stringa "0", o la stringa vuota "". In array che possono contenere questi valori "vuoti" non c'è modo semplice di capire se siamo veramente alla fine della lista tramite le funzioni key, current, o next, basandosi sul fatto che queste restituiscono false, perché ci sono questi valori vuoti che vengono interpretati come false. Quindi potremmo fermarci su uno di questi valori, non considerando così tutti i dati inseriti successivamente nell'array.

Per queste ragioni è stata creata un'altra funzione per traversare gli array, each, che non presenta di questi inconvenienti, ma è un po' più complicata da usare:

array **each** (array *array*)

questa funzione ritorna in un sol colpo sia la chiave che il valore dell'elemento corrente e poi avanza il puntatore sul prossimo elemento dell'array. Essa ha per argomento ovviamente l'array su cui operare il prelievo e restituisce un altro array di 4 elementi. La chiave e il valore vengono cioè "impacchettati" in un array di 4 elementi, in cui le 2 chiavi 0 e "key" sono associate alla chiave dell'elemento corrente, mentre le altre 2 chiavi 1 e "value" sono associate al valore dell'elemento corrente. Arrivati oltre la fine dell'array, each ritorna false. In pratica potete scegliere se riferirvi alla chiave dell'elemento corrente con l'indice "0" oppure "key" e al valore dell'elemento elemento con l'indice "1" o "value" all'interno dell'array ritornato da each. Gli indici numerici sono più compatti e veloci da scrivere, quelli letterali sono più prolissi ma più semplici da interpretare.

Capite subito che ritornando un array, each può essere usato come condizionale senza problemi anche quando l'elemento corrente ha chiave e/o indice tra quei valori "vuoti".

Se tutto questo suona complicato, ecco un esempio molto semplice:

```
<%
/* output:
$n[1]=1
$n[value]=1
$n[0]=0
$n[key]=0
*/

$i=array(1,2,3);
$n = each ($i);
echo "\$n[1]=$n[1]<br>
 \$n[value]=$n[value]<br>
 \$n[0]=$n[0]<br>
 \$n[key]=$n[key]";
%>
```

La chiave del primo elemento è `$n[0]`, oppure `$n[key]` (questo è uno dei pochi casi in cui è sicuro risparmiarsi di quotare l'indice letterale, poiché ci troviamo all'interno di una stringa). Questi due ultimi valori risulteranno sempre uguali. Il valore del primo elemento è dato da `$n[1]==$n[value]`. Un ciclo basato su key per stampare questo array non stamperebbe niente, provare per credere:

```
<%
$i=array(1,2,3);
for (reset($i); $idx=key($i); next($i))
   echo "\$[$idx]=$i[$idx]<br>\n";
%>
```

già sapete perché...

È importante capire che l'ordine negli array di PHP è significativo e che l'ordine degli elementi nell'array ritornato da each è esattamente quello con cui sono stati stampati nell'esempio precedente. Volendo comporre un ciclo di stampa con each, si può fare come in questo esempio:

```
<%
/* output:
qui ha preso 8
quo ha preso 7
qua ha preso 6
*/

$voti=array("qui"=>8, "quo"=>7, "qua"=>6);
```

```
while ($voto=each($voti))
   echo "$voto[key] ha preso $voto[value]<br>";
%>
```

volendo essere più criptici l'istruzione di stampa si può abbreviare così:

```
echo "$voto[0] ha preso $voto[1]<br>";
```

Spesso avete la necessità di assegnare i valori di elementi di un array a variabili distinte, scartandone eventualmente alcuni. Per semplificare le cose PHP include il costrutto:

void **list**(...);

il quale riceve una lista di variabili e gli assegna alcuni valori di un array, come se stessimo assegnando ad un altro array. Questo costrutto rappresenta un'estensione sintattica del lato sinistro di una assegnazione e quindi va usato in quel posto.

Più precisamente list permette di assegnare ad una lista di variabili alcuni valori di un array con chiave numerica in un un'unica, compatta operazione, scartando tutti i rimanenti. L'esempio precedente riguardante i voti scolastici dei nipoti del ben noto Paperino, può essere riscritto in questo modo equivalente utilizzando list:

```
<%
$voti=array("qui"=>8, "quo"=>7, "qua"=>6);
while (list($nipote, $voto)=each($voti))
   echo "$nipote ha preso $voto<br>";
%>
```

Il valore con chiave numerica 0 dell'array restituito da each viene assegnato alla variabile `$nipote`, quello con chiave numerica 1 alla variabile `$voto`. **list** vi permette di assegnare esattamente e solo gli elementi che volete, semplicemente lasciando il posto vuoto agli elementi che non volete assegnare:

```
<%
$a=array(1,"test"=>2,3,3=>5,2=>4,6);
/* dopo questa assegnazione risulta:
$a[0]==1
$a["test"]=2
$a[1]==3
$a[2]==4
```

```
$a[3]==5
$a[4]==6
*/
list(, $a, , $b) = $a; // quindi $a diventa 3 e $b diventa 5
echo "$a<br>$b";        // stampa 3<br>5
%>
```

Si ricorda che list vede solo gli elementi con indice numerico, li considera in ordine di indice numerico e tra questi seleziona. Notare anche che il fatto che si riassegni `$a` a se stesso non produce problemi.

Un uso tipico di questi due costrutti che abbiamo imparato, list ed each, nell'ambito della programmazione CGI, consiste nel traversamento dell'array `$HTTP_POST_VARS` oppure `$HTTP_GET_VARS`. Questi sono array associativi già predisposti per voi in questo formato dall'interprete PHP che li ottiene comunicando col server web. In pratica una form HTML trasmette delle coppie nome=valore opportunamente codificate al server web, che si ritrovano negli array `$HTTP_POST_VARS` e `$HTTP_GET_VARS` a seconda che rispettivamente venga utilizzato il metodo di codifica POST o GET dal browser del client. Con PHP non è più necessario fare il lavoro di decoding e interpretazione della variabile di ambiente QUERY_STRING o dello standard input, poiché questo lavoro lo fa ogni volta l'interprete e crea questi due array globali e crea inoltre per ogni coppia nome=valore una variabile `$nome` inizializzata con il valore valore. In ogni caso comunque php mette a disposizione una funzione **urldecode** già pronta per effettuare il decoding dei parametri di un URL e la funzione **parse_str** che oltre ad effettuare il decoding, divide le coppie nome-valore e dichiara automaticamente delle variabili **$nome** e le inizializza con i rispettivi valori.

Solitamente uno script sa quale variabili gli vengono passate e può accedervi ad una ad una tramite **$nome**. Ci sono tuttavia casi in cui i nomi e/o il numero delle variabili passate non sono noti allo script e in questi casi occorre percorrere uno degli array `$_HTTP_POST_VARS` o `$HTTP_GET_VARS` a seconda del metodo utilizzato.

Questo script ispirato dalla documentazione di PHP visualizza nomi e valori di tutte le variabili web passate ad uno script col metodo GET (sostituite GET con POST per avere una rappresentazione dell'altro array):

```
<%
 echo "QUERY_STRING: $QUERY_STRING<p>";

 echo "Values submitted via GET method:<br>";
 reset ($HTTP_GET_VARS);
 while (list ($key, $val) = each ($HTTP_GET_VARS)) {
```

```
    echo "$key => $val<br>";
 }
%>
```

Se ad es. richiamete questo script con l'URL:

http://localhost/prova.php?prova=ciao&nome=Antonio

l'output html renderizzato sarà del tipo:

```
QUERY_STRING: prova=ciao&nome=Antonio

Values submitted via GET method:
prova => ciao
nome => Antonio

Nota: QUERY_STRING è settata solo col metodo GET.
```

Poiché gli hash in PHP hanno associato un ordinamento, ha senso ordinare un hash! Il termine hash in inglese significa mucchio e il mucchio è per antonomasia qualcosa di non ordinato, tuttavia in PHP gli hash sono speciali e portano dentro sé l'informazione relativa all'ordine in cui sono disposti gli elementi. Esistono delle funzioni per facilitarvi il compito.

La funzione sort ordina gli elementi di un array:

void **sort** (array *array* [, int *sort_flags*])

questo stamperà:

```
Londra-Madrid-New York-Roma-San Francisco

<%
$città = array("Roma", "New York", "Madrid", "San Francisco", "Londra");
sort($città);
echo "$città[0]-$città[1]-$città[2]-$città[3]-$città[4]";
%>
```

Per ordinare in ordine inverso usate:

void **rsort** (array *array* [, int *sort_flags*])

nell'esempio precedente **rsort** al posto di **sort** farebbe stampare

```
San Francisco-Roma-New York-Madrid-Londra
```

Le funzioni **sort** e **rsort** andrebbero usate solo nel caso di array che possiedono le stesse limitazioni degli array del C. Usare **sort** o **rsort** su un array in cui almeno un indice è una stringa, o ci sono dei buchi, produce un corretto ordinamento dei valori, ma tutte le chiavi vengono perse e rimpiazzate con chiavi numeriche contigue a partire da 0 (indici di un normale array del C).

Per ordinare senza perdere l'associazione degli indici, usate le funzioni:

void **asort** (array *array* [, int *sort_flags*])
void **arsort** (array *array* [, int *sort_flags*])

rispettivamente per l'ordinamento crescente e decrescente.

Il tipo di ordinamento predefinito per tutte queste funzioni è l'ordinamento regolare, che generalmente va bene per array con tipi misti (numerici e stringa). Se si vuole un ordinamento strettamente basato sul valore numerico, e su conversioni a valore numerico, passare `SORT_NUMERIC`. `SORT_STRING` invece forza a considerare tutti gli elementi dell'array come stringhe e a confrontarli come tali in base al codice ASCII.

```
<%
/*
 *  chiamata funzone ordinamento    risultato ordinamento
 *  ------------------------------------------------------------------
 *  sort($a)                        1ant    daffy   zorro   3       10
 *  sort($a, SORT_REGULAR);         1ant    daffy   zorro   3       10
 *  sort($a, SORT_NUMERIC);         zorro   daffy   1ant    3       10
 *  sort($a, SORT_STRING);          10      1ant    3       daffy   zorro
 */

$a=array(10, "1ant", 3, "zorro", "daffy");

for (reset($a); list($key,$value)=each($a); )
  echo "$key => $value<br>\n";

echo "<p>\n\n";

sort($a);
#sort($a, SORT_REGULAR);
#sort($a, SORT_NUMERIC);
#sort($a, SORT_STRING);

for (reset($a); list($key,$value)=each($a); )
  echo "$key => $value<br>\n";
%>
```

La funzione:

bool **in_array** (mixed *needle*, array *haystack*)

presente però solo a partire da PHP4 vi evita di dovervi programmare ogni volta anche una semplice ricerca sequenziale:

bool **bool** in_array (mixed *s*, array *a*)

questa ricerca s nell'array a e ritorna true se la trova, altrimenti false.

La cancellazione di un elemento da un array, con effettiva liberazione della memoria occupata si svolge ancora tramite la funzione unset vista prima per le variabili scalari:

```
unset($a["test"]); #toglie solo l'elemento di indice "test" da $a
unset($a); #l'intero array $a non è più isset
```

La lista completa delle numerose funzioni sugli array presenti in PHP si trova nella sottosezione II del manuale: *"II. Array Functions"*, all'interno della sezione *"IV. Function Reference"*. Qui inoltre mi sono limitato a trattare gli array unidimensionali. Sono sicuro che adesso non avrai molta difficoltà a capire la documentazione, una volta acquisite le basi.

1.8 Costanti

La definizione di semplici costanti è molto simile alla `#define` del C, solo che si usa la funzione **define**():

int **define** (string *name*, mixed *value* [, int *case_insensitive*])

Questa funzione non supporta le macro come in C. *name* è una stringa indicante il nome della costante. *value* è il valore, con la limitazione che deve essere uno scalare, perciò non potete definire un array costante. Il terzo parametro vale 0 per default, se specificate 1, la relativa costante sarà riferibile in modo case-insensitive. Per default quindi anche i nomi di costante sono case-sensitive. Una volta definita una costante, provare a ridefinirla non ne altera il valore e non c'è modo di annullarne la definizione (non esiste la corrispondente della `#undef` del C).

```
define("ANT_ETA", 23);
define("ANT_ETA", 18); // ANT_ETA rimane 23 (ahimé)

define("PROMPT", "> ", 1); // PROMPT, prompt, Prompt, ecc. valgono "> "
```

Per indicare una costante basta scriverne semplicmente il nome, che NON deve essere preceduto da `$` (in questo modo PHP distingue tra costanti e variabili):

```
echo ANT_ETA;
```

Nel caso dobbiate scrivere script che devono effettuare calcoli matematici, sappiate che PHP predefinisce con la migliore approssimazione parecchie costanti matematiche come `M_E` per il numero di Nepero, `M_PI` per π (pigreco). ecc... (per l'elenco completo vedere la sottosezione *"XXXIII. Mathematical Functions"* della sezione *"IV. Function Reference"* del manuale PHP).

1.9 Operatori logici

```
Esempio Nome Risultato

$a and $b  And   Vero se sia $a che $b sono Veri.
$a or $b   Or    Vero se $a oppure $b è Vero.
$a xor $b  Or    Vero se $a oppure $b è Vero, ma non entrambi.
! $a       Not   Vero se $a è non vero.
$a && $b   And   Vero se is $a che $b sono Veri.
$a || $b   Or    Vero se $a oppure $b è Vero.
```

Il nome riportato nella tabella precedente è il nome in inglese dell'operatore. Quelli in italiano sono "e" per And, "o" per Or e "non" per Not. Queste due lingue naturali tuttavia sono ambigue riguardo il significato di Or che può essere inclusivo (or) o esclusivo (xor) a seconda che la verità di entrambi i suoi argomenti dia verità o falsità rispettivamente.

Gli operatori logici `or`, `and` differiscono da `||` e `&&` solo perché i primi hanno una precedenza minore; in pratica ricordate che `or`, `and` hanno una precedenza minore degli operatori di assegnamento (in tutte le salse, quindi oltre che `=`, anche `+=`, `.=`, `&=`, ecc...) mentre `||` e `&&` hanno una precedenza maggiore degli operatori di assegnamento. Per i dettagli consultate la tabella delle precedenze riportate sopra. L'operatore logico `xor` esiste solo con precedenza più bassa degli operatori di assegnamento.
ES.
Supponendo che `$a` e `$b` siano integer, il seguente spezzone di codice assegna `$b` ad `$a` poi verifica che il valore assegnato sia diverso da 0 e inoltre che sia minore di 3:

```
if (($a=$b) && $b<3)
...
```

Le parentesi attorno all'assegnazione `$a=$b` sono necessarie (a meno di non spostarla fuori, prima dell'`if`), poiché l'operatore `&&` ha maggior precedenza di `=`. L'uso dell'operatore `and` permette di evitarle:

```
if ($a=$b and $b<3)
...
```

PHP supporta inoltre gli operatori logici bit a bit (bitwise operator in inglese) del C (i simboli sono gli stessi). Questi vengono usati più raramente per fare manipolazioni di bit a basso livello. Per chi fosse interessato la pagina relativa del manuale che li elenca è:

language.operators.bitwise.html

1.10 Strutture di controllo

Esattamente come in C, ogni istruzione in PHP è una espressione terminata da un punto e virgola (`;`). Esattamente come in C si può avere una istruzione vuota costituita solo dal `;` (che potrebbe sembrare una sciocchezza, ma è utile come segnaposto nel corpo di altre istruzioni quando non si desidera fare nulla) e si possono raggruppare più istruzioni in un blocco racchiudendole tra `{ }`. Un blocco di istruzioni è ancora una istruzione.

La sintassi del costrutto di selezione è la stessa del C (le parti racchiuse tra `[]` sono opzionali, quelle racchiuse tra `[ ]*` sono opzionali e ripetibili):

```
if (espressione)
     istruzione
[else
     istruzione]
```

L'unica differenza è che se la parte `else` è seguita da un `if` è anche possibile scrivere attaccate le due keyword `else` ed `if`; sintatticamente si ottiene una nuova keyword (`elseif`) e non si aumenta il livello logico di nesting, ma dal punto di vista pratico non vi è alcuna differenza tra scrivere "`else if`" e scrivere "`elseif`" tutto attaccato, solo che vi risparmiate di digitare un carattere. La sintassi completa dell'if in PHP è quindi la seguente:

```
if (espressione)
     istruzione
[elseif (espressione)
     istruzione
]*
[else
     istruzione]
```

La parte `else`, se presente, viene eseguita se nessuna delle espressioni è risultata vera.

Anche il ciclo `while`, il ciclo `do while` e il `for` sono identici al C. Come vedete, nel caso del `for`, tutte e tre le espressioni sono opzionali.

```
while (espressione)
     istruzione

do
     istruzione
while (espressione)

for ([espr1]; [espr2]; [espr3])
     istruzione
```

Per chi non conosce il C, la semantica del `for` è equivalente a quella del seguente ciclo `while`:

```
[espr1];
while (espr2|TRUE)
{
   istruzione
   [espr3];
}
```

in più occorre aggiungere che quando *espr2* manca, il linguaggio la considera sempre TRUE (il simbolo `|` indica che si deve scegliere tra *espr2* e TRUE).

Il costrutto `switch` di PHP segue la stessa sintassi e semantica del C, ma poiché in PHP le stringhe sono tipi nativi, è possibile anche avere un insieme di casi stringa e persino double (in C si possono avere solo interi):

```
switch (espressione)
{
     [case espr:
[istruzione]*]*
     [default:
[istruzione]*]
}
```

Tra le istruzioni una break interrompe l'esecuzione di un certo `case` intrapreso (altrimenti si proseguirebbe con le istruzioni sotto il `case` successivo).

Come in C l'istruzione break termina l'esecuzione di un ciclo `for`, `while`, o `switch`. In realtà la `break` del PHP è più potente. Se scrivo:

```
break;
```

equivale a:

```
break 1;
```

e significa appunto di terminare l'esecuzione della struttura annidata a livello 1 rispetto all'istruzione in cui si trova `break`. Si può terminare l'esecuzione anche ad un livello maggiore, cosa che in C si può ottenere solo ricorrendo ad una label e alla istruzione goto. Ad es.:

```
while (...)
{
   ...
   switch (...)
   {
break 2; // fa saltare fuori di 2 livelli
   }
   ...
}
// break 2 fa saltare qui
```

Anche l'istruzione `continue`, che non interrompe il ciclo ma salta alla fine del ciclo stesso accetta un argomento opzionale `n` indicante il numero di livello di nesting rispetto alla posizione corrente.

Oltre alla sintassi C-like per i costrutti di controllo, PHP offre anche una sintassi alternativa detta *colon syntax* (sintassi coi due punti). In questa forma sintattica la parentesi graffa aperta `{` che segna l'inizio di un blocco viene sostituita da `:`, mentre quella chiusa `}` che ne indica la fine viene sostituita da una keyword composta da `end` più il nome del ciclo tutto attaccato.

```
if (espressione):
     [istruzione]*
[elseif (espressione):
     [istruzione]*
]*
[else:
     [istruzione]*]
endif
```

```
while (espressione):
     [istruzione]*
endwhile
```

```
for ([espr1]; [espr2]; [espr3]):
     [istruzione]*
endfor
```

```
switch (espressione):
     [case espr:
[istruzione]*]*
     [default:
[istruzione]*]
endswitch;
```

In PHP inoltre il parser vi permette, quando siete nel mezzo di un blocco, di uscire dal modo PHP, ritornando nel modo HTML, per poi continuare di nuovo. L'HTML compreso tra le due parti del blocco viene stampato solo se il flusso del programma vi fa entrare in quel blocco. Questo può essere utile, in quanto vi fa, a volte, risparmiare istruzioni di stampa e rende più leggibile lo script.

Il seguente esempio illustra la stampa di un messaggio in una cornice centrata qualora una condizione sia vera, altrimenti non viene stampato nulla:

```
<% if (...): %>
<table cellspacing="0" cellpadding="1" border="0" align="center"
bgcolor="#548B54"><tr><td>

<table cellspacing="0" cellpadding="6" border="0" bgcolor="#FFFFFF">
<tr><td align="center">Ecco il messaggio incorniciato</td></tr>
</table>

</td></tr></table>
<% endif; %>
```

Qui si utilizza un trucco probabilmente noto a molti webmaster per controllare lo spessore di un bordo di tabella regolare senza utilizzare tag proprietari della microsoft. Si crea una tabella più esterna col colore di sfondo voluto per il bordo (nell'esempio il colore di X Window "PaleGreen4"), consistente di un'unica cella con all'interno un'altra tabella col colore di sfondo pari al colore che volete dentro la cornice. Regolando il cellpadding della tabella più esterna potete regolare lo spessore del bordo.

Poiché il giochino richiede parecchie istruzioni HTML, piuttosto che affidarsi ad una o più istruzioni echo, ho preferito saltare ad un blocco HTML, in modo da rendere il programma più leggibile. Naturalmente il programma precedente si sarebbe potuto scrivere anche utilizzando la sintassi tradizionale per l'`if`, comunque la sintassi con `endif` risulta più leggibile in questo caso.

```
<% if (...) { %>
... HTML di prima ...
<% } %>
```

A partire dalla versione 4, PHP supporta anche una sintassi particolare per specificare delle stringhe lunghe, copiata dal Perl e detta *HERE DOC(ument)* (letteralmente "eccolo qui il documento"). Secondo questa sintassi una stringa inizia con `«<IDENTIFICATORE`, dove `IDENTIFICATORE` è scelto dall'utente (scelte frequenti sono `EOD`, che sta per **E**nd **O**f **D**ocument, oppure `EOT` (**E**nd **O**f **T**ext) o `EOF` (**E**nd **O**f **F**ile)) e termina con IDENTIFICATORE, posto a sé stante su una linea e a partire dalla 1a colonna:

```
<% if (...)
      echo <<<EOD
<table cellspacing="0" cellpadding="1" border="0" align="center"
bgcolor="#548B54"><tr><td>

<table cellspacing="0" cellpadding="6" border="0" bgcolor="#FFFFFF">
<tr><td align="center">Ecco il messaggio incorniciato</td></tr>
</table>

</td></tr></table>
EOD;
%>
```

Un vantaggio della sintassi here doc in PHP rispetto alle virgolette doppie è che il testo all'interno viene analizzato come nel caso delle virgolette doppie (sequenze di escape e variabili vengono sostituite), ma siccome l'intera stringa non è racchiusa tra virgolette doppie, non è necessario escapare le virgolette doppie all'interno di un here doc (nell'esempio di sopra notate che infatti non le abbiamo escapate).

Abbiamo già visto come traversare un array con il costrutto `while` e la funzione `each` (eventualmente in combinazione con il costrutto `list`). PHP versione 4.0 offre persino un altro costrutto per facilitare questo compito, simile a quello presente in Perl. Ecco le sintassi e il codice con `while`, `list` ed `each` semanticamente equivalente a ciascuna:

```
1) foreach(array_expression as $value) statement
       ||
   $copy=array_expression;
   while (list(, $value) = each ($copy))
      statement
   unset($copy);

2) foreach(array_expression as $key => $value) statement
```

```
    ||
$copy=array_expression;
while (list($key, $value) = each ($copy))
   statement
unset($copy);
```

Le cose da notare sono:

- non è necessario eseguire un reset prima di usare il costrutto `foreach`, poiché esso opera su una copia dell'array e copiando un array in un'altro il puntatore all'elemento corrente del primo si resetta automaticamente nella copia `$copy`.
- poiché `foreach` opera su una copia, il puntatore all'elemento corrente dell'array percorso non viene modificato, mentre l'uso della funzione each come visto prima lo incrementa ogni volta che viene eseguita.
- la variabile `$copy` è stata usata per mostrare la semantica operazionale in termini di `while`, `list` ed `each`, ma naturalmente l'interprete usa una variabile non visibile al programma.
- la prima sintassi di `foreach` è utile se servono solo i valori, la seconda è utile quando servono sia i valori che le rispettive chiavi.

Seguono un paio di esempi tratti dalla documentazione PHP:

```
<%
/* foreach example: value only */

/* output is:
Current value of $a: 1.
Current value of $a: 2.
Current value of $a: 3.
Current value of $a: 17.
*/

 $a = array (1, 2, 3, 17);

 foreach ($a as $v)
    print "Current value of \$a: $v.<br>\n";
%>


<%
/* foreach example: key and value */
```

```
/* output is:
$a[one] => 1.
$a[two] => 2.
$a[three] => 3.
$a[seventeen] => 17.
*/

 $a = array (
     "one" => 1,
     "two" => 2,
     "three" => 3,
     "seventeen" => 17
 );

 foreach($a as $k => $v)
     print "\$a[$k] => $v.<br>\n";
%>
```

1.11 Riferimenti e Variabili variabili

In questo capitolo introduciamo due concetti tra i più trucchettosi del PHP.

I riferimenti sono stati introdotti a partire dalla versione 4 di PHP. Per capire il concetto considerate il seguente esempio:

```
$b = 5;
$a = $b; // $a e $b sono due variabili indipendenti
```

Quando assegnate una espressione (`$b` nell'esempio) ad una variabile, l'espressione viene valutata e il valore viene copiato nella variabile di destinazione. Dopo l'assegnazione `$a` e `$b` sono due variabili indipendenti: anche se per ora hanno lo stesso valore, cambiare il valore di una non influirà sul valore dell'altra e viceversa.

Tutto questo sembra ovvio, ma con i riferimenti non succede così. Quando si prepende il nome di una variabile con il simbolo `&` nel lato destro di una assegnazione si intende ottenere non il valore della variabile, bensì un riferimento ad essa, l'equivalente di quello che in C si chiama puntatore, seppur con alcune differenze e limitazioni rispetto ai puntatori del C, come vedremo. Se si assegna questo riferimento ad un'altra variabile, quest'ultima diventa un alias della prima o come si dice anche "punta" alla prima variabile. Le due variabili condividono lo stesso spazio di memorizzazione e quindi un cambiamento al valore dell'una si riflette subito sul valore dell'altra.

```
$b = 5;
$a = &$b; // $a ed $b sono la stessa variabile!
```

Per ragioni di leggibilità alcuni usano inserire uno spazio tra `&` e `$`:

```
$a = & $b;
```

oppure scelgono questo stile:

```
$a =& $b;
```

Ovviamente un valore costante non ha un riferimento, altrimenti sarebbe possibile modificare una costante vedendola come variabile, il che non ha senso e non serve (piuttosto si usi una variabile anziché una costante se la si intende modificare):

```
$b = &5; // errore
```

Al contrario del C, in PHP una variabile come `$a` non ha un tipo "puntatore ad integer", ma il tipo rimane ancora un integer:

```
<%
$b=5;
$a=&$b;
echo gettype($a); // stampa integer
%>
```

Traduco testualmente dalla documentazione di PHP: "`$a` e `$b` dopo `$a = & $b` sono esattamente la stessa cosa: non c'è tanto un `$a` che punta a `$b` o viceversa, piuttosto si deve dire che `$a` e `$b` puntano alla stessa locazione di memoria". Questa affermazione potrebbe sembrare strana per chi è abituato a pensare in termini di puntatori in C, ma è naturale alla luce del modo in cui i riferimenti sono implementati nell'interprete, ossia come *alias* nella tabella dei simboli.

I riferimenti del PHP non sono così flessibili come i puntatori in C. Essi sono semplicemente degli alias nella tabella dei simboli dell'interprete. Non esiste aritmetica dei puntatori in PHP e ad es. non è possibile stampare un indirizzo di riferimento. Essi sono più simili ai riferimenti del linguaggio C++.

Il modo migliore per assorbire questo concetto di riferimento è osservare che in realtà ogni nome di variabile è un riferimento al valore e che incidentalmente ci possono essere più nomi che si riferiscono allo stesso valore. Così se fate `unset` di una variabile, quello che accade è che la funzione `unset` spezza il legame tra un nome e il valore a cui si riferisce, ma solo per quel nome che avete specificato; lo spazio di memoria occupato dal valore viene liberato da `unset` solo se non ci sono altri riferimenti a quel valore:

```
<%
$a = 1;
$b =& $a;
unset ($a); // $b non perde il suo valore
```

```
echo "\$a=$a<br>\$b=$b"; // stampa "$a=<br>$b=1"
%>

<%
$a = 1;
$b =& $a;
unset ($b); // $a non perde il suo valore
echo "\$a=$a<br>\$b=$b";        // stampa "$a=1<br>$b="
%>
```

Qual'è l'utilità di creare riferimenti o alias per le variabili? Come vedremo in un linguaggio come PHP questi saranno utili soprattutto allorquando discuteremo il concetto di funzione. Un parametro di una funzione può essere un riferimento ad una variabile passata dal chiamante, anziché una sua copia e questo permette alla funzione di modificare la variabile passata. Questo tipo di modifica è detta in gergo *"effetto collaterale"*.

Usare i riferimenti per parametri di funzioni, se non ci sono "effetti collaterali" o è appropriato averli, è utile anche per ottimizzare l'interpretazione di alcune parti di codice, perché assegnare per riferimento è molto più veloce che usare la normale assegnazione per copia, ovviamente perché non viene effettuata nessuna copia di valori e viene copiato semplicemente un riferimento. In pratica la funzione condivide spazio di memoria già allocato dal chiamante, piuttosto che allocare per sé del nuovo spazio e copiarci i parametri del chiamante. Comunque, i benefici in termini di velocità si notano soprattutto nel caso di assegnazioni ripetute in ciclo di grossi array o oggetti.

Una variabile di variabile (o variabile variabile) è una variabile il cui nome è esso stesso una variabile. In pratica può essere usata come una specie di puntatore letterale, poiché permette di computare a runtime il nome della variabile a cui ci si riferisce. È questo il concetto del PHP probabilmente più vicino ai puntatori del C: un nome di variabile può essere usato per riferirsi alla variabile con quel nome, siccome un indirizzo di variabile può essere usato per riferirsi alla variabile tramite un puntatore in C.

Per utilizzare un nome di variabile variabile, basta semplicemente scrivere `$` e poi, al posto del nome di variabile, l'indicazione di un'altra variabile. Il seguente esempio mostra un ciclo il quale stampa tutte le variabili definite con i nomi `$a...$z`: invece di verificare con 26 `if` se tutte queste variabili esistono, ricorrendo al concetto di variabile di variabile si può ciclare, abbreviando notevolmente e generalizzando il programmino:

```
for ($varname="a"; $varname<="z"; $varname++)
   if (isset($$varname))
      echo "\$$varname = ${$varname}<br>";
```

Notate due cose nuove: l'incremento unitario di una stringa come `$varname` (`$varname++`) consiste nell'incrementare il codice ascii dell'ultimo carattere non cifra. Inoltre nella stringa argomento di `echo`, occorre scrivere `${$varname}`, e non `$$varname`, altrimenti vengono stampati i nomi delle variabili (nella forma `$h`, il primo `$` viene stampato come se fosse escapato) e non i valori. A questo proposito l'escapatura del primo `$` in `\$$varname` si poteva quindi omettere. Le parentesi graffe servono quindi a dire che nel secondo `$$varname`, la parte finale "`$varname`" va considerata come un tuttuno. L'interprete sostituirà prima la parte tra `{ }` con il contenuto della stringa `$varname` e poi prenderà in considerazione la variabile che ha quel nome e la sostituirà col suo valore[7].

Probabilmente avrete bisogno raramente delle variabili di variabili nella programmazione CGI di tutti i giorni; tuttavia è così semplice fornire una tale potente funzionalità in un linguaggio interpretato che i progettisti dell'interprete PHP non hanno resistito alla tentazione.

1.12 Funzioni

Quando una serie di compiti deve essere ripetuta più volte al variare di alcuni parametri, è conveniente definire questi compiti in una funzione, in modo da rendere il programma più breve, più leggibile e più facilmente modificabile. Allo scopo di aumentare la leggibilità e la riusabilità del codice spesso si definisce una funzione in un programma anche se viene usata una sola volta.

In PHP la sintassi di definizione di una funzione è del tipo:

```
function nome_funzione ([ [&]$arg[=espr] [,[&]$arg[=espr]]* ])
{
   [istruzione]*
}
```

Notate che l'insieme delle istruzioni che fanno parte di una funzione devono essere racchiuse tra `{ }`, anche se si tratta di una sola istruzione e che una funzione che non ha nessun argomento deve essere definita e chiamata con un nome seguito comunque da una coppia di parentesi tonde, come in C. Inoltre non si specificano i tipi degli argomenti che possono essere qualsiasi (anche array ed oggetti), anche se all'interno della funzione si possono forzare delle conversioni ricorrendo a casting o a settype:

```
<%
function foo($a)
{
   // forza l'argomento ad intero
   settype($a, integer);
```

[7] Le parentesi `{ }` sono usate in questo modo anche per poter referenziare array multidimensionali all'interno delle quote `" "`. Vedi a proposito la sezione *"Arrays"* del capitolo 6 *"Types"* del manuale di PHP.

```
    echo $a;
}

echo foo("101dalmata"); // stampa 101
%>
```

`[istruzione]` può essere in particolare "**return** *espr;*", nel senso che restituire un valore di ritorno è opzionale. Ogni tipo di valore può essere ritornato da una funzione (anche un array). Poiché si può ritornare una sola variabile, nel caso si vogliano ritornare più valori, un workaround ovvio è ritornare un array nell'istruzione `return` e poi raccogliere il valore di ritorno con un altro array o col costrutto `list`:

```
// funzione che ritorna valori multipli tramite un array
function returns_multiple_values()
{
    // computo dei valori $val1 e $val2
    ...
    return array($val1, $val2);
}

list ($val1, $val2) = returns_multiple_values();
// oppure
$val = returns_multiple_values();
```

Le funzioni in PHP possono essere nidificate, come in Pascal, ossia all'interno di una funzione possono apparire definizioni di altre funzioni e persino di oggetti.

Di default le variabili vengono passate per valore, come in C. Se una funzione cambia un parametro passato per valore, questi cambiamenti non si riflettono nel parametro del chiamante (ammesso che sia una variabile), poiché la funzione opera su una copia di tale parametro. Le variabili sono passate da sinistra verso destra; tuttavia scrivere codice che si avvantaggia di questo è sconsigliato, perché, anche se corretto, risulta poco chiaro.

Si specifica il passaggio per riferimento di un parametro facendolo precedere dal simbolo di e commerciale (`&`) nella definizione della funzione. Un tale parametro sarà sempre passato per riferimento. Questo implica naturalmente che solo una variabile può essere passata per riferimento[8]:

```
<%
function foo(&$par)
```

[8] Se non avete letto nel capitolo precedente la spiegazione sui riferimenti consiglio di farlo.

```
{
   echo "$par";
}

// Fatal error: Only variables can be passed by reference
foo("uffa");
// è corretto scrivere
foo($a="uffa");
%>
```

Il passaggio per riferimento di `$a` fa sì che la variabile locale `$par` della funzione `foo` venga inizializzata in questo modo:

```
$par =& $a;
```

Poiché PHP è un linguaggio interpretato, vi permette addirittura di forzare un passaggio per riferimento su un certo parametro anche quando la funzione era stata definita in modo da passare quel parametro per valore. Basta inserire il simbolo `&` davanti al nome della variabile da passare nella espressione di chiamata della funzione.

```
function foo ($bar)
{
   $bar .= ' and something extra.';
}

$str = 'This is a string, ';
foo ($str);
echo $str;    // outputs 'This is a string, '
foo (&$str);
echo $str;    // outputs 'This is a string, and something extra.'
```

Questa sintassi è in accordo con la sintassi di assegnazione di un riferimento:

```
$bar = &$str;
```

Il passaggio per riferimento è preferibile quando si devono passare grossi array o grossi oggetti e non è un problema (oppure si vuole proprio) che la funzione chiamata modifichi la variabile passata dal chiamante, oppure questa modifica non avviene.

