Capitolo 41

Introduzione a PHP

PHP[1] sta per *hypertext preprocessor* e, originariamente, per *personal home page*. Si tratta in pratica di un interprete di un linguaggio che ha lo stesso nome, attraverso il quale si genera al volo una pagina ipertestuale (di norma HTML). Pertanto, il linguaggio PHP si usa per realizzare degli script, la cui interpretazione avviene presso un servente HTTP-CGI, dove si associa l'estensione del file (di solito è '`.php`') all'esecuzione dell'interprete PHP, in qualità di programma CGI. Per esempio, nella configurazione del servente HTTP, potrebbe apparire una direttiva simile a quella seguente che si riferisce precisamente al caso di Mathopd (sezione 40.2):

```
    Control {
        Alias /
        Location /var/www
        External {
            /usr/bin/php-cgi { .php }
        }
    }
```

Qui si intende dire al servente HTTP che, nel caso venga richiesto

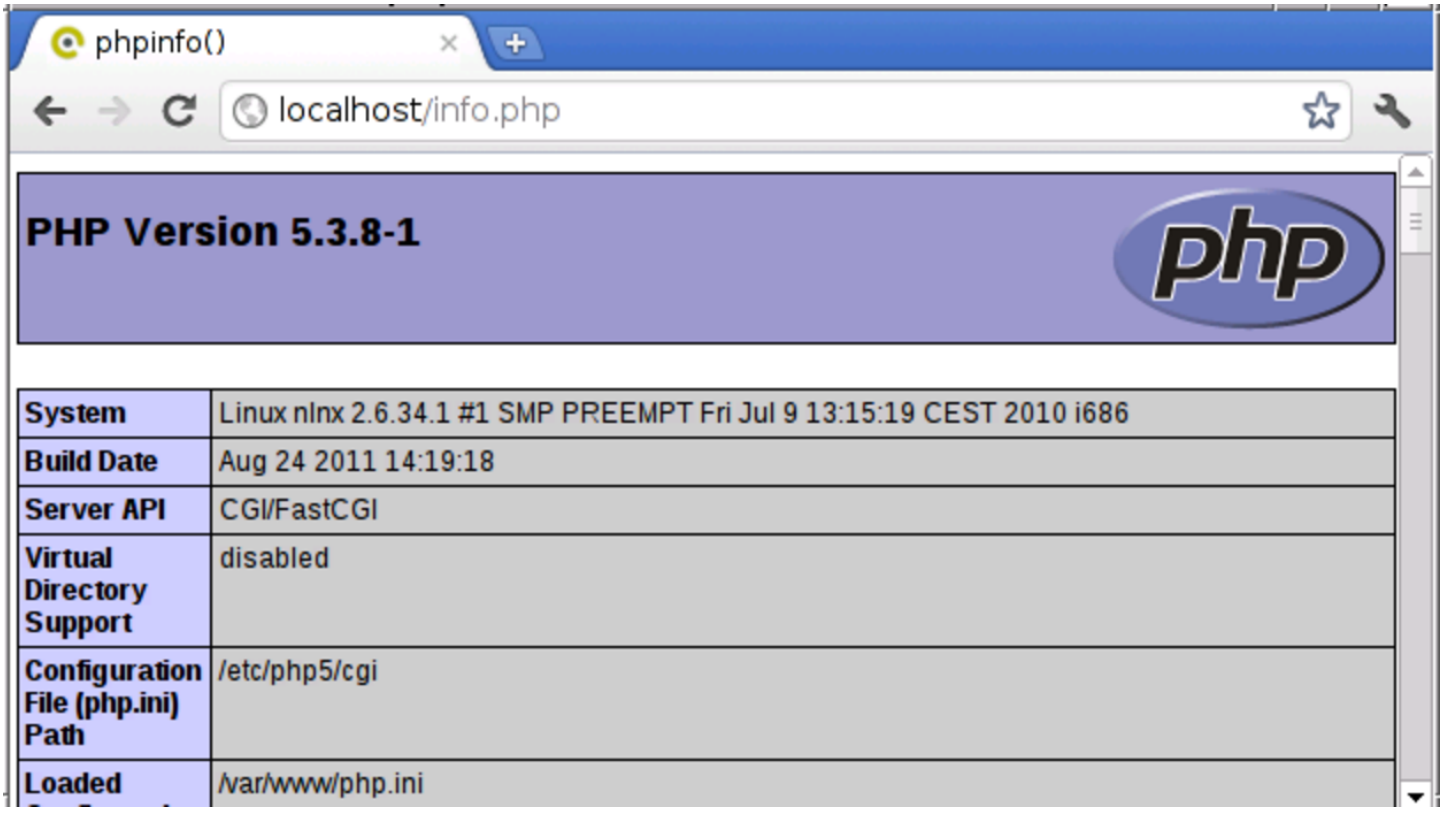
di accedere a un file con estensione '`.php`', deve utilizzare il programma '`/usr/bin/php-cgi`' per interpretarlo, restituendo poi il risultato come se fosse il contenuto del file richiesto originariamente.