I valori di default per un parametro si specificano come in C++, facendo seguire il nome del parametro da `=espressione_costante` nella dichiarazione della funzione. Il seguente è l'esempio di una funzione che calcola la media o la media pesata tra due numeri, in cui entrambi i pesi hanno il valore di default 1:

```
function mean($a, $b, $pa=1, $pb=1)
{
   return ($a*$pa + $b*$pb)/2;
}

echo mean(6,4); // stampa 5
echo mean(6,4,2); // stampa 8
echo mean(6,4,2,3); // stampa 12
echo mean(6,4,1,3); // stampa 9
```

Non è consigliabile assegnare dei valori di default ad un parametro e non assegnare poi un valore di default a tutti i parametri successivi nella dichiarazione. Se avessimo definito la funzione in questo modo:

```
function mean($a, $pa=1, $b, $pb=1)
{
   return ($a*$pa + $b*$pb)/2;
}
```

una chiamata del tipo:

```
echo mean(6,4);
```

pone `$a=6`, `$pa=4`, `$b` non settata, `$pb=1`, che non è proprio quello che volete, ma PHP assegna sempre i valori dei parametri da sinistra verso destra in ordine, sia che abbiamo valori predefiniti che non. Si ottiene in output qualcosa del tipo:

```
Warning: Missing argument 3 for mean() in
/usr/local/apache/htdocs/prova.php on line 7
12
```

Poiché non c'è modo di assegnare `$b` senza saltare l'assegnazione di `$pa`, è stato inutile assegnare il valore di default a `$pa`. Seguite sempre la regola di dichiarare ogni argomento con valore di default dopo quello senza valore di default.

In PHP, dove tra l'altro non ci sono dichiarazioni di variabili, le regole di *scoping* (o visibilità delle variabili) sono leggermente diverse da quelle del linguaggio C o del Pascal. Lo scope (ambito di visibilità) di una variabile è il contesto in cui questa è definita. Di default tutte le variabili in PHP hanno uno scope singolo.

Una variabile usata al di fuori di una funzione ha scope globale e basta e non è disponibile di default all'interno di una funzione definita dall'utente, perché lo scope globale non è ereditato implicitamente dalle funzioni come in C. Questa è una misura di sicurezza per impedire che per errore una funzione definita dall'utente cambi inavvertitamente il valore di una variabile globale. Quindi quando usate una variabile all'interno di una funzione viene ricercata per default solo nello scope locale della funzione, mentre in C viene cercata prima nello scope locale e poi, se non vi è trovata, viene implicitamente ricercata nello scope globale. Tutte le variabili globali sono accessibili ad una funzione C se dichiarate o definite prima della funzione, anche se la loro definizione o dichiarazione è esterna alla funzione stessa.

```
$a = 1; /* global scope */

Function Test () {
    echo $a; /* riferimento ad una variabile nello scope locale */
}

Test (); // non stampa niente, $a non è settata nello scope locale
```

Il programma precedente non stampa niente, perché `$a` viene ricercata solo nello scope locale, dove non è settata. In C invece una variabile globale come `$a`, definita o dichiarata prima di una funzione, oppure dichiarata all'interno di una funzione (come variabile extern) è automaticamente disponibile alla funzione, a meno che non venga mascherata da una definizione locale con lo stesso nome. I programmatori C sappiano che devono utilizzare sempre l'equivalente di extern del C in PHP, che è il costrutto `global`, all'interno delle funzioni per importare variabili dallo scope globale in quello locale se intendono utilizzarle; poiché lo scope globale non viene ereditato nello scope locale, global è sempre necessaria per accedere a variabili globali nello scope locale.

```
$a = 1; /* global scope */

Function Test () {
    global $a;
    echo $a; /* riferimento ad una variabile nello scope globale */
}

Test ();        // stampa 1
```

Nel caso dobbiate accedere a più variabili, basta specificarne la lista separata da virgole nel costrutto global, anziché usare più costrutti global:

```
global var1, var2, ..., varn;
```

Naturalmente purché la variabile globale venga settata prima della chiamata della funzione, la funzione non ha problemi a vederla:

```
Function Test () {
    global $a;
```

```
    echo $a; /* riferimento ad una variabile nello scope globale */
}

$a = 1; /* global scope */
Test ();        // stampa 1

#######################################################

Function Test () {
    global $a;
    echo $a; /*  riferimento ad una variabile
              */ non settata nello scope globale
}

Test ();        // non stampa niente
$a = 1; /* global scope */
```

Un'alternativa all'utilizzo del costrutto global per accedere a variabili globali da uno scope locale consiste nell'uso dell'array (associativo) `$GLOBALS`. In pratica in uno scope locale PHP importa di default un array associativo globale in cui memorizza tutti i riferimenti alle variabili globali create dal programma; in effetti quando si importa una variabile con:

```
global $var;
```

questo equivale a creare un riferimento ad un elemento omonimo dell'array

```
$GLOBALS:
$var =& $GLOBALS["var"];
```

Questo array contiene le coppie nome=>valore di tutte le variabili globali, quindi l'esempio precedente si può riscrivere in questo modo:

```
$a = 1; /* global scope */

Function Test () {
    echo $GLOBALS["a"]; /*  riferimento ad una
                        */ variabile nello scope globale
}

Test ();        // stampa 1
```

Le variabili con scope locale sono normalmente disallocate dall'interprete quando l'interpretazione di una funzione finisce, allo scopo di economizzare lo spazio di memoria, così ogni volta che viene chiamata una funzione, una variabile locale presenta un valore non settato.

Poniamo il caso in cui si debba utilizzare una variabile solamente all'interno di una funzione e si voglia che il suo valore venga conservato tra una chiamata e l'altra della funzione, come nel caso banale di una funzione che conta il numero di volte che è stata chiamata. Una possibile soluzione è utilizzare allo scopo una variabile globale ed importarla nello scope locale con global:

```
function counts_itself()
{
   global $i;
   return ++$i;
}

echo counts_itself(); // stampa 1
echo counts_itself(); // stampa 2
echo counts_itself(); // stampa 3
```

Questo metodo funziona, ma presenta l'inconveniente che la variabile, avendo scope globale, può essere accidentalmente alterata da codice presente in questo scope e persino da funzioni che per sbaglio la importino. Il nome scelto per la variabile è così comune come valore indice che quasi certamente la variabile è esposta a questo rischio.

La soluzione consiste nel conferirle scope locale, ma tempo di vita (lifetime) globale tramite una dichiarazione `static`, che è del tutto analoga a quella del C:

```
function counts_itself()
{
   static $i=0;
   return ++$i;
}

echo counts_itself();   // stampa 1
echo counts_itself();   // stampa 2
echo counts_itself();   // stampa 3
```

In PHP non è necessario inizializzare la variabile scrivendo =0 dopo static `$i`, ma è comunque bene farlo per ragioni di chiarezza del codice. In questo modo si può avere una variabile globale `$i` che non avrà niente a che fare con la `$i` vista da `counts_itself` e nello stesso tempo far sì che quest'ultima sia permanente per tutta la durata del programma.

Inoltre in PHP le `{}` che delimitano un blocco non iniziano un nuovo ambito come in C. Questo programma stamperà quindi 5:

```
<%
function prova()
{
   $a=5;
   {
      echo $a;
   }
}

prova();
%>
```

mentre questo stamperà due volte 6 (66):

```
<%
function prova()
{
   $a=5;
   {
      $a=6;
      echo $a;
   }
   echo $a;
}

prova();
%>
```

Infine PHP supporta la ricorsione (una funzione può richiamare se stessa per svolgere un compito ricorsivo – avremo presto occasione di apprezzare questa proprietà), e come il C supporta la dichiarazione di funzioni con un numero variabile di argomenti a partire della versione 4 e infine come il C++ supporta la restituzione di un riferimento da una funzione. Spiegherò qualcosa su queste feature.

Il seguente è un esempio di funzione definita dall'utente con un numero variabile di argomenti: si tratta di una funzione che ritorna il valore medio degli elementi di un array (anche associativo) se usata con un solo argomento. Se il secondo argomento viene specificato, rappresenta il numero di elementi di cui si deve calcolare la media.

```
// restituisce la media dei primi $dim elementi
// dell'array $dim è opzionale: se manca si considera
// la dimensione dell'intero array

function array_mean($array, $dim)
```

```
{
  $sum=0;

  if (func_num_args()>1)
     $n=func_get_arg(1);
  else
     $n=sizeof($array);

  for (reset($array), $i=$n; $i>0; next($array), $i--)
     $sum += current($array);

  return $sum / $n;
}
```

Avremmo potuto scrivere il prototipo della funzione anche omettendo la dichiarazione di `$dim` (il corpo rimane invariato), ma io preferisco mantenere la dichiarazione di tutti i parametri, anche quelli opzionali per ragioni di leggibilità. Notate che questa funzione funziona anche con array associativi del tipo:

```
$gol_segnati = array("Juventus"=>30,
                     "Fiorentina"=>"20gol",
                     "Milan"=>30);
```

e la conversione dei valori da stringa a numero è automatica. Notate che passare un numero di argomenti maggiore rispetto a quello richiesto ad una funzione definita dall'utente non è un errore in php: gli argomenti inutili vengono semplicemente ignorati, a meno che la funzione definita dall'utente non si prenda la briga di fare un controllo ed emettere un warning (cosa che andrebbe fatta e che io non ho fatto sopra, la lascio per esercizio), come succede per le funzioni predefinite.

Notare che la funzione `array_mean` precedente non avrebbe potuto essere implementata usando un parametro con valore di default, poiché c'è la restrizione che questo deve essere una espressione costante:

```
function array_mean($array, $dim=sizeof($array)) // errore
```

Per maggiori dettagli sulle funzioni con un numero qualsiasi di argomenti, consultare la documentazione di PHP[9]:

Per ritornare per riferimento un valore da una funzione, occorre usare l'operatore di riferimento nella dichiarazione della funzione. Quando poi si assegna il valore di ritorno ad una variabile occorre ancora utilizzare l'operatore di riferimento:

[9]function.func-num-args.html function.func-get-arg.html function.func-get-args.html

```
function &returns_reference() {
    ...
    return $someref;
}

$newref = &returns_reference();
```

In pratica il meccanismo di ritorno di un riferimento da una funzione serve per far sì che sia la funzione a stabilire a quale variabile creare il riferimento. È necessario indicare nel prototipo della funzione che intendiamo ritornare per riferimento perché ovviamente il comportamento di default è di ritornare per copia. È necessario scrivere:

```
$newref = &returns_reference();
```

e non semplicemente:

```
$newref = returns_reference();
```

perché vogliamo creare una variabile `$newref` che sia l'alias della variabile indicata dalla funzione. Piuttosto che contenerne una copia del valore, vogliamo che ne condivida l'indirizzo di memoria.

Le funzioni in PHP sono abbastanza flessibili, ma presentano comunque alcune limitazioni, tuttosommato non gravi, ma che ricordo a beneficio di chi provenga da altri linguaggi:

- non è possibile l'overloading del nome di funzione, cioè non è possibile avere funzioni che hanno lo stesso nome ma lista di argomenti diversa (come in C++)

- non è possibile annullare la definizione di una funzione già definita o ridefinirla.

PHP versione 3 presenta poi queste ulteriori limitazioni, che invece non sussistono a partire dalla versione 4:

- prima di poter chiamare una funzione occorre che sia definita; è un errore chiamare una funzione definita dopo del punto di chiamata.

- non supporta funzioni con un numero variabile di argomenti, ma solo argomenti con valore di default. Un workaround consiste nel passare un array con gli argomenti alla funzione, il quale può contenere un numero qualsiasi di elementi di ogni tipo.

1.13 Accesso a file

PHP comprende una nutrita serie di funzioni per la gestione del filesystem (dal lato server, ovviamente ;) da fare invidia alla libreria di I/O standard del C `<stdio.h>`. Sui sistemi UNIX, PHP provvede anche dei wrapper per accedere ad alcune funzioni del kernel per la gestione del filesystem a basso livello[10].

Prima di poter operare su un file bisogna aprirlo nella modalità opportuna. La funzione `fopen` di PHP è molto simile alla omonima del C:

> int **fopen** (string *filename*, string *mode* [, int *use_include_path*])

Questa funzione ritorna un piccolo intero non negativo che può essere usato per riferirsi al file nel seguito ed è detto puntatore al file. Se qualcosa dovesse andare storto (ad es. il processo del server non ha il permesso di accedere al file o di crearlo, oppure state cercando di aprire in modalità 'r' un file che non esiste) `fopen` ritorna false. Questo intero ritornato da `fopen` è l'analogo del descrittore di file ritornato dalla system call `open(2)` di UNIX. Di solito il valore ritornato non è interessante di per sé, interessa solo salvarlo in una variabile (di solito si assegna a questa variabile un nome descrittivo come `$fp` (file pointer) o `$fd` (file descriptor), ma ovviamente potete scegliere il nome che volete).

Il modo può essere uno dei seguenti:

- `'r'` – Apre il file per sola lettura; posiziona il puntatore del file all'inizio. Se il file non esiste, è errore.
- `'r+'` – Apre il file in lettura e scrittura; posiziona il puntatore del file all'inizio. Se il file non esiste, è errore.
- `'w'` – Apre il file per sola scrittura; posiziona il puntatore del file all'inizio e tronca il file a lunghezza zero. Se il file non esiste, prova a crearlo.
- `'w+'` – Apre il file per lettura e scrittura; posiziona il puntatore del file all'inizio del file e tronca il file a dimensione zero. Se il file non esiste, tenta di crearlo.
- `'a'` – Apre il file per sola scrittura; posiziona il puntatore nel file alla fine del file. Se il file non esiste, prova a crearlo.
- `'a+'` – Apre il file per lettura e scrittura; posiziona il puntatore alla fine del file. Se il file non esiste, prova a crearlo.

Come potete arguire facilmente il flag `+` aggiunge la possibilità di fare sia lettura che scrittura.

Se il parametro mode contiene anche la lettera `b` (ad es. `"rb"`) il file viene considerato come file binario. Sui sistemi UNIX non c'è differenza tra file binari e

[10]La sezione del manuale che tratta di questi argomenti è: *IV. Function Reference XX. Filesystem functions.*

file di testo, quindi la `b` viene ignorata. Nei sistemi Windows invece in un file di testo una coppia di caratteri `CR+LF` viene vista come un unico carattere di finelinea, nei file binari vengono visti come due caratteri distinti, di codici ASCII rispettivamente 13 e 10 (valori decimali). Sotto UNIX il carattere di finelinea è unico ed è `LF`.

La funzione `fopen` è network-trasparente, nel senso che potete aprire anche una pagina HTML situata su qualche computer della rete, specificando un filename che inizia con `http://`. PHP si occupa di aprire una connessione HTTP al server specificato e di ritornare un file pointer al testo della risposta. Ad es. il seguente comando apre in lettura un file di testo presente su un'altro host, col nome immaginario `paride.dei.it`:

```
$fd=fopen("http://paride.dei.it/giudizio.txt", "r");
```

questo apre un file dal server ftp locale (supponendo che questo supporti il trasferimento passivo di file e permetta di scrivere il file all'utente anonymous):

```
$fd=fopen("ftp://localhost/pub/README", "w");
```

questo apre in lettura il file dati della sottodirectory `"data"` rispetto a quella in cui è posizionato lo script che contiene il comando:

```
$fd=fopen("data/dati", "r");
```

questo apre in lettura il file

`/home/ant/.netscape/bookmarks.html`

contenente i bookmark di **lynx**, che l'utente **ant** ha magnanimamente deciso di rendere accessibile in lettura a tutti, quindi anche all'utente sotto cui gira il server web (tanto per esemplificare l'uso di un path assoluto sul server):

```
$fd=fopen("/home/ant/lynx_bookmarks.html", "r");
```

Le principali funzioni per leggere da un file, in ordine di dimensione dei dati che leggono in una chiamata, sono:

string **fgetc** (int *fp*);
: legge un carattere alla volta dal file puntato da fp e ritorna una stringa contenente un solo carattere, oppure FALSE se incontra EOF, ossia la fine del file (End Of File).

string **fgets** (int *fp*, int *length*);
: legge una intera riga (fino a length-1 bytes) dal file puntato da *fp*. La lettura termina quando sono stati letti length-1 bytes, oppure se viene incontrato un newline (che viene incluso nella stringa ritornata), o se viene incontrato EOF, a seconda di quello che capita prima. In caso di errore oppure se viene incontrata la fine del file questa funzione ritorna false, anziché una stringa.

int **readfile** (string *filename* [, int `use_include_path`]);
apre il file col nome specificato, lo legge e lo stampa su standard output e poi lo chiude. È network-trasparente. Ritorna il numero di byte letti dal file.

array **file** (string *filename* [, int `use_include_path`]);
legge un intero file e restituisce un array in cui ogni elemento è una riga del file (comprensiva del newline).

Un esempio classico è la lettura di un certo file linea per linea. Il manuale consiglia in generale il seguente codice.

```
$fd = fopen ("/tmp/inputfile.txt", "r");
while (!feof ($fd)) {
    $buffer = fgets($fd, 4096);
    echo $buffer;
}
fclose ($fd);
```

Viene usata la funzione:

int **feof** (int *fp*)

che ritorna true se il puntatore del file è oltre la fine del file (o come si dice è su EOF (End Of File)), oppure se si è verificato un errore, altrimenti ritorna false.

Potreste anche pensare di abbreviare le cose in questo modo:

```
$fd = fopen ("/tmp/inputfile.txt", "r");
while ($buffer = fgets($fd, 4096))
    echo $buffer;
fclose ($fd);
```

ma c'è il rischio di non beccare l'ultima linea nel caso si valuti a false e non sia seguita da un carattere di EOL, poiché quando non c'è EOL una tale linea si valuta a false. Per capire il problema provate a creare questo file:

```
echo -n 0 >/tmp/inputfile.txt
```

avviate lo script precedente e vedrete che non stampa nulla, mentre la versione del manuale visualizza uno zero. Se siete sicuri che ogni linea del file è terminata da newline (come la maggior parte degli editor assicurano), potete usare il valore di ritorno di `fgets` come condizionale, altrimenti usate la funzione `feof()`.

Nel caso dobbiate semplicemente leggere un intero file per stamparlo, potete anche affidarvi alla funzione già pronta `readfile`:

```
readfile("/tmp/inputfile.txt");
```

Se un programma deve effettuare una elaborazione sequenziale su un grosso file, per economizzare l'uso della memoria, è preferibile leggere il file linea per linea, piuttosto che leggerlo tutto in memoria con la funzione file, la quale invece è utile per elaborazioni di un intero file di natura non sequenziale da svolgere interamente in memoria.

Le funzioni `fopen`, `readfile` e `file` accettano un parametro opzionale che, se vale 1 indica di cercare il file anche nelle locazioni specificate nella direttiva `include_path` del file di configurazione `php.ini`, nel caso non venga trovato nel path specificato. La direttiva `include_path` ha valore di default `.` e segue la stessa sintassi usata nella variabile di ambiente UNIX PATH per specificare i percorsi di ricerca, che vengono provati da sinistra verso destra.

È buona norma chiudere i file non appena non servono più, usando la funzione:

int **fclose** (int *fp*)

che riceve come parametro il descrittore del file da chiudere e restituisce true in caso di successo e false in caso di errore (in quest'ultimo caso non è che ci sia molto da fare, tranne che avvisare l'utente dell'anomalia). Al termine dello script comunque tutti i file vengono chiusi automaticamente.

Per scrivere su un file potete usare la funzione:

int **fputs** (int *fp*, string *str* [, int *length*])

che scrive sul file puntato da *fp* l'intera stringa *str*, oppure i primi length bytes se viene specificato il parametro opzionale *length*. Se viene specificato *length* e la stringa è più corta di length bytes, ci si ferma comunque alla fine della stringa.

Queste funzioni sono sufficienti per gestire l'I/O su file visti come una successione di linee, che è il caso più comune. Nei casi più rari in cui state trattando dei file "binari", ossia non strutturati a linee, è più conveniente utilizzare queste altre due funzioni per l'I/O:

string **fread** (int *fp*, int *length)*

`fread` è analoga a `fgets`, solo che legge esattamente fino a length bytes (e non length-1 come `fputs`) e non si ferma quando legge un newline, ma solo quando sono stati letti length bytes, oppure se sopraggiunge la fine del file (EOF).

int **fwrite** (int *fp*, string *string* [, int *length*])

questa è del tutto analoga a `fputs` vista pocanzi.

Infine PHP permette anche di accedere casualmente ai contenuti di un file aperto, posizionando il puntatore alla posizione corrente del file nella posizione voluta. Le funzioni che si utilizzano a questo scopo sono `fseek`, `ftell`, `rewind`. Se poi avete bisogno di creare file temporanei, sono a disposizione le funzioni `tmpnam`, `tmpfile`: è essenziale usare queste funzioni in un ambiente multitasking per evitare problemi. Tenete conto infatti che potrebbero esserci più copie del vostro script in esecuzione concorrente, quindi se due di queste copie creano ed usano lo stesso file (temporaneo o no), possono succedere casini apparentemente inspiegabili. Il modo migliore per capire il problema è simularlo.

Predisponete questi due script nella document root del vostro server:

```
<%
// test.php
$fd=fopen("file","w+");
fputs($fd, "test");
echo "scritto test nel file";
sleep(30);
rewind($fd);
echo "<p>riletto ". fgets($fd, 100). " nel file";
echo "<p>fclose ha ritornato: ". fclose($fd);
%>
```

```
<%
// prova.php
$fd=fopen("file","w+");
fputs($fd, "prova");
echo "scritto prova nel file";
rewind($fd);
echo "<p>riletto ". fgets($fd, 100). " nel file";
echo "<p>fclose ha ritornato: ". fclose($fd);
%>
```

Ciascuno di questi script accedono allo stesso file chiamato "`file`" e lo aprono in lettura/scrittura (modalità "`w+`", tentando di crearlo se non esiste e troncandolo se esiste già. Poi il primo script ci scrive dentro la stringa "`test`", mentre il secondo ci scrive "`prova`", ossia ci scrivono il nome del rispettivo script. Poi il primo script attende 30 secondi, mentre il secondo non perde tempo. Infine entrambi gli script riposizionano il puntatore all'inizio del file e rileggono ciò che hanno scritto (si spera). Infine chiudono il file con `fclose` e visualizzano il valore di ritorno di `fclose`.

Assicuratevi che ogni script possa creare un file con questo nome nella stessa directory in cui risiedono. Se non volete concedere il permesso generale di scrittura in quella directory all'utente su cui gira il server web (solitamente nobody), potete creare un file di nome file con:

```
touch file
```

e assegnargli permesso di riscrittura a tutti con:

```
chmod a+w file
```

Se provate con un browser ad eseguire non concorrentemente i due script, noterete che entrambi funzionano a dovere e che non c'è nessun problema. Il primo script ritorna l'output (dopo aver atteso circa 30 secondi):

```
scritto test nel file

riletto test nel file

fclose ha ritornato: 1
```

mentre il secondo:

```
scritto prova nel file

riletto prova nel file

fclose ha ritornato: 1
```

Provate però successivamente a fare una cosa del genere: con il browser mandate in esecuzione il primo script `test.php`. Poi subito dopo, e comunque entro la terminazione del primo script, aprite un'altra finestra (con netscape basta premere <Alt>n) e mandate in esecuzione anche il secondo `prova.php`. Appare subito l'output del secondo script che dovrebbe essere del tipo:

```
scritto prova nel file

riletto prova nel file

fclose ha ritornato: 1
```

quindi passate ad attendere l'output del primo script che sorprendentemente sarà:

```
scritto test nel file

riletto testa nel file

fclose ha ritornato: 1
```

se avevamo scritto `"test"`, come diavolo è che rileggiamo `"testa"`? Sembra incredibile, ma la ragione è che mentre il primo processo (`test.php`) dorme, il secondo processo (`prova.php`) ha riscritto lo stesso file che teneva aperto il primo processo con `"prova"`. Quando il processo `test.php` riprende l'esecuzione, la sua istruzione `rewind` causa lo svuotamento dei buffer, in pratica viene scritto `test` nel file sovrascrivendo ciò che vi aveva scritto l'altro processo. Se scrivete `"test"` sopra `"prova"` lettera per lettera, il risultato è `"testa"`, ed è questa la stringa che viene letta. Se spostate l'istruzione `rewind` prima di `sleep(30)` nel file `test.php` e ripetete la simulazione allo stesso modo, quello che vedrete come output dello script `test.php` sarà addirittura:

```
scritto test nel file

riletto prova nel file

fclose ha ritornato: 1
```

Questo perché `rewind` ha provocato lo svuotamento dei buffer di scrittura, prima che venga eseguito `prova.php` che sovrascrive il file non ancora chiuso da `test.php`.

Morale della favola: poiché gli script sono eseguiti in un ambiente multitasking, occorre prendere delle precauzioni quando accedono a particolari risorse condivise. Se uno script ha bisogno di un file temporaneo per svolgere un certo lavoro, meglio crearlo con un nome unico, usando le funzioni `tmpfile` e `tmpnam`, piuttosto che usare un nome di file costante. Se uno script ha bisogno di aggiornare un file, è opportuno che questi aggiornamenti siano atomici. I sistemi operativi multitasking forniscono meccanismi di locking sui file, ma non c'è uno standard, perciò PHP definisce una propria funzione per attuare il locking. Se tutti gli script che condividono una risorsa, come ad es. l'aggiornamento di un file, usano `flock` per bloccare l'accesso ad altri al file, e dopo aver finito col file ed averlo chiuso, tolgono il blocco (un blocco viene comunque chiuso quando si chiama `fclose` sul file o il programma termina), non si possono verificare casini: quello che ha bloccato il file deve prima terminare il suo lavoro e solo dopo un altro script può bloccare il file e portare a termine il suo aggiornamento, che pure non può essere interrotto. Infatti `flock` mette in attesa uno script che tenta di *"lockkare"* un file già *"lockkato"* da un'altro, finché il primo non lo *"unlockka"*. In pratica per evitare la sitazione di conflitto, basta aggiungere un'istruzione `flock` ad entrambi i programmi precedenti subito dopo la `fopen`:

```
<%
        // test.php
$fd=fopen("file","w+");
echo "flock ha ritornato: ". flock($fd, LOCK_EX);
fputs($fd, "test");
echo "scritto test nel file";
sleep(30);
rewind($fd);
echo "<p>riletto ". fgets($fd, 100). " nel file";
```

```
echo "<p>fclose ha ritornato: ". fclose($fd);
%>

<%
        // prova.php
$fd=fopen("file","w+");
echo "flock ha ritornato: ". flock($fd, LOCK_EX);
fputs($fd, "prova");
echo "scritto prova nel file";
rewind($fd);
echo "<p>riletto ". fgets($fd, 100). " nel file";
echo "<p>fclose ha ritornato: ". fclose($fd);
%>
```

se ora avviate la simulazione come prima, noterete che `prova.php` si blocca attendendo che `test.php` tolga il blocco dal file. Alla fine la versione del file "`file`" che rimane è quella creata da chi ha terminato l'esecuzione per ultimo, cioè `prova.php`, quindi il file "`file`" conterrà sempre la stringa "`prova`".

1.14 Oggetti e classi

Assumerò che conosciate la programmazione ad oggetti e vi propongo questa volta una traduzione letterale dal manuale di PHP. PHP implementa il paradigma degli oggetti in modo piuttosto semplice, perciò se provenite da linguaggi in cui le cose sono molto più complicate (come ad es. nel C++), sarà facile per voi capire come programmare ad oggetti in PHP. In PHP non c'è modo di creare nuovi tipi di dati che non siano classi (non esiste il concetto di struttura come in C). In fondo le classi sono una estensione del concetto di struttura, quindi potete utilizzare le classi come semplici strutture se proprio non volete programmare ad oggetti: basta che non vi incapsuliate nessuna funzione.

Occorre notare che la maggior parte delle librerie di funzioni predefinite in PHP sono imperniate sul meccanismo della programmazione procedurale, quindi PHP si configura piuttosto come un linguaggio misto procedurale-oggetti, piuttosto che come linguaggio solo ad oggetti. Per la programmazione web questo secondo me va benissimo. La virtù (a volte) sta nel mezzo.

1.14.1 Classi e Oggetti, (traduzione Capitolo 13 manuale di PHP)

class

Una classe è una collezione di variabili e funzioni che lavorano con queste variabili. Una classe viene definita usando la seguente sintassi:

```
<?php
class Cart {
    var $items; // Articoli nella nostra shopping bag

    // Aggiunge $num articoli di tipo $artnr alla shopping bag

    function add_item ($artnr, $num) {
$this->items[$artnr] += $num;
    }

    // Elimina $num articoli di tipo $artnr dalla shopping bag

    function remove_item ($artnr, $num) {
if ($this->items[$artnr] > $num) {
    $this->items[$artnr] -= $num;
    return true;
} else {
    return false;
}
    }
}
?>
```

Questo definisce una classe chiamata `Cart` che consiste di un array associativo di articoli nella borsa e due funzioni per aggiungere e rimuovere articoli da questa borsa virtuale della spesa.

Le classi rappresentano dei tipi, cioè sono gli *"stampi"* con i quali si costruiscono le vere variabili. Dopo aver definito una classe per usarla devi creare una variabile del tipo desiderato con l'operatore `new`.

```
$cart = new Cart;
$cart->add_item("10", 1);
```

Questo codice crea un oggetto `$cart` della classe `Cart` e poi viene chiamata la funzione `add_item()` di quell'oggetto per aggiungere la quantità 1 dell'articolo numero 10 alla borsa.

Le classi possono essere estensioni di altre classi (concetto noto col nome di ereditarietà o derivazione o estensione). Le classi *"estese"* o *"derivate"* hanno tutte le variabili e le funzioni della classe base e in più quello che viene aggiunto nella definizione estesa. Una classe estesa si dichiara usando la parola chiave `extends`. L'ereditarietà multipla non è supportata in PHP.

```
class Named_Cart extends Cart {
    var $owner;
```

```
    function set_owner ($name) {
$this->owner = $name;
    }
}
```

Viene definita una classe `Named_Cart` che ha tutte le variabili e le funzioni di `Cart` e in più una variabile addizionale `$owner` e una ulteriore funzione `set_owner()`. Si crea una borsa della spesa con nome nello stesso modo in cui si creava la borsa anonima e si può settare e ottenere il nome del possessore della borsa. Sulle borse della spesa con nome si possono ancora usare le normali funzioni definite per le borse della spesa anonime:

```
$ncart = new Named_Cart;       // Crea una borsa con nome
$ncart->set_owner ("kris");    // Assegna il nome a quella borsa
print $ncart->owner;           // stampa il nome del possessore
                               // della borsa
$ncart->add_item("10", 1);     // (funzionalità ereditata da cart)
```

All'interno delle funzioni della classe la variabile `$this` indica l'oggetto su cui è invocata la funzione. Si usa la sintassi `$this->something` per accedere ad una qualunque variabile o funzione chiamata `something` all'interno dell'oggetto corrente[11].

I costruttori sono funzioni di una classe che vengono chiamati automaticamente quando viene creata una nuova istanza della classe. Una funzione diventa un costruttore quando ha lo stesso nome della classe.

```
class Auto_Cart extends Cart {
    function Auto_Cart () {
$this->add_item ("10", 1);
    }
}
```

Abbiamo definito una classe `Auto_Cart` che rappresenta una `Cart` con in più un costruttore che inizializza la cart con la quantità 1 dell'articolo numero "10" ogni volta che viene creato un nuovo oggetto di tipo `Auto_Cart` con l'operatore "`new`". I costruttori possono avere anche degli argomenti e questi argomenti possono avere dei valori di default, il che li rende molto più flessibili.

```
class Constructor_Cart extends Cart {
    function Constructor_Cart ($item = "10", $num = 1) {
```

[11] In un metodo di un oggetto, `$this` è sempre un riferimento all'oggetto che chiama il metodo. Vedere il capitolo sui riferimenti se non si sa cos'è un riferimento in PHP.

```
$this->add_item ($item, $num);
    }
}

// Riempie la borsa con la solita cosa.

$default_cart = new Constructor_Cart;

// Inizializza la borsa con qualcosa di diverso

$different_cart = new Constructor_Cart ("20", 17);
```

Attenzione
Nelle classi derivate, il costruttore della classe genitore non viene chiamato automaticamente quando viene chiamato il costruttore della classe derivata, perciò dovete chiamarlo manualmente voi all'interno del costruttore della classe derivata se desiderate sfruttarne i servigi.

Le definizioni di classe non possono essere nidificate ma possono apparire anche all'interno di una funzione.

L'assegnazione di un oggetto ad un altro o il passaggio di un oggetto come argomento di una funzione, di default avviene per copia. Tutti i membri dati dell'oggetto vengono copiati nell'oggetto destinazione, come mette in evidenza il programma seguente:

```
<%
class test {
   var $num;
}

$t = new test;
$t->num = 7;

echo "\$t->num => $t->num<BR>";         // stampa: $t->num => 7

$t2 = $t;       // copia di oggetti

echo "\$t2->num => $t2->num<BR>";       // stampa: $t2->num => 7

$t2->num=5;

echo "<P>\$t->num => $t->num<BR>";      // stampa: $t->num => 7
echo "\$t2->num => $t2->num<BR>";       // stampa: $t2->num => 5
%>
```

se invece sostituite nel programma precedente l'istruzione:

```
$t2 = $t;
```

con:

```
$t2 =& $t;
```

l'output anziché:

```
$t->num => 7
$t2->num => 7

$t->num => 7
$t2->num => 5
```

risulta:

```
$t->num => 7
$t2->num => 7

$t->num => 5
$t2->num => 5
```

poiché gli oggetti `$t` e `$t2` convidono lo stesso spazio di memorizzazione.

PHP, essendo un linguaggio interpretato, rende molto semplice ad un programma ottenere informazioni sulle classi definite[12].