Per poter utilizzare o iniziare a studiare il linguaggio PHP, occorre disporre di un servente HTTP, predisposto in modo tale da poter interpretare i file PHP. Per verificare tale funzionalità, è sufficiente predisporre un file come quello seguente, il cui nome potrebbe essere '`info.php`':

```
<?php
phpinfo();
?>
```

Accedendo attraverso un navigatore ipertestuale al file, tramite il servente HTTP, si dovrebbe ottenere un elenco delle funzionalità disponibili e della configurazione attuale dell'interprete PHP:

Figura 41.3. Esito dell'interpretazione del file '`info.php`' di esempio.



L'interprete PHP viene configurato con un file di configurazione generale, il quale potrebbe corrispondere a '`/etc/php/cgi/php.ini`', e da file di configurazione particolari, relativi soltanto ai file PHP che si trovano nella stessa directory. Le direttive di questo file di configurazione sono molto semplici e consistono nell'assegnamento di un valore a delle variabili di configurazione prestabilite, come nell'esempio seguente:

```
; Questo è un commento
magic_quotes_gpc = Off
; Le direttive seguenti evidenziano tutti gli errori in fase
; di esecuzione degli script PHP.
error_reporting = E_ALL | E_NOTICE | E_STRICT
display_errors = On
```

41.1 Delimitazione del codice PHP

«

Il codice PHP contenuto in un file, deve essere delimitato, attraverso un marcatore di apertura e uno di chiusura:

```
<?php
...
?>
```

Il file che contiene il codice PHP potrebbe contenere più di un blocco di codice delimitato in questo modo. In tal caso, di solito il file è scritto secondo il linguaggio HTML, dove, nelle sole zone delimitate, si utilizza del codice PHP per rendere dinamica la composizione complessiva:

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Data e orario</title>
  </head>
  <body>
    <p>Data e orario:
    <em><?php echo (date ("r")); ?></em></p>
    <p>Il tuo indirizzo IP è:
    <em><?php echo ($_SERVER["REMOTE_ADDR"]); ?></em></p>
  </body>
</html>
```

L'esempio appena apparso mostra due piccole inserzioni di codice PHP, con le quali si genera prima una stringa contenente la data e l'orario, poi l'indirizzo IP del nodo cliente.

Tuttavia, mescolare codice PHP dentro codice di tipo differente, potrebbe risultare in un lavoro confuso e disordinato. La stessa cosa apparsa sopra avrebbe potuto essere scritta, in modo più coerente, come segue:

```
<?php
echo ("<!doctype html>\n");
echo ("<html>\n");
echo ("  <head>\n");
echo ("    <meta charset=\"UTF-8\">\n");
echo ("    <title>Data e orario</title>\n");
echo ("  </head>\n");
echo ("  <body>\n");
echo ("    <p>Data e orario: <em>"
      .date ("r")."</em></p>\n");
echo ("    <p>Il tuo indirizzo IP è: <em>"
      .$_SERVER["REMOTE_ADDR"]."</em></p>\n");
echo ("  </body>\n");
echo ("</html>\n");
?>
```

In questo capitolo introduttivo a PHP, si usa sempre solo questa seconda modalità di codifica.

41.2 Struttura fondamentale del linguaggio

«

Il linguaggio PHP è strutturato nello stesso modo del linguaggio C (capitolo 66) e del linguaggio Perl (capitolo 24); pertanto è uguale la forma dei commenti, il modo di terminare le istruzioni con il punto e virgola e il modo di raggrupparle con le parentesi graffe:

```
/* commento
   [...]
*/
```

```
// commento
```

```
istruzione ;
```

```
{ istruzione ;  istruzione ;  istruzione ; }
```

Dal momento che il linguaggio PHP non utilizza direttive del preprocessore, anche il cancelletto (‘**#**’) può essere usato per segnalare dei commenti, come avviene nel linguaggio Perl:

```
# commento
```

I nomi delle variabili e delle funzioni seguono regole analoghe a quelle del linguaggio C e del linguaggio Perl, per cui si usano lettere, cifre numeriche e trattino basso, con il vincolo di iniziare con una lettera (il trattino basso iniziale è riservato per nomi di sistema). Tut-

tavia, rispetto al linguaggio C, solo i nomi di variabili si distinguono anche in base alla scelta di lettere maiuscole o minuscole, mentre non è così per i nomi di funzione; inoltre, i nomi delle variabili sono sempre contrassegnati dal prefisso '**$**', come avviene nel linguaggio Perl.

Quindi, i nomi ***mamma_mia()*** e ***Mamma_Mia()***, rappresentano indifferentemente la stessa funzione, mentre le variabili ***$mia*** e ***$Mia*** sono entità distinte.

41.3 Analisi sintattica

Il linguaggio PHP viene interpretato in qualità di script, pertanto non richiede una fase di compilazione per la produzione di un file in linguaggio macchina. Tuttavia, è opportuna un'analisi sintattica preventiva. Di norma si ottiene con il comando '**php**', con l'opzione '**-l**':

```
$ php -l file.php[Invio]
```

In questo esempio si analizza il file '`file.php`'. Nella migliore delle ipotesi si ottiene questo messaggio:

```
No syntax errors detected in file.php
```

Tuttavia, ci sono errori che si possono rivelare solo in fase di utilizzo. Per questi è necessario un controllo preliminare, attivando opportunamente alcune opzioni di configurazione dell'interprete PHP. L'estratto seguente della configurazione di un file '`php.ini`' mostra l'uso di tali direttive per mettere in evidenza ogni tipo di errore e di avvertimento in fase di esecuzione degli script:

```
error_reporting = E_ALL | E_NOTICE | E_STRICT
display_errors = On
```

41.4 Variabili e costanti

Le variabili per il linguaggio PHP hanno il tipo che deriva da ciò che viene loro assegnato. Per esempio, una variabile è di tipo numerico se le si assegna un valore numerico, ma può trasformarsi in una stringa se successivamente le si assegna un dato di questo tipo. In pratica, con il PHP non ci si deve preoccupare di definire il tipo delle variabili.

41.4.1 Valore nullo

Il PHP definisce il valore '**NULL**', pari a un nulla, non meglio precisato, che può essere uguale solo a un altro '**NULL**'. Una variabile corrisponde a '**NULL**' se le è stato assegnato tale valore, oppure se non è ancora stata definita.

```
$var0 = NULL;
```

41.4.2 Variabili e costanti logiche (booleane)

Una variabile è di tipo logico se gli si assegna un valore logico, *Vero* o *Falso*, espresso dalle costanti '**TRUE**' e '**FALSE**':

```
$var1 = TRUE;
$var2 = FALSE;
```

Una variabile logica può servire come condizione, per esempio in una struttura condizionale:

```
if ($var1)
  {
    echo ("<p>La variabile contiene il valore VERO!</p>\n");
  }
```

Tuttavia, ai fini della valutazione in qualità di variabile logica, qualunque altro tipo di variabile può essere usato come condizione, nel qual caso si considera che un valore «nullo», inteso però in base al contesto, sia pari a «FALSO», mentre qualunque altra cosa sia pari a «VERO».

41.4.3 Variabili e costanti numeriche

Una variabile è di tipo numerico intero quando le si assegna un valore numerico intero; è in virgola mobile se le si assegna un valore numerico con virgola. La costante si scrive semplicemente, usando il punto come separatore decimale, se serve; inoltre, si mette il segno meno ('**-**') prima delle cifre numeriche, se si tratta di un valore negativo. Non c'è bisogno di specificare il rango del valore.

```
$var3 = 12345;
$var4 = 678.901;
$var5 = -234.56;
```

Si possono utilizzare costanti numeriche intere in base sedici e in base otto, come avviene nel linguaggio C:

```
$var6 = 0x123AF;          // valore esadecimale
$var7 = 01237;            // valore ottale
```

41.4.4 Stringhe

Le stringhe vengono trattate dal PHP come se fossero dei valori scalari (al pari dei valori numerici). Una variabile è di tipo stringa se le si associa una stringa:

```
$var8 = 'Ciao a tutti';
$var9 = "$var8, ma proprio a tutti";
```

Le stringhe costanti devono essere delimitate. Per questo si possono usare, a scelta, gli apici doppi o quelli singoli; tuttavia il loro comportamento è differente, come si intuisce dall'esempio.

41.4.5 Stringhe delimitate da apici singoli

Le stringhe delimitate da apici singoli sono stringhe letterali, nel senso che ciò che contengono viene letto per quello che è, con due sole eccezioni, '`\'`' e '`\\`' che vengono rese come se fossero, rispettivamente, l'apice singolo e la barra obliqua inversa. Si osservi l'esempio:

```
$var10 = 'L\'apostrofo e la barra obliqua inversa (\\).';
echo ("<p>");
echo ($var10);
echo ("</p>");
```

In questo caso, quando viene visualizzato il contenuto della variabile ***$var10***, si ottiene il testo:

```
L'apostrofo e la barra obliqua inversa (\).
```

È evidente che sia necessario un modo per rappresentare l'apostrofo, all'interno di una stringa delimita da apici singoli, senza che ciò interrompa la stringa stessa. Poi, dato che per questo si usa una sequenza composta con la barra obliqua inversa, quando si vuole rappresentare la barra obliqua, senza ambiguità, diventa necessario indicarla due volte. Nell'esempio precedente non sarebbe necessario farlo, perché dopo la barra obliqua inversa non c'è un apostrofo, mentre in quello successivo diventa indispensabile:

```
$var11 = 'L\'apostrofo si inserisce nelle stringhe ';
$var12 = 'letterali con la sequenza \\'.';
```

```
echo ("<p>");
echo ($var11);
echo ($var12);
echo ("</p>");
```

Ecco il risultato del nuovo esempio:

```
L'apostrofo si inserisce nelle stringhe letterali ↩
↪con la sequenza \'.
```

41.4.6 Stringhe delimitate da apici doppi

Le stringhe delimitate da apici doppi sono soggette a un'interpretazione; in modo particolare vengono riconosciute le variabili ed espanse in forma di stringhe:

```
$var13 = 123;
$var14 = "La variabile \$var13 contiene il valore $var13.";
echo ("<p>");
echo ($var14);
echo ("</p>");
```

Ecco il risultato:

```
La variabile $var13 contiene il valore 123.
```

Oltre a questo, è possibile inserire alcuni codici speciali, tra cui il più importante è rappresentato da '**\n**', con il quale si ottiene di inserire un'interruzione di riga:

```
$var15 = "Prima riga,\nseconda riga.";
echo ("<pre>");
echo ($var15);
echo ("</pre>");
```

Ecco il risultato:

```
Prima riga,
seconda riga.
```

Si comprende che l'uso della barra obliqua inversa cambia con le stringhe delimitate da apici doppi, arricchendosi di qualche nuovo codice:

Sequenza	Risultato
`\\`	Barra obliqua inversa, singola: '****'.
`\"`	Apice doppio: '**"**'.
`\$`	Dollaro: '**$**'.
`\n`	Interruzione di riga, corrispondente al codice <*LF*>, pari al valore $0A_{16}$.
`\r`	Codice <*CR*>, pari al valore $0D_{16}$.
`\t`	Codice di tabulazione orizzontale, <*HT*>, pari al valore 09_{16}.
`\v`	Codice di tabulazione verticale, <*VT*>, pari al valore $0B_{16}$.
`\e`	Codice <*ESC*>, pari al valore $1B_{16}$.
`\f`	Codice <*FF*>, pari al valore $0C_{16}$.

41.4.7 Cast

«

Il cast, ovvero la trasformazione esplicita del tipo di una variabile scalare, è previsto anche nel linguaggio PHP, principalmente come ausilio a una programmazione ordinata e chiara, anche se ci posso-

no essere situazioni in cui la conversione esplicita è necessaria. Si osservi l'esempio seguente:

```
$a = "15.5 anni";
$b = $a;
```

In questo caso, entrambe le variabili (***$a*** e ***$b***) sono stringhe e contengono ciò che si vede: «15.5 anni». In PHP è ammessa anche la conversione da stringa a valore numerico:

```
$a = "15.5 anni";
$b = (int) $a;
```

In questo caso, la variabile ***$b*** diventa di tipo intero, in quanto contiene il valore 15, perdendo il resto delle informazioni contenute nella stringa originale.

```
$a = "15.5 anni";
$b = (float) $a;
```

Qui invece la variabile ***$b*** ottiene il valore 15,5, in virgola mobile.

Va osservato che in PHP le variabili scalari non hanno un tipo fisso, quindi, una variabile che prima è di tipo stringa, può poi diventare di tipo numerico, per il solo fatto che gli si assegna un dato nuovo di tale tipo. Quindi, quando si assegna un valore a una variabile, non avviene un cast implicito, ma al massimo una trasformazione del tipo della variabile ricevente.