Infine un ultimo esempio. Per misurare i tempi di esecuzione degli script allo scopo di confrontare in pratica l'effetto di due versioni dello stesso script per capire qual'è la più ottimizzata, mi servo della classe seguente, la quale implementa il comportamento di un cronometro con i classici tasti LAP/RESET e START/STOP. L'implementazione di tali funzioni come metodi della classe si basa sulla funzione `time()` che ritorna il timestamp di UNIX, ossia ritorna la data corrente misurata come numero di secondi trascorsi dall'inizio dell'Epoca di Unix, fissata al 1 Gennaio 1970 ore 00:00:00 GMT. Tutte le funzioni membro restituiscono semplicemente un numero di secondi.

```
chronometer.inc
---------------
<?php

class chronometer
```

[12]Per approfondire questo aspetto vedere la sottosezione *"VIII. Class/Object Functions"* della sezione *"IV. Function Reference" del manuale.*

```
{
   var $start_time, $stop_time;

   function chronometer()
   {
      $this->reset();
   }

   function reset()
   {
      return $this->start_time = $this->stop_time = 0;
   }

   function start()
   {
      $this->start_time += time() - $this->stop_time;
      return $this->stop_time;
   }

   function lap()
   {
      return time() - $this->start_time;
   }

   function stop()
   {
      return $this->stop_time=time() - $this->start_time;
   }
}

?>
```

La risoluzione di questo cronometro non è molto elevata (essendo di ben 1 secondo), e inoltre esso non tiene conto del tempo di esecuzione del solo processo, ma del tempo complessivo di esecuzione, comunque è sufficiente per fare delle prove su programmi che impiegano alcuni secondi di esecuzione. Ecco un esempio di come impiegare tale classe: questo script crea un nuovo oggetto cronometro, che all'inizio è automaticamente resettato. Avvia quindi il cronometro (che restituisce il valore 0 all'inizio, proprio come un vero cronometro). Dopo 3 secondi effettua una lettura senza fermare il conteggio (che restituisce ovviamente il valore 3). Dopo 5 secondi dall'avvio ferma il cronometro, che restituirà quindi il valore 5. Attende ancora un secondo e poi avvia di nuovo il cronometro (sul cui "visore" si leggerà il valore 5) e così via... insomma si usa proprio come un vero cronometro sportivo ;)

```
<?php
   require 'chronometer.inc';

   /* output:
start: 0
lap: 3
stop: 5
start: 5
   */
   $chrono = new chronometer;
   echo 'start: ', $chrono->start(), '<br>';
   sleep(3);
   echo 'lap: ', $chrono->lap(), '<br>';
   sleep(2);
   echo 'stop: ', $chrono->stop(), '<br>';
   sleep(1);
   echo 'start: ', $chrono->start(), '<br>';
   ...
?>
```

1.15 Require e Include

Lo statement:

require (`path_or_url_to_file`);

è l'analogo della `#include` del C ed in più è network-trasparente. Attenzione che questo costrutto provoca l'inclusione del contenuto del file specificato nella modalità HTML e non PHP. Questo significa che quando l'interprete incontra una istruzione `require`, esce dalla modalità PHP, passa ad analizzare il file specificato e quando ha finito ritorna nella modalità PHP e continua con la prossima istruzione del file che aveva fatto l'inclusione. Questo comportamento può essere schematizzato in questo modo:

```
<% <%
... ...
require('mylib.inc'); <=>%>inserisce qui il contenuto di mylib.inc<%
... ...
%>%>
```

Questo fatto implica che anche se il file `mylib.inc` è costituito da solo codice PHP, questo codice deve essere racchiuso tra `<%, %>`, altrimenti il codice non verrà interpretato e verrà stampato come HTML. Quando viene incluso del codice PHP ecco cosa accade:

```
<% <%
... ...
%><% codice_php_incluso %><% <=>codice_php_incluso
... ...
%>%>
```

Tuttavia quando viene richiamata una pagina php tramite il protocollo http sullo stesso host o su un altro configurato in modo da fare il parsing delle pagine PHP, il codice php non può essere incluso, bensì viene incluso l'output HTML prodotto dal parsing e dall'esecuzione. È naturalmente possibile passare dei parametri nell'URL tramite il metodo GET, così come quando si richiama un'URL da un browser:

```
require('http://localhost/pagina.php?paramuno=1&paramdue=2');
```

questa istruzione è analoga a:

```
readfile('http://localhost/pagina.php?paramuno=1&paramdue=2');
```

ma c'è la sottile differenza che se il parsing di pagina.php ha fatto stampare del codice PHP, questo verrà interpretato dopo una require, mentre readfile lo stampa solo su output.

Essendo `require` non una funzione, ma un costrutto del linguaggio, il suo comportamento è speciale. `require` può apparire ovunque in una sezione PHP, non restituisce nulla e viene comunque espanso, anche il flusso dell'esecuzione non raggiunge mai il codice (html o php) che viene espanso. Viene espanso quindi ad es. anche se c'è una istruzione `if` che lo controlla, anche se la condizione risulta sempre false. Quando `require` si trova nell'istruzione controllata da un ciclo, la sua espansione avviene una sola volta, e le istruzioni espanse vengono ripetute normalmente. Poiché l'espansione avviene una sola volta, se passate un nome di file variabile a require all'interno di un loop, non avrete l'inclusione di codice diverso ad ogni iterazione, ma il codice iterato sarà sempre quello incluso la prima volta.

Quando viene richiesta l'inclusione di un file, il codice che questo file contiene eredita l'ambito di visibilità della linea che contiene l'istruzione `require`. Tutte le variabili che sono visibili in quella linea saranno visibili anche al codice contenuto nel file richiesto. Questo significa che se l'istruzione `require` viene utilizzata all'interno di una funzione del file chiamante, allora tutto il codice contenuto nel file chiamato si comporterà come se fosse stato definito all'interno di quella funzione.

`require` è molto utile soprattutto per includere in ogni script eventuali file di configurazione o librerie di classi e/o funzioni e definizioni di costanti comuni a tutti gli script. Il fatto di non dover ripetere il codice in ogni script è un grande beneficio, poiché altrimenti bisognerebbe ripetere eventuali modifiche o aggiunte di codice in ogni script. In questo modo potete riunire in un luogo centralizzato le opzioni di configurazione o le funzioni comuni a una serie di script appartenenti tutti ad un unico programma e così faremo anche noi nel progetto del **bookmarksDB**.

Questa semplicissima istruzione è sufficiente a realizzare il riutilizzo del codice in un linguaggio interpretato come PHP in molti casi. Tuttavia possono sorgere dei sottili problemi.

Considerare ad es. il caso seguente: esiste un file `common.inc` che definisce (tra le altre eventuali cose) una funzione php chiamata `common()`:

```
common.inc
----------
<%
// ...
function common() {}
// ...
%>
```

c'è poi un file detto `lib.inc` che tra le altre cose include con `require` il file `common.inc` perché ne ha bisogno (vuole ad es. utilizzare la funzione `common`):

```
lib.inc
-------
<%
// ...
require('common.inc');
// ...
%>
```

infine c'è un terzo file, `script.php` il quale include entrambi i file:

```
script.php
----------
<%
// ...
require('common.inc');
require('lib.inc');
// ...
%>
```

quando viene lanciato lo script `script.php`, l'interprete terminerà col seguente errore fatale:

```
Fatal error: Cannot redeclare common() in
/usr/local/apache/htdocs/common.inc on line 3
```

L'errore poteva essere evitato ricordandosi nello script `script.php` che `lib.inc` include `common.inc` ed eliminando la riga:

```
require('common.inc');
```

Ma occorre ricordarsi di questo fatto e comunque se supponiamo ad es. di aver diviso la libreria di funzioni in due file, `lib.inc` e `libbis.inc`, di utilità generale, tanto che possono essere usati anche separatamente e che anche quest'ultimo abbia bisogno di includere il file di configurazione `common.inc`:

```
libbis.inc
----------
<%
// ...
require('common.inc');
// ...
%>
```

e che `script.php` debba utilizzare entrambe le librerie, allora deve per forza includerle entrambe, anche se sa di non dover includere `common`:

```
script.php
----------
<%
// ...
require('lib.inc');
require('libbis.inc');
// ...
%>
```

allora non c'è modo di evitare l'errore con `require`, che è sempre lo stesso di prima:

```
Fatal error: Cannot redeclare common() in
/usr/local/apache/htdocs/common.inc on line 3
```

La soluzione a questo problema consiste nell'utilizzare il costrutto

```
require_once()
```

`require_once` ha le stesse proprietà di `require`, tuttavia il processore PHP registra quando ogni file viene incluso la prima volta con `require_once()` ed eviterà di includere lo stesso file una seconda volta: non potranno quindi sorgere problemi di riassegnazione di valori di variabili non desiderate o tentativi di ridefinizione di funzioni.

Nell'esempio precedente, basta che entrambi i file di libreria

`lib.inc` e
`libbis.inc`

includano `common.inc` usando `require_once()` anziché `require` e non si verifica più nessun errore.

I programmatori in C avranno capito che `require` è del tutto simile alla `#include` del preprocessore C e `require_once` permette di ottenere l'inclusione condizionale esattamente come si può fare con le direttive `#if` e `#define` del preprocessore C:

```
#if !defined(COMMON)
#define COMMON

/* contenuto dell'header COMMON.h */

#endif
```

Bene, sappiate che PHP mette a disposizione anche un altro tipo di funzione per l'inclusione:

include (*path_or_url_to_file*);
include_once (*path_or_url_to_file*);

questa funzione è simile a `require`, ma l'inclusione invece di essere fatta a *"compile time"*, o meglio si dovrebbe dire quando viene fatto il *parsing* dello script, in un linguaggio interpretato come PHP, viene fatta ogni volta a *"runtime"*, ossia quando lo script viene eseguito. Nei casi in cui avete la necessità di inserire dinamicamente (ossia variabilmente durante la sua esecuzione) un contenuto diverso di codice nello script, dovete usare `include` (o `include_once`) al posto di `require` (o `require_once`).

Riprendo un esempio dal manuale di PHP:

```
$files = array ('first.inc', 'second.inc', 'third.inc');
 for ($i = 0; $i < count($files); $i++) {
     include $files[$i];
 }
```

questo codice include tutti i file i cui nomi sono elencati nell'array `$files` e ne esegue il codice. Se avessimo usato `require` invece che `include`, solo il primo file (`first.inc`) sarebbe incluso e il suo contenuto sarebbe stato interpretato ed eseguito 3 volte.

Quindi la differenza tra `include` e `require` è che lo statement `include` viene rivalutato ogni volta che si incontra. Inoltre se il flusso del programma non passa per un `include` non avviene nessuna inclusione. Con `require` invece l'inclusione avviene comunque, sia che il contenuto incluso sia poi interpretato che non, poiché si tratta di una sostituzione della istruzione `require` col codice contenuto nel file indicato come parametro e l'inclusione avviene una sola volta. Per il resto `include` si comporta come `require`.

`include_once` è in tutto e per tutto simile a `include`, con la differenza che consulta una lista dei file già inclusi e non include niente se il file è stato già incluso. Questo evita i problemi di ripetizione indesiderata dell'esecuzione del codice incluso una seconda volta, errori fatali di ridefinizione di funzioni, errori di indesiderata riassegnazione dei valori originali a variabili definite nel file incluso che erano state modificate dopo l'inclusione.

Capitolo 2

mSQL e MySQL API: esempi pratici

2.1 mSQL API e il bookmarksDB

Il DBMS **Mini SQL** include nella distribuzione una libreria API (Application Programming Interface) che permette ad un qualsiasi programma scritto nello stesso linguaggio nativo del database server, ossia il C, di comunicare con l'applicazione server stessa. Inoltre mSQL comprende nella distribuzione un interprete del linguaggio Lite, che è un linguaggio simile al C, ma molto meno potente sia del C che del PHP e che ha comunque il vantaggio di avere integrata una API mSQL per accedere al database server mSQL. Questo linguaggio può essere utilizzato anche embedded in HTML, poiché viene fornito l'interprete anche sotto forma di applicazione CGI "wrapper".

PHP comunque mette a disposizione una API per accedere a mSQL che vi permette di fare tutto quello che potete fare con la API C o Lite di mSQL. In questa lezione vedremo come usare questa API mSQL di PHP[1].

Per prima cosa assicuratevi di avviare il demone **msql2d**, se questo non è già avviato.

```
[17:24:29 root@localhost ~]# /usr/local/Hughes/bin/msql2d &
[1] 300
[17:24:29 root@localhost ~]#

Mini SQL Version 2.0.11
Copyright (c) 1993-94 David J. Hughes
Copyright (c) 1995-99 Hughes Technologies Pty Ltd.
All rights reserved.
```

[1]Se siete interessati a Lite o alla API C di mSQL vi rimando alla documentazione che trovate sul sito http://www.hughes.com.au.

```
        Loading configuration from '/usr/local/Hughes/msql.conf'.
        Server process reconfigured to accept 200 connections.
        Server running as user 'msql'.
        Server mode is Read/Write.

Warning : No ACL file.  Using global read/write access.
```

Dopodiché potete provare a connettersi al database server da uno script PHP. Per fare questo dovete usare la funzione API:

int **msql_connect** (string *hostname*)

questa funzione accetta per argomento il nome dell'host su cui gira il database server (oppure il suo indirizzo IP). Se il database server si trova sulla stessa macchina che esegue lo script PHP, l'argomento hostname si può omettere, poiché di default viene assunto localhost:

```
$cd=msql_connect();
```

Notare che per abilitare le connessioni al demone su una determinata porta via TCP/IP, occorre editare il file di configurazione:

```
/usr/local/Hughes/msql.conf
```

e cambiare il valore di default dell'opzione `Remote_Access`:

```
Remote_Access = False
```

con True:

```
Remote_Access = True
```

La porta viene invece specificata dalla direttiva `TCP_Port`. Il default è:

```
TCP_Port = 1114
```

Fatte queste modifiche, avviare il server[2]. Ci si può connettere quindi ad un DB server mSQL sulla macchina di nome `machine` col programma monitor richiamandolo con questa sintassi:

```
msql -h machine nome_db
```

[2]Se è già avviato, killatelo – il pid si trova nel file /usr/local/Hughes/msql2d.pid – e poi avviatelo di nuovo.

È raccomandabile attivare l'accesso via TCP solamente su una rete privata fidata, o se siete dietro un firewall. Se non siete in questi casi e comunque ne avete bisogno, almeno cercate di regolare l'accesso ai db creando un opportuno file

/usr/local/Hughes/msql.acl

in modo da impedire che chiunque possa accedere o cancellare i vostri dati da qualsiasi host.

L'intero ritornato da `msql_connect` rappresenta l'id della connessione e deve essere conservato poiché occorrerà passarlo alle altre funzioni che useremo. Nel caso ci sono problemi di connessione (vi siete ricordati di lanciare il demone msql2d? vedi sopra), `msql_connect` ritorna false. C'è una limitazione: uno script PHP non può stabilire più di una connessione ad uno stesso server mSQL, quindi se chiamate `msql_connect` più volte nell'ambito di uno stesso script, otterrete sempre lo stesso valore di ritorno e non verranno create ulteriori connessioni.

Questo tipo di connessione viene chiusa automaticamente alla fine dello script, ma potete anticiparne la chiusura in qualsiasi momento usando:

int **msql_close** (int *link_identifier*)

anche qui l'argomento `link_identifier` è opzionale, visto che ogni script può avere un solo id di connessione a mSQL per server (e di solito ci si connette ad un solo server in uno script, a meno che non si voglia accedere a più database residenti su server diversi in un solo script), viene usato automaticamente quello, quindi basta scrivere:

```
msql_close();
```

questa funzione ritorna false se non avevate aperto nessuna connessione o se si è verificato un errore (non potete chiudere connessioni permanenti con questa funzione), true altrimenti. In generale comunque, in mancanza di argomento, `msql_close` usa l'ULTIMO link aperto.

È consigliato non connettersi scrivendo:

```
$cd=msql_connect("localhost");
```

oppure col monitor:

```
msql -h localhost nome_db
```

poiché la connessione al database server locale tramite socket di UNIX è molto più veloce che tramite TCP/IP. Piuttosto non indicare niente e verrà usato un socket di UNIX:

```
$cd=msql_connect();
```

```
msql nome_db
```

PHP è in grado di offrire connessioni persistenti (o "permanenti") ad un database. È consigliabile usare questo tipo di connessioni. Non occorre fare altro che usare:

int **msql_pconnect** (string *hostname*)

al posto di `msql_connect` e NON usare mai `msql_close()`.

Mentre una connessione aperta con `msql_connect` viene chiusa alla fine dello script, una connessione permanente persiste anche dopo la fine dello script che l'ha richiesta e rimane associata al processo figlio di Apache che ha servito la richiesta. Quando questo script o persino un altro script viene eseguito di nuovo, se Apache riutilizza lo stesso processo figlio per servire la nuova richiesta, quando questo ripete la chiamata a `msql_pconnect` (con lo stesso nome di host) viene riutilizzata la connessione che persisteva insieme a questo processo figlio. Poiché il solo fatto di stabilire una connessione richiede un certo tempo (soprattutto via TCP/IP, come avviene quando ci si connette ad un host diverso da quello in cui sono eseguiti gli script e soprattutto se il server SQL in questione è usato da molti utenti e quindi è oberato di lavoro), le connessioni permanenti possono incrementare in modo significativo le prestazioni.

Affinché PHP possa usare connessioni persistenti al DB di mSQL, l'opzione:

`msql.allow_persistent`

deve essere `On` nel file `php.ini` (questo è il default).

Da tutto questo è ovvio che quando PHP viene compilato come CGI "wrapper", anziché come modulo di Apache, non si possono avere delle connessioni permanenti: in questo caso infatti ogni volta che un client fa la richiesta di una pagina web viene avviata una istanza dell'interprete PHP, stabilite eventuali connessioni a database richieste dal codice PHP e poi distrutte le connessioni e l'istanza dell'interprete PHP in memoria, e quindi usare `msql_pconnect` in questo caso risulta equivalente a `msql_connect`: le connessioni permanenti semplicemente non permangono. L'uso di PHP come CGI è lento per questo motivo e non è consigliato su un server.

Una volta connessi, prima di poter mandare delle query, occorre scegliere il database sul quale si vuole lavorare, tramite la funzione:

int **msql_select_db** (string *database_name* [, int *link_identifier*])
int **msql_selectdb** (string *database_name* [, int *link_identifier*])

il primo argomento è il nome del DB. Il secondo deve corrispondere al valore ritornato da `msql_connect` o `msql_pconnect` (comunque se non specificato viene usata l'ultima connessione aperta al db con una di queste due funzioni; se nessuna di queste due funzioni era stata chiamata, `msql_select_db` invoca automaticamente la prima (`msql_connect`) senza argomenti). Se avete finito di lavorare su un db e volete passare ad un altro, basta chiamare `msql_select_db`, senza prima dover chiudere e poi riaprire la connessione al db server che sarebbe più lento. Anche questa funzione ritorna true se tutto va bene, false se c'è qualche errore (ad es. il database cercato non esiste, o non riesce a connettersi al db server).

Aperta la connessione e selezionato il db su cui operare si possono fare delle query SQL su quel db tramite:

int **msql_query** (string *query* [, int *link_identifier*])

questa funzione, nel caso si ometta `link_identifier` si comporta come `msql_select_db`. `msql_query` restituisce un id positivo che identifica la query, e false in caso di errore. Conservate questo id nel caso la query generi dell'output (ad es. è una query di SELECT): infatti i dati vengono conservati in opportuni buffer gestiti dall'API e potete recuperarli proprio utilizzando l'id restituito da `msql_query`. Non è obbligatorio terminare di recuperare i dati di una query prima di effettuarne un'altra, poiché la API è in grado di utilizzare più buffer, ciascuno associato ad un diverso query id.

La funzione seguente combina le due azioni di selezionare un database ed eseguire una query ed è molto utile per abbreviare il codice se il vostro script deve eseguire una sola query al db in questione, oppure una sola query per ogni db. Come `msql_query` ritorna un query id tramite il quale potete poi recuperare i dati. Tenete presente che questa funzione cambia il db selezionato con quello indicato e che si può avere un solo database corrente.

int **msql** (string *database*, string *query* [, int *link_identifier*])

Come saprete il risultato di una query SQL è una tabella, composta di righe e colonne. Potete conoscere il numero di righe e colonne di questa tabella passando l'id della query ritornato da `msql_query` o `msql` rispettivamente a queste due funzioni:

int **msql_num_rows** (int *query_identifier*)
int **msql_num_fields** (int *query_identifier*)

I valori restituiti da queste funzioni sono quelli logicamente accettabili per ogni tipo di query. Sia le righe che le colonne possono essere 1; in realtà il valore minimo delle righe è 0, quando la tabella generata da una SELECT è vuota; il numero delle colonne quando si fa una SELECT è pari al numero di colonne che si proiettano, anche se le righe sono 0.

Nel caso di query che non ritornano alcuna tabella, nemmeno la cornice, come una query di DELETE o UPDATE, entrambi i valori ritornati dalle due funzioni precedenti sono zero. Per conoscere il numero di righe che una query di questo tipo ha cancellato o modificato, usate questa funzione:

int **msql_affected_rows** (int *query_identifier*)

Questa funzione funziona naturalmente nel caso di una SELECT, ritorna il numero di righe recuperate, esattamente come `msql_num_rows`.

Ci sono sinonimi di queste funzioni per quelli che non amano usare troppi underscore (tranne per la l'ultima):

int **msql_numrows** (int *query_identifier*);
int **msql_num_fields** (int *query_identifier*);

Per recuperare i dati della tabella ci sono 4 funzioni. La prima restituisce una sola cella della tabella alla volta, le altre tre (che iniziano tutte con `msql_fetch`) operano tutte su una riga alla volta, ma offrono la possibilità di pacchettare i dati in modo diverso.

string `msql_result` (int *query_identifier*, int i, mixed *field*)

ritorna il contenuto della cella alla riga i e alla colonna numero *field* della tabella. I numeri di riga e di colonna sono contati a partire da zero. Specificare un numero di colonna o di riga non valido produce un Warning in output e `msql_result` ritorna il boolean false, anziché una stringa. È possibile specificare nel parametro *field* sia il numero di colonna oppure il nome del campo corrispondente, oppure nometabella.nomecampo, in tal caso nometabella deve corrispondere all'alias e non al nome vero della tabella se la query stabilisce un alias per una tabella. In pratica fate corrispondere il parametro *field* con il nome del campo che avete indicato nella lista di proiezione dopo la SELECT e non avrete problemi. C'è da notare che usare i nomi dei campi, anziché i numeri d'ordine, rende più leggibile il codice, ma è molto più lento. L'utilizzo di questa funzione è di per sè lento e dovrebbe essere evitato soprattutto quando c'è la possibilità che una query ritorni una tabella molto grossa. È più veloce fare il fetch per righe con una delle funzioni seguenti, e anche se fate il fetch per righe non siete limitati ad un accesso alle righe strettamente sequenziale (vedi la funzione `msql_data_seek`).

array **msql_fetch_row** (int *query_identifier*)

questa funzione ritorna in sequenza le righe della tabella associata al query id passato sotto forma di un array con soli indici numerici (array scalare o enumerato), a partire dall'indice 0. Arrivato dopo l'ultima riga, ritorna false, quindi può essere usata come condizionale di un ciclo.

array **msql_fetch_array** (int *query_identifier* [, int *result_type*])

questa funzione è una estensione della funzione precedente. Quando viene invocata in questo modo:

array **msql_fetch_array** (int *query_identifier*, `MSQL_NUM`)

è equivalente alla funzione precedente. Passando la costante `MSQL_ASSOC` al parametro opzionale *result_type*, invece di un array scalare viene ritornato un array associativo con i nomi dei campi come chiavi. Passando la costante `MSQL_BOTH`, viene creato un array associativo con entrambi gli indici numerici e letterali, quindi potete anche usare entrambi i tipi di indicizzazione. Questa funzione è piuttosto ottimizzata e non è molto più lenta della precedente, perciò ne è consigliato l'uso anche col valore di default di *result_type* (che è `MSQL_BOTH`).

object **msql_fetch_object** (int *query_identifier* [, int *result_type*])

questa funzione è analoga alla precedente, solo che invece di un array ritorna un oggetto con i nomi delle variabili membro uguali ai nomi dei campi. Non usate `MSQL_NUM` come secondo argomento, poiché le variabili non possono avere un nome costituito da (o iniziante con) un numero. È consigliabile non specificare niente per *result_type*.

Nei buffer associati al query id viene mantenuto un puntatore alla riga corrente. Come accennato prima è possibile (ri)posizionare questo puntatore alla riga voluta, tramite la funzione:

int **msql_data_seek** (int *query_identifier*, int *row_number*)

dopodiché la prossima chiamata alla funzione `msql_fetch_row`, `msql_fetch_array` o `msql_fetch_object` ritornerà la riga *row_number*.

Quando il processore PHP termina il suo lavoro di elaborazione di una pagina, svuota automaticamente i buffer dell'API associati al query id che avete utilizzato. Se non avete intenzione di estrarre tutti i dati, o avete finito di estrarre tutti i dati da una query, e siete molto prima della fine dello script e volete liberare quindi la memoria associata a questi buffer anticipatamente rispetto alla fine dello script, dovete chiamare esplicitamente questa funzione:

int **msql_free_result** (int *query_identifier*)

alias

int **msql_freeresult** (int *query_identifier*)

Dopo aver chiamato **msql_free_result**, il it query_identifier non sarà più valido, quindi non tentate più di estrarre dati[3].

Con queste funzioni siete in grado di interrogare con delle SELECT e manipolare un qualsiasi database mSQL da uno script PHP.

[3]Un valore NULL nel database, viene ritornato come stringa vuota `""`.

Tuttavia potreste avere la necessità di creare database da uno script PHP, di ottenere l'elenco dei database presenti sul server, oppure di reperire informazioni sulla struttura di un database. Esistono delle funzioni API apposite, grazie alle quali ho creato **mSQLadmin**, un insieme di semplici script PHP per la gestione di db mSQL tramite un browser, che in futuro potrete trovare nel sito indicato alla fine di questo tutorial. Mi limiterò a trattare le due funzioni API per creare a cancellare DB. Vi rimando alla documentazione di PHP per il resto.

Per poter creare e cancellare database, il modo più semplice è che l'utente sotto cui gira il server web (tipicamente `nobody`), sia definito come amministratore, cambiando la riga:

```
Admin_User = root
```

con:

```
Admin_User = nobody
```

Questa è una cosa potenzialmente pericolosa, anche se l'account `nobody` risulta disabilitato e solo il `root` può diventare l'utente `nobody`. Tuttavia gli altri utenti possono scrivere script PHP, che girano sotto l'utente `nobody` e che creano e cancellano database nelle loro home directory, se avete abilitato la possibilità di crearsi un sito web nella propria home directory. Se tutti gli utenti sono fidati questo non è un problema.

In generale però non ci si può fidare degli utenti[4]. Se qualcuno riesce ad accedere al suo account, vi può spazzare via tutti i database, scrivendo uno script PHP nella sua `public_html` e lanciandolo via web. È possibile impedire a questi utenti di poter eseguire qualsiasi script PHP, tramite opportuna configurazione di Apache, comunque non è una soluzione molto felice per l'utente, che non potrà usare PHP[5]. Quindi pensateci bene, prima di assegnare privilegi di amministratore del db all'utente `nobody`. È consigliabile che il `root` si sobbarchi manualmente l'onere di cancellare e creare nuovi DB (cosa tutto sommato non molto frequente).

Ammesso che possiate farlo, per creare un database si usa la seguente funzione:

int **msql_create_db** (string *database_name* [, int *link_identifier*])
int **msql_createdb** (string *database_name* [, int *link_identifier*])

se tutto va bene questa restituisce il booleano true, altrimenti restituisce false.

Per distruggere un intero database, con tutte le tabelle e i dati che contiene:

int **msql_drop_db** (string *database_name*, int *link_identifier*)
int **msql_dropdb** (string *database_name*, int *link_identifier*)

[4] C'è sempre qualche utente *"scapestrato"* che per esempio sceglie password facili da indovinare, oppure si scrive le password in modo che altri possano carpirla.

[5] Quindi che lo avete installato a fare se dovete impedirne l'uso ai vostri utenti?.

Un'altra funzione molto utile per il debugging è:

string **msql_error** ()

questa funzione può essere usata per stampare il messaggio di errore corrente, quello che si è eventualmente verificato dopo l'ultima chiamata ad una funzione API di mSQL. L'API memorizza ogni messaggio di errore ritornato dal server in una stringa e `msql_error` semplicemente ritorna il contenuto di questa stringa. La stringa viene sovrascritta solo se si verifica un altro errore, quindi conterrà sempre l'ultimo errore verificatosi o sarà vuota se non si è verificato alcun errore[6].
ES.

```
msql_connect() or die(msql_error());
```

Siamo pronti per creare lo script PHP che predisporrà la struttura del nostro database di bookmarks, che si chiama `install.php`. Eccolo qui:

```
install.php
-----------
<html>
<head><title>BookmarksDB installation</title></head>
<body bgcolor="#ffffff">
<h1 align="center">BookmarksDB installation</h1>
<hr noshade size=1>
<?php
// The BOOKMARKSDB installation program //
// this creates the empty bookmark database //
// //
// distributed under GPL (see COPYING) //

// please set here host to which connect and database name to create
// they defaults to localhost and bookmarksDB respectively.
require 'CONF.php';

// connect to the mSQL database server
// i prefer using non-permanent connection here
// cause permanent connection will be reused even
// if the server has been killed and restarted and
// when you rerun this script you will get "MSQL server has gone away"
HOSTNAME?msql_connect(HOSTNAME):msql_connect()
   or die(msql_error(). '<br><font color="green">
Please check HOSTNAME in the configuration file (CONF.php).'. HTMLend);

// try to create db DBNAME
msql_createdb(DBNAME);
```

[6]La lista dei messaggi di errore con relativa spiegazione si trova alla fine del manuale di mSQL, nelle sezioni *"API Library Error Messages"* e *"Server Error Messages"*.

```
$uinfo=posix_getpwuid(posix_getuid());

// try to open db DBNAME
msql_selectdb(DBNAME)
    or die(msql_error(). '<br>
<font color="green">
     Can\'t create or open database '. DBNAME. ' from a web script.<br>
     If this database doesn\'t exist, please ask mSQL admin user
     (usually root) to create an<br>
     empty database named '. DBNAME. ' and then re-run this script.<br>
     If it exists, ask administrator to ensure that the web user
     ('. $uinfo["name"] . ') has permission<br>
     to read/write to this database.<br>
     Check the configuration file CONF.php if you want to change the
     bookmarks db name.'. HTMLend);

// try to create bookmarks table
msql_query('CREATE TABLE bookmarks (
id  UINT,
parent  UINT,
name  CHAR('. nameCHARlen .'),
url TEXT('. urlTEXTlen .'),
comment TEXT('. commentTEXTlen .')
)')
    or die(msql_error(). '<br>
<font color="green">It seems that the database already exists.<br>
It will be a good idea to remove this script.'. HTMLend);

// create indexes
msql_query('CREATE UNIQUE INDEX idx1 ON bookmarks (id)')
    or print('Warning: can\'t create UNIQUE INDEX ON bookmarks(id).<br>');
msql_query('CREATE INDEX idx2 ON bookmarks (parent)')
    or print('Warning: can\'t create INDEX ON bookmarks(id).<br>');
?>
<h2>Database successfully created.</h2>
You can remove this script so that anyone can't access it anymore.
<hr noshade size=1>
<h6 align="center">Copyright 2001
<a href="mailto:antonio.b@infinito.it">Antonio Bonifati</a>.
Distributed under <a href="COPYING">GPL</a>.</h6>
</body>
</html>
```

Il file di configurazione `CONF.php`, definisce alcune costanti utili per la configurazione dell'intero programma di gestione dei bookmark:

```
CONF.php
--------
```

```
<?php
// The BOOKMARKSDB configuration file //
// copyright 2001 Antonio Bonifati    //
// see COPYING for license details    //

// default database has the following structure:
/*
Database = bookmarksDB
Table    = bookmarks

 +-----------------+----------+--------+----------+--------------+
 |     Field       |   Type   | Length | Not Null | Unique Index |
 +-----------------+----------+--------+----------+--------------+
 | id              | uint     | 4      | N        | N/A          |
 | parent          | uint     | 4      | N        | N/A          |
 | name            | char     | 50     | N        | N/A          |
 | url             | text     | 50     | N        | N/A          |
 | comment         | text     | 100    | N        | N/A          |
 | idx1            | index    | N/A    | N/A      | Y            |
 | idx2            | index    | N/A    | N/A      | N            |
 +-----------------+----------+--------+----------+--------------+
*/

// for install: mSQL server in which to create the bookmark database
// for other scripts: mSQL server which contains the bookmark database
// '' means the same server which runs the scripts (i.e. localhost)
define('HOSTNAME', '');

// install.php will try to create a database with this name, but usually
// web server user hasn't permission to create/drop databases. If so
// you will have to create this database by hand before running
// install.php. As mSQL Admin_User (usually root) issue this
// shell command: /usr/local/Hughes/bin/msqladmin create DBNAME
// please substitute the correct path to your mSQL executables
// (/usr/local/Hughes is the default) and the DBNAME defined below.
define('DBNAME', 'bookmarksDB');

// these defines various text field length which are used
// when creating the database. See CREATE query in install.php.
define('nameCHARlen',  50); // set to MAX bookmark name length
define('urlTEXTlen',  50); // set to MEAN url length
define('commentTEXTlen', 100); // set to MEAN comment length

// html to print after a custom fatal error message
define('HTMLend', "</font>\n</body>\n</html>\n");


//// you can change these as you want, but please don't remove %s's
// markup to use to represent the previous directory
define('PREVDIRMARK', '<a href="?parent=%s">
                       <img src="folder.png"
                        width=15 height=13
                        border=0 hspace=3>..</a><br>');

// markup to use to represent directories
// please note that the first %s get substituted
// with the url parameter value,
// the second one with the directory name, so you can't have
// directory name printed before setting the associated url
define('DIRMARK', '<a href="?parent=%s">
                   <img src="folder.png"
                    width=15 height=13
```

```
                 border=0 hspace=3>%s</a><br>');

// markup used to represent "url files"
// same warning as above
define('URLMARK', '<img src="spacer.png" width=3><a href="%s">%s</a><br>');

// markup printed when a directory is empty Optional: use
// define('EMPTYMARK', '')
// to print nothing in this case.
define('EMPTYMARK', '<img src="spacer.png" width=3><i>empty</i>');

// configuration file ends here //
?>
```

Anche per un programma semplice, le possibilità di configurazioni sono spesso molteplici. È buona norma concentrare tutte le definizioni che l'utilizzatore vorrebbe poter cambiare in uno o più file di configurazione (più nel caso possano essere suddivise in classi più o meno distinte). Il vantaggio è che chi vuole *"customizzare"* l'applicativo non deve per forza studiarsi tutto il codice sorgente e può farlo apportando modifiche ad un solo file.

La cosa più interessante riguarda la struttura del database. Si tratta di un database molto semplice, composto da una sola tabella. Tuttavia i record di questa tabella sono in una relazione gerarchica. I bookmark sono disposti secondo una struttura gerarchica molto diffusa e importante in informatica, detta **albero n-ario**. Questa struttura è del tutto analoga a quella del filesystem UNIX, quindi sarà ben nota ai più. Vale comunque la pena di ricordarne le caratteristiche. Un albero n-ario è un particolare tipo di grafo, composto da **nodi** e da **frecce** che li collegano con le seguenti **regole**:

1. esiste un nodo principale da cui tutti gli altri nodi si dipartono. Questo nodo è l'analogo della radice di un albero naturale, considerando la radice come un tuttuno (volendo fare i pignoli anche la radice infatti ha una struttura ad albero, ma essendo sotterrata la consideriamo alla stregua di un nodo);

2. ogni nodo può avere un qualsiasi numero di nodi "figli" (nella usuale rappresentazione grafica una freccia collega un nodo padre a ciascuno dei suoi figli), siccome da un ramo si possono dipartire un qualsiasi numero di altri rami (di sezione media minore). Un nodo che non abbia nessun nodo figlio è detto nodo terminale (o nodo foglia, oppure file nel filesystem UNIX);

3. ogni nodo "figlio" deve avere uno e un solo nodo padre (o directory nel filesystem di UNIX), ad eccezione del nodo principale (detto root nel filesystem di UNIX), il quale è l'unico ad essere "orfano", in quanto è il progenitore di tutti.

Una rappresentazione lineare di una struttura di questo genere si ottiene assegnando ad ogni nodo (compreso il nodo padre) un numero unico. Si rappresenta quindi per ogni nodo il numero del nodo (id), il numero del nodo genitore (parent in inglese), il nome del nodo (name) e il resto del contenuto informativo associato al nodo. I nodi "directory" hanno come unico contenuto informativo (oltre al nome) un commento (che descrive meglio il contenuto di quella directory); i nodi foglia sono i bookmark veri e propri e oltre ad un commento contengono naturalmente l'informazione dell'URL associato. Questa rappresentazione non è naturalmente l'unica possibile: ne avevo considerato anche altre, ma questa mi sembra la più semplice e quella che riduce al minimo la ridondanza dei dati nel db.

Ad esempio supponete di avere il seguente albero di bookmark:

```
ROOT (/)
|
|- PHP (bookmark sul linguaggio PHP; NULL)
|  |
|  |- main (sito ufficiale; www.php.net)
|  `- tutorial (sito ricco di tutorial su PHP; www.devshed.com)
|
`- Perl (bookmark sul linguaggio Perl; NULL)
   |
   |- main (sito ufficiale; www.perl.com; NULL)
   `- tutorial (NULL; NULL)
      |
      |- cgi-lib.pl (libreria web per Perl; cgi-lib.berkeley.edu)
      `- it_tut (tutorial in italiano; www.venis.it/francesco)
```

Ogni nodo è stato indicato con il suo nome. Le informazioni contenute nel nodo sono state rappresentate con questa convenzione:

(comment; url)

Una possibile rappresentazione tabulare con i campi stabiliti prima è la seguente:

```
id | parent | name       | url                    | comment
---+--------+------------+------------------------+----------------------------
 1 | 0      | PHP        | NULL                   |bookmark sul linguaggio PHP
 2 | 0      | Perl       | NULL                   |bookmark sul linguaggio Perl
 3 | 1      | main       | www.php.net            |sito ufficiale
 4 | 1      | tutorial   | www.devshed.com        |sito ricco di tutorial su PHP
 5 | 2      | main       | www.perl.com           |sito ufficiale
 6 | 2      | tutorial   | NULL                   |NULL
 7 | 6      | cgi-lib.pl | cgi-lib.berkeley.edu   |libreria web per Perl
 8 | 6      | it_tut     | www.venis.it/francesco |tutorial in italiano
```

Qui solo per caso la tabella è ordinata nel senso delle id crescenti, come sapete, in un RDBMS l'ordine delle righe (o tuple) non conta.

Allo scopo di salvare spazio, e supponendo che la maggior parte dei link che uno vuole memorizzare inizino con `http://`, ho (almeno inizialmente) convenuto (come del resto è uso comune) che se nessun protocollo viene indicato si intenda il protocollo http[7], il protocollo più usato sul web e adatto alla trasmissione delle pagine web). Se volete un altro tipo di protocollo, occorre indicarlo esplicitamente nella stringa dell'URL[8]. Gli script PHP che generano le pagine aggiungono automaticamente `http://` se la stringa URL non inizia con qualcosa del tipo

```
nomeprotocollo://
```

Se dovete memorizzare molti collegamenti tramite gli altri tipi di protocolli, la soluzione più conveniente è aggiungere un campo indicante il tipo di protocollo tramite un byte carattere, piuttosto che indicare il nome del protocollo in ogni stringa url. In effetti è così che faremo, ma prima voglio evitare questa complicazione.

Allo stesso scopo di compattare i dati, ho preferito far indicare il fatto che un nodo è una directory dal fatto che il campo url ha il valore NULL. In effetti una directory è un nodo speciale che non ha nessun contenuto di url[9], ma piuttosto contiene (nel senso che da essa si diramano) altri nodi foglia e directory. Un nodo directory può contenere un commento nel campo comment o NULL per indicare nessun commento proprio come un nodo "file".

Lo script che visualizza il contenuto di una directory e permette di navigare in questo "bookmarksystem", riceve tramite `GET` o `POST` il codice id della directory nella variabile globale `$parent` ed effettua una query molto semplice:

```
SELECT * FROM bookmarks WHERE parent=$parent
```

Lo script assegna il valore 0 a `$parent` nel caso questa non sia settata, quindi di default mostrerà il contenuto della directory padre, che nel nostro caso sarà costituito dai dati seguenti:

```
id | parent | name       | url       | comment
---+--------+------------+-----------+----------------------------
 1 | 0      | PHP        | NULL      |bookmark sul linguaggio PHP
 2 | 0      | Perl       | NULL      |bookmark sul linguaggio Perl
```

Naturalmente tutto sarà opportunamente formattato in HTML. Quando l'utente clicca ad es. su PHP, poiché url è NULL, lo script avrà creato un collegamento a se stesso passando però col metodo `GET` il parametro `parent=1`. La stessa query di prima produrrà il contenuto della directory PHP:

```
id | parent | name     | url             | comment
---+--------+----------+-----------------+----------------------------
 3 | 1      | main     | www.php.net     |sito ufficiale
 4 | 1      | tutorial | www.devshed.com |sito ricco di tutorial su PHP
```

[7] Hypertext Transfer Protocol.
[8] ftp, mailto, file sono gli altri più comuni.
[9] O clorofilla nel caso dell'albero :)

La struttura quindi è adeguata per la produzione del codice HTML. Tuttavia per facilitare la navigazione desidero includere un bookmark speciale chiamato, in analogia col filesystem UNIX, `".."`. Cliccando su questo bookmark voglio che si ritorni a listare il contenuto della directory di bookmark precedente, la directory padre rispetto a quella che è listata. Il problema però è che lo script non conosce l'id di questa directory, ma solo quello della directory corrente, che è la directory padre dei nodi listati. La soluzione più semplice per ottenere l'id della directory padre della directory corrente consiste nel fare un'altra query che la cerca sulla tabella:

```
SELECT parent FROM bookmarks WHERE id=$parent;
```

Tra l'altro ho creato un indice unico sul campo id e un altro sul campo parent della tabella (che non può essere unico, poiché più record possono avere lo stesso campo parent), in modo da velocizzare notevolmente entrambi gli unici due tipi di query a cui la sottoponiamo.

Semplice e veloce il DB, semplici gli script! Adesso non ci resta che scrivere il codice:

```
index.php
---------
<html>
<head><title>BookmarksDB</title></head>
<body bgcolor="#ffffff">
<h1 align="center">BookmarksDB</h1>
<hr noshade size=1>
<?php
                // The BOOKMARKSDB navigator //
                // permission is granted to use modified copies //
// of this program on condition that this //
// original copyright notice remains intact. //
                //                                              //
// Copyright 2001 Antonio Bonifati //
// <antonio.b@infinito.it>//
// <kiss@clubnautilus.net>//
// //
                // distributed under GPL (see COPYING)          //

// all that you might want to change is here
require 'CONF.php';

// connect to the mSQL database server
// using permanent connection if you can
HOSTNAME?msql_pconnect(HOSTNAME):msql_pconnect()
   or die(msql_error(). '<br><font color="red">
Please check HOSTNAME in the configuration file (CONF.php)<br>
and ensure a mSQL server is running there.</font>'. HTMLend);

// try to open db DBNAME
msql_selectdb(DBNAME)
    or die(msql_error(). '<br>
<font color="red">Cannot open database '. DBNAME. '.<br>
Have you set up the program correctly?<br>
Please use the install.php script to create the database first</font>'. HTMLend);
```

```
if (!isset($parent))
   $parent=0; // starts from root for default

$qid=msql_query("SELECT * FROM bookmarks WHERE parent=$parent")
    or die(msql_error(). '<br>
<font color="red">Query failed.<br>
Please check parent parameter.<br>
If it is right please check query in index.php and '. DBNAME .' database structure.<br>
To restore it you can drop the database and then ricreate it with install.php<br>
but doing so you will lose all the data you put in it.'. HTMLend);


//// print bookmark directory

// if we are not at root print the special parent link
if ($parent)
{
   $qid2=msql_query("SELECT parent FROM bookmarks WHERE id=$parent");
   printf(PREVDIRMARK . "\n", msql_result($qid2,0,0));
   msql_free_result($qid2);
}

if (!msql_affected_rows($qid))
   echo EMPTYMARK, "\n";
else
   while ($row=msql_fetch_array($qid, MSQL_ASSOC))
   {
      if ($row["url"]) // is a bookmark "file"
 printf(URLMARK . "\n", ereg('^[[:alpha:]]+://', $row["url"]) ?
($row["url"]) :("http://".$row["url"]), $row["name"]);
      else // is a bookmark "directory"
 printf(DIRMARK . "\n", $row["id"], $row["name"]);
   }
?>

<hr noshade size=1>
<h6 align="center">Copyright 2001
<a href="mailto:antonio.b@infinito.it">Antonio Bonifati</a>.
Distributed under <a href="COPYING">GPL</a>.</h6>
</body>
</html>
```

A parte qualche funzione nuova[10], di cui potete trovare la descrizione nel manuale PHP; per quanto riguarda il supporto per le espressioni regolari che è stato implementato in PHP: man 7 regex), non c'è niente di difficile o che non abbiamo spiegato prima.

Se volete provare gli script, potete caricare i dati esemplificativi visti prima nel DB usando l'utility msqlimport e passandogli questo file:

```
bookmarksDB.sample.asc
----------------------
1,0,PHP,,bookmark sul linguaggio PHP
2,0,Perl,,bookmark sul linguaggio Perl
3,1,main,www.php.net,sito ufficiale
4,1,tutorial,www.devshed.com,sito ricco di tutorial su PHP
```

[10] printf, posix_getuid, posix_getpwuid, ereg

```
5,2,main,www.perl.com,sito ufficiale
6,2,tutorial,,
7,6,cgi-lib.pl,cgi-lib.berkeley.edu,libreria web per Perl
8,6,it_tut,www.venis.it/francesco,tutorial in italiano
```

Il comando è semplicemente:

```
msqlimport bookmarksDB bookmarks <bookmarksDB.sample.asc
```

Se avete provato il programma, potete vedere che si comporta correttamente, però la lista di ogni direttorio non è ordinata. Sarebbe preferibile poter avere la possibilità di ottenere una lista ordinata in senso alfabetico. Anzi sarebbe meglio presentare prima tutte le directory ordinate in senso alfabetico e poi tutti i file, pure ordinati in senso alfabetico.

Questo ci porta purtroppo a modificare la struttura del db. Una soluzione che implica comunque una complicazione piuttosto bassa è la seguente. Occorre aggiungere un altro campo di un solo carattere, che indicherà il tipo di protocollo oppure se quella entry è una directory, anziché un file di collegamento. La convenzione è la seguente:

```
type CHAR(1) // campo da aggiungere nella tabella bookmarks

type | significato
-----+---------------------------------------
 D   | directory
 H   | file collegamento con protocollo HTTP
 F   | file collegamento con protocollo FTP
 f   | file collegamento con protocollo FILE
 G   | file collegamento con protocollo GOPHER
 M   | file collegamento con protocollo MAILTO
 N   | file collegamento con protocollo NEWS
 n   | file collegamento con protocollo NNTP
 T   | file collegamento con protocollo TELNET
 W   | file collegamento con protocollo WAIS
```

Il caso vuole (per nostra fortuna :) che la lettera `D` sia nel set ASCII alfabeticamente minore di tutte le altre, quindi facendo un doppio ordinamento in base al campo `type` e poi in base al campo `name`, si ha proprio il risultato voluto di listare prima le directory e dopo tutti gli altri file. Inoltre non è più necessario usare la funzione `ereg` per riconoscere se un url inizia con `ftp://` ecc... in quanto adesso l'informazione del tipo di protocollo è codificata nel campo `type`. Ordinando prima rispetto al campo `type` e poi rispetto al campo `name`, non si ottiene in generale un ordinamento alfabetico rispetto al campo `name`, come se si ordinasse solo in base a questo campo. Bensì gli URL vengono prima divisi in gruppi ordinati in base alla lettera rappresentativa del protocollo e poi questi gruppi sono ordinati in senso alfabetico. Questo è facile da implementare, in quanto lasciamo al db server l'onere di fare gli ordinamenti e facciamo una sola query per listare tutti gli elementi; tutto sommato la divisione in categorie ordinate dei file è accettabile.

All'inizio non avevamo voluto introdurre questa complicazione, poiché personalmente preferisco non introdurre complicazioni finché non sono necessarie. A questo punto è necessaria e le modifiche da fare per implementare il meccanismo sono piuttosto semplici. Ho dato la possibilità nel file di configurazione di scegliere se si vuole il precedente tipo di ordinamento oppure un altro possibile con quello schema di db oppure nessun tipo di ordinamento. Basta inserire una definizione di questo tipo:

```
define("ORDERBY", 'type, name')
oppure, se non si vuole alcun tipo di ordinamento:
define("ORDERBY", '')
```

Siamo giunti alla versione vicina a quella finale degli script

`CONF.php`, `install.php` e `index.php`

per mSQL, ossia agli script di installazione e visualizzazione del nostro programma web di gestione di un db di bookmarks. Un qualsiasi webmaster (anche se non conosce PHP) può inserire il db di bookmark in una pagina web: basta che sostituisca il codice HTML iniziale e finale in `index.php`; può inserire il codice compreso tra `<?php` e `?>` dove vuole che il listing interattivo appaia (ad es. potrebbe scegliere di farlo apparire in una cella di una tabella che definisce il layout della pagina).

Un webmaster che sa poco di mSQL e in generale di SQL e database (e purtroppo non vuole impararlo) non sarebbe comunque in grado di manipolare il db dei bookmark tramite il programma monitor di mSQL. Quindi per facilitare la vita a questi tipi ;) e anche per esercizio, svilupperemo alcuni script di amministrazione, che permetteranno di manipolare il DB dei bookmark senza sapere un'acca di SQL o della struttura del DB stesso.

Questi script comunque possono essere utili anche se chi deve manipolare il DB sarà il programmatore stesso. Un motivo è che mSQL non possiede un tipo enumerativo, quindi non è possibile porre dei vincoli al campo `type`. Se qualcuno per sbaglio assegna il valore '`z`' al campo `type` che non corrisponde ad alcun tipo di protocollo internet, il database server accetterà senza problemi quel valore. Bisogna ammettere quindi che gli script di amministrazione possono essere talvolta più comodi e sicuri rispetto all'interfaccia testuale in linguaggio SQL, impedendovi di fare errori (se sono ben scritti), ma naturalmente è possibile fare solo quell'insieme di manipolazioni predisposte dal programmatore con questa interfaccia.

2.2 Amministrare il bookmarksDB via web

Prima ancora di creare l'interfaccia voglio apportare alcune modifiche ed aggiungere altri script di visualizzazione.

Ho subito creato un nuovo file `lib.php` in cui condivido alcune funzioni comuni alla maggior parte degli script (la funzione `protocol_name` che implementa la corrispondenza tra codice e nome del protocollo di un collegamento) e due funzioni per provare a connettersi permanentemente al db e per provare a selezionare il DB opportuno. Questo salva una inutile ripetizione del codice. Notare che la funzione che stabilisce il link al db, `try_msql_pconnect`, salva in una variabile globale `$db` l'id della risorsa, in modo che la funzione che effettua la query di listing di una directory possa utilizzarla (addirittura implicitamente) senza che occorra passargliela per parametro.

Inoltre `do_listing_query` utilizza il globale `$parent`, ancora una volta per evitare un passaggio di un parametro. Essa restituisce il query id `$qid`, piuttosto che utilizzare un valore `$qid` globale, perché servirà avere più valori di `$qid` contemporaneamente sullo stack in uno dei processi ricorsivi che vedremo in seguito.

L'uso (accorto) di variabili globali può semplificare il codice, ma non bisogna esagerare, poiché dalla chiamata non è evidente quali variabili globali una funzione modifica. Questo è vero soprattutto nei casi in cui ci sono molte variabili globali.

Inoltre ho pensato di mettere il **require** di `CONF.php` solo in `lib.php`, visto che tutti gli script includono `lib.php`, eccetto `install.php` che include solo `CONF.php` e di rifinire la query di select in `index.php`. Invece di:

```
SELECT * FROM bookmarks WHERE parent=$parent ORDER BY ...
```

ho scritto

```
SELECT id, name, url, type FROM bookmarks WHERE parent=$parent
ORDER BY ...
```

È inutile infatti proiettare anche i campi `comment` e `id` se non li utilizziamo. In particolare per il campo comment rallenterebbe leggermente il programma, poiché il tipo TEXT di mSQL permette di memorizzare testi di lunghezza qualsiasi; testi di lunghezza minore o uguale alla dimensione media del campo vengono inseriti totalmente nella tabella. Il resto di testi più lunghi verrà inserito in un buffer di overflow esterno. Ovviamente questo è il motivo per cui l'accesso a dati TEXT è più lento rispetto ai dati CHAR di pari lunghezza, tuttavia permettono di risparmiare notevole spazio nei file del db (non è necessario dichiarare un CHAR(255) per una lunghezza media dei dati di 40 caratteri, solo perché ci sono pochi campi molti più lunghi di 40 caratteri).

Tra gli zuccherini che ho aggiunto vi è uno script che vi permette di vedere la struttura di directory espansa completamente o che vi espande l'intero bookmarksystem (file e directory) in una sola pagina. Infine ho aggiunto uno script che permette di effettuare una ricerca nel bookmarksystem di tutti i file e/o directory che contengono una stringa specificata. Il primo dei due script usa il metodo programmativo detto ricorsione, che risulta il più semplice e naturale su una struttura intrisencamente ricorsiva come un albero n-ario.

```
all.php
-------
<html>
<head><title>BookmarksDB: all in one page</title></head>
<body bgcolor="#ffffff">
<h1 align="center">BookmarksDB - all</h1>
<hr noshade size=1>
<?php
                // The BOOKMARKSDB one page navigator //
                // permission is granted to use modified copies //
// of this program on condition that this //
// original copyright notice remains intact. //
                //                                              //
// Copyright 2001 Antonio Bonifati //
// <antonio.b@infinito.it>//
// <kiss@clubnautilus.net>//
// //
                // distributed under GPL (see COPYING)          //

// include the common function library
require 'lib.php';

// try to connect to the mSQL database server
// using permanent connection
try_msql_pconnect();

// try to open db DBNAME
try_msql_selectdb();

if (!isset($parent))
   $parent=0; // starts from root for default

// print entire tree recursively
$reclevel=-1;
print_tree();

function do_indent()
{
   global $reclevel;
   if ($reclevel)
```

```
      printf(INDENTMARK, INDENTWIDTH * $reclevel);
}

function print_tree()
{
   global $parent, $withfile, $reclevel;

   $reclevel++;
   $qid=do_listing_query();

   //// print bookmark directory

   if (!msql_affected_rows($qid))
   {
      if (EMPTYMARK)
      {
 do_indent();
 echo EMPTYMARK, "\n";
      }
   }
   else
      while ($row=msql_fetch_array($qid, MSQL_ASSOC))
 if ($row["type"]=='D') // directory
 {
    do_indent();

    printf(DIRMARK . "\n", $row["id"], $row["name"]);
    $parent=$row["id"];
    print_tree();
 }
 elseif (isset($withfile)) // file
 {
    do_indent();

    printf(URLMARK . "\n",
protocol_name($row["type"]) .'://'. $row["url"], $row["name"]);
 }

   $reclevel--;
   msql_freeresult($qid);
}
?>
<hr noshade size=1>
```

```
<font size=-2><a href="./">Root index</a> |
<a href="search.php">Search</a> |
<?php if (isset($withfile)): ?>
<a href="all.php">Expand only directories</a>
<?php else: ?>
<a href="all.php?withfile=">Expand all</a>
<?php endif; ?></font>
<h6 align="center">Copyright 2001
<a href="mailto:antonio.b@infinito.it">Antonio Bonifati</a>.
Distributed under <a href="COPYING">GPL</a>.</h6>
</body>
</html>
```

Lo script di sopra funziona benissimo, ma anche qui allo scopo di ottimizzare è preferibile evitare che vengano prelevati anche tutti i record rappresentanti file se l'utente non ha passato il parametro vuoto `withfile`, cioè desidera che venga disegnata la struttura ad albero delle sole directory, senza includere i file contenuti in ciascuna directory. Questo si può ottenere complicando un po' la funzione `do_listing_query` passandogli una condizione da aggiungere alla sezione WHERE della SELECT (valore di default del parametro condizione è la stringa vuota). La versione finale ottimizzata è inclusa nello zip allegato a questo articolo.

Per quanto riguarda lo script di ricerca, anche questo è un po' complicato, in quanto ho dato la possibilità di delimitare a piacimento il campo di ricerca. È solo in questo script che viene stampato il campo commento (se non è vuoto). In pratica si inserisce un pezzo della stringa da cercare.

Si può scegliere se includere nella ricerca della keyword un qualsiasi sottoinsieme dei campi name, url, comment e se si vuole cercare nei record rappresentanti file e/o directory. Il lavoro usuale di uno script di ricerca in generale è di comporre la stringa rappresentante la query SQL e di mandarla al db server. Questo si occupa quindi della ricerca vera e propria. Lo script `search.html` non fa eccezione a questo schema elaborativo. Purtroppo i campi TEXT non sono confrontabili (è un limite di mSQL), quindi ho dovuto convertire tutti i campi TEXT nella tabella in CHAR, per implementare una ricerca così sofisticata a livello del db.

Ecco come appare (fig. 2.1) la maschera di ricerca con Lynx, un famoso browser in modo testo:

Passiamo ora agli script di amministrazione. Cominciate col creare una sottodirectory `admin` (o con un altro nome che preferite) dentro la directory in cui avete sviluppato il programma; io ho scelto `bookmarksDB`, quindi ho creato la directory `bookmarksDB/admin`.

L'idea è piuttosto semplice: prima di poter effettuare una operazione di cancellazione o modifica occorre trovare il file o la directory che si vuole modificare. Quando si vuole aggiungere un file o una directory, occorre trovare la directory padre che dovrà contenerlo. Dobbiamo fornire quindi strumenti di navigazione anche

```
                                                    BookmarksDB search

                          BookmarksDB search
   ------------------------------------------------------------------

  Keyword ____________________ Search

  names    [X]   files       [X]
  urls     [X]   directories [X]
  comments [X]
   ------------------------------------------------------------------

  Root index | Expand only directories | Expand all

         Copyright 2001 Antonio Bonifati. Distributed under GPL.
```

Figura 2.1: Come appare lo script in Lynx.

negli script di amministrazione. Siccome abbiamo già sviluppato questi strumenti di navigazione, perché allora non copiare? Basterà fare:

```
cp index.php all.php search.php admin
```

Inoltre ho deciso di spostare lo script `install.php` nella directory `admin`. Poiché l'accesso a questa directory sarà sottoposto ad una password, è meglio metterlo lì e inoltre l'installazione è un'operazione che comunque compete all'amministratore:

```
mv install.php admin
```

ho tolto da questo file il suggerimento di cancellarlo una volta usato, poiché può essere utile conservarlo dentro `admin`. Anche il file `CONF.php` l'ho spostato dentro `admin`. Anche se da web richiamando questo file si ottiene il classico errore di Netscape "The document contained no data. Try again later, or contact the server's administrator", poiché non contiene nemmeno una linea vuota dopo la fine, `CONF.php` sta logicamente meglio dentro `admin`, così come `bookmarksDB.sample.asc`.

Infine ho spostato anche `lib.php` dentro `admin`, in modo da lasciare nella root del programma solo la licenza COPYING, i tre file php visibili senza password sul web e i due file di immagine collegati a quest'ultimi.

È opportuno creare dentro admin dei collegamenti simbolici a queste immagini col comando UNIX `ln -s`, altrimenti gli script di visualizzazione di admin non li vedrebbero. Analogamente occorre creare un collegamento per il file `CONF.php` di `admin` nella root del programma.

```
[17:26:39 root@localhost /usr/local/apache/htdocs/bookmarksDB]# ln -s
admin/CONF.php .
```

quindi modificate tutti i titoli delle pagine in admin aggiungendo

`"- ADMIN", per ricordare che siamo negli script di amministrazione.`

L'idea è di aggiungere due iconcine subito dopo una entry: una con una X per cancellare quel file o quella directory (che chiederà conferma e poi ritornerà alla pagina di listing precedente mediante un semplice codice JavaScript - chiamando il metodo `go` dell'oggetto `history`) e una con una matita per modificarlo. Infine in testa ad ogni listato di directory aggiungeremo una iconcina con la forma di un foglio bianco (a righe ;) su cui cliccare per creare una nuova entry in quella directory. Per rendere possibile stabilire il markup di queste iconcine nel file `CONF.php`, ho dovuto aggiungere la definizione di ENDMARK, poiché il markup delle icone va inserito prima di un `<br>` o simili. Siccome il `<br>` era compreso nelle definizioni di DIRMARK, PREVDIRMARK, ecc..., non c'è un modo semplice di aggiungere il markup delle icone, perciò ho tolto i `<br>` da lì e li ottengo facendo stampare ENDMARK (che contiene di default `<br>`) dopo ogni DIRMARK, PREVDIRMARK, ecc...

Quindi ho aggiunto in `CONF.php` le definizioni dei markup per le iconcine. Poi ho scritto lo script `delete.php` che viene chiamato allorquando si clicca sulla icona di cancellazione a forma di x. A `delete.php` viene passato semplicemente l'id dell'entry da cancellare. Se viene passato anche il parametro vuoto `sure`, allora lo script non chiederà conferma. Notare che la conferma consiste nel cliccare di nuovo sulla x accanto ai dati dell'entry che si vuole cancellare che vengono riproposti per comodità. In effetti dopo la conferma lo script richiama se stesso con lo stesso argomento id ma con in più il flag `sure` e questo innesca la funzione di cancellazione, che in generale sarà ricorsiva e permetterà la cancellazione di intere directory in un solo colpo.

Dopo la cancellazione viene emesso del codice JavaScript che accedendo alla history list del browser redirige il client alla pagina precedente automaticamente, in modo da risparmiare all'utente la pressione del tasto back del browser per due volte dopo una cancellazione per poter ritornare alla pagina di listing precedente (che può essere `index.php`, `search.php` oppure `all.php`). Se il vostro browser non supporta JavaScript o è settato per ignorare JavaScript, viene proposto un messaggio che dice che la cancellazione è stata effettuata e che per tornare indietro occorre premere il tasto back del browser due volte (o una volta se si è impostato il programma per non chiedere conferma in caso di cancellazione).

Naturalmente, anche nel caso in cui il browser supporta JavaScript e questo è attivato, potete scegliere di disattivare la redirezione automatica, settando la booleana REDIRONDEL nel file `CONF.php`. Un'altro dettaglio degno di nota è che la redirezione dopo una cancellazione potrebbe portare ad una pagina non aggiornata (l'entry cancellata è ancora presente nella visualizzazione della pagina), a causa del fatto che molti browser sono impostati per fare il caching delle pagine web almeno durante una stessa sessione di navigazione. Per ovviare a questo problema, nei tre file che fanno il listing di `admin` ho fatto emettere degli header HTTP che indicano ai browser di non fare il caching della pagina e ho dato la possibilità in `CONF.php` di non farlo per tutte le pagine[11]. Infine non c'è modo di cancellare di botto tutto il contenuto del database (l'analogo del pericoloso comando UNIX "`rm -fr /`"). Quindi ho provveduto facendo apparire la scritta "root" seguita da una crocetta cliccabile nel listing della root in `index.php` e anche in `all.php`. Inoltre occorre gestire questo caso particolare di conferma in `delete.php`. Invece di srotolare il processo ricorsivo anche quando si cancella tutto, per ottimizzare, preferisco fare una sola query DELETE priva della condizione di WHERE. Se dovreste abbattere un intero grosso albero, evidentemente vi conviene imbracarlo e tagliarlo dal tronco, piuttosto che mettersi a tagliare singolarmente ogni ramo.

Dapprima avevo scritto la funzione ricorsiva di cancellazione in questo modo:

```
// deletes an entire directory tree
// starting from the entry with number $id
function delete_tree()
{
   global $id;

   // first delete this directory
   msql_query("DELETE FROM bookmarks WHERE id=$id");

   // then recursively all the files and directories below
   // we can avoid a recursive call for a file
   // i think this saves a bit of stack space and is quicker
   $qid=msql_query("SELECT id, type FROM bookmarks WHERE parent=$id");
   while ($row=msql_fetch_array($qid))
      if ($row["type"]=='D')      // directory
      {
 $id=$row["id"];
 delete_tree();
      }
      else                       // file
 msql_query("DELETE FROM bookmarks WHERE id=$row[id]");
}
```

È sicuramente corretta e abbastanza efficiente, ma può essere migliorata riducendo di molto il numero di query effettuata al db server. Infatti piuttosto che cancellare ogni file di una directory singolarmente, è più veloce cancellarli tutti in una volta e risparmiarsi un maggior prelievo di dati dal db server. Ecco quindi la versione ottimizzata, che è quella che ho incluso nella versione finale del programma:

```
// deletes an entire directory tree
// starting from the entry with number $id
function delete_tree()
```

[11]Vedi a proposito nel manuale PHP l'help sulla funzione `header`, file `function.header.html`.

```
{
   global $id;

   // first delete this directory and all its files
   msql_query("DELETE FROM bookmarks WHERE id=$id OR
              (parent=$id AND type<>'D')");

   // then recursively all the directories below
   $qid=msql_query("SELECT id FROM bookmarks WHERE parent=$id");
   while ($row=msql_fetch_array($qid))
   {
 $id=$row["id"];
 delete_tree();
   }
   msql_freeresult($qid);
}
```

Notate che l'unica variabile "replicata" dal processo ricorsivo è `$qid` e i buffer a cui è collegata, che preferisco liberare quanto prima. Questa funzione dimostra l'utilità di uno script: cancellare un intero ramo a partire da un certo modo (operazione analoga alla potatura di un albero ;) può essere seccante poiché occorre dare parecchie query. La funzione `delete_tree()` ci evita la seccatura.

Vorrei sottolineare che qui l'automazione è fornita da uno script: non è mai l'interfaccia grafica che fornisce la possibilità di automazione, bensì sono i linguaggi. Questa affermazione potrebbe sembrare banale, ma ci sono molti che sostengono che non si può utilizzare efficientemente un computer senza interfaccia grafica.

Chi lavora con UNIX sa bene che le interfaccie grafiche sono solo un optional e rispetto alle interfaccie in modo testo hanno vantaggi e svantaggi evidenti. È molto meglio disporre di entrambi i paradigmi di comunicazione con l'elaboratore piuttosto che di uno solo (come succede in certi blasonati sistemi operativi il cui nome inizia con W), poiché si può scegliere quello più rapido e conveniente a seconda dei casi. La grande fortuna di UNIX sta proprio nel fatto di essere un sistema operativo modulare e in grado di operare in rete e conforme a dei solidi standard. I detrattori di UNIX sostengono che è ostico e difficile da usare, ma è ovvio che UNIX non potrebbe fornire all'utente tanta potenza e flessibilità senza che questo sappia cosa sta facendo. Molti altri sistemi operativi vi permettono di fare poche cose e come tali sono certamente facili da usare, sicché potete imparare tutto quel poco che vi serve più di frequente in poco tempo, ma rendono impossibile fare compiti più difficili. La complessità non è un prezzo troppo alto che si paga per la potenza, che è spesso necessaria e inoltre anche UNIX rende facili i compiti più frequenti rendendo praticabili e automatizzabili anche quelli più difficili. Man mano che la mia conoscenza di UNIX cresce, lo trovo sempre più coerente, facile e rapido da usare e piacevolmente ricco più di ogni altro sistema operativo che usa solo il paradigma grafico.

Abbiamo finito con la cancellazione, occupiamo ora delle operazioni di inserimento e di modifica. L'operazione di modifica potremmo anche pensare di non fornirla, poiché si può ottenere tramite una cancellazione e poi un successivo inserimento. Sarebbe tuttavia scomodo per l'utente, come sarebbe scomodo se non si potesse editare un testo e si dovrebbe cancellarlo e ricrearlo daccapo per correggere un qualsiasi errore, anche banalissimo. Quindi ho deciso di fornirla. Ho sviluppato prima lo script di inserimento di una nuova entry (`new.php`), che è più semplice e poi l'ho copiato e modificato per ottenere lo script di modifica.

Lo script `new.php` ha bisogno di utilizzare un valore del campo id (che deve essere obbligatoriamente specificato, poiché un campo indice non può essere NULL) che non coincida con nessun valore già specificato (perché l'indice è del tipo UNIQUE). Questo valore può essere ottenuto dal db server mSQL mediante il meccanismo delle sequenze. In pratica è possibile associare ad una tabella un numero che viene memorizzato dal db server insieme alla tabella e che può essere inizializzato con un valore qualsiasi (il default è 1) e incrementato con un passo qualsiasi, anche negativo (per essere più precisi in tal caso sarebbe "decrementato", anziché "incrementato").

Per creare una sequenza, occorre far mandare anche questa semplice query ad `install.php`:

```
CREATE SEQUENCE ON bookmarks
```

che esprime semplicemente di associare un valore di sequenza alla tabella bookmarks e inizializzarlo col valore di default (1) e incrementarlo del valore di default (ancora 1) dopo ogni accesso, in modo che accesso e incremento siano un'operazione atomica, non interrompibile da altri processi. Per accedere al valore attuale della sequenza e incrementarlo (un po' come se scrivessi `_seq++` in PHP), basta lanciare la query seguente:

```
SELECT _seq FROM bookmarks
```

la prima volta restituirà:

```
+----------+
| _seq     |
+----------+
| 1        |
+----------+
```

la seconda volta:

```
+----------+
| _seq     |
+----------+
| 2        |
+----------+
```

ecc...

Nello script `new.php` lanciamo questa query per ottenere un valore unico per il campo id prima di effettuare l'inserimento con una query INSERT.

Non è una buona idea crearsi una tabella a parte di una sola riga e colonna e pensare di poter gestire correttamente autonomamente dal proprio script un valore di sequenza senza ricorrere all'istruzione CREATE SEQUENCE, non solo perché quest'ultima è più semplice da usare, ma anche perché non avremmo modo di garantire che non si verifichi una "race condition" quando accediamo ad un valore di sequenza.

La "race condition" è un casino raro ma possibile quando lo stesso script è usato da più utenti, che si verifica quando uno script vuole accedere al valore della sequenza; lo legge e subito dopo capita che il sistema operativo decida di interrompere quel processo per continuare l'esecuzione di un altro. Se inoltre capita che quest'altro processo è un'altra copia dello stesso script in esecuzione e riesce a leggere il valore di sequenza prima che il controllo sia ritornato al primo script che lo incrementerà, entrambi gli script si ritroveranno con lo stesso valore di sequenza e chi lo utilizzerà per ultimo per fare l'inserimento fallirà con errore o verranno inseriti due record con lo stesso id (a seconda rispettivamente che l'id sia indice unico o no). La "condizione di gara" invece non si può mai verificare se l'operazione di selezione e incremento del valore di sequenza è nel complesso atomica, ossia non interrompibile. Il DB server garantisce questo, mentre non avete modo di garantirlo da un'applicazione che lo usa, quindi occorre utilizzare le sequenze messe a disposizione dal DB server.