```
$a = "15.5 anni";
$a = 15;
$a = 15.5;
```

Questo esempio ulteriore, serva a capire che la variabile ***$a***, in momenti diversi, si trasforma da stringa, a valore intero, a valore in virgola mobile, mano a mano che accoglie dati diversi al suo interno.

Tabella 41.30. Cast disponibili nel linguaggio PHP.

Cast	Tipo di trasformazione ottenuta
`(int)` `(integer)`	Trasformazione in tipo intero. Nel caso il dato originario sia una stringa, si estrapolano le prime cifre numeriche, ammesso che la stringa inizi con cifre numeriche, altrimenti si ottiene il valore zero.

`(float)`
`(double)`
`(real)`
	Trasformazione in tipo a virgola mobile. La precisione di questo tipo di rappresentazione dipende dalla realizzazione e dal sistema in cui si trova a funzionare.

`(string)`	Trasformazione in stringa.
`(array)`	Trasformazione in un array. La conversione di un tipo scalare in un array, produce un array contenente un solo elemento, pari al valore scalare originale.
`(object)`	Trasformazione in oggetto. Si veda *http://php.net/manual/en/language.types.object.php* .
`(unset)`	Trasformazione nel valore '**`NULL`**'.

41.4.8 Array

Gli array del linguaggio PHP andrebbero considerati tutti come array associativi, nel senso che l'indice usato per accedere agli elementi può essere arbitrario, concretizzandosi in pratica in una chiave di ricerca. Per creare velocemente un array si può usare l'istruzione ***array()*** che si presenta come una funzione standard:

```
$arr = array (1, 1, 2, 3, 5, 8);
```

```
$arr = array (0 => 1, 1 => 1, 2 => 2,
              3 => 3, 4 => 5, 5 => 8);
```

I due esempi sono equivalenti, nel senso che producono lo stesso tipo di array. Nel primo caso si considera che gli elementi siano associati all'indice predefinito, costituito da un numero intero, dove il primo elemento ha indice zero e l'ultimo ha indice $n-1$, con n corrisponde alla quantità di elementi.

Il secondo esempio mostra la dichiarazione esplicita dell'indice di accesso. L'esempio successivo mostra che si può usare anche una stringa o qualunque altro valore «scalare» in qualità di indice per un array:

```
$arr = array ("aa" => 1, "bb" => 1, 123 => 2,
              123.5 => 3, -123 => 5, -123.5 => 8);
```

Per accedere a un elemento di un array, si usa la forma consueta, con la quale l'indice si colloca tra parentesi quadre:

```
$arr["aa"] += 2;
```

```
$i = "aa";
$arr[$i] += 2;
```

Per aggiungere un elemento a un array, è sufficiente fare riferimento a un indice che non sia ancora stato utilizzato:

```
$i = "zz";
$arr[$i] = 999;
```

Quando l'indice degli elementi è strutturato nel modo tradizionale (con indice da zero a $n-1$) e gli elementi sono ordinati effettivamente secondo l'indice, è possibile aggiungere un elemento attribuendo automaticamente l'indice successivo, nel modo seguente:

```
$arr[] = 111;
```

Anche per questa ragione, è frequente osservare nel codice PHP la creazione di array vuoti che poi vengono popolati in base alle necessità:

```
$arr = array ();
```

Dal momento che gli elementi di un array hanno comunque un ordine al loro interno, se l'indice usato non è più adeguato, è possibile attribuire agli elementi un nuovo indice ordinato:

```
$arr1 = array ("aa" => 1, "bb" => 1, 123 => 2,
               123.5 => 3, -123 => 5, -123.5 => 8);
$arr2 = array_values ($arr1);
```

In questo modo, l'array ***$arr2[]*** ottiene una copia degli elementi di ***$arr1[]***, ma con un indice ordinato, costituito da un intero a partire da zero, fino a $n-1$.

Gli elementi di un array possono essere, a loro volta, degli array. Ciò consente di produrre array a più dimensioni, a cui si accede con due o più indici. Pertanto, ***$arr[1][2]*** fa riferimento all'elemento con indice 2 di un array che a sua volta si colloca nell'elemento con indice 1 dell'array principale.

41.4.9 Stringhe trattate come array

Le stringhe possono essere trattate come se fossero array di byte. Ma occorre fare attenzione: si tratta proprio di array di byte, non di array di caratteri.