Una cosa da ricordare con le sequenze è che se si importano alcuni dati in un db tramite `msqlimport`, occorre indicare al programma di fare una query di selezione del valore di sequenza (cioè `'SELECT _seq from table'`, dove `table` è il parametro `table` sulla riga di comando) prima di ogni query di tipo INSERT e bisogna inoltre indicare in quale numero di campo (o "colonna") (contato da 0) inserire questo valore di `_seq` al posto di quello letto dal file. Ciò si effettua con la sintassi seguente:

```
msqlimport -m n:s database table <table.dump
```

Quindi il comando corretto per importare il database di esempio in formato ASCII è il seguente:

```
msqlimport -m 0:s bookmarksDB bookmarks <bookmarksDB.sample.asc
```

Nella nostra tabella in cui ci sono legami tra le varie tuple del tipo padre/figlio, non propriamente relazionali, l'importazione va a buon fine, se la sequenza non è stata mai incrementata è perché `bookmarksDB.sample.asc` contiene la lista delle tuple ordinata nel senso crescente del campo id. In pratica fatela subito dopo aver lanciato `install.php`.

Cosa succede se non ci sono più valori di sequenza? Una sequenza viene memorizzata da mSQL in un valore senza segno a 4 byte (32 bit), quindi potrà variare tra [0, 2^32-1=4294967295] dopodiché i valori ciclano, ripartendo da 0, come tipico della rappresentazione in complemento a due. Il problema "si aggrava" poiché i valori di sequenza che erano associati ad elementi cancellati non vengono riutilizzati. Non cercherò di fornire rimedi per questo spinoso problema, semplicemente lo ignorerò, poiché è praticamente impossibile in questa applicazioncina avere un tale numero di inserimenti nel db in modo da effettuare 2^32 volte o più l'incremento della sequenza, a meno che non state pensando di inserire tutti i possibili link presenti su internet, cosa praticamente impossibile :)

Nello script `edit.php`, notate che ho utilizzato la funzione `str_replace` che permette di sostituire una sottostringa con un'altra per aggiungere l'attributo SELECTED nel giusto tag `<option>`. Questo è un utilizzo tipico. Ho inoltre espresso la lunga stringa HTML in cui effettuare la sostituzione con la sintassi `here doc`, anche se avrei potuto usare le virgolette singole e spezzare la stringa su più linee. Se siete preoccupati con la compatibilità con le versioni precedenti al 4, in cui la sintassi `here doc` non c'è, non dovreste usarla.

Una cosa interessante da notare è che in entrambi gli script di inserimento e di modifica non mi sono preoccupato di programmare l'escapatura di eventuali caratteri come `'` che all'interno di stringhe quotate tra `'` nelle query SQL vanno naturalmente sempre escapati, pena errore di sintassi nella query. Se provate ad inserire caratteri di questo tipo nei campi di input, notate che non si verificano problemi. Come è possibile? La ragione è che se nel file `php.ini` l'opzione `magic_quotes_gpc` è settata ad On, PHP automaticamente escapa i caratteri `'` (virgoletta-singola), `"` (virgoletta doppia), \ (backslash) e `NUL` (carattere di codice 0) con un backslash nei dati provenienti dalle operazioni di Get/Post/Cookie (abbreviato gpc). Questo è di solito quello che volete, perché spesso questi dati dovete inserirli in un DB perciò è desiderabile lasciare ad On questa opzione nel file di configurazione. Tuttavia se dovete salvare i dati provenienti dalle operazioni di Get/Post/Cookie in un file di testo o ristamparli a video, vi ritroverete i caratteri di prima con un fastidioso escape che li precede. Per ovviare a questo potete utilizzare la funzione:

string **stripslashes** (string *str*)

questa funzione restituisce una stringa in cui ogni `\'` viene sostituito con `'`, ogni \\ con \ ecc..

In generale non c'è una scelta migliore in assoluto per il valore di questa opzione e purtroppo non si può settarla a runtime solo per un determinato script, essendo un comportamento fortemente cablato nell'interprete. Poiché uno per default può aver deciso di metterla ad off, dobbiamo far sì che i nostri script si comportino correttamente (escapando gli `'` nelle query) anche in questo caso. Ci viene in aiuto una funzione predefinita sulle stringhe:

string **addslashes** (string *str*)

questa è l'inverso della funzione `stripslashes` vista prima: `stripslashes` toglie le \ davanti a `",\,\0` mentre questa funzione ce lo mette. Poiché chiamare `addslashes` su qualcosa di già escapato produce una doppia escapatura (\' diventa \\\'), prima di chiamarla occorre verificare che la direttiva `magic_quotes_gpc` non sia già on. Per ottenere questa informazione PHP mette a disposizione la funzione:

int **get_magic_quotes_gpc** (`void`)

che ritorna 1 per on e 0 per off. Quindi aggiungeremo del codice tipo:

```
if (!get_magic_quotes_gpc)
{
   $name = addslashes($name);
   $comment = addslashes($comment);
}
```

I file in cui queste istruzioni sono necessarie sono

`admin/new.php`, `admin/edit.php`, `admin/search.php` e `search.php`.

In particolare in questi ultimi due occorre fare:

```
// quote ' in $keyword if PHP doesn't do that automagically
if (!get_magic_quotes_gpc())
   $keyword = addslashes($keyword);
```

Questo rende i nostri script più robusti.

Oltre alla modifica dei valori dei campi `name`, `url`, `comment`, ho ritenuto utile anche poter modificare il campo `parent`. In effetti durante l'utilizzo degli script, con i quali gestisco la mia collezione di link, mi è capitato di frequente di voler spostare dei file da una parte all'altra, perciò ho creato un'altra iconcina (quella a forma di doppia freccia) che richiama un altro script che presenta l'albero delle directory in modo analogo allo script `all.php` richiamato senza il parametro `withfile` e permette quindi di selezionare la directory padre a cui si vuole spostare semplicemente cliccandoci sopra.

In pratica l'id della directory selezionata da questo navigatore andrà a costituire il valore (eventualmente diverso da quello presente in precedenza) del campo `parent` del record modificato. Il file `move.php` l'ho ottenuto copiando `all.php` e modificandolo, piuttosto che partire da zero.

Cliccando sull'icona della doppia freccia in uno dei navigatori (`index.php`, `all.php` o `search.php`) viene richiamato lo script `move.php` passandogli l'id del record del quale vogliamo modificare il campo `parent`. Cliccando su una directory del navigatore viene ripassato questo id nel parametro di nome `id` e l'id della nuova directory padre nel parametro di nome `parent`, come al solito.

Occorre osservare che nel listing ricorsivo di `move.php` bisogna evitare di listare le sottodirectory della directory da spostare e la directory stessa (in altri termini bisogna escludere l'intero ramo che parte dal nodo da spostare, includendo il nodo stesso). Infatti non ha senso cercare di spostare una directory in una delle proprie sottodirectory o in sè stessa; l'operazione produrrebbe un riferimento circolare tra le due directory (ciascuna è padre dell'altra, oppure una directory è padre di sè stessa) e la struttura risultante non sarebbe più ad albero. Questo è piuttosto facile da ottenere (guardate il codice della funzione `print_rest_of_tree` in `move.php` che è una versione leggermente modificata della `print_tree` di `all.php`). Inoltre per permettere lo spostamento di una directory a livello della root, operazione sempre possibile, poiché la root contiene tutte le directory e nessuna directory contiene la root, occorre aggiungere un link speciale alla root nel navigatore di `move.php`.

Se siete curiosi di sapere perché mi serviva quest'ultima funzionalità di spostamento (a cui dapprima in verità non avevo pensato) ecco la spiegazione: spesso creavo una directory su un certo argomento (es. Php) inserendoci man mano link. Quando il numero di link cresceva, sentivo il bisogno di organizzarli meglio all'interno di questa directory, quindi creavo altre sottodirectory (es. tutorial per tutti i link riguardanti i tutorial su PHP e code per tutti i link riguardanti siti in cui si può scaricare codice sorgente in php, ecc...) e poi avevo bisogno di spostare i file di un livello inferiore in queste sottodirectory. Cancellare e ricreare i file, oppure lavorare con due finestre e fare taglia e incolla dei dati da una parte all'altra era molto scomodo, così ho deciso di aggiungere `move.php`.

Se è stata un po' dura affrontare la comprensione di questo programma consolatevi col fatto che è stata dura anche per me scriverlo ;) In ogni caso penso ne sia valsa la pena; infatti a parte il fatto che a me il database di bookmark serviva, alcune applicazioni web, come ad es. un forum hanno una struttura gerarchica similare e adesso probabilmente sarete in grado di crearne una sulla falsa riga di questa esperienza.

Se per caso doveste trovare errori negli script fatemelo sapere. Per favore cercate di includere quante più informazioni potete affinché possa riprodurre e capire l'errore. Sarò lieto di risolvere l'eventuale problema.

2.2.1 Limitazioni, altri dettagli aggiunti e consigli d'uso del programma

- il programma permette di creare una directory o file con un nome che già esiste nella stessa directory. Non ci sono problemi, poiché anche se hanno lo stesso nome, le due entry hanno un id diverso e siccome vengono listate entrambe a video e si possono selezionare cliccandoci, piuttosto che indicandone il nome, non c'è di fatto ambiguità. Tuttavia è consigliabile non farlo, perché confonde chi deve utilizzare il programma avere più file o directory con lo stesso nome allo stesso livello.

- un altro dettaglio di interesse da curare è che tutte le URL contenute in una pagina web devono essere scritte in modo che i caratteri definiti dallo standard RFC1738 come *"unsafe"* siano encodati nella forma ‘`%xy`’, dove ‘`xy`’ è la rappresentazione esadecimale del valore ASCII del carattere. I caratteri "unsafe" comprendono tutti i caratteri non alfanumerici, eccetto questi tre "`-_.`". Ad es. ‘`%`’ viene encodato come ‘`%25`’, ‘`:`’ come ‘`%3A`’, ‘’ come ‘`%40`’ e ‘ ’ come ‘`%20`’. La lista completa di trova nel documento RFC1738. PHP mette a disposizione una funzione già pronta per effettuare la codifica che è:

string **rawurlencode** (string *str*)

e la sua inversa:

string **rawurldecode** (string str)

Però, poiché gli spazi nei parametri vanno encodati come `+` anziché come `%20` (come fa la funzione `urlencode`, questa è la differenza con `rawurlencode`), e occorre distinguere le varie parti dell'URL da encodare e quali no, cosa che le precedenti funzioni non fanno, per semplicità il programma non cerca di encodare gli url. Dovete fornirli già nella forma encodata nella form di `new.php` o `edit.php`.

Questo per pigrizia del programmatore ;). Si potrebbero individuare le varie parti dell'url tramite la funzione:

array **parse_url** (string *url*)

e quindi codificare quelle che vanno codificate nel modo opportuno e quindi ricostruire l'intera url correttamente codificata. Si può pensare di memorizzare URL non codificate nel DB (decodificandole se vengono fornite in forma codificata nelle form) e poi di codificarle al volo con una apposita funzione negli script di visualizzazione, ma non voglio calcar troppo la mano ;)

Questo programma dimostra cosa fa la funzione `parse_url`:

```
<html>
<head>
 <title>URL parser</title>
</head>
<body>
<form>
<h1>URL parser</h1>
url <input type=text name=url>
</form>
<?php
   $urlpiece=parse_url($url);
   echo "$url<p>";
```

```
    foreach ($urlpiece as $key => $value)
      echo "$key => $value<br>\n";
?>
</body>
</html>
```

- in `search.html`, mettendo come keyword '`%`' la keyword fa match con qualsiasi elemento selezionato nella ricerca.
- ho aggiunto una iconcina a forma di punto interrogativo vicino ad ogni entry negli script di visualizzazione. Soffermando il mouse su quella iconcina viene mostrato il commento relativo a quell'entry sotto forma di ALTernativa testuale associata all'immagine (tranne in `search.html` dove i commenti vengono già mostrati su linea a sé stante). Gli script sono un po' più pesanti, ma volevo avere i commenti accessibili non solo dalla ricerca.
- aggiunta icona che rimanda al listing della directory padre quando il risultato di una ricerca è un file (a forma di una piccola cartella).
- notate che nel campo comment si può (e si deve) inserire del testo in formato html. Questo significa che potete includere dei marcatori, quindi per forzare un accapo occorre scrivere `<br>`, ecc... Il testo presente in `comment` viene interpretato come HTML. Questo offre la possibilità di formattare a piacimento i commenti. Tuttavia la formattazione si perde quando il commento viene presentato come ALT della iconcina col punto interrogativo (a tal scopo ho usato la funzione `strip_tags`). Il campo `name` invece viene sempre sottoposto alla funzione:

  string **htmlspecialchars** (string *string*)

  prima di essere stampato, la quale effettua le seguenti sostituzioni di caratteri speciali in entità html:

  ```
  '&' (ampersand) =>'&amp;'

  '"' (double quote) =>'&quot;'

  '<' (less than) =>'&lt;'

  '>' (greater than) =>'&gt;'
  ```

- è opportuno restringere l'accesso alla directory di `admin` solo a determinati indirizzi IP, oppure solo a chi sia in grado di fornire una giusta accoppiata username, password. L'argomento non è tanto banale e fa parte della documentazione di Apache[12].

[12] Un tutorial si trova qui: http://www.apacheweek.com/features/userauth.

- ho aggiunto il titolo contenente il nome della directory corrente formattabile in modo personalizzato tramite la definizione di TITLEMARK in `CONF.php`.

2.3 MySQL API: bookmarksDB per MySQL

Lo scopo di questa lezione è di trasportare i concetti visti nella lezione precedente nel caso in cui come db server, anziché mSQL, si utilizzi MySQL. MySQL è un db server molto più potente di mSQL (ad es. offre una implementazione del linguaggio SQL più potente ed una maggiore varietà di tipi di dati) ed è addirittura gratuito in Unix, quindi vale certamente la pena di apprenderne l'utilizzo.

Alla fine di questa lezione sarete in grado di convertire la nostra utile applicazione bookmarkDB in modo che possa funzionare con MySQL. In caso non vi riuscite o non avete tempo per affrontare questo esercizio, potete limitarvi a dare un'occhiata al codice della versione per MySQL di bookmarksDB fatta da me (vedi link all'inizio del tutorial).

Il sito principale di MySQL è http://www.mysql.com, che è mirrorato in parecchi paesi (tra cui anche l'Italia). Qui potete scaricare sorgenti e pacchetti binari precompilati. Il file `install.txt` allegato a questo tutorial descrive l'installazione dell'ultima versione binaria stabile per Linux, precompilata per 686.

Le API per accesso ai database di PHP si assomigliano concettualmente tutte, quindi non avrete molta difficoltà ad imparare quella di MySQL, anzi la spiegherò mettendone spesso in evidenza le differenze con quella di mSQL, che sono piuttosto poche, infatti molte volte basta semplicemente sostituire un "msql" con "mysql", quindi probabilmente vi annoierete un po' perché ripeterò alcune descrizioni; vi assicuro tuttavia che la prossima lezione sarà più divertente ;)

La prima funzione da capire è quella che apre una connessione ad un server MySQL:

int **mysql_connect** ([string *hostname* [:*port*] [:*/path/to/socket*] [, string *username* [, string *password*]]])

Analogamente a `msql_connect`, questa restituisce un id positivo che identifica la risorsa di connessione. In caso di errore la funzione restituisce il booleano FALSE e fa stampare un warning del tipo:

```
Warning: MySQL Connection Failed: DESCRIZIONE_DELL'ERRORE
in __FILE__ on line __LINE__
```

dove `__FILE__` indica il nome e il percorso completo dello script dove si è verificato l'errore, `__LINE__` il numero di linea in cui l'errore si è verificato e DESCRIZIONE_DELL'ERRORE non ha bisogno di spiegazioni.

La differenza con `msql_connect` è che non solo si può specificare l'host a cui connettersi (che per default è lo stesso che esegue gli script, cioè '`localhost`'), ma che si può specificare anche un nome utente ed una password, che invece non sono supportate da mSQL. Il nome utente infatti con MySQL può essere anche diverso dal nome utente Unix di chi si connette, in quanto gli utenti di Unix e gli utenti di MySQL non hanno nulla a che vedere tra loro, al contrario di mSQL. L'uso delle password permette ovviamente di poter verificare l'identità dichiarata da chi si connette: tanto per fare un esempio si può essere sicuro che l'utente **ant** sia davvero **Antonio Bonifati** e non un detrattore ;) se gli si permette di connettersi solo se fornisce la giusta password, che il buon Antonio ha scelto in modo che sia difficilissimo indovinare e che custodisce gelosamente in qualche anfratto della sua memoria ;). In effetti MySQL fornisce meccanismi di autenticazione e di controllo dei permessi su un database molto più sofisticati di mSQL (il manuale di MySQL li tratta ampiamente, per cui vi rimando a quello). Il nome utente di default di `mysql_connect` è quello dell'utente sotto cui gira il server web, tipicamente **nobody**, mentre la password di default è vuota (ossia ”).

Nota: sul mio sistema l'utente nobody non ha nessun permesso globale e occorre fornirgli i privilegi necessari per amministrare i singoli db. Fornisco per ogni db solo i privilegi strettamente necessari e questo assicura una certa sicurezza. Nel caso del db bookmarks DB ho dato il seguente comando come utente che ha il privilegio di amministrare le tabelle dei privilegi:

```
GRANT SELECT,INSERT,UPDATE,DELETE
   ON bookmarksDB.*
   TO nobody@localhost
```

In pratica nel caso più frequente in cui ci si collega al server MySQL sullo stesso host, come utente nobody (o altro utente su cui gira il server web), il quale non ha bisogno di nessuna password, i comandi:

```
mysql_connect('localhost','nobody','');
mysql_connect('localhost','','');
mysql_connect('','nobody');
ecc...
```

possono essere abbreviati al massimo così:

```
mysql_connect();
```

Nel caso di fallimento la maggior parte delle volte non ha molto senso continuare l'esecuzione dello script, quindi potete terminarla subito dopo l'emissione del Warning scrivendo codice del tipo:

```
if (!mysql_connect())
    exit();

mysql_connect('anotherhost') or exit();
```

```
$link = mysql_connect ('kamikazen', 'antoine', 'secret')
  or die ('Could not connect');
print ('Connected successfully');
```

La funzione di PHP:

void **exit**(void);

semplicemente termina l'esecuzione dello script corrente come se fossimo giunti alla fine dello script e non restituisce niente.

Lo stesso la funzione:

void **die** (string *message*)

che però prima stampa il messaggio "message".

Per evitare la stampa del messaggio di warning preperendere alla chiamata di funzione l'operatore :

```
$link = @mysql_connect ('kamikazen', 'antoine', 'secret')
or die ('Could not connect');
print ('Connected successfully');
```

Il simbolo di at () è un operatore di controllo dell'errore. Quando viene prefisso ad una espressione, qualsiasi messaggio di errore che quella espressione potrebbe generare non verrà stampato e sarà ignorato. Non è consigliabile fare abuso di questo operatore. Considerate il seguente esempio in cui viene chiamata una funzione non definita:

```
<%
@uffa();
echo "testo successivo"; // non viene stampato
%>
```

A causa della presenza di davanti all'espressione (che in tal caso consiste semplicemente di una chiamata di funzione), lo script termina senza alcuna indicazione di errore. L'operatore è utile soprattutto se avete intenzione di fornire un messaggio di warning personalizzato (come nell'esempio di `mysql_connect` precedente), mentre non è consigliabile far sì che non venga emesso neanche un messaggio di errore, che può aiutare a far luce sul problema. Anche se può essere imbarazzante per un webmaster il fatto che agli utenti escano messaggi d'errore nelle pagine, è sempre meglio che esca un messaggio di errore e che l'utente abbia la possibilità di comunicare qual'è il problema al webmaster, piuttosto che una pagina vuota o piena a metà, no?

Con `msql_connect` c'è la stessa limitazione che si può avere una sola connessione a db server per script CON GLI STESSI ARGOMENTI nella chiamata a `mysql_connect`; in altri termini, richiamando `mysql_connect` con gli stessi argomenti viene ritornato lo stesso id e non viene creata una nuova connessione.

Analogamente a come succedeva con mSQL il link al server verrà chiuso automaticamente non appena lo script termina, ma la sua chiusura può essere anticipata chiamando la funzione:

int **mysql_close** ([int *link_identifier*])

questa (ancora come `msql_close`) in mancanza dell'argomento usa l'ultimo link aperto.

Come nel caso di mSQL, è meglio utilizzare dei link persistenti: la funzione `mysql_pconnect` realizza una connessione persistente e si usa allo stesso modo di `mysql_connect` (dal punto di vista dell'utilizzazione cambia solo il nome e c'è il fatto che `mysql_close` non chiuderà un link persistente):

int **mysql_pconnect** ([string *hostname* [:it port] [`:/path/to/socket`] [, string *username* [, string *password*]]])

come spiegato nel caso mSQL, un link persistente non viene chiuso alla fine dello script, ma rimane aperto per usi futuri, infatti ogni chiamata di `mysql_pconnect` cerca prima nell'elenco dei link persistenti già aperti se è aperto un link sullo stesso host con lo stesso nomeutente e password. Se viene trovato un link di questo tipo, viene ritornato l'id relativo a questa risorsa già allocata, anziché aprire una nuova connessione, e questo può incrementare le prestazioni su un server molto trafficato, quindi è consigliato l'uso di link persistenti.

Come con mSQL, affinché PHP possa usare connessioni persistenti al DB di MySQL, l'opzione:

```
mysql.allow_persistent
```

deve essere `On` nel `file php.ini` (questo è il default).

Anche qui si può lavorare con un solo db per volta e per scegliere il db corrente per una data connessione, si usa una funzione simile a `msql_select_db` (difatti la sua descrizione è pari pari quella di `msql_select_db`, dove al posto di `msql` ho sostituito `mysql` ;):

int **mysql_select_db** (string *database_name* [, int *link_identifier*])

il primo argomento è il nome del DB. Il secondo deve corrispondere al valore ritornato da `msql_connect` o `msql_pconnect` (comunque se non specificato viene usata l'ultima connessione aperta al db con una di queste due funzioni; se nessuna di queste due funzioni era stata chiamata, `mysql_select_db` invoca automaticamente la prima (`mysql_connect`) senza argomenti). Se avete finito di lavorare su un db e volete passare ad un altro, basta chiamare `mysql_select_db`, senza prima dover chiudere e poi riaprire la connessione al db server che sarebbe più lento. Anche questa funzione ritorna TRUE se tutto va bene, FALSE se c'è qualche errore (ad es. il database cercato non esiste, o non riesce a connettersi al db server).

La funzione API esternamente analoga a `msql_query` è (indovinate un po'):

int **mysql_query** (string *query* [, int *link_identifier*])

che agisce sul db corrente della connessione identificata da `link_identifier`. Questa funzione, nel caso si ometta

`link_identifier`

si comporta come

`mysql_select_db`.

`mysql_query` restituisce un id positivo che identifica la query, e FALSE in caso di errore (ad es. una query non è sintatticamente o semanticamente valida, oppure le politiche di accesso ai db di MySQL, che possono essere piuttosto raffinate, non permettono all'utente proprietario della connessione di accedere a certe tabelle di un db o addirittura a certi campi). Conservate questo id nel caso la query generi dell'output (ad es. è una query di SELECT): infatti i dati vengono conservati in opportuni buffer gestiti dall'API e potete recuperarli proprio utilizzando l'id restituito da `mysql_query`. Non è obbligatorio terminare di recuperare i dati di una query prima di effettuarne un'altra, poiché la API è in grado di utilizzare più buffer, ciascuno associato ad un diverso query id.

La funzione seguente combina le due azioni di selezionare un database ed eseguire una query ed è molto utile per abbreviare il codice se il vostro script deve eseguire una sola query al db in questione, oppure una sola query per ogni db. Come `mysql_query` ritorna un query id tramite il quale potete poi recuperare i dati. Tenete presente che questa funzione cambia il db selezionato con quello indicato e che si può avere un solo database corrente.

int **mysql_db_query** (string *database*, string *query* [, int *link_identifier*])

Questa funzione è l'analoga della funzione `msql` vista prima per mSQL. Prima si chiamava `mysql`, ma ora è stata rinominata con `mysql_db_query`, anche se il vecchio nome `mysql` può essere ancora utilizzato (è consigliabile comunque usare il nuovo nome nei nuovi script che si scrivono).

Le funzioni:

int **mysql_num_rows** (int *query_identifier*)
int **mysql_num_fields** (int it query_identifier)

permettono di conoscere la dimensione della tabella restituita da una query, come le analoghe di mSQL. Il loro argomento è l'id che identifica il risultato di una query come ritornato da `mysql_query` o `mysql_db_query`.

Come nel caso di mSQL, le query DELETE, INSERT, REPLACE, e UPDATE ritornano una "tabella con 0 righe e 0 colonne", solo la SELECT ritorna almeno un numero di colonne non nullo (il numero delle colonne proiettate). Per conoscere il numero di record sui quali ha agito una query di questo tipo utilizzate la funzione seguente:

int **mysql_affected_rows** ([int *link_identifier*])

Ci sono notevoli differenze tra questa funzione e l'analoga di mSQL:

int **msql_affected_rows** (int *query_identifier*)

La prima che salta subito all'occhio è che mentre `msql_affected_rows` accetta per parametro un query id, `mysql_affected_rows` accetta per parametro l'id della connessione (se nessun parametro viene passato usa l'ultimo link aperto). Questo perché `mysql_affected_rows` ritorna sempre il numero di righe toccate dall'ULTIMA query di modifica, poiché questo valore non viene salvato dalla API MySQL per ogni query effettuata come avviene con la API di mSQL, ma solo per l'ultima query in una variabile comune. Nota bene che al contrario di `msql_affected_rows`, `mysql_affected_rows` NON ritorna il numero di righe selezionate da una SELECT. Dovete usare `mysql_num_rows()` per conoscere quel dato.

Un'altra differenza sottile è che una query di DELETE che non abbia la clausola WHERE, e che quindi cancellerà tutti i record della tabella, fa ritornare 0 dalla funzione `mysql_affected_rows` (`msql_affected_rows` invece non ha questa limitazione e ritorna il numero di record cancellati).

Passiamo ora alle funzioni per recuperare i dati. L'analoga della funzione `msql_result` che ritorna una cella della tabella risultante da una SELECT è naturalmente:

int **mysql_result** (int *result*, int *row* [, mixed *field*])

gli argomenti sono identici e come nel caso di `msql_result` questa funzione è lenta, soprattutto se si usa il nome letterale del campo anziché l'offset numerico. In più c'è la limitazione che non dovreste mischiare chiamate di `mysql_result()` con le altre funzioni che trattano l'insieme dei risultati descritte in seguito.

Le altre funzioni che recuperano una riga per volta (che in gergo si indicano come le funzioni `mysql_fetch*`, poiché iniziano tutte con `mysql_fetch`) sono analoghe a quelle dell'API mSQL.

Quella che restituisce una riga sotto forma di array scalare (enumerativo) è:

array mysql_fetch_row (int result)

è analoga a `msql_fetch_row` (result è un query id naturalmente) e come quella restituisce FALSE se non ci sono più righe.

Quest'altra invece la restituisce sotto forma di array (associativo):

array mysql_fetch_array (int result [, int result_type])

`result_type` può assumere i valori

MYSQL_ASSOC, MYSQL_NUM e MYSQL_BOTH

(quest'ultimo è il default) per indicare che si vogliono solo chiavi letterali, solo chiavi numeriche o entrambe rispettivamente.

Infine questa restituisce la riga come un oggetto:

object **mysql_fetch_object** (int *result* [, int *result_type*])

quindi i campi andranno acceduti col loro nome e usando l'operatore di selezione di membro dati di un oggetto `->`.

Infine la funzione:

int **mysql_free_result** (int *result*)

libera la memoria associata ad un query id, che comunque viene liberata automaticamente alla fine dello script, quindi chiamarla serve soprattutto se si è preoccupati di consumare troppa memoria mentre lo script sta ancora girando.

Ammesso che l'amministratore (o gli amministratori, poiché MySQL può essere configurato anche in modo che ci siano più amministratori di sistema ciascuno col proprio insieme di privilegi differente dall'altro) delle tabelle di privilegi di MySQL abbiano dato all'utente che utilizzate negli script il permesso di creare e cancellare db, potete farlo con le seguenti funzioni:

int **mysql_create_db** (string *databas_ename* [, int *link_identifier*])
int **mysql_drop_db** (string *database_name* [, int *link_identifier*])

Un'altra funzione molto utile per il debugging è:

string **mysql_error** ([int *link_identifier*])

questa funzione può essere usata per stampare il messaggio di errore corrente, quello che si è eventualmente verificato dopo l'ultima chiamata ad una funzione API di MySQL. L'api memorizza ogni messaggio di errore ritornato dal server in una stringa e `mysql_error` semplicemente ritorna il contenuto di questa stringa. Al contrario di `msql_error`, la stringa viene sovrascritta con '' (la stringa vuota) se si chiama un'altra funzione API (esclusa questa funzione stessa e la funzione `msql_errno`) e non si verifica un altro errore, quindi il suo valore va usato prima di chiamare un'altra funzione di MySQL.

La funzione

int **mysql_errno** ([int *link_identifier*])

restituisce invece il codice di errore (0 se non si è verificato nessun errore) e questo valore viene aggiornato dopo ogni chiamata di funzione API, come per la funzione `mysql_error`. ES:

```
mysql_connect() or
  echo mysql_errno().": ".mysql_error()."<BR>";
```

Infine vi segnalo che nel manuale di MySQL esiste una sezione dedicata alle differenze tra MySQL e mSQL[13].

[13] *"21 How MySQL compares to other databases" "21.1 How MySQL compares to mSQL".*

2.4 Generazione dinamica di immagini

Con PHP potete generare non solo HTML al volo, ma anche immagini in una varietà di formati, tra cui quelli più usati sul web (gif, png, jpg). Per fare questo occorre che PHP sia compilato con la libreria GD. Su un sistema Linux RedHat, il manuale d'uso della libreria GD (che se non installata dovete installare prima di compilare PHP) si trova qui (il numero di versione può ovviamente variare)[14]:

```
/usr/doc/gd-1.3/index.html
/usr/doc/gd-1.3/readme.txt
```

Più avanti affronterò l'argomento dell'installazione di una versione recente di GD.

Per avere un'idea veloce di quello che si può fare, prima ancora di spiegare qualcuna delle funzioni fondamentali di GD, considerate un esempio tratto dalla documentazione di PHP[15]:

```
button.php?text
---------------
<?php
     Header("Content-type: image/gif");
     $string=implode($argv," ");
     $im = imagecreatefromgif("button.gif");
     $orange = ImageColorAllocate($im, 220, 210, 60);
     $px = (imagesx($im)-7.5*strlen($string))/2;
     ImageString($im,3,$px,9,$string,$orange);
     ImageGif($im);
     ImageDestroy($im);
?>
```

Lo script precedente sovrappone il testo che gli viene passato come parametro (per semplicità il testo è il nome stesso del parametro, essendoci un solo parametro) al centro dell'immagine `button.gif`, generando quindi una nuova immagine Gif sullo standard output. Questo script può essere utile nel caso volete generare in un documento HTML una bottoniera composta da bottoni aventi tutti lo stesso sfondo ma titoli diversi al volo: anziché disegnare uno per uno i bottoni in tanti file immagini separate, possiamo generare tutte le immagini dinamicamente sovrapponendo il testo voluto ad uno sfondo comune. Questa tecnica ha inoltre il vantaggio che è facile cambiare lo sfondo comune a tutti i bottoni semplicemente cambiando un solo file `button.gif`. Nella pagina html si richiama l'immagine dinamica(mente

[14] L'ultima versione di GD (che è stata scritta in C da Thomas Boutell) si può trovare qui: http://www.boutell.com/gd/

[15] Se copiate questo file da questo testo, non inserite linee bianche dopo i tag <?php e ?>. La ragione sarà presto chiara.

generata) esattamente come se fosse una gif statica, col nome dello script che la genera, seguito dagli eventuali argomenti CGI:

```
<img src="button.php?text">
```

Scaricatevi l'archivio `button.tar.gz` (vedi link all'inizio di questo tutorial) per un esempio realistico dello script precedente, che qualsiasi webmaster potrà personalizzare (anche se non conosce PHP!).

In qualità di sviluppatori PHP naturalmente vogliamo essere in grado di scrivere i programmi per creare le nostre "immagini dinamiche", piuttosto che essere solamente in grado di usare codice PHP scritto da altri. Tra un po' sarete in grado di capire completamente il programma `button.php`. Scriveremo poi un generatore di insiemi di Mandelbrot via web.

La prima cosa da ricordarsi quando si scrivono script che generano immagini direttamente, come quello visto prima, è che occorre inviare, PRIMA di ogni altro output, sia effettuato dal codice PHP che HTML, un header HTTP indicante al browser il tipo di immagine che si genererà. Questo viene fatto con la funzione:

int **header** (string *string*)

è importante notare che anche l'inclusione (con **require** o **include**) di un file php contenente anche solo una riga vuota prima di inviare l'header fa fallire la corretta interpretazione dell'header stesso e che potranno essere prodotte immagini non valide mischiando i dati dell'immagine con altro output (è comunque molto difficile far piantare un browser, di solito immagini non valide risulteranno come immagini rotte). In questo caso comunque PHP si prende la briga di emettere un Warning per avvisarvi del problema (a meno che non mettete l'operatore di soppressione del messaggio di errore `@` davanti a "header"):

```
Warning: Cannot add header information - headers already sent by
(output started at NOME_FILE:NUMERO_LINEA) in NOME_FILE
on line NUMERO_LINEA
```

Quindi occhio a che non sia stampato nient'altro da uno script che genera un'immagine: solo i dati rappresentativi del formato dell'immagine devono essere mandati in output, preceduti dal giusto content-type header.

Per le immagini gif (Graphics Interchange Format) basta fare:

```
Header("Content-type: image/gif");
```

Per le jpeg (Joint Photographic Experts Group):

```
Header("Content-type: image/jpeg");
```

Dalla versione 1.6 in poi, la libreria GD, che è free software non include più il supporto per le GIF, che sono un formato proprietario della società Unisys (infatti il vecchio algoritmo di compressione LZW utilizzato nelle gif, è patentato, e difatti le vecchie versioni di GD, come ad es. la 1.3, proprio per questo fatto generano gif non compresse con LZW, ma con una semplice compressione RLE (Run Length Encoding) che è in genere molto meno efficace). Poco male, poiché queste versioni supportano le PNG (Portable Network Graphics) al posto delle gif, che sono un formato non proprietario e leggibile da qualsiasi browser moderno[16]:

```
Header ("Content-type: image/png");
```

Personalmente ritengo (e non sono il solo) che brevettare gli algoritmi è un po' come aver la assurda pretesa di patentare le idee. Le idee non appartengono in esclusiva a nessuno, poiché più individui possono pensare idee simili e di qualità praticamente equivalente. Brevettare un algoritmo è simile a brevettare l'idea matematica di integrale, o voler pretendere l'uso esclusivo di una legge fisica, che come la storia dimostra può essere scoperta indipendentemente da più persone. Spesso chi brevetta le cose non è colui che ha lavorato e le ha scoperte per primo. Il telefono è stato inventato dall'italiano Antonio Meucci, che però non aveva i soldi di Bell per poter ottenere un brevetto ed arricchirsi con la sua straordinaria invenzione. Se anche voi la pensate in questo modo la cosa migliore per battersi contro i brevetti del software (che spesso sfiorano l'assurdità come nel caso di Amazon che ha preteso e ottenuto di brevettare una forma di utilizzo dei cookie) è boicottarli e usare le alternative non patentate. La ragione per cui vengono concessi in alcuni paesi, tra cui gli Stati Uniti, dei brevetti così assurdi è facilmente indovinabile, visto gli interessi economici che sono coinvolti.