```
$str = "Perché?";
$str[0] = "p";
```

L'esempio mostra la dichiarazione della variabile ***$str*** contenente la stringa «Perché». Poi, il primo byte della stringa viene modificato, facendo sì che la variabile ***$str*** contenga complessivamente la stringa «perché» («p» minuscola). Si osservi però cosa accade qui:

```
$str = "Perché?";
$x = $str[5];
```

La variabile ***$x*** ottiene il sesto byte della stringa ***$str***. Tuttavia, dipende dalla codifica usata effettivamente nel sistema in cui funziona il PHP, a cosa corrisponda effettivamente tale byte. Per esempio, se è in uso la codifica UTF-8, quello che si ottiene è semplicemente il codice $C3_{16}$, perché complessivamente, la lettera «é» si rappresenta con due byte: $C3A9_{16}$.

41.4.10 La definizione di costanti

È possibile dichiarare delle costanti con l'ausilio della funzione ***define()***; tuttavia, ci sono circostanze in cui il risultato non è propriamente quello che ci si aspetterebbe: per evitare complicazioni è bene dichiarare tali costanti in una posizione che sia, anche formalmente, accessibile da tutto il programma:

```
define ("CIAO", "Ciao a tutti!");
echo ("<p>");
echo (CIAO);
```

```
echo ("</p>");
```

Si osserva che le costanti non hanno il prefisso '**$**' delle variabili; inoltre, va chiarito che la funzione ***define()*** può essere usata con un argomento aggiuntivo che però è sconsigliabile sfruttare.

41.5 Campo di azione delle variabili

Il PHP distingue tre tipi di campo di azione per le variabili: ***superglobali***, ***globali*** e ***locali***. Le variabili superglobali sono predefinite e si distinguono perché il loro nome inizia con il trattino basso e sono composte poi da lettere maiuscole. Per esempio, l'array ***$_GET[]*** serve a recepire i valori di una chiamata con il metodo GET del protocollo HTTP.

Le variabili superglobali sono accessibili in qualunque parte del programma PHP, senza distinzioni.

Le variabili globali sono quelle definite al di fuori delle funzioni, ma all'interno delle funzioni non sono visibili automaticamente: perché lo siano occorre ridichiararle espressamente in qualità di variabili globali. Esistono delle variabili globali predefinite, il cui utilizzo è però sconsigliato in favore della scelta di variabili superglobali che possono offrire le stesse informazioni. Eventualmente, va tenuto conto che le variabili globali predefinite possono essere trasmesse alle funzioni solo ridichiarando la loro natura di variabili globali nelle funzioni stesse.

Le variabili locali sono quelle definite all'interno delle funzioni e rimangono visibili solo nell'ambito della funzione che le contiene, senza trasmettersi alle funzioni che da lì potrebbero essere chiamate. Le variabili locali possono essere rese «statiche», nel senso che

conservino il loro valore fino alla prossima chiamata della stessa funzione.

```
$a = 10;
$b = 20;
function c ()
{
  global a;
  static d = 5;
  b = 0;
  e = 10;
  a++;
  b++;
  d++;
  e++;
  return (a+b+d+e);
}
```

L'esempio mostra la dichiarazione di due variabili globali, ***a*** e ***b***, a cui viene assegnato inizialmente un valore. Poi si vede la funzione ***c()***, la quale acquisisce la variabile globale ***a***, definisce la variabile statica ***d*** e le variabili locali ***b*** ed ***e***. All'interno della funzione, le tre variabili ***a***, ***d*** ed ***e***, vengono incrementate di una unità, poi di queste viene restituita la somma.

Quando questa funzione viene chiamata la prima volta, la variabile statica ***d*** ottiene il suo valore iniziale, pari a 5, ma poi, alle chiamate successive, tale variabile non viene più inizializzata e continua a conservare il valore ottenuto nella chiamata precedente. Dato che la funzione lo incrementa di una unità, alla chiamata successiva si trova ad avere inizialmente ***6***, poi ***7***,...

La variabile globale ***a*** viene recepita dalla funzione, con il valore

che possiede al momento della chiamata; poi il contenuto di questa variabile viene incrementato e tale modifica risulta anche al di fuori della funzione.

All'interno della funzione viene dichiarata la variabile locale ***b***, il cui nome coincide con quello di una variabile globale, la quale però viene ignorata all'interno della funzione. Pertanto, la modifica che viene apportata alla variabile locale ***b*** non si trasmette alla variabile globale che ha lo stesso nome.

Le variabili locali ***b*** ed ***e*** vengono formalmente distrutte al termine dell'esecuzione della funzione; pertanto, sono utili solo in quanto partecipano alla definizione del valore restituito dalla funzione ***c()***.

41.6 Riferimento a una variabile

Il linguaggio PHP si astrae notevolmente dalla realtà del linguaggio macchina che serve per pilotare la CPU. Pertanto, certe questioni che riguardano la gestione delle variabili e degli array, nel linguaggio PHP sono risolte in modo apparentemente semplice: per esempio, le stringhe sono gestite come se fossero valori scalari (mentre in realtà si sa che sono array di byte). Ciò comporta il fatto che il linguaggio gestisca in modo trasparente tutte le questioni relative ai puntatori delle variabili.

Il linguaggio PHP prevede l'operatore '**&**', per indicare che si intende fare riferimento a una variabile, ma va usato secondo le modalità previste e non esiste un operatore analogo di dereferenziazione, perché questa è implicita. Si osservi l'esempio seguente:

```
$a = 10;
$b = &$a;
```

Il secondo assegnamento dell'esempio, fa sì che la variabile ***$b*** sia un alias della variabile ***$a***, semplicemente. Così, se si assegna un valore diverso a ***$b*** questo cambiamento si ripercuote anche nell'altro alias.

L'unica situazione in cui può essere utile l'uso dei riferimenti alle variabili riguarda la chiamata delle funzioni, dove è possibile passare un parametro per riferimento ed è possibile restituire una variabile locale per riferimento (in tal caso la variabile locale non verrebbe distrutta alla conclusione del funzionamento della funzione).

41.7 Operatori ed espressioni

L'operatore è qualcosa che esegue un qualche tipo di funzione, su uno o più operandi, restituendo un valore. Gli operandi descritti di seguito sono quelli più comuni e importanti. Le espressioni sono formate spesso dalla valutazione di sottoespressioni (espressioni più piccole).

Tabella 41.45. Ordine di precedenza tra gli operatori principali previsti nel linguaggio PHP. Gli operatori sono raggruppati a livelli di priorità equivalente, partendo dall'alto con la priorità maggiore, scendendo progressivamente alla priorità minore. Le variabili ***a***, ***b*** e ***c*** rappresentano la collocazione delle sottoespressioni da considerare ed esprimono l'ordine di associatività: prima ***a***, poi ***b***, poi ***c***.

Operatori	Annotazioni
(***a***)	Le parentesi tonde usate per raggruppare una porzione di espressione hanno la precedenza su ogni altro operatore.

Operatori	Annotazioni
`[`*a*`]`	Le parentesi quadre che delimitano l'indice o la chiave di accesso a un elemento di un array.
`++`*a* `--`*a* *a*`++` *a*`--`	Incremento e decremento.
`~`*a* `-`*a* (*tipo*)	L'operatore '**–**' di questo livello è da intendersi come «unario», ovvero si riferisce al segno di quanto appare alla sua destra. Le parentesi tonde si riferiscono al cast.
`!`*a*	Negazione logica.
a`*`*b* *a*`/`*b* *a*`%`*b*	Moltiplicazione, divisione e resto della divisione intera.
a`+`*b* *a*`-`*b* *a*`.`*b*	Somma, sottrazione e concatenamento di stringhe.
a`<<`*b* *a*`>>`*b*	Scorrimento binario.
a`<`*b* *a*`<=`*b* *a*`>`*b* *a*`=>`*b*	Confronto.
a`==`*b* *a*`===`*b* *a*`!=`*b*	Confronto.
a`&`*b* `&`*a*	AND bit per bit e riferimento alla variabile.
a`^`*b*	XOR bit per bit.
a`|`*b*	OR bit per bit.
a`&&`*b*	AND nelle espressioni logiche.

Operatori	Annotazioni
a || *b*	OR nelle espressioni logiche.
a ? *b*1 : *b*2	Operatore condizionale.
b = *a* *b* += *a* *b* -= *a* *b* *= *a* *b* /= *a* *b* %= *a* *b* &= *a* *b* ^= *a* *b* |= *a* *b* <<= *a* *b* >>= *a* *b* .= *a*	Operatori di assegnamento.
a and *b*	AND logico.
a xor *b*	XOR logico.
a or *b*	OR logico.
a , *b*	Sequenza di espressioni (espressione multipla).

Tabella 41.46. Elenco degli operatori binari. Gli operatori devono riferirsi a valori interi.

Operatore e operandi	Descrizione
op1 & *op2*	AND bit per bit.
op1 | *op2*	OR bit per bit.
op1 ^ *op2*	XOR bit per bit (OR esclusivo).

Operatore e operandi	Descrizione
~op1	Complemento a uno, ovvero inversione binaria.
op1 << ***op2***	Scorrimento binario a sinistra, ottenuto come $op1 \cdot 2^{op2}$.
op1 >> ***op2***	Scorrimento binario a destra, ottenuto come $op1 \cdot 2^{-op2}$. In pratica si ottiene dividendo il valore di ***op1*** per due, ***op2*** volte.