Soprattutto per il web è auspicabile che le tecnologie software che si utilizzano siano libere (come sono liberi HTML e il protocollo HTTP), in modo che chiunque possa farne uso liberamente, senza che sia necessario acquistare costose licenze o concessioni (in gergo "royalty") a suon di migliaia di dollari anche nel caso si intenda impiegare un software in attività no-profit.

Anche se foste daccordo con le leggi sui brevetti del software presenti in alcuni paesi (come ad es. negli Stati Uniti) non c'è comunque più nessuna ragione tecnica per usare il vecchio formato GIF, tranne il fatto che i vecchi browser (vecchie

[16]Nota bene: se vi scordate l'header, un browser potrebbe non essere in grado di riconoscere il formato dell'immagine indicata dal tag img e quindi non visualizzarla. Anche se netscape riconosce bene il formato quando lo script viene chiamato poiché il suo nome è presente nell'attributo SRC del tag `<IMG>`, comunque se lo script viene chiamato direttamente, ottenete la visualizzazione della codifica associata all'immagine (nel caso di una GIF, otterrete `GIF87a` oppure `GIF89a` seguito da "strani caratteri"), anziché l'immagine stessa, poiché i dati dell'immagine vengono interpretati come HTML (questo perché PHP emette l'header Content-type col mime type text/html se non si indica diversamente prima di qualsiasi altro output). Meglio mandare sempre l'header per evitare problemi di misinterpretazione del contenuto da parte del browser dell'utente.

versioni di Internet Explorer o Netscape Navigator che praticamente nessuno che sia interessato al web usa più) non supportano le PNG. È PNG il nuovo standard raccomandato dal W3C. Ha molte più feature ed è gratuito. Molti webmaster non lo sanno e continuano ad usare le GIF. Per informazioni sul formato PNG che è stato introdotto da alcuni anni a questa parte visitate la sua homepage:

http://www.libpng.org/pub/png/

Questa pagina contiene molti link utili per attuare la conversione GIF → PNG (compresi link a programmi di conversione gratuiti), nonché molte opinioni (protese in favore di PNG che è open-source) sulla questione GIF:

http://burnallgifs.org/

Nel caso abbiate delle GIF animate (`GIF89a`, poiché la vecchia versione delle GIF, `GIF87a` non supporta l'animazione e nemmeno la trasparenza), potete convertirle nel formato MNG (Multiple-image Network Graphics) che non è altro che un superset di PNG per supportare l'animazione.

Ma torniamo a cose più tecniche. Per poter creare delle immagini, occorre prima allocare in memoria una opportuna struttura dati atta a contenere l'immagine in un buffer e le altre informazioni ad essa associata. Infatti negli script PHP non c'è uno schermo predefinito direttamente accessibile su cui disegnare, ma si "disegna in memoria" se mi passate il termine e poi, come visto nell'esempio di prima, si pacchetta l'immagine in un formato riconoscibile da un browser e la sia invia sullo standard output.

Per allocare un buffer inizialmente "vuoto" si usa la seguente funzione:

int **imagecreate** (int *x_ size*, int *y_ size*)

che ritorna l'id della risorsa (o image identifier) rappresentante una immagine della dimensione specificata (*x_ size* pixel orizzontali per *y_ size* pixel verticali). All'inizio il buffer rappresenta una immagine con tutti i pixel con lo stesso colore, che è il colore numero 0 della palette. Come al solito conservate l'id della risorsa perché va usato ogni volta che ci si vuole riferire a quel buffer, ad es. salvatelo in una variabile di nome `$im` (che sarà del tipo speciale "resource"):

```
$im = imagecreate(100,50);
```

Ovviamente potete scegliere il nome della variabile come volete.

```
<?php
   // genera una immagine "vuota" 100x50
   Header("Content-type: image/gif");
   ImageGif(imagecreate(100, 50));
?>
```

Creato un buffer, per disegnarci sopra occorre utilizzare una delle primitive grafiche. La più elementare è quella che disegna un punto, ma prima di poter disegnare qualcosa occorre ovviamente stabilire il colore col quale si disegna e il colore dello sfondo dell'immagine. Per definire un colore da utilizzare poi nel seguito, si usa la funzione:

int **imagecolorallocate** (int *im*, int *red*, int *green*, int *blue*)

questa alloca un colore per l'immagine identificata da *im* (il valore che abbiamo visto essere ritornato da **imagecreate**). Gli interi *red*, *green*, *blue* sono i componenti (o valori) RGB dell'immagine. Il valore restituito è un numero che identifica il colore composto dalla terna RGB specificata e può essere usato in seguito per riferirsi al colore allocato. L'insieme dei colori allocati viene comunemente definito "palette" dell'immagine.

Più precisamente la palette di una immagine è un vettore di terne RGB, in cui ogni terna è individuata da un numero ordinale a partire da 0. **imagecolorallocate** (che alcuni per ragioni di leggibilità preferiscono scrivere **ImageColorAllocate** oppure **imageColorAllocate**), semplicemente aggiunge un elemento all'array palette e restituisce il nuovo indice (quindi la prima volta restituirà 0, la seconda 1, la terza 2, ecc...).

Stabilendo il primo colore della palette (il colore numero 0) si stabilisce il colore di sfondo di un buffer appena creato vuoto (è buona norma farlo sempre per essere sicuri che sia il colore desiderato per lo sfondo):

```
<?php
   // genera una immagine "vuota" bianca 100x50
   Header("Content-type: image/gif");
   $im = imagecreate(100, 50);
   ImageColorAllocate($im, 0xFF, 0xFF, 0xFF);  // setta colore dello sfondo
   ImageGif($im);
?>
```

La funzione **ImageColorAllocate** deve essere chiamata per creare ciascun colore che deve essere utilizzato nell'immagine rappresentata da *im*, poiché ciascun colore utilizzato deve essere definito. In altri termini prima di utilizzare un indice nell'array palette dovete definire il valore dell'elemento (terna RGB) a quell'indice, altrimenti non saprete cosa troverete a quell'indice (probabilmente valori impostati da altri script).

Come sicuramente sapete, ogni pixel sullo schermo è composto da 3 fosfori molto vicini in grado di emettere ognuno luce rossa, verde e blu con differenti intensità in modo che l'occhio umano posto a sufficiente distanza percepisca un colore unico (appunto quello di un pixel). Regolando l'intensità del livello di rosso, verde e blu indipendentemente, si possono ottenere moltissimi colori differenti di un unico pixel. Quanti? Se le intensità di rosso verde e blue variano tra 0 (assenza di colore) e 255 (massima intensità di quel colore primario), come tipicamente avviene, il numero complessivo di colori ottenibili è pari ovviamente a tutte le possibili disposizioni con ripetizione di 256 elementi su 3 posti (compresa la terna (0,0,0) che ovviamente identifica il nero più buio), cioè:

$$256 * 256 * 256 = 256^3 = 16777216 \text{ (abbreviato 16M)}$$

vale a dire più di 16 milioni di colori differenti, vale a dire "16 mega colori". I valori delle componenti RGB vengono spesso espressi per convenienza in esadecimale. Siccome variano tra 0 e 255 ognuno sarà espresso da una coppia di cifre esadecimali.

```
$white = imagecolorallocate($im, 0xFF, 0xFF, 0xFF);
$black = imagecolorallocate($im, 0x0, 0x0, 0x0);
```

Sfortunatamente le schede video, i monitor e le preferenze degli utenti non sono tutti uguali. Alcuni utenti preferiscono lavorare con una profondità di 64k colori (detta anche high color mode), altri che posseggono schede grafiche più avanzate e computer più moderni, sono in grado di lavorare a 16M di colori (modo detto "true color"). Alcuni tradizionalisti come me usano ancora anche la profondità di 1 byte per pixel ossia 256 colori, anche se occorre dire che oramai sempre più utenti sono in grado di usare 16M di colori senza che il loro sistema si rallenti apprezzabilmente.

Inoltre non tutti i formati supportano 16M di colori. Il formato JPEG li supporta, ed è quindi più adatto per la rappresentazione di immagini "fotografiche", ossia di immagini prese dalla realtà che tipicamente richiedono un elevato numero di colori per essere rappresentate in modo realistico. Il vecchio formato GIF (se avete intenzione di usarlo nonostante le raccomandazioni contrarie) supporta al più 256 colori differenti e il metodo di compressione che utilizza si comporta molto bene nel caso di immagini regolari, come un diagramma di flusso o schema a blocchi o, ad es., una immagine rappresentante una finestra di `xterm`, in genere immagini in cui è presente un alto contrasto e poca continuità dei toni (PNG tuttavia va mediamente meglio in compressione di GIF oltre a non presentare problemi di licenza e non è limitato a 256 colori; come GIF anche PNG offre la possibilità di avere un colore trasparente, mentre JPEG no).

JPEG è un metodo di compressione con perdita (lossy), anche se la scelta di un opportuno fattore di compressione può rendere inapprezzabile o quasi la perdita di qualità. PNG e GIF invece sono formati con algoritmi di compressione lossless, "senza perdita". Probabilmente sapete già queste cose, eppure ho visto molti webmaster che avevano salvato immagini regolari e con pochi colori in formato JPEG con bassa compressione anziché GIF o PNG. Sbagliatissimo: i file erano più grossi e la qualità inferiore rispetto alla versione GIF o PNG e questo dimostra che non avevano idea della elementare differenza tra una GIF (o PNG) e una JPEG!

Non ci vuole grande scienza a fare delle prove di salvataggio in diversi formati, per rendersi conto quale è il formato più adatto all'immagine (e il rapporto di compressione a cui ci si può spingere prima di ritenere la qualità dell'immagine troppo scadente nel caso di JPG). Un buon webmaster deve curare bene l'aspetto della compressione delle immagini, poiché il web è ancora molto lento per l'utente medio. Anche riempire le pagine con troppe immagini può essere controproduttivo: chi ha connessioni lente (come la maggior parte delle persone che si collegano con modem 56k da casa) abbandonerà la pagina se per caricarsi ci mette troppo.

Cercare le informazioni che servono in un sito in cui le pagine sono sovraccariche di immagini di abbellimento molto lente a caricarsi è snervante e potete stare sicuri che molti visitatori premeranno dopo un po' sul tasto X del browser per bloccare il caricamento della pagina e cercheranno un altro sito. Potete trovare un esempio di una prova che ho fatto io nel file di testo `provaimg.txt` (vedi archivio allegato a questo corso).

Molti webmaster creano immagini con un numero di colori maggiore di 65536, sicché quelli che hanno 65536 colori non vedranno molto bene queste immagini; quelli che stanno usando solo 256 colori poi le vedranno malissimo (comunque per quest'ultimi, soprattutto nel caso di immagini fotografiche non c'è molto da fare). Se ad es. una immagine ha 100000 colori, e la profondità di colore è solo di 65536 colori, il browser dovrà trovare i 65536 colori che si avvicinano di più a quelli dell'immagine. Su alcuni tipi di display potrà definire i livelli RGB di ciascuno di questi 65536 colori scegliendogli ad es. tra 16M di colori possibili in modo che corrispondano a 65536 dei 100000 presenti nell'immagine. Gli altri 100000-65536=34464 dovranno essere resi con colori uguali a quelli già usati e più "vicini" al colore mancante, perdendo così alcuni dettagli nelle sfumature (roba comunque che a volte il mio occhio miope quasi non riesce ad apprezzare ;) oppure dovrà essere utilizzata una tecnica nota come dithering che consiste nel simulare un colore che non si ha disegnando una "scacchiera" di altri due colori, facendo quindi a livello di pixel l'effetto della fusione dei colori che fanno i 3 fosfori che compongono ogni pixel. Questa tecnica tuttavia spesso produce dei brutti effetti di punteggiatura[17].

Tenete conto anche che se una pagina web utilizza solo immagini con meno di 65536 colori, non è detto che si veda bene con una profondità di 65536 colori: infatti ogni immagine potrebbe utilizzare 30000 colori tutti differenti da quella delle altre due, in modo che i colori diversi siano in totale 90000 che è ben maggiore di 65536. Anche in tal caso quindi è inevitable la scelta tra il ricorso alla perdita di alcuni colori o al dithering e quindi la perdita di alcuni dettagli o peggio vedere alcune parti dell'immagine punteggiata; ma si può limitare il danno se i 65536 colori vengono scelti tra quelli più usati e quelli che risaltano di più nell'immagine.

Tutto questo parlando in generale. Tuttavia poiché spesso l'hardware non supporta la definizione della palette nel modo 65536 colori (e le SVGA non lo supportano, hanno quindi 65536 colori fissi, che non si possono scegliere nell'ambito dei 16M), è meglio far sì che le vostre immagini JPEG contengano solo 64K colori o meno e che comunque anche se ne contengono di più, che 64K di questi colori siano quelli più usati e più visibili. Se create l'immagine, disegnatene le aree principali con 64K colori e poi passate ai 16M solo per rifinire i dettagli, se possibile: in questo modo chi visualizzerà l'immagine su un display a soli 64K colori con palette fissa

[17] In realtà nell'esempio precedente, non tutti i 65536 colori potrebbero essere ridefiniti a piacimento, qualora la ridefinizione dei colori sia possibile. È possibile che alcuni siano riservati per i colori dalla taskbar, dei bottoni del browser, ecc... che si vuole rimangano fissi. Questo non fa che aggravare il problema della rappresentazione di immagini che hanno più colori di quelli che si possono avere a video.

(profondità ancora molto usata), la vedrà non molto dissimile da come si vede in modalità 16M. Potete anche pensare di salvare l'immagine su cui avete lavorato in 16M riducendo i colori a 64K con uno dei metodi disponibili, anziché lasciare più di 64K colori, per ragioni di economia di spazio. Infatti in generale più colori ci sono in un JPG, a parità di rapporto di compressione e più il file risulterà "grosso".

È ovvio che con circa 16 milioni di colori si possono rappresentare le immagini fotografiche con una qualità eccezionale, ma anche con soli 64k colori molte foto non sono poi così brutte. Per le immagini più semplici inoltre 256 colori possono essere più che sufficienti. Un buon webmaster usa le varie profondità a seconda del tipo di immagini e fa in modo che la scelta dei colori più usati ricada tra quelli disponibili con le profondità inferiori di 65536 e 256. Molti webmaster non si preoccupano molto, come se tutti navigassero in internet con Microsoft Explorer e in 1024x768 a 16 milioni di colori. Non è così, anche se la diminuzione dei prezzi e l'aumento delle prestazioni delle schede grafiche e dei monitor ha permesso a molte più persone di acquistare periferiche più potenti e raffinate. Nel momento in cui scrivo tuttavia ha ancora molto senso far sì che le proprie pagine si vedano bene con 65536, anche perché le immagini con molti colori sono più pesanti in termini di KB e internet è ancora molto lenta per l'utente domestico medio (soprattutto in paesi come l'Italia).

La domanda che sorge spontanea è la seguente: esiste un set di terne RBG in qualche modo "standard", ossia che garantisca che su tutte le piattaforme e su tutti i sistemi operativi, il corrispondente colore sia rappresentabile e non venga sostituito con un altro simile oppure ditherizzato? Purtroppo la risposta è negativa: il set di colori "sicuri" è fortemente dipendente dall'hardware e dal software, quindi purtroppo c'è poco da fare. Un colore che si vede in un certo modo su una piattaforma con un certo sistema operativo, in una certa modalità potrebbe essere reso in modo molto diverso su un'altra piattaforma, o anche solo in un altro sistema operativo o addirittura anche quando cambia solo la modalità. Qualche colore potrebbe essere reso allo stesso modo di un altro "sufficientemente vicino" e si perde così la differenza tra i due colori oppure mediante la ditherizzazione di due colori diversi... in pratica avete capito che in questo campo non c'è la garanzia assoluta che l'utente veda lo stesso colore che vede il programmatore.

Ho letto che sul SunOS con monitor impostato con 8-bit sono colori Web-safe tutti quelli con valori RGB ottenuti dalle disposizioni con ripetizione delle coppie di numeri esadecimali 00, 40, 80, BF, o FF, ma questi colori non sono Web-safe sotto Linux quando la profondità di colore è sempre di soli 8 bit (la maggior parte sono rappresentati da Netscape usando il dithering).

Ho poi sperimentato che su Linux sono web-safe i colori ottenuti dalle coppie 00, 30, 65, 9A, CF, FF per lo meno con Netscape. Se sviluppate pagine prevalentemente per computer Linux con monitor ad 8 bpp conviene quindi usare questi colori[18].

Sul tutorial HTML di Joe Barta (http://junior.apk.net/~jbarta) è spiegato molto bene l'argomento della scelta dei colori e sono indicati 216 colori che nel modo

[18]Per avviare X Window con risoluzione 8 bpp usate il comando: `startx - -bpp 8`.

a 256-colori sono Web-safe sia per Netscape Navigator che per Microsoft Internet Explorer sia sui sistemi Windows che Macintosh (ma non su Linux). Questi colori hanno terne RGB ottenibili disponendo con ripetizione le coppie di valori esadecimali 00, 33, 66, 99, CC, FF (che corrispondono ai valori decimali di, rispettivamente, 0, 51, 102, 153, 204 e 255). Tuttavia effettuando delle prove è stato scoperto che questi colori andrebbero ridotti a 212, poiché Internet Explorer su Windows non renderizza correttamente 4 dei 216 colori, che sono: #0033FF (0,51,255), #3300FF (51,0,255), #00FF33 (0,255,51), e #33FF00 (51,255,0), tuttavia ignorerò questo fatto[19].

Tuttavia come visto prima i browser quando si trovano in questa modalità a soli 256 colori pongono delle limitazioni, quindi per evitare brutti effetti, è meglio settare la vostra palette nello stesso modo in cui lo setta il browser Netscape o Explorer, oppure un "comune denominatore" dei due. Questo sempre se intendete supportare gli utenti che usano 256 colori (cosa che se possibile è ancora consigliabile).

Solitamente le immagini generate con PHP sono disegni regolari e non devono avere molte sfumature (anche se questo non è sempre vero; le immagini del frattale di Mandelbrot che genereremo non sono affatto regolari). Quindi usare le palette a 216 colori ha ancora molto senso.

In questo tutorial mi limiterò a considerare questa palette "web-safe" di 216 colori. Scriviamo anzitutto una funzione per impostare facilmente una palette di questo tipo:

```
// setta la palette websafe di 216 colori sull'immagine $im
function setWebSafePalette($im)
{
   // setta i primi 216 colori websafe
   for ($b=0; $b <= 0xFF; $b += 0x33)
      for ($g=0; $g <= 0xFF; $g += 0x33)
 for ($r=0; $r <= 0xFF; $r += 0x33)
    imageColorAllocate($im, $r, $g, $b);

   // setta i 40 colori rimanenti col nero
   for ($i=0; $i<40; $i++)
      imageColorAllocate($im, 0, 0, 0);

   return $im;
}
```

[19] Nella VGA i 256 colori delle modalità 8 bpp si possono scegliere tra una palette piuttosto ampia. Si può assegnare un valore da 0 a 63 estremi compresi ad ogni elemento della terna RGB che nell'array palette definisce uno dei 256 colori e questo ci dà la possibilità di scegliere 256 tra: $64^3 = 262144$.

Il passaggio dell'immagine su cui operare il cambiamento di palette può essere effettuato anche semplicemente per valore, trattandosi di un handle. Usare questa funzione è semplice come fare:

```
$im = imageCreate(100, 50);
setWebSafePalette($im);
```

o abbreviando (ecco perché faccio ritornare `$im`):

```
$im = setWebSafePalette(imageCreate(100, 50));
```

La controparte in Linux della funzione setWebSafePalette è la seguente:

```
// set the linux-netscape 216 color web safe palette
function setLinuxWebSafePalette($im)
{
   // setta i primi 216 colori websafe
   $int = array(0x00, 0x30, 0x65, 0x9a, 0xcf, 0xff);

   for ($b=0; $b < 6; $b++)
      for ($g=0; $g < 6; $g++)
 for ($r=0; $r < 6; $r++)
    imageColorAllocate($im, $int[$r], $int[$g], $int[$b]);

   // setta i 40 colori rimanenti col nero
   for ($i=0; $i<40; $i++)
      imageColorAllocate($im, 0, 0, 0);

   return $im;
}
```

Poiché 0x30 è molto vicino a 0x33, 0x65 è vicino a 0x66, 0x9A è vicino a 0x99, 0xCF è vicino a 0xCC la palette web-safe di linux è molto simile a quella di Windows. Purtroppo Linux *ditherizza* molti colori della palette websafe di Windows e viceversa quando siamo in modalità a 256 colori.

Per comprimere i dati nel buffer secondo un ben preciso formato e inviarli allo standard output oppure ad un file si usano le funzioni:

int **imagegif** (int *im* [, string *filename*])
int **imagepng** (int *im* [, string *filename*])
int **imagejpeg** (int *im* [, string *filename* [, int *quality*]])

Nel caso della funzione **ImageGIF** si noti che il formato sarà GIF87a a meno che un colore dell'immagine sia stato reso trasparente tramite la funzione **ImageColorTransparent**; in questo caso infatti, poiché GIF87a non supporta la trasparenza, il formato sarà GIF89a. Dalla versione 1.6 in poi la libreria GD ha rimosso il supporto per le GIF per i motivi citati prima e quindi non è più possibile generare immagini di questo tipo con GD ≥ 1.6

La funzione **ImageJPEG** accetta anche un terzo parametro che indica il valore di qualità che da un punto di vista pratica conviene scegliere nell'intervallo 0-95: un maggior valore di qualità di solito implica sia una maggiore qualità che un file più grosso. In generale il valore di qualità di default (che si applica quando non specificate nessun valore di qualità, oppure specificate un valore quality negativo, es. -1) rappresenta un buon rapporto qualità/dimensione per la maggior parte delle immagini.

Oltre che "al volo", ossia direttamente come output di uno script, le immagini possono essere salvate in un file sul server con nome a piacere, per essere riferite successivamente dal codice HTML (anche dello script stesso che le ha generate), solitamente dal tag `<IMG>`. Occorre in tal caso che lo script abbia il permesso di scrivere quel file. Basta settare l'argomento filename col nome del file immagine da generare nelle tre funzioni precedenti.

Se volete settare l'argomento quality senza settare l'argomento filename, basta usare una stringa vuota ('') al posto di filename. L'ultima versione della libreria GD (nel momento in cui scrivo la 1.8.3) non supporta ancora le immagini truecolor (16.7 milioni di colori) JPEG o PNG, ma le supporterà a partire dalla versione 2.0.

Per la scelta di quale formato utilizzare, si applicano le considerazioni generali viste prima sulla differenza tra i formati. Tenete inoltre presente che il supporto per JPEG è disponibile solo se PHP è stato compilato con la versione 1.8 di GD o successiva.

Per avere il supporto per JPEG e PNG, siccome avevo installato la vecchia gd-1.3-6 (insieme con gd-devel) quando installai RedHat 6.2, ho dovuto compilare la nuova, scaricata dal sito http://www.boutell.com/gd e poi ricompilare PHP. Le istruzioni su come (ri)compilare PHP si trovano nel file allegato install.txt. Qui vediamo le differenze rispetto a quello che si dice in quel file. Cominciamo ad installare gd dai sorgenti:

```
tar zxvf gd-1.8.3.tar.gz -C /usr/src/
cd /usr/src/gd-1.8.3/
```

La documentazione si trova nel file di testo semplice `readme.txt`. A questo punto editate il `Makefile`, seguendo le istruzioni ivi contenute sotto forma di commenti. Nel mio caso, avendo già installate la **libjpeg** (necessaria per poter creare immagini JPG - occorre la versione 6b o successiva e occorre installare anche il pacchetto **libjpeg-devel**), la libreria bf zlib per la compressione dei dati (necessaria a gd - a proposito occorre installare anche la **zlib-devel**[20]), e la **libpng** (pure richiesta da GD per poter generare immagini PNG, poiché ripeto che GIF non è più supportato - ricordo anche che occorre installare anche la **libpng-devel** per poter compilare programmi che usano la **libpng**), e il pacchetto **xpm** (contenente

[20]Se non lo è già:
`rpm -ivh zlib-devel-1.1.3-6.i386.rpm`

la libreria **pixmap** per X Window, nonché **xpm-devel**), ho fatto semplicemente i cambiamenti indicati da questo diff file:

```
file generato da "diff Makefile.ORIG Makefile"
----------------------------------------------
14c14
< CFLAGS=-O
---
> #CFLAGS=-O
17c17
< #CFLAGS=-O  -DHAVE_XPM -DHAVE_JPEG -DHAVE_LIBTTF
---
> CFLAGS=-O  -DHAVE_XPM -DHAVE_JPEG -DHAVE_LIBTTF
26c26
< LIBS=-lm -lgd -lpng -lz
---
> #LIBS=-lm -lgd -lpng -lz
36c36
< #LIBS=-lm -lgd -lpng -lz -ljpeg -lttf -lXpm -lX11
---
> LIBS=-lm -lgd -lpng -lz -ljpeg -lttf -lXpm -lX11
43c43
< INCLUDEDIRS=-I. -I/usr/local/include -I/usr/include/X11
-I/usr/X11R6/include/X11
---
> INCLUDEDIRS=-I. -I/usr/local/include -I/usr/include/X11
-I/usr/X11R6/include/X11 -I/usr/include/freetype
```

Tra l'altro devo dire che, avendo scelto di includere il supporto per i font TrueType, ho dovuto installare i pacchetti relativi prima di compilare GD:

```
rpm -ivh freetype-devel-1.3.1-5.i386.rpm
```

Qui ho prima copiato `Makefile` in `Makefile.ORIG` e poi l'ho modificato. Se avete installato patch[21], e state per compilare la stessa mia versione di GD con tutti i pacchetti accessori come me, soprattutto se avete Linux RedHat, potete apportare le modifiche precedenti anche facendo semplicemente:

```
patch Makefile file_generato_da_diff
```

anziché procedere con un text-editor.

Non ho disinstallato la vecchia GD, poiché viene usata da alcuni programmi sul mio sistema. Uno di questi giorni mi deciderò a reinstallare tutto usando una

[21] altrimenti potete farlo con un comando del tipo
`rpm -ivh patch-2.5-10.i386.rpm`

distribuzione di Linux più recente; di un certo Linux esce una distribuzione nuova così spesso che se vuoi tenerti aggiornato devi sempre ricominciare daccapo ;) Linux è un fenomeno in continua evoluzione. Fintantoché non vi decidete a reinstallare tutto, potete compilare alcune nuove versioni dei pacchetti quando vi servono nuove funzionalità come stiamo facendo adesso. Tuttavia installare RPM è molto più semplice, quindi se i programmi da aggiornare sono molti forse è meglio procurarsi una distribuzione recente e rifare l'installazione, oppure effettuare un aggiornamento.

Quindi per costruire e installare una GD super-accessoriata ho fatto semplicemente:

```
make
make install (da root)
```

Quest'ultimo comando vi dice esattamente tutto quello che viene installato:

```
sh ./install-item 644 libgd.a /usr/local/lib/libgd.a
sh ./install-item 755 pngtogd /usr/local/bin/pngtogd
sh ./install-item 755 pngtogd2 /usr/local/bin/pngtogd2
sh ./install-item 755 gdtopng /usr/local/bin/gdtopng
sh ./install-item 755 gd2topng /usr/local/bin/gd2topng
sh ./install-item 755 gd2copypal /usr/local/bin/gd2copypal
sh ./install-item 755 gdparttopng /usr/local/bin/gdparttopng
sh ./install-item 755 webpng /usr/local/bin/webpng
sh ./install-item 755 bdftogd /usr/local/bin/bdftogd
sh ./install-item 644 gd.h /usr/local/include/gd.h
sh ./install-item 644 gdcache.h /usr/local/include/gdcache.h
sh ./install-item 644 gd_io.h /usr/local/include/gd_io.h
sh ./install-item 644 gdfontg.h /usr/local/include/gdfontg.h
sh ./install-item 644 gdfontl.h /usr/local/include/gdfontl.h
sh ./install-item 644 gdfontmb.h /usr/local/include/gdfontmb.h
sh ./install-item 644 gdfonts.h /usr/local/include/gdfonts.h
sh ./install-item 644 gdfontt.h /usr/local/include/gdfontt.h
```

Quindi non resta che ricompilare PHP come spiegato in `install.php` e avrete PHP con la nuova GD. Le uniche variazioni sono che prima di rifare il configure occorre rimuovere il file `config.cache` e poi passare al configure ulteriori parametri:

```
# rm -f config.cache
# ./configure --with-mysql=/usr/local/mysql --with-msql \
--with-apxs=/usr/local/apache/bin/apxs \
--with-gd=/usr/local -with-jpeg-dir --with-xpm-dir --with-gdbm
```

Ho aggiunto il path `/usr/local` al parametro `-with-gd` per essere sicuro che venga utilizzata la nuova versione di gd (poiché come ho già detto mantengo entrambe le versioni sul mio sistema RedHat 6.2 perché ho vecchie applicazioni che usano la gd 1.3.6). `-with-jpeg-dir` e `-with-xpm-dir` assicurano che vengano incluse le rispettive librerie shared necessarie a `gd-1.8+` (il configure genererà un warning su `stderr` se non riesce a trovarle).

Anche se tutto va bene, non guasta dedicare un po' di tempo a dare una controllatina al `config.log`, per vedere se il supporto per le nuove librerie sia stato compilato correttamente, ossia se tutte le librerie sono state trovate. Potete ignorare i messaggi di errore riguardanti la mancanza di funzioni relative alle Gif (come **gdImageCreateFromGif**), che come detto è roba vecchia venduta a caro e ingiustificato prezzo.

Ho fatto poi il make:

```
make
```

Sfortunatamente ho ottenuto questo errore:

```
/usr/local/lib/libgd.a(gd.o): In function ‘gdImageColorResolve’:
gd.o(.text+0x6a0): multiple definition of ‘gdImageColorResolve’
ext/gd/.libs/libgd.al(gd.lo):/usr/src/php-4.0.3pl1/ext/gd/gd.c:308: first
defined here
/usr/bin/ld: Warning: size of symbol ‘gdImageColorResolve’ changed from 255
to 226 in gd.o
collect2: ld returned 1 exit status
make[1]: *** [libphp4.la] Error 1
make[1]: Leaving directory ‘/usr/src/php-4.0.3pl1’
make: *** [all-recursive] Error 1
```

Qualche linea sopra la 308 di

`/usr/src/php-4.0.3pl1/ext/gd/gd.c`

ho trovato questa direttiva C:

```
#ifndef HAVE_GDIMAGECOLORRESOLVE
```

In pratica questo dice che la funzione **gdImageColorResolve** va inclusa nell'interprete solo se non è stata definita la costante

```
HAVE_GDIMAGECOLORRESOLVE.
```

Siccome il `configure` mi dava:

```
checking for gdImageColorResolve in -lgd... yes
```

vuol dire che questa funzione è già presente in GD e quindi la costante

```
HAVE_GDIMAGECOLORRESOLVE
```

non dovrebbe essere definita, invece lo è e vale 1. Deve esserci un errore negli script di configurazione. Secondo me c'è un errore: la direttiva alla riga 308 di `ext/gd/gd.c` va riscritta in questo modo:

```
#if !HAVE_GDIMAGECOLORRESOLVE
```

L'ho riscritta così e il problema si è risolto rifacendo il make. Più in là però ho incontrato un altro problema.

```
.libs/libphp4.a(gd.o): In function ‘php_imagettftext_common’:
/usr/src/php-4.0.3pl1/ext/gd/gd.c:2046: undefined reference to ‘gdttf’
collect2: ld returned 1 exit status
make[1]: *** [php] Error 1
make[1]: Leaving directory ‘/usr/src/php-4.0.3pl1’
make: *** [all-recursive] Error 1
```

Ora questa funzione è stata rinominata **gdImageStringTTF** nella nuova versione di GD, quindi un patch veloce è di sostituire alla linea 2046 di `gd.c` la chiamata di `gdttf` con **gdImageStringTTF**.

Fatto questo, non mi restava che dare il fatidico:

```
make install
```

et voilà, finalmente PHP supporta la generazione di immagini JPG e PNG[22]!

D'ora in poi genero solo immagini JPG e PNG, poiché la nuova GD non supporta più GIF.

Tornando al tutorial dopo questa parentesi di "amministrazione di sistemi" (compito tuttavia che molte volte tocca anche al povero programmatore ;), è molto utile sapere che le immagini, oltre che da zero con la funzione **ImageCreate**, possono essere generate anche partendo da immagini già esistenti. È cioè possibile caricare nel buffer immagini già esistenti, disegnarci sopra e restituire l'immagine modificata. Anche qui ci sono 3 funzioni per 3 differenti formati:

int **imagecreatefromgif** (string *filename*)
int **imagecreatefromjpeg** (string *filename*)
int **imagecreatefrompng** (string it filename)

Tutte le funzioni ritornano l'id dell'immagine, similmente a **imagecreate**. Lo script seguente legge l'immagine `test.png` nella sua stessa directory e la restituisce tale e quale. Chiamare questo script è la stessa cosa che richiamare da web direttamente l'immagine `test.png`, quindi questo esempio non è di grande utilità pratica, ma illustra l'uso della funzione **ImageCreateFromPng**.

```
<?php
 Header("Content-type: image/png");
 $im = ImageCreateFromPng("test.png");
 ImagePng($im);
 ?>
```

[22]Prima di fare `make install`, stoppate apache. Appena finita l'installazione potete riavviarlo.

In caso di errore le tre funzioni **ImageCreateFrom*** ritornano il booleano FALSE e inoltre provocano l'emissione di un Warning. Tuttavia sfortunatamente il warning non apparirà a video, poiché verrà interpretato dal browser come una immagine PNG, risultando in una immagine rotta. Questo rende difficile il debugging. Un workaround semplice per ottenere i messaggi di errore consiste nel mandate l'header che indica se si tratta di una immagine dopo la chiamata ad **ImageCreateFromPng**:

```
<?php
 $im = ImageCreateFromPng("test.png");
 Header("Content-type: image/png");
 ImagePng($im);
 ?>
```

In questo modo, se ad es. `test.png` non esiste, anziché una immagine rotta ottenete tre warning, di cui il primo indica l'errore riscontrato da **ImageCreateFromPng**:

```
Warning: imagecreatefrompng: Unable to open 'test.png' for reading in
/usr/local/apache/htdocs/tests/palette/prova.php on line 2

Warning: Cannot add header information - headers already sent by (output
started at
/usr/local/apache/htdocs/tests/palette/prova.php:2) in
/usr/local/apache/htdocs/tests/palette/prova.php on line 3

Warning: Supplied argument is not a valid Image resource in
/usr/local/apache/htdocs/tests/palette/prova.php on line 4
```

Notare che se usate **ImagePng**, **ImageCreateFromPng**, **ImageGif** ecc... e il supporto per PNG,GIF, ecc.. non è stato compilato nel PHP che state usando, viene emesso pure un Warning. In generale meglio ritardare l'emissione dell'header subito prima della **ImagePng**, **ImageGif** o **ImageJpeg**, in modo da poter vedere eventuali warning (tranne quelli che vengono generati da quest'ultime tre, tuttavia se PHP supporta il formato non dovrebbero esserci problemi con queste funzioni).

Le funzioni:

int **imagesx** (int *im*)
int **imagesy** (int *im*)

restituiscono rispettivamente la larghezza e l'altezza dell'immagine il cui id viene passato con argomento e sono utili soprattutto quando caricate delle immagini con **ImageCreateFrom*** e avete poi bisogno di conoscerne le dimensioni.

La funzione:

int **imagedestroy** (int *im*)

libera tutta la memoria associata all'immagine il cui id è *im*. Non è quasi mai necessario chiamare esplicitamente questa funzione, poiché la memoria allocata viene liberata al termine della richiesta. Può essere utile solo se il vostro script deve fare una elaborazione complessa dopo una **ImageGif**, **ImageJpeg**, ecc... insomma dopo aver utilizzato per l'ultima volta i dati di una immagine e volete anticipare la liberazione dello spazio di memoria.

Passiamo finalmente a descrivere qualche primitiva grafica, iniziando da quella che disegna un punto, o meglio un pixel:

int **imagesetpixel** (int *im*, int *x*, int *y*, int *col*)

im è naturalmente l'id dell'immagine in cui accendere il pixel. *col* è il colore definito con **imagecolorallocate**. *x*,*y* sono le coordinate. Il sistema di riferimento sullo schermo, come usuale in informatica è convenzionalmente scelto in maniera piuttosto diversa da come si fa di solito in matematica. L'origine, ossia il punto con coordinate 0,0 si trova nell'angolo in alto a sinistra dello schermo, anziché al centro o nell'angolo in basso a sinistra e quindi l'asse Y è rivolto verso il basso anziché verso l'alto.

```
0,0
+------------------>  X
|
|
|
|
|
|
v

Y
```

Questa sarà l'unica primitiva grafica che serve per disegnare l'insieme di Mandelbrot e in effetti tutte le altre primitive grafiche si possono implementare usando questa. Per risparmiarvi la fatica comunque PHP definisce per voi molte altre primitive.

Usando un opportuno algoritmo si possono disegnare con la migliore approssimazione le linee rette (che non sono altro che un insieme di punti a scalino). La funzione che fa questo è:

int **imageline** (int *im*, int *x1*, int *y1*, int *x2*, int *y2*, int *col*)

x1,*y1* e *x2*,*y2* sono i punti di partenza, *col* e *im* sapete già cosa sono. In particolare si possono disegnare delle linee orizzontali o verticali, come in quest'esempio.

```
<?php
// il giallo sfuma verticalmente nel bianco
$im=imageCreate(100, 256);
for ($i=0; $i<256; $i++)
{
   $color=imageColorAllocate($im, 255, 255, $i);
   imageLine($im, 0, $i, 99, $i, $color);
}

header("Content-type: image/png");
imagePng($im);
?>
```

Per disegnare delle linee punteggiate anziché "continue" si usa la seguente funzione analoga:

int **imagedashedline** (int *im*, int *x1*, int *y1*, int *x2*, int *y2*, int *col*)

provate a vederne l'effetto sostituendo **imageDashedLine** a **imageLine** nel programmino precedente.

Il demo seguente disegna una immagine con un po' di linee a caso su sfondo nero ogni volta diverse e di colori diversi, in un quadrato 100x100:

```
<?php
########## linee casuali
define('QUANTE', 20);

function setWebSafePalette($im)
{
   ... come prima
}

$im=setWebSafePalette(imageCreate(100, 100));

// seed with microseconds since last "whole" second
srand ((double) microtime() * 1000000);

for ($i=0; $i<QUANTE; $i++)
   imageLine($im, rand(0,99), rand(0,99),
rand(0,99), rand(0,99), rand(1,215));

header("Content-type: image/png");
imagePng($im);
?>
```

Anche qui vi invito a provare l'effetto di **imageDashedLine**.

È possibile disegnare con una sola chiamata di funzione anche dei poligoni.

int **imagepolygon** (int *im*, array *points*, int *num_points*, int *col*)

Questa funzione crea un poligono nell'immagine individuata da *im* avente tutti i lati di un unico colore *col*. *points* è semplicemente un array ad una sola dimensione che contiene ordinatamente ascisse e ordinate dei vari vertici del poligono. *num_points* è il numero totale di vertici.

Ad es. se volete disegnare un triangolo di vertici (x_0, y_0), (x_1, y_1) e (x_2, y_2), dovete impostare l'array da passare alla funzione in questo modo:

```
$points[0] = x0;
$points[1] = y0;
$points[2] = x1;
$points[3] = y1;
$points[4] = x2;
$points[5] = y2;
```

e passare 3 per il parametro *num_points*. Ecco un esempio:

```
<?php
########## un triangolo

function setWebSafePalette($im)
{
   ... come già visto
}

$im=setWebSafePalette(imageCreate(70, 70));

$points[0] = 10;
$points[1] = 20;
$points[2] = 60;
$points[3] = 30;
$points[4] = 30;
$points[5] = 50;

imagePolygon ($im, $points, 3, 200);

header("Content-type: image/png");
imagePng($im);
?>
```

Se dichiaraste 2 come terzo parametro di **imagePolygon** nell'esempio precedente verrebbe disegnato solo il lato superiore del triangolo: un poligono composto da due punti si riduce infatti ad una linea. Banalmente un poligono che ha un solo vertice si riduce ad un unico punto.

Naturalmente l'esempio precedente poteva essere abbreviato usando il costrutto `array`, persino direttamente nel secondo argomento di **imagePolygon**:

```
imagePolygon ($im, array(10,20, 60,30, 30,50), 3, 200);
```

imagePolygon unisce i punti specificati nell'ordine specificato tramite delle linee. Infine unisce l'ultimo punto col primo. Ne segue che per un triangolo l'ordine con cui si specificano i punti è ininfluente, ma quando i lati son più di tre, l'ordine conta. Ad es. confrontare l'output delle istruzioni:

```
imagePolygon ($im, array(20,20, 40,30, 60,20, 40,50), 4, 200);
imagePolygon ($im, array(20,20, 60,20, 40,30, 40,50), 4, 200);
```

in cui sono stati scambiati il secondo e il terzo punto. Come potete vedere i quadrilateri sono ben diversi: il primo è una freccia diretta verso il basso, il secondo una freccia diretta obliquamente.

La funzione:

int **imagefilledpolygon** (int *im*, array *points*, int *num_points*, int *col*)

è la stessa di **imagepolygon**, solo che riempe il poligono con lo stesso colore dei lati.

Naturalmente un tipo di poligono molto comune tanto da avere delle funzioni apposita per disegnarlo vuoto o pieno è il rettangolo:

int **imagerectangle** (int *im*, int *x1*, int *y1*, int *x2*, int *y2*, int *col*)
int **imagefilledrectangle** (int *im*, int *x1*, int *y1*, int *x2*, int *y2*, int *col*)

In entrambe queste funzioni si forniscono solamente due vertici opposti del rettangolo. Se dovessimo utilizzare **imagepolygon** per disegnare un rettangolo sarebbe più noioso perché dovremmo fornire sempre tre punti.

Un esempio utile e semplice allo stesso tempo di applicazione della funzione **imageFilledRectangle** si trova in questa funzione, la quale vi permette di ottenere un file Png che rappresenta graficamente i primi 256 colori della palette di un'immagine il cui handle viene passato per argomento.

```
// shows the 256 color palette of an image or the web-safe one
//(when $im==0) returns html to display $filename or nothing
// if $filename=''
function imagePng256colorPalette($im=0, $filename='')
{
   define('SWIDTH', 15);        // square color width
   define('IWIDTH', SWIDTH*16); // image width (and height)
```

```
   $pal_im=imageCreate(IWIDTH, IWIDTH);

   if ($im)
      // copy $im palette into $pal_im
      for ($i=0; $i<256; $i++)
      {
         $col = @imageColorsForIndex($im, $i);
         imageColorAllocate($pal_im, $col["red"], $col["green"], $col["blue"]);
      }
   else
      // no image specified, display 216 color web-safe palette
      setWebSafePalette($pal_im);

   // draw the "chessboard" rappresentation of the palette
   $i=0;
   for ($y=0; $y < 16; $y++)
      for ($x=0; $x < 16; $x++)
      {
         $x1 = $x*SWIDTH; $y1 = $y*SWIDTH;
         imageFilledRectangle($pal_im, $x1, $y1, $x1+SWIDTH, $y1+SWIDTH, $i++);
      }

   // output buffer to file or stdout
   if (!$filename)
   {
      header('Content-type: image/png');
      imagePng($pal_im);
   }
   else
   {
      imagePng($pal_im, $filename);
      return "<IMG SRC=\"$filename\" WIDTH=". IWIDTH ." HEIGHT=". IWIDTH .">\n";
   }
}
```

Se nessun nome di file Png viene specificato, il file viene emesso su stdout preceduto dalla necessaria intestazione HTTP. Viene generata una immagine contenente una scacchiera 16*16 di caselle colorate. La scacchiera viene dipinta per righe da destra verso sinistra usando colori di numero crescente da 0 a 255. Variate a piacere la costante `SWIDTH` per ottenere una immagine più grande o più piccola. Per convenienza la funzione **imagePng256colorPalette** ritorna il codice html necessario a visualizzare il file creato sul server, se viene specificato il parametro *filename* altrimenti non ritorna niente. Questo permette di scrivere codice del tipo:

```
<%
   require '256color.inc';

   ... creazione immagine $im
%>
...
<% echo imagePng256colorPalette($im, "impal.png"); %>
...
```

Questo codice permette di visualizza la palette dell'immagine `$im` a scopo di debugging. Tutto quello che dovete fare è aggiungere da qualche parte nell'html o nei tag php l'istruzione:

```
<% echo imagePng256colorPalette($im, "impal.png"); %>
```

Ricordate comunque che il file `impal.png` viene scritto nella directory dello script sul server (ed occorre che lo script abbia il permesso di scriverlo) e poi alla fine dello sviluppo del programma rimuoverlo, tanto per non lasciare cose inutili.

Il programma ricopia la palette dell'immagine `$im` nell'immagine locale: per far questo usiamo una funzione che data un'immagine e un indice nella palette di quell'immagine, ritorna i valori RGB a quell'indice:

array **imagecolorsforindex** (int *im*, int *index*)

I valori vengono ritornati in un array associativo contenente le chiavi red, green, e blue.

L'operatore `@` serve per evitare l'emissione di fastidiosi Warning del tipo:

```
Warning: Color index out of range in /path_to/256color.inc on line 37
```

Anche se l'ordine con cui sono presentati i colori non è dei migliori, con questa funzione è possibile ottenere una rappresentazione grafica della palette websafe di 216 colori più volte citata, semplicemente scrivendo un programmino così semplice (che non crea alcun file sul server):

```
<?php
   require '256color.inc';

   imagePng256colorPalette();
?>
```

Il file `256color.inc` contenente questa funzione e **setWebSafePalette** potete scaricarlo pronto all'uso (vedi inizio del tutorial).

Nel caso vogliate rappresentare la palette websafe di Linux potete fare così:

```
<?php
   require '256color.inc';

   $im=imageCreate(0,0);
   setLinuxWebSafePalette($im);
   imagePng256colorPalette($im);
?>
```

Con un'unica funzione potete disegnare cerchi, ellissi e i rispettivi archi:

int **imagearc** (int *im*, int *cx*, int *cy*, int *w*, int *h*, int *s*, int *e*, int *col*)

dove:

im immagine su cui disegnare

cx,cy centro dell'arco di ellisse

w,h larghezza ed altezza dell'arco di ellisse (diametri)

s,e angolo di inizio (start) e fine (end) dell'arco di ellisse

col colore da usare

Infatti un cerchio è un caso particolare di ellisse e un ellisse è un caso particolare di arco di ellisse (l'arco di ellisse completo, che spazza 360 gradi). Ecco perché ho usato solo il termine arco di ellisse, che comprende sia gli ellissi, che i cerchi che gli archi di cerchio. La convenzione sugli angoli : da 0 a 360 in senso orario e con l'angolo di ampiezza zero coincidente col raggio orizzontale destro dell'ellisse.

```
# ecco un arco di 60° a partire da 0°, di un ellisse
# di centro 35,35 e di diametri orizzontale,verticale
# 50,40 disegnato col colore numero 200
# col colore 100 viene disegnato il centro
# col colore 50 vengono disegnati i diametri
imageSetPixel ($im, 35,35, 100);
imageLine ($im, 35+1,35, 60,35, 50);
imageLine ($im, 35,35+1, 35,55, 50);
imageArc ($im, 35,35, 50,40, 0,60, 200);
```

Non c'è una funzione per riempire un cerchio o un ellissi di un colore, tuttavia potete usare la funzione:

int **imagefill** (int *im*, int *x*, int *y*, int *col*)

che implementa l'algoritmo di flood fill partendo dalle coordinate *x*, *y* e usando il colore numero *col* dell'immagine *im*.

```
# ecco un cerchietto pieno
imageArc ($im, 35,35, 50,50, 0,360, 200);
imageFill ($im, 35,35, 3);

# osservate cosa accade se c'è un buco nel recinto del cerchio
# la vernice esce anche al di fuori.
imageArc ($im, 35,35, 50,50, 0,350, 200);
imageFill ($im, 35,35, 3);
```

Una specializzazione di **imagefill** è:

int **imagefilltoborder** (int *im*, int *x*, int *y*, int *border*, int *col*)

la quale "spande il colore" scavalcando anche i bordi di colore diverso dal colore border specificato. Tutti gli altri parametri hanno la stessa funzioni che in *imagefill*.

Qui non discuterò tutte le funzioni relative alle immagini (vi rimando al manuale PHP per una lista completa), tuttavia questo tutorial non sarebbe completo se non parlassi un po' delle primitive per emettere del testo.

La funzione:

int **imagestring** (int **im**, int *font*, int *x*, int *y*, string *s*, int *col*)

disegna la stringa *s* col colore *col* nell'immagine *im* a partire dalle coordinate *x*, *y* (che rappresentano l'angolo in alto a sinistra della striscia di testo, o si può anche dire le coordinate dell'angolo in alto a sinistra del primo carattere della stringa) usando il numero di font specificato. I font numero 1, 2, 3, 4 e 5 rappresentano 5 diverse grandezze di un font di default che sicuramente c'è e che è del tipo fixed-width, ossia i caratteri hanno tutti la stessa dimensione come nei modi testuali.

```
<?php
// demo di imageString

require '256color.inc';

$i=setWebSafePalette(imageCreate(120,120));

for ($fs=1; $fs<=5; $fs++)
   imageString($i, $fs, 5,($fs-1)*20+5, 'php is cool!', 200);

header('Content-type: image/png');
imagePng($i);
?>
```

Numeri di font maggiori di 5 rappresentano font definiti dall'utente (sempre di tipo fixed) che è possibile caricare da un file esterno usando la funzione **ImageLoadFont**[23].

La funzione seguente è analoga a imagestring, ma disegna la stringa in verticale.

int **imagestringup** (int *im*, int *font*, int *x*, int *y*, string *s*, int *col*)

in tal caso le coordinare *x*, *y* sono sempre quelle dell'angolo superiore sinistro del blocco carattere, ma il blocco carattere va visto nella sua orientazione naturale (in altri termini girate la testa di 90 gradi antiorari ;)

[23]Vedi manuale.

```
<?php
// demo di imageStringUp

require '256color.inc';

$i=setWebSafePalette(imageCreate(120,120));

for ($fs=1; $fs<=5; $fs++)
   imageStringUp($i, $fs, ($fs-1)*20+5,110, 'php is cool!', 200);

header('Content-type: image/png');
imagePng($i);
?>
```

Gli ultimi due esempi vi fanno vedere le 5 grandezze dei font, ma notate che le scritte che si succedono non sono equispaziate. Riscriviamo il primo esempio in modo che le scritte siano equispaziate di `SPACE` pixel. Per ottenere l'altezza di un carattere in un font fixed di numero *font*, dove è costante, si usa la funzione:

int **imagefontheight** (int *font*)

analogamente esiste la funzione che ritorna la larghezza:

int **imagefontwidth** (int *font*)

```
<?php
// demo di imageString migliorato

require '256color.inc';

define('SPACE', 5);

$i=setWebSafePalette(imageCreate(120,120));

$y=10;
for ($fs=1; $fs<=5; $fs++)
{
   imageString($i, $fs, 5,$y, 'php is cool!', 200);
   $y += imageFontHeight($fs)+SPACE;
}

header('Content-type: image/png');
imagePng($i);
?>
```

Nei rari casi in cui dovete stampare un solo carattere (il primo carattere di una stringa) potete usare le funzioni:

int **imagechar** (int *im*, int *font*, int *x*, int *y*, string *c*, int *col*)
int **imagecharup** (int *im*, int *font*, int *x*, int *y*, string *c*, int *col*)

rispettivamente per la stampa del carattere in orizzontale e verticale.

Se i font fixed non sono di vostro gradimento e avete compilato PHP includendo il supporto per i font TrueType che sono antialiased (nel senso che presentano meno frastagliature), scalabili e rotabili potete utilizzarli tramite la funzione **imagettftext**[24].

Nel mio caso quando ho installato GNOME, è stato installato il pacchetto bf freetype che è un motore per il rendering dei font TrueType free e portabile, ma per ragioni di copyright, non sono inclusi molti file `.ttf` contenenti i font. Se avete Windows li trovate nella directory `windows/fonts`.

Ricopio un esempio tratto dalla documentazione di PHP:

```
<?php
 Header ("Content-type: image/png");
 $im = imagecreate (400, 30);
 $black = ImageColorAllocate ($im, 0, 0, 0);
 $white = ImageColorAllocate ($im, 255, 255, 255);
 ImageTTFText ($im, 20, 0, 10, 20, $white, "/mnt/win/windows/fonts/arial.ttf",
"Testing... Omega: &#937;");
 ImagePng ($im);
 ImageDestroy ($im);
 ?>
```

Provate a sostituite `arial.ttf` con un altro tipo di font di Windows (ad es. `times.ttf`).

Nel mio sistema RedHat ho trovato alcuni font ttf installati, facenti parte del pacchetto enlightenment[25] (il window manager di GNOME):

```
/usr/share/enlightenment/E-docs/aircut3.ttf
/usr/share/enlightenment/E-docs/benjamingothic.ttf
/usr/share/enlightenment/E-docs/neuropol.ttf
/usr/share/enlightenment/E-docs/rothwell.ttf
/usr/share/enlightenment/E-docs/unionform.ttf
/usr/share/enlightenment/E-docs/x-files.ttf
/usr/share/enlightenment/themes/ShinyMetal/ttfonts/rothwell.ttf
/usr/share/enlightenment/themes/ShinyMetal/ttfonts/zirkle.ttf
```

Quest'ultimo esempio mostra uno stesso font (lo zirkle, ma potete usare ovviamente anche un altro a vostra scelta) renderizzato a varie dimensioni (tra 1 e 20):

---

[24] Vedi manuale.

[25] La **libttf** library necessaria per utilizzare questi tipi di font nel caso non sia installata sul vostro sistema è disponibile al sito http://www.freetype.org/).

```
<?php
 define('FONTPATH',
'/usr/share/enlightenment/themes/ShinyMetal/ttfonts/zirkle.ttf');
 define('STRING', 'PHP can use ttf fonts!');
 define('SPACER', 4);

 Header ("Content-type: image/png");
 $im = imagecreate (400, 350);
 $black = ImageColorAllocate ($im, 0, 0, 0);
 $white = ImageColorAllocate ($im, 255, 255, 255);

 $y=0;
 for ($i=1; $i<=20; $i++)
 {
    /*
ImageTTFBBox() returns an array with 8 elements representing
four points making the bounding box of the text:

0 lower left corner, X position
  1 lower left corner, Y position
2 lower right corner, X position
3 lower right corner, Y position
4 upper right corner, X position
5 upper right corner, Y position
6 upper left corner, X position
7 upper left corner, Y position
    */

    $box=imageTTFBBox ($i, 0, FONTPATH, STRING);
    $y += -$box[5] + SPACER; // or -$box[7] + SPACER;
    ImageTTFText ($im, $i, 0, 10,$y, $white, FONTPATH, STRING);
 }
 ImagePng ($im);
 ImageDestroy ($im);
 ?>
```

Sostituite `$white` con `-$white` nel programma precedente; questo disabilita l'anti-aliasing ed è istruttivo proprio per rendersi conto di come l'anti-aliasing migliora la qualità.

L'esempio seguente stampa in una rosa la parola Ninuzzo (il mio nickname):

```
<?php
                // copyright 2001 Antonio Bonifati
                // distributed under GPL
```

```
define('FONTPATH', '/mnt/win/windows/fonts/arial.ttf');
define('STEP', 20);
define('RADIUS', 40);
define('CENTERX', 100);
define('CENTERY', 100);
define('SIZE', 10);
define('STR', 'ninuzzo');

Header ("Content-type: image/png");
$im = imagecreate (200, 200);
$black = ImageColorAllocate ($im, 0, 0, 0);
$white = ImageColorAllocate ($im, 255, 255, 255);

for ($g=0; $g<360; $g+=STEP)
{
   $r=$g*M_PI/180;
   $x=RADIUS*cos($r)+CENTERX;
   $y=CENTERY-RADIUS*sin($r);
   # for debugging only
   #ImageSetPixel($im, $x,$y, $white);
   ImageTTFText ($im, SIZE, $g, $x,$y, $white, FONTPATH, STR);
}

ImagePng ($im);
ImageDestroy ($im);
?>
```

Chi conosce un po' di matematica (trigonometria in particolare) non avrà difficoltà a capire quali sono le trasformazioni coinvolte nel problema.

Un'altra categoria di font che potete utilizzare in PHP sono i font "PostScript Type 1". Per maggiori informazioni vi rimando al manuale di PHP.

Abbiamo descritto tutte le funzioni grafiche più comuni. Siete ora in grado di capire completamente lo script proposto all'inizio per etichettare uno sfondo di un bottone.

Ci sarebbero altre funzioni grafiche che non descriverò che permettono di settare la modalità interlacciata e la trasparenza per i formati che le supportano, che permettono di operare alcune altre operazioni sull'array della palette, ecc... In particolare vi invito ad approfondire per conto vostro le seguenti che a volte possono essere utili:

```
GetImageSize()
ImageColorTransparent()
ImageInterlace()
```

```
ImageColorAt()
ImageColorExact()
ImageColorsTotal()
ImageColorDeAllocate()
ImageTypes(void)
```

2.5 Un esempio: l'insieme di Mandelbrot

-John Hubbard, descrivendo l'insieme di Mandelbrot, 1985

Un po' di tempo fa scrissi per il DOS, col compilatore DJGPP[26] un programmino molto semplice per disegnare l'insieme di Mandelbrot.

Ho deciso di propinarvi questa curiosità matematica stavolta in PHP. Alla fine di questa sezione (e di questo tutorial) avremo creato un generatore di immagini dell'insieme di Mandelbrot (che prende il nome dal suo scopritore, il matematico della IBM Benoit Mandelbrot) che si può usare via web e che rappresenta un gadget che potete aggiungere nel vostro sito per attirare i visitatori ed invogliargli a ritornarvi. Avrei potuto scegliere di affrontare altre applicazioni, probabilmente di valenza pratica maggiore (avevo in mente anche di scrivere qualche generatore di grafici a barre e a torta), ma alla fine ho scelto di proporvi questo famoso frattale, assecondando la mia passione per la matematica e la programmazione pure.

La lezione è stata scritta per chi sa poco o ricorda poco di matematica, quindi se siete dei bravi matematici vi avverto che mi soffermerò su concetti che avete sicuramente già afferrato e quindi vi annoierò con un po' di ripetizione e qualche dimostrazione facile.

L'insieme di Mandelbrot scaturisce da una rappresentazione grafica dell'insieme di non divergenza di una famiglia di successioni di numeri complessi, detta successione di Mandelbrot. Spiegherò man mano il significato profondo di queste parole.

La successione di Mandelbrot si può definire facilmente in modo ricorsivo. siano z e c due numeri complessi e sia n un numero naturale. La successione di Mandelbrot è costituita dalla sequenza di numeri z_n tali che:

$$z_0 = 0$$
$$z_n = z_{n-1}^2 + c$$

come si vede si tratta di una successione parametrica, ossia di infinite successioni che si ottengono fissando di volta in volta i valori di c nel campo complesso.

Ricordo a proposito che i numeri complessi sono un'estensione del campo dei numeri reali e costituiscono un campo in cui è sempre possibile effettuare tutte le operazioni che si possono fare nel campo complesso con in più la possibilità di fare sempre l'estrazione di radice (o l'elevamento a potenza), cosa non sempre possibile

[26]http://www.delorie.com/djgpp

nel campo reale. Per chi non ne abbia mai sentito parlare ecco un ragionamento che induce a definire tali tipi di numeri.

Considerate ad es. la seguente radice quadrata:

$$\sqrt{-3}$$

questa non è possibile effettuarla nel campo dei numeri reali, poiché dovrebbe essere uguale ad un numero che elevato al quadrato dà -3. Per la regola dei segni ogni numero elevato al quadrato è positivo, quindi non esiste alcun numero reale il cui quadrato sia negativo. Se conveniamo di definire i un numero (che non è un numero reale, ma è un nuovo tipo di numero) tale che:

$$i^2 = -1$$

possiamo scrivere, sfruttando le proprietà della radice e la definizione di i:

$$\sqrt{-3} = \sqrt{3i^2} = \sqrt{i^2}\sqrt{3} = i\sqrt{3}$$

Il numero i è il numero il cui quadrato è -1; è esso stesso un numero complesso e viene detto *"unità immaginaria"*.

Per la precisione le radici di -3 nel campo complesso sono due. Oltre a $i\sqrt{3}$ infatti anche $-i\sqrt{3}$ elevata al quadrato dà -3. In generale si può dimostrare che le radici ennesime di un qualsiasi numero complesso (eccetto lo zero) sono n distinte e si può ricavare una formula per calcolarle. Il calcolo precedente va quindi scritto più correttamente in questo modo, senza scartare l'altra radice:

$$\sqrt{-3} = \sqrt{3i^2} = \sqrt{i^2}\sqrt{3} = \pm i\sqrt{3}$$

Il calcolo precedente ci suggerisce di definire un nuovo campo di numeri, che si può verificare gode di tutte le proprietà del campo R e ne rappresenta una estensione, in quanto tutte le operazioni definite in R in C si possono sempre effettuare. Questi numeri, detti "complessi" si possono definire in modo che costituiscano un campo estensione di R in questo modo:

un numero complesso è un numero della forma $a + bi$ dove a e b sono numeri reali e i è l'unità immaginaria.

È evidente che quando $b = 0$ il numero complesso è anche un numero reale. Ogni numero reale è complesso, ma non viceversa, poiché ovviamente se $b \neq 0$ l'oggetto $a + bi$ è davvero qualcosa di nuovo. Le operazioni di somma algebrica, moltiplicazione e divisione tra questi numeri si effettuano secondo le usuali regole dell'algebra dei binomi. La somma è definita in questo modo:

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

la moltiplicazione non è altro che somme ripetute e si può ottenere velocemente applicando la formula generale seguente:

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i$$

Altri modi diffusi di rappresentare i numeri complessi sono:

$a + bj$ (l'unità immaginaria viene chiamata j)

$bi + a$ (si scrive prima la parte immaginaria e poi quella reale)

(a, b) (coppia ordinata)

Il metodo di rappresentazione non ha nessuna importanza tranne che pratica, così come tutta la matematica è indipendente dalla base del sistema di numerazione posizionale usato o persino dal sistema di numerazione usato. Una simbologia è migliore di un'altra se facilita la manipolazione dei simboli rispetto alla prima (ad es. è molto più semplice fare calcoli con numeri arabi che romani), ma da un punto di vista formale simbologie equivalenti non possono comportare alcuna differenza nel significato degli assiomi e nei teoremi che ne derivano.

Mentre i numeri reali si possono rappresentare su una retta, per rappresentare univocamente un numero complesso occorre uno spazio bidimensionale, ossia un piano. La convenzione è di fissare due assi reali perpendicolari sul piano orientati in questo modo:

```
^ Im
|
|
|
|
--------+--------> Re
|
|
|
|
```

e rappresentare la parte reale del numero complesso sull'asse orizzontale (detto per l'appunto asse reale o asse Re) e la parte immaginaria sull'asse verticale, nello stesso modo in cui si rappresentano numeri reali su una retta (ricordate che abbiamo detto che la parte reale e immaginaria sono entrambi numeri reali). Il punto corrispondente al numero si trova all'incrocio delle parallele all'altro asse condotte per i punti segnati sugli assi.

Tornando alla successione di Mandelbrot, vediamo adesso di capire di che cosa si tratta. Queste due formule definiscono il valore di z per tutti gli infiniti n naturali:

$$z_0 = 0$$
$$z_n = z_{n-1}^2 + c$$

e rappresentano un modo non ambiguo per definire questi infiniti numeri. Equivale a scrivere la prima parte della seguente catena infinita di uguaglianze lasciando poi dei puntini (in cui sta purtroppo l'ambiguità della definizione: i ... a rigore non hanno infatti un significato preciso in matematica, anche se si intuisce come debbano proseguire le cose):

$$\begin{gathered} z_0 = 0 \\ z_1 = z_0^2 + c \\ z_2 = z_1^2 + c \\ z_3 = z_2^2 + c \\ \ldots \end{gathered}$$

Volendo essere ancora "infinitamente più prolissi", se chiamiamo a la parte reale di c, che ricordo è definito essere un numero complesso e b la sua parte immaginaria possiamo scrivere le precedenti definizioni così:

$$\begin{gathered} z_0 = 0 \\ z_1 = a + bi \Rightarrow z_1 = z_0^2 + c = 0^2 + a + bi \\ z_2 = (a+bi)^2 + a + bi \Rightarrow z_2 = z_1^2 + c = (a+bi)^2 + c \\ z_3 = ((a+bi)^2 + a + bi)^2 + a + bi \Rightarrow z_3 = z_2^2 + c = ((a+bi)^2 + c)^2 + c \\ \ldots \end{gathered}$$

Spero con questo di avere chiarito la definizione matematica di successione di Mandelbrot. Questi numeri complessi z formano una successione i cui primi termini sono di questo tipo:

$$0, a + bi, (a+bi)^2 + a + bi, ((a+bi)^2 + a + bi)^2 + a + bi, \ldots$$

se dovesse descriverla a parole basterebbe dire che il primo termine è zero e che tutti gli altri termini si ottengono sommando al quadrato del precedente il numero complesso c; appunto questo dice la definizione ricorsiva, che ha il pregio di essere compatta e non ambigua. Daccordo, mi sono preso la licenza di fare qualche semplificazione di zeri, altrimenti avrei dovuto scrivere:

$$0, 0^2 + a + bi, (0^2 + a + bi)^2 + a + bi, ((0^2 + a + bi)^2 + a + bi)^2 + a + bi, \ldots$$

Poco prima abbiamo visto la rappresentazione cartesiana dei numeri complessi. Occorre dire che se immaginate di congiungere il punto che rappresenta un numero complesso $c = a + bi$ con l'origine degli assi e applicate il teorema di Pitagora per calcolare la distanza cartesiana del punto dall'origine, ecco cosa ottenete:

$len(c) = \sqrt{a^2 + b^2}$

la funzione *len* associa a ciascun numero complesso un numero reale (come si dice va da $C \to R$) e può essere chiamata, tanto per intenderci, *"lunghezza di un numero complesso"*. Notate che essendo $a^2 + b^2$ sempre positivo, al più nullo quando $a = b = 0$, la radice quadrata si può sempre fare e quindi *len* per l'esattezza ha come immagine $\Re^+$. Notate poi che quando $b = 0$, ossia c si particolarizza in un numero reale, si ha che:

$$len(c) = |c|$$

ossia potete considerare len come l'analogo della funzione valore assoluto in R. In effetti anche la funzione valore assoluto che associa ad ogni numero reale il numero stesso privato del segno ha analoga interpretazione geometrica di *len*: esprime infatti sulla retta reale la distanza del numero dall'origine. La retta reale è uno spazio di dimensione uno. Il piano complesso è uno spazio di dimensione due[27].

Siccome il valore assoluto della differenza tra due numeri reali esprime la distanza tra questi due numeri (che appunto è una quantità positiva), così pure $|a-b| = |b-a|$, dove a e b sono due numeri complessi esprime la distanza tra questi due numeri, ossia la distanza in linea retta tra i due punti che li rappresentano. È molto semplice rendersi conto di questo fatto geometricamente. Faremo presto uso di questa semplice formula per definire il concetto di limite.

Definita anche questa funzione *len*, torniamo alla successione di Mandelbrot definita dalle equazioni di ricorrenza:

$$z_0 = 0$$
$$z_n = z_{n-1}^2 + c$$

Nonostante tutti i nostri sforzi su questi nuovi enti astratti che sono i numeri complessi (che in fondo poi non sono meno astratti dei numeri reali, è solo che non siamo così abituati ad usarli) non abbiamo ancora definito cosa sia l'insieme di Mandelbrot, ma abbiate pazienza ci siamo vicini.

Considerate la successione di Mandelbrot ottenuta per un fissato valore di c (vale a dire, essendo $c = a + bi$, per un fissato valore di a e b). Ci si chiede, intuitivamente, se questa sequenza di numeri complessi presenti numeri che al crescere indefinitamente di n, siano in lunghezza sempre più grandi, oppure che ci sia un valore di tendenza a cui la lunghezza si avvicina sempre di più all'aumentare di n. Naturalmente in matematica le frasi *"siano in lunghezza sempre più grandi"* oppure *"si avvicina sempre di più"* non hanno alcun significato. Occorre precisarne il senso usando opportune proposizioni con i simboli e gli operatori non ambigui della matematica.

Il *"valore di tendenza"* di una successione di numeri complessi z_n, che viene anche detto *"valore limite"* o semplicemente *"limite"* l, esiste se e solo se assegnato ad arbitrio un numero reale $\varepsilon > 0$, si può trovare un naturale positivo p tale che per ogni naturale $n > p$ risulta che $|z_n - l| < \varepsilon$.

Per chi non è familiare col concetto di limite, che in matematica si definisce in maniera analoga in molte altre occasioni (come nel caso delle successioni reali o delle funzioni reali di variabile reale), è bene ripetere la definizione a parole: l (che in generale è un numero complesso) si dice limite di z_n se si verifica che comunque venga assegnato un numero reale e scelto sempre positivo, che rappresenta una

[27]Spesso in matematica sia la funzione valore assoluto | | che *len* vengono indicata racchiudendo tra due stanghette il suo argomento: così per $|c|$ si intende $len(c)$. Useremo d'ora in poi questa notazione che ha il vantaggio della brevità.

tolleranza rispetto al valore limite l, è possibile trovare un indice p della successione (in generale dipendente da ε, la dipendenza precisa dipende ovviamente dal tipo di successione che si considera) tale che da quell'indice in poi tutti i termini della successione (ossia tutti i termini z_n con $n > p$) risultano sempre ad una distanza da l minore della tolleranza ε assegnata. Si ricorda infatti che z_n e l sono punti del piano e l'interpretazione geometrica della definizione di limite di successione complessa data sopra è molto semplice: si vuol dire che l è limite di z_n se e solo se da un certo p in poi tutti i termini della successione cadono all'interno di un cerchio di centro il punto l e raggio pari ad ε, comunque sia scelto $\varepsilon > 0$. La cosa importante che si afferma e che rende quindi significativo il valore l che verifica questa proprietà è che la proprietà deve valere PER OGNI ε.

Riflettete bene sulla definizione precedente finché non l'avete digerita. Questa definizione non è niente di difficile, è solo il modo più semplice di tradurre in linguaggio rigoroso la proprietà che "al crescere indefinitamente di n i valori di z_n si avvicinano sempre di più a l". Qui l'ambiguità sta soprattutto nella parola "avvicinano". A rigore non ha senso dire che un numero è vicino ad un altro; ha senso invece dire, stabilita una tolleranza, che un numero dista dall'altro meno di quella tolleranza. È questa osservazione, unita alla definizione di distanza tra due numeri che conduce alla traduzione in termini rigorosi del concetto a volte persino intuitivo di limite, senza alcuna difficoltà.

La definizione rigorosa di limite permette di dimostrare alcuni teoremi e proprietà fondamentali di una nuova operazione, che nel nostro caso si applica alle successioni a valori complessi, operazione che si chiama *"passaggio al limite"* o se volete *"ricerca dei valori di tendenza"*. I teoremi e le proprietà di questa operazione, unite ad alcuni risultati raggiunti nel caso z_n sia qualche tipo particolarmente semplice di successione, permettono di calcolare i limiti in molti altri casi di z_n, senza dover per forza applicare la definizione, la quale richiede la verifica di una disequazione (parametrica in ε) che può essere molto complessa e quindi impraticabile.

Quando l appartenente a C esiste per z_n si dice anche che z_n è *"convergente a l"* o che *"converge ad l"*.

Abbiamo quindi chiarito il significato della proprietà "una sequenza di numeri complessi presenta numeri che man mano che procediamo nella sequenza si avvicinano sempre di più ad un numero complesso l". Il valore limite l può essere raggiunto una, infinite volte o nessuna volta; la nostra definizione non esclude che il valore limite possa essere raggiunto. Infatti se il valore limite viene raggiunto in qualche punto si ha che in quel punto:

$$|z_n - l| < \varepsilon$$

diventa:

$$0 < \varepsilon$$

che essendo *varepsilon* positivo ma non nullo è certamente verificata.

Non abbiamo però ancora chiarito in termini rigorosi cosa si deve intendere quando diciamo: mi pare che valga questa proprietà: che al crescere indefinitamente di n, i termini z_n sono sempre più lunghi, ossia si allontano sempre di più verso i bordi del piano complesso. Questa proprietà si formula in termini rigorosi in questo modo:
assegnato ad arbitrio un numero reale $M > 0$, si può trovare un naturale positivo p tale che per ogni naturale $n > p$ risulta che $|z_n| > M$.
in altri termini questo equivale a dire che i termini della successione da un certo punto in poi sono tutti maggiori di M, qualsiasi sia la tolleranza positiva M fissata. Ho indicato la tolleranza con due lettere diverse, ma questo non è necessario; l'ho fatto solo perché usare lettere diverse aiuta a ricordare che ε è una tolleranza che non deve essere superata, mentre M è una tolleranza che deve essere superata. Vi ricordo che, come già visto, $|z_n|$ non è altro che $|z_n - 0|$, ossia la distanza del punto z_n dall'origine del piano complesso.

Quando vale la proprietà precedente si dice brevemente che la successione (in generale complessa) z_n "diverge", oppure che è "divergente", una parola un po' ricercata, ma il significato preciso rimane quello che abbiamo visto. Si dice anche che il valore di tendenza o limite sia infinito, ma attenzione che si tratta solo di un modo di dire: infinito non è un numero né reale né complesso.

Riprendiamo il problema che ci aveva condotto a definire i concetti di successione convergente e divergente in modo rigoroso. Ci chiedevamo appunto per quali valori di c la successione di Mandelbrot:

$$\begin{gathered} z_0 = 0 \\ z_n = z_{n-1}^2 + c \end{gathered}$$

sia convergente o divergente. In realtà potrebbe non fare nessuna delle due cose, potrebbe ad es. essere alternante come ad es. la successione:

$$-1-j, 1+j, -1-j, 1+j, -1-j, 1+j, ...$$

meglio definita così:

$$\begin{gathered} z_n = -1-j \text{ per } n = 0 \text{ o pari} \\ z_n = 1+j \text{ per } n \text{ dispari} \end{gathered}$$

Facendo un po' di calcoli per valori particolari di c si vede che esistono dei punti c del piano complesso per i quali converge e altri per i quali diverge e altri ancora per i quali né diverge e né converge. Questi punti sono disposti in modo molto caotico e nessuno è mai riuscito a trovare una formula che permetta di ricavare questi punti, risolvendo in qualche modo la disequazione della definizione di limite.

L'insieme di Mandelbrot (abbreviato in mset - Mandelbrot Set) è definito rigorosamente come l'insieme di tutti i punti c del piano complesso per cui la successione generata dall'equazione di ricorrenza $z \leftarrow z^2 + c$ con valori iniziale $z = 0$ (un altro modo più compatto di indicare le due equazioni viste sopra) non diverge.

ES.
considerate l'origine, il punto $c = (0, 0)$. In questo punto l'equazione di ricorrenza diventa $z \leftarrow z^2$ sempre con z che parte dall'origine. Si ha banalmente che

$$z_0 = 0,\ z_1 = z_0^2 = 0^2 = 0,\ z_2 = z_1^2 = 0^2 = 0$$

insomma avrete capito che la successione di Mandelbrot si riduce alla successione costante 0 (o 0+0i se preferite) e che questa converge banalmente a 0 (infatti verifica la definizione di limite con $l = 0$). Quindi il punto (0,0) appartiene all'insieme di Mandelbrot.

dal'altro canto il punto $c = 1 + 0i = 1$ particolarizza la famiglia di successioni di Mandelbrot nella seguente successione:

$$\begin{gathered} z_0 = 0 \\ z_1 = z_0^2 + c = 1 \\ z_2 = z_1^2 + c = 1^2 + 1 = 2 \\ z_3 = z_2^2 + c = 5 \\ z_4 = z_3^2 + c = 26 \\ \ldots \end{gathered}$$

anche un bambino intuisce che i termini tendono rapidamente ad aumentare e che quindi la successione diverge, tuttavia i matematici hanno imparato a non fidarsi dell'intuizione che a volte risulta ingannevole, e tendono a dimostrare e a verificare tutto, quindi occorre dimostrarne la divergenza in base alla definizione. La dimostrazione è piuttosto semplice, poiché la successione si è ridotta ad una successione di numeri reali che si scrive così:

$$\begin{gathered} z_0 = 0 \\ z_n = z_{n-1}^2 + 1 \end{gathered}$$

stavolta pensando z semplicemente come un numero reale. La successione diverge poiché è possibile trovare una successione i cui elementi sono tutti minori di z_n (a parità di n) che diverge. La successione divergente che "minòra" la successione z_n è la seguente (ho semplicemente tolto il quadrato):

$$\begin{gathered} m_0 = 0 \\ m_n = m_{n-1} + 1 \end{gathered}$$

questa successione è coincidente con la successione dei numeri naturali:

$$0, 1, 2, 3, 4, 5, 6, \ldots$$

infatti si potrebbe definire semplicemente così, in modo non ricorsivo:

$$m_n = n$$

Dimostrazioni: ecco la dimostrazione che $m_n \le z_n$ per ogni n che si consegue facilmente usando il principio di induzione:

1. $m_0 \leq z_0$ è vera, infatti $m_0 = 0$ e $z_0 = 0$, quindi diventa $0 \leq 0$
2. mostriamo che $m_n \leq z_n$ implica che $m_{n+1} \leq z_{n+1}$. Supponendo vera la prima si ha:

$$m_n \leq z_n; m_n + 1 \leq z_n + 1$$

(ho sommato 1 ad entrambi i membri)

$$m_{n+1} \leq z_n + 1;$$

($m_n + 1$ vale m_{n+1} secondo definizione)

ora siccome è facile riconoscere che i z_n sono tutti numeri positivi, si può aumentare il secondo membro della diseguaglianza moltiplicando z_n per z_n e la diseguaglianza continuerà a valere. Si ottiene così:

$$m_{n+1} \leq z_n^2 + 1$$

cioè (vedi definizione di z_n):

$$m_{n+1} \leq z_{n+1}$$

e quindi il secondo passo induttivo è dimostrato. Abbiamo così dimostrato che $m_n \leq z_n$ per tutti gli infiniti n naturali.

Dimostriamo ora che una successione è divergente se esiste una successione minorante divergente. Il risultato si dimostra in generale nel caso di successioni a valori complessi e vale naturalmente anche per successioni a valori reali che non sono altro che un caso particolare di quelle complesse. Dire che m_n è divergente equivale a dire, secondo definizione, che:
assegnato ad arbitrio un numero reale $M > 0$, si può trovare un naturale positivo p tale che per ogni naturale $n > p$ risulta che $|m_n| > M$.

Ora per ipotesi si sa che per ogni n (ma basta anche da un certo n in poi) tutti i termini di z_n sono di lunghezza maggiore o tutt'alpiù uguale ai corrispondenti (cioè stesso n) termini di m_n, in formule:

$$|z_n| \geq |m_n| \; \forall \; n$$

Inserendo questo risultato nella proposizione precedente per ipotesi vera si ha che:
assegnato ad arbitrio un numero reale $M > 0$, si può trovare un naturale positivo p tale che per ogni naturale $n > p$ risulta che:

$$|z_n| \geq |m_n| > M.$$

siccome $a \geq b > c$ implica che $a > c$, posso semplificare la proposizione precedente in questo modo:
assegnato ad arbitrio un numero reale $M > 0$, si può trovare un naturale positivo p tale che per ogni naturale $n > p$ risulta che $|z_n| > M$
ma questa è esattamente la definizione di divergenza della successione z_n. Quindi la successione z_n nelle ipotesi fatte diverge.

Infine terzo esempio: un punto in cui la successione di Mandelbrot né converge e né diverge, cioè non verifica né la definizione di convergenza né di divergenza (un tale punto appartiene comunque all'insieme di Mandelbrot; nella definizione di insieme di Mandelbrot oltre a *"non diverge"* si può anche dire *"è limitata"*.

$$\begin{gathered} c = j \\ z_0) = 0 \\ z_1 = 0^2 + j = j \\ z_2 = j^2 + j = j - 1 \\ z_3 = (j-1)^2 + j = j^2 + 1 - 2j + j = -j \\ z_4 = (-j)^2 + j = j^2 + j = j - 1 \end{gathered}$$

il calcolo fino ad $n = 4$ basta per dimostrare che da $n = 2$ i valori $j - 1$,$-j$ si ripetono sempre uguali. La successione è quindi "oscillante" tra questi due valori e non tende ad alcun limite né finito né infinito nel punto $c = j$ (l'unità immaginaria).

Senza altri indugi, cerchiamo ora di stabilire un algoritmo che cerchi di dare una rappresentazione dell'insieme di Mandelbrot. Una idea semplice potrebbe essere di rappresentare tutti i punti che appartengono all'insieme in nero e tutti i punti del piano che non vi appartengono in bianco o viceversa. Si ottiene così una rappresentazione "monocromatica" dell'insieme caotico di Mandelbrot. Più avanti vedremo come ottenere qualche rappresentazione a colori (l'assegnazione dei colori è naturalmente artificioso e può essere fatto arbitrariamente, anche se i migliori effetti si ottengono con schemi di colorazione ben precisi).

Poiché non sappiamo una formula che ci dia l'elenco preciso dei punti di non divergenza, come faremo a disegnare l'insieme? Evidentemente non possiamo verificare per ispezione diretta se un punto converge o diverge. Anche se per un certo c siamo arrivati a $n = 1000$ e z_n è un numero grandissimo nessuno ci assicura che z_n continuerà a crescere. Ad un certo punto potremmo andare in overflow e il nostro calcolatore sbaglierà i calcoli. Per la stessa ragione, anche se dopo 1000 iterazioni abbiamo trovato che un punto sembra convergere ad un valore costante o oscillare, non possiamo essere sicuri che si tratti di una falsa convergenza cioè che con un numero maggiore di iterazioni la tendenza si inverta. Con un calcolatore di potenza finita, in grado di trattare numeri di dimensione e precisione finita e con un tempo finito a nostra disposizione non possiamo calcolare l'infinito e a rigore non saremmo in grado di dimostrare la convergenza o la divergenza nemmeno in un solo punto.

Anche se non possiamo ricavare una rappresentazione esatta dell'insieme di Mandelbrot (che tra l'altro sullo schermo discreto di un calcolatore non potremmo rappresentare esattamente), possiamo tuttavia ottenererne una ragionevole approssimazione, che risulterà essere molto suggestiva e praticamente impossibile da ottenere manualmente, vista la notevole mole di calcoli che sono richiesti. Una rappresentazione finita e imprecisa di qualcosa che è infinitesimamente complesso e caotico sta per uscire dal vostro fido pc.

L'idea è piuttosto bruta: come detto prima è vero che non possiamo essere certi della convergenza o della divergenza di un punto dopo aver calcolato il 1000esimo termine della successione di Mandelbrot (d'ora in poi al posto del valore 1000 usiamo un valore costante MAXITER(ATIONS)), tuttavia ai fini di ottenere un disegno approssimato, possiamo benissimo dire che se dopo MAXITER il valore non supera in modulo una certa soglia, il punto appartiene all'insieme, altrimenti no. Resta il problema di stabilire come scegliere questo valore di soglia, poiché è una scelta che influenza di molto la precisione della nostra rappresentazione. Vedremo tra un po' che possiamo fare a meno per fortuna di incorrere nell'imbarazzo di questa scelta, grazie ad un ulteriore teorema sull'insieme di Mandelbrot.

Occupiamo prima ancora di un po' di matematica di numeri complessi, necessaria per poter formulare il programma. In PHP non possiamo effettuare calcoli tra numeri complessi, tuttavia possiamo scindere il calcolo di z_n nel calcolo della sua parte reale e della sua parte immaginaria, che sono due calcoli reali. Affinché ciò sia possibile infatti si è indotti a definire l'uguaglianza tra due numeri complessi come l'uguaglianza delle rispettive parti reali e immaginarie. Le relazioni $<$ e $>$ non sono invece di solito definite sui numeri complessi, ma solo sui loro moduli (cioè i risultati della funzione $|\ |$ applicata a ciascun numero).

Applichiamo la formula del prodotto tra due numeri complessi nel caso particolare in cui il primo numero è uguale al secondo (ometto il simbolo di moltiplicazione *):

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i$$

se $c = a$ e $d = b$ diventa:

$$(a + bi)(a + bi) = (aa - bb) + (ab + ba)i = (a^2 - b^2) + 2abi$$

si ha poi che:

$$z_n = z_{n-1}^2 + C$$

si può calcolare così, computando separatamente la parte reale e la parte immaginaria:

$$\begin{array}{c} Re(z_n) = Re(z_{n-1}^2) + Re(c) \\ Im(z_n) = Im(z_{n-1}^2) + Im(c) \end{array}$$

essendo

$$z_{n-1} = Re(z_{n-1}) + Im(z_{n-1})i$$

si ha: $z_{n-1}^2 = Re(z_{n-1})^2 - Im(z_{n-1})^2 + 2Re(z_{n-1})Im(z_{n-1})i$ essendo poi

$$c = Re(c) + Im(c)i$$

si ha infine:

$Re(z_0) = 0, Im(z_0) = 0$ $Re(z_n) = Re(z_{n-1})^2 - Im(z_{n-1})^2 + Re(c)$ $Im(z_n) = 2Re(z_{n-1})Im(z_{n-1}) + Im(c)$
successione di Mandelbrot espressa in termini delle sue componenti reali ed immaginarie

dove Re ed Im sono delle funzioni che restituiscono la parte reale e immaginaria rispettivamente di un numero complesso passato per argomento e che, si può facilmente verificare, sono funzioni lineari, ossia passano per le somme di numeri complessi e anche per i prodotti di numeri complessi per una costante reale.

$$Re(a + bi) = a \ Im(a + bi) = b$$

Le formule racchiuse nel box sono quelle che sono state applicate direttamente nel programma.

Come dicevo prima per fortuna esiste un teorema che dimostra che l'insieme di Mandelbrot è tutto compreso all'interno di un cerchio centrato sull'origine del campo complesso (0,0) di raggio 2. In termini analitici questo equivale ad affermare che la successione di Mandelbrot non può che divergere in un punto c distante più di 2 dall'origine o che se converge converge sempre a punti la cui distanza dall'origine risulta minore o uguale a 2 (e lo stesso nel caso in cui sia indeterminata). Questo risultato è di notevole importanza, poiché definisce i limiti grafici dell'insieme; nel programma precedente quando il punto c preso in considerazione è al di fuori del cerchio c, ossia $|c| > 2$ invece di dover perdere tempo ad iterare per calcolare la ridotta della successione di Mandelbrot, possiamo semplicemente concludere che quel punto non appartiene all'insieme di Mandelbrot.

Ma c'è di più. Supponete di aver iniziato l'iterazione per il calcolo degli elementi della successione per un punto c interno al cerchio ($|c| \leq 2$). Se ad un certo n capita che $|z_n| > 2$, si può concludere che la successione diverge in quel punto ed evitare persino di effettuare MAXITER iterazioni. Infatti la sottosuccessione formata da tutti gli elementi da questo n in poi è identica ad una successione di Mandelbrot che parte da $c = z_n$ e siccome $|z_n| > 2$, si ha che $|c| > 2$, quindi la sottosuccessione di z_n diverge. Allora diverge anche la successione z_n. Tutto questo è dimostrato in base alla definizione di divergenza e al teorema precedente (che in questa sede non abbiamo dimostrato).

Dopo questi risultati, decidiamo di modificare il nostro algoritmo approssimante in senso migliore in questo modo: per ogni punto di cui vogliamo stabilire se appartiene all'insieme di Mandelbrot o no, iteriamo per calcolare un massimo di MAXITER primi elementi della successione di Mandelbrot in quel punto, fermandoci prima se troviamo che $|z_n| > 2$ e concludendo in tal caso che il punto non appartiene all'insieme. Se invece dopo MAXITER elementi non è mai successo che $|z_n| > 2$, assumiamo che il punto appartenga all'insieme (quest'ultima è una congettura, che sarà comunque più precisa quanto più il valore di MAXITER è elevato). In termini algoritmici si tratta di un ciclo molto semplice:

```
per ogni c=(cr,ci):
member=TRUE;
iter=0;
zr=cr; zi=ci;
for (;;)
{
   zr2=zr*zr;
   zi2=zi*zi;
   if (zr2 + zi2 > 2)
   {
      member=FALSE;
      break;
   }
   elseif (++iter = MAXITER)
      break;
   zi=2*zr*zi+ci;
   zr=zr2-zi2+cr;
}
```

Si noti che quando $|c| > 2$, poiché partiamo dal secondo elemento della successione che è sempre c, si ha subito che $zr2 + zi2 > 2$ e quindi si conclude subito che il punto al di fuori del cerchio non appartiene all'insieme, come stabilito dal teorema fondamentale. Si noti inoltre che conviene calcolare prima zi e poi zr per evitare l'uso di una variabile temporanea e che la verifica che $|z_n| > 2$ è stata scritta come $zr^2 + zi^2 > 4$ anziché $\sqrt{zr^2 + zi^2} > 2$, per risparmiarsi l'uso della radice quadrata (che costituisce una significativa ottimizzazione pratica del programma). Le variabili $zr2$ e $zi2$ non sono strettamente necessarie e sono state introdotte per non ripetere il calcolo dei quadrati di zr e zi nella condizione di divergenza e nel calcolo dell'elemento successivo.

Naturalmente piuttosto che calcolare il valore della booleana member vogliamo disegnare. Possiamo creare un'immagine con sfondo bianco e disegneremo in nero solo i punti che troviamo essere molto probabilmente appartenenti all'insieme. Dobbiamo anche dettagliare l'istruzione *"per ogni* $c = (cr, ci)$*"*. Chiederemo all'utente tramite form html quale dimensione desidera abbia l'immagine da generare (indichiamo la larghezza in pixel con `X` e la lunghezza, pure in pixel con `Y`). Gli permetteremo poi di settare il valore di MAXITER desiderato e gli estremi dell'intervallo in cui far variare i valori `cr` e `ci`. Questi ultimi non varieranno in modo continuo, ma piuttosto con dei passi che possiamo calcolare una volta per tutte facendo:

$$\frac{crh-crl}{X} = stepr,\ \frac{cih-cil}{Y} = stepi$$

dove $[crl, crh]$ è l'intervallo di variazione della parte reale di c (ad es. $[-2, +2]$) e $[cil, cih]$ quello della parte immaginaria (ad. es. $[-2, +2]$).

A questo punto possiamo effettuare la scalatura dell'intervallo rettangolare definito dai due punti $[crl, cil]$, $[crh, cih]$ sull'intera immagine di dimensione $X * Y$ in due modi. Il primo consiste nell'iterare sui valori decimali di *cr* e *ci* e ricavare le coordinate del punto da plottare x, y di volta in volta, come fatto qui, oppure contemporaneamente iterare anche sui valori interi x e y:

```
(sono scelti dall'utente i valori di crl,crh, cil,cih, maxiter, X,Y)
stepr=(crh-crl)/X;
stepi=(cih-cil)/Y;
for (cr=crl; cr<=crh; cr+=stepr)
   for (ci=cil; ci<=cih; ci+=stepi)
   {
iter=0;
zr=cr; zi=ci;
for (;;)
{
   zr2=zr*zr;
   zi2=zi*zi;
   if (zr2 + zi2 > 4)
      break;
   elseif (++iter = maxiter)
   {
      plotta il punto ((cr-crl)/stepr, (ci-cil)/stepi);
      break;
   }
   zi=2*zr*zi+ci;
   zr=zr2-zi2+cr;
}
   }
```

Il secondo modo che io trovo più semplice e che ho prescelto itera invece sui valori interi di x, y e contemporaneamente sui valori di *ci* e *cr*:

```
(sono scelti dall'utente i valori di crl,crh, cil,cih, maxiter, X,Y)
stepr=(crh-crl)/X;
```

```
stepi=(cih-cil)/Y;
cr=crl; ci=cil;
for (y=0; y<Y; y++, ci+=stepi)
{
   for (x=0; x<X; x++, cr+=stepr)
   {
iter=0;
zr=cr; zi=ci;
for (;;)
{
   zr2=zr*zr;
   zi2=zi*zi;
   if (zr2 + zi2 > 4)
      break;
   elseif (++iter = maxiter)
   {
      plotta il punto (x,y);
      break;
   }
   zi=2*zr*zi+ci;
   zr=zr2-zi2+cr;
}
   }
}
```

Passiamo alla pratica, ecco una prima semplice versione di MSET generator:

```
mandel.php
----------
<?php
   require 'chronometer.inc';

   // limit here the maximum execution time (in seconds)
   // if seconds is set to zero, no time limit is imposed
   set_time_limit(5 * 60); // max 5 minutes

   // set integer parameter types
   settype($X, integer);
   settype($Y, integer);
   settype($maxiter, integer);

   // set default value
   if (!$X) $X=100;
   if (!$Y) $Y=100;
   if (!$maxiter) $maxiter=50;
   if (ltrim($crl)=='') $crl=-1.5;
   if (ltrim($cil)=='') $cil=-1;
   if (ltrim($crh)=='') $crh=0.5;
   if (ltrim($cih)=='') $cih=1;

   // set double parameter types
   settype($crl, double);
   settype($cil, double);
   settype($crh, double);
   settype($cih, double);
?>
<html>
<head><title>MSET generator</title></head>
<body>
<h1>MSET generator</h1>
```

```
<form>
X <input type=text name=X size=5 value=<?php echo $X; ?>>
Y <input type=text name=Y size=5 value=<?php echo $Y; ?>><br>
crl <input type=text name=crl size=8 value=<?php echo $crl; ?>>
crh <input type=text name=crh size=8 value=<?php echo $crh; ?>><br>
cil <input type=text name=cil size=8 value=<?php echo $cil; ?>>
cih <input type=text name=cih size=8 value=<?php echo $cih; ?>><br>
maxiter <input type=text name=maxiter size=8 value=<?php echo $maxiter; ?>><br>
<input type=submit value=" Draw ">
</form>
<?php

$fname="${X}_${Y}_${crl}_${crh}_${cil}_${cih}_$maxiter.png";

if (!file_exists ($fname))
{
   $im=ImageCreate($X,$Y);
   $white=ImageColorAllocate($im, 255,255,255);
   $black=ImageColorAllocate($im, 0,0,0);

   $chrono=new chronometer;
   $chrono->start();

   $stepr=($crh-$crl)/$X;
   $stepi=($cih-$cil)/$Y;
   $ci=$cil;
   for ($y=0; $y<$Y; $y++, $ci+=$stepi)
   {
      $cr=$crl;
      for ($x=0; $x<$X; $x++, $cr+=$stepr)
      {
 $iter=0;
 $zr=$cr; $zi=$ci;
 for (;;)
 {
     $zr2=$zr*$zr;
     $zi2=$zi*$zi;
     if ($zr2 + $zi2 > 4)
break;
     elseif (++$iter == $maxiter)
     {
ImageSetPixel($im, $x,$y, $black);
break;
     }
     $zi=2*$zr*$zi+$ci;
     $zr=$zr2-$zi2+$cr;
 }
      }
   }

   echo '<small>Drawing took ', $chrono->stop(), ' seconds.</small><br>';

   ImagePng($im, $fname);
}

echo "<img src=\"$fname\" width=".$X.' height='.$Y.'>';
?>

<h6>copyright 2001
<a href="mailto:antonio.b@infinito.it">
  Antonio Bonifati
</a></h6>
</body>
</html>
```

Se lanciate il programma precedente e provate a disegnare una grossa immagine con grande precisione (un valore di maxiter elevato), noterete che occorre tantissimo tempo. Se allo script occorrono più di 30 secondi per portare a termine il suo lavoro, otterrete un messaggio di errore del tipo:

```
Fatal error: Maximum execution time of 300 seconds exceeded in
/usr/local/apache/htdocs/tests/mandel.php on line 71
```

Infatti di default dopo 30 secondi di esecuzione di uno script PHP esso va in timeout e viene chiuso. Questa è una misura per impedire che uno script dia troppo carico sul server; il valore di default di 30 secondi può essere aumentato a piacimento, tramite la funzione:

void **set_time_limit** (int *seconds*)

e lo stato di timeout può essere anche disabilitato ponendo come argomento di questa funzione il valore 0 (in tal caso uno script può girare quindi per tutto il tempo che vuole). Chiamando questa funzione prima dello scadere del tempo, il tempo di esecuzione viene aumentato della quantità *seconds*. Se si chiama questa funzione all'inizio di un programma, il tempo di esecuzione massimo è praticamente uguale a quello stabilito da questa funzione.

Anche il valore di default di 30 secondi può essere cambiato ridefinendo la costante:

```
max_execution_time integer
```

nel file di configurazione[28] col nuovo valore.

Tuttavia quando PHP gira in *"safe mode"*[29] la costante `safe_mode` è `On`) la funzione **set_time_limit** non ha alcun effetto e solo l'amministratore tramite una modifica del valore di `max_execution_time` in `php.ini` può cambiare il massimo tempo di esecuzione[30].

Una ottimizzazione che ho già inserito nel programma consiste nel fatto che tutte le approssimazioni dell'insieme di Mandelbrot richieste vengono it "cached" dal lato server; in pratica a tutte le immagini generate viene dato un nome unico, composto dai parametri matematici separati da `_` (underscore) e dall'estensione `.png`. Questo significa che la directory che contiene lo script `mandel.php` deve avere i permessi di scrittura da parte dell'utente bf nobody e che si riempirà piuttosto presto di tanti piccoli e grandi file `png`. Visto comunque che gli hard-disk costano sempre meno, penso che possiamo permettercelo e comunque questa è una applicazione piuttosto particolare.

Una nota dolente è che PHP, essendo un linguaggio interpretato, è molto più lento del C (anche se più semplice): se questo programma fosse riscritto in C sarebbe parecchio più veloce (un esperto programmatore in assembly poi potrebbe guadagnare ancora qualcosa in più in velocità su una determinata piattaforma). Se avete bisogno di velocità in genere non resta quindi che ricorrere al C (ed eventualmente sostituire i calcoli con numeri decimali con numeri interi che rappresentano i numeri decimali nella notazione a punto fisso, in modo da guadagnare velocità, anche se si perde un po' di precisione). In ogni caso la stesura PHP non è stata inutile: infatti convertire un programma PHP in C qualora si senta bisogno di maggior

[28]Usualmente posto in /usr/local/lib/php.ini

[29]nel file `php.ini`

[30]Manuale di PHP, *"Chapter 21. Connection handling"* e *"XLVII. PHP options & information"*.

velocità è sempre più facile che non dover progettare un programma dall'inizio in C o PHP ;)

Il programma in C infatti può utilizzare la stessa libreria GD per generare immagini PNG, mentre per usare l'interfaccia CGI, potete utilizzare una delle tante librerie CGI per il C.

I limiti di PHP nei confronti del C sono soprattutto quindi nella minore velocità di esecuzione. I limiti di PHP nei confronti di Java stanno invece nel fatto che Java offre una interattività molto maggiore di PHP. La scelta migliore tra C, PHP e Java dipende ovviamente dall'applicazione che si vuole effettuare e se si intende spostare l'elaborazione più dal lato server o di più dal lato client.

A questo punto ci restano da implementare solo gli schemi di colorizzazione dell'insieme di Mandelbrot. Finora avete visto l'insieme di Mandelbrot in monocromatico, che è piuttosto bruttino, anche se è comunque l'oggetto matematico vero e proprio; come già accennato tuttavia i colori non sono altro che fronzoli aggiunti alla rappresentazione e difatti per aggiungere il colore basta aggiungere un po' di codice al nostro algoritmo, mentre la logica dell'algoritmo rimane immutata.

L'idea generale è di basare la scelta del colore di un punto (o pixel corrispondente) sul potenziale o altezza di quel punto, essendo il potenziale di un punto c definito come il numero di iterazioni (`$iter` nell'algoritmo visto prima) che sono state effettuate prima che si sia usciti dal ciclo a causa del fatto che $|z| > 2$. Ci si inventa poi una funzione che fa corrispondere alle varie altezze determinati colori, che saranno ovviamente comprese tra 0 e `$maxiter` nell'algoritmo precedente. Non c'è quindi nessun significato matematico o particolarmente profondo nella scelta dello schema di colorazione, tranne che l'effetto estetico che si può ottenere con alcuni schemi è notevole.

Si può ad es. lasciare che quando `$iter==$maxiter`, ossia consideriamo il punto come appartenente all'insieme di Mandelbrot venga disegnato col colore nero come nel caso della rappresentazione monocromatica. Per tutti gli altri punti, che consideriamo come non appartenenti all'insieme di Mandelbrot, possiamo plottarli in nero come quelli appartenenti se la differenza tra `$maxiter` e `$iter` risulta un numero pari (cioè divisibile per 2), non plottarli in caso contrario. Questa rappresentazione si ottiene semplicemente sostituendo nel programma precedente lo spezzone di codice:

```
if ($zr2 + $zi2 > 4)
   break;
elseif (++$iter == $maxiter)
{
   ImageSetPixel($im, $x,$y, $black);
   break;
}
```

con:

```
             if ($zr2 + $zi2 > 4)
             {
                if (($maxiter-$iter)%2)
                   ImageSetPixel($im, $x,$y, $black);
                break;
             }
             elseif (++$iter == $maxiter)
             {
                ImageSetPixel($im, $x,$y, $black);
                break;
             }
```

Questo schema di *"colorazione"* è detto *"monochrome banding"* o *"delle bande monocromatiche"*, proprio perché con questo algoritmo l'insieme di mandelbrot vero e proprio viene circondato da suggestive bande di colore bianco e nero alternati.

Infine un ultimo schema, del tutto arbitrario pure questo, per una rappresentazione con i colori gialli e verdi (supponendo per semplicità che maxiter valga 50).

Cambiate le istruzioni

```
$im=ImageCreate($X,$Y);
$white=ImageColorAllocate($im, 255,255,255);
$black=ImageColorAllocate($im, 0,0,0);
```

con:

```
$im=ImageCreate($X,$Y);
// setta una palette con:

// 16 verdi sempre più intensi
    for ($i=0; $i<=15; $i++)
   ImageColorAllocate($im, 0,$i*17,0);
// 34 verdi tendenti al giallo
for ($i=16; $i<=49; $i++)
   ImageColorAllocate($im, $i*7,255,0);
```

cambiate poi il solito spezzone

```
             if ($zr2 + $zi2 > 4)
                break;
             elseif (++$iter == $maxiter)
             {
                ImageSetPixel($im, $x,$y, $black);
                break;
             }
```

con:

```
if ($zr2 + $zi2 > 4)
{
   ImageSetPixel($im, $x,$y, $iter);
   break;
}
elseif (++$iter == $maxiter)
{
   ImageSetPixel($im, $x,$y, 0);
   break;
}
```

provate a plottare un'immagine 300x300 tra -2,1 e -1.5,1.5 con maxiter=50 (occorre circa un minuto).

Ho fatto questo esempio per mostrare come in prossimità dei limiti dell'insieme di Mandelbrot (ossia dei confini di questo insieme), le altezze dei punti (ossia il numero di iterazioni sufficienti per stabilire che il punto non appartiene all'insieme) tendono ad aumentare rapidamente. Intuitivamente, avvicinandosi ad un punto membro dall'esterno, l'altezza dei punti sulla traiettoria che si percorre tende ad aumentare piuttosto rapidamente. Per conseguenza di ciò nel grafico precedente non si vede praticamente giallo (i punti in giallo in prossimità dei limiti sono pochissimi. Per aumentare la quantità di giallo visibile nei punti, possiamo prendere il logaritmo dell'altezza prima di applicare lo schema di colorazione, oppure utilizzare una palette con variazioni di colori più lente negli indici più alti, in modo che segua meglio i gradienti delle altezze ad es. così:

```
for ($i=0; $i<=15; $i++)
   ImageColorAllocate($im, 0,$i*17,0);
for ($i=0; $i<=15; $i++)
   ImageColorAllocate($im, $i*17,255,0);
for ($i=0; $i<=17; $i++)
   ImageColorAllocate($im, 255,255,0);
```

Ci sono altre ottimizzazioni e schemi di colore applicabili all'insieme di Mandelbrot che non tratterò (siamo già andati piuttosto fuori dall'argomento di questo tutorial, che è la programmazione CGI in PHP). Se siete interessati vi rimando alla letteratura presente in rete.

Come contattare l'autore

Se volete conttarmi per offerte di lavoro o altro ecco i miei dati.

Antonio Bonifati sviluppa pagine web statiche e dinamiche e basi di dati sotto server di tipo UNIX. Collabora gratuitamente a progetti culturali e no-profit sul web. Può essere contatto all'indirizzo:

antonio.b@infinito.it
oppure
kiss@clubnautilus.net

sito web:
http://go.to/ninuzzo
http://king.rett.polimi.it/~ant
posta delle lumache :)
Via Ernesto Moneta, 11
87012 Castrovillari (CS)
tel. 0981/26247