Tabella 41.47. Elenco degli operatori di confronto.

Operatore e operandi	Descrizione
op1 == ***op2***	*Vero* se gli operandi si equivalgono.
op1 === ***op2***	*Vero* se gli operandi si equivalgono e sono anche dello stesso tipo.
op1 != ***op2*** ***op1*** <> ***op2***	*Vero* se gli operandi sono differenti.
op1 |== ***op2***	*Vero* se gli operandi sono differenti per contenuto o per tipo.
op1 < ***op2***	*Vero* se il primo operando è minore del secondo.
op1 > ***op2***	*Vero* se il primo operando è maggiore del secondo.
op1 <= ***op2***	*Vero* se il primo operando è minore o uguale al secondo.
op1 >= ***op2***	*Vero* se il primo operando è maggiore o uguale al secondo.

Tabella 41.48. Elenco degli operatori di incremento e di decremento.

Operatore e operandi	Descrizione
`++`***op***	Incrementa di un'unità l'operando prima che venga restituito il suo valore.
op`++`	Incrementa di un'unità l'operando dopo averne restituito il suo valore.
`--`***op***	Decrementa di un'unità l'operando prima che venga restituito il suo valore.
op`--`	Decrementa di un'unità l'operando dopo averne restituito il suo valore.

Tabella 41.49. Elenco degli operatori logici. Va osservato che gli operatori '**`and`**', '**`or`**' e '**`xor`**', hanno una precedenza molto bassa: in generale, sarebbe meglio evitare il loro utilizzo per evitare inutili confusioni.

Operatore e operandi	Descrizione
`!` ***op***	Inverte il risultato logico dell'operando.
op1 `&&` ***op2*** ***op1*** `and` ***op2***	Se il risultato del primo operando è *Falso* non valuta il secondo.
op1 `||` ***op2*** ***op1*** `or` ***op2***	Se il risultato del primo operando è *Vero* non valuta il secondo.
op1 `xor` ***op2***	Se uno dei due operandi dà un risultato pari a *Vero*, mentre l'altro dà il valore *Falso*, produce complessivamente un risultato pari a *Vero*.

Tabella 41.50. Concatenamento di stringhe. Il concatenamento può avvenire con valori di tipo diverso dalla stringa, i quali vengono convertiti contestualmente in stringhe.

Operatore e operandi	Descrizione
op1 `.` ***op2***	Concatena le stringhe ***op1*** e ***op2***.

Tabella 41.51. Operatori relativi agli array.

Operatore e operandi	Descrizione
op1 `+` ***op2***	Unisce l'array ***op1[]*** con l'array ***op2[]***.
op1 `==` ***op2***	Confronta i due array e restituisce *Vero* se questi hanno le stesse coppie chiave-valore (l'ordine degli elementi è però indifferente).
op1 `===` ***op2***	Confronta i due array e restituisce *Vero* se questi hanno le stesse coppie chiave-valore, nello stesso ordine e se corrispondono anche i tipi relativi.
op1 `!=` ***op2*** ***op1*** `<>` ***op2***	`!(`***op1*** `==` ***op2***`)`
op1 `!==` ***op2***	`!(`***op1*** `===` ***op2***`)`

Tabella 41.52. Elenco degli operatori di assegnamento. Va tenuto in considerazione che le espressioni di assegnamento restituiscono lo stesso valore assegnato.

Operatore e operandi	Descrizione
var = *valore*	Assegna alla variabile il valore alla destra.
op1 += *op2*	*op1* = (*op1* + *op2*)
op1 -= *op2*	*op1* = (*op1* - *op2*)
op1 *= *op2*	*op1* = (*op1* * *op2*)
op1 /= *op2*	*op1* = (*op1* / *op2*)
op1 %= *op2*	*op1* = (*op1* % *op2*)
op1 .= *op2*	*op1* = (*op1* . *op2*)
op1 &= *op2*	*op1* = (*op1* & *op2*)
op1 |= *op2*	*op1* = (*op1* | *op2*)
op1 ^= *op2*	*op1* = (*op1* ^ *op2*)
op1 <<= *op2*	*op1* = (*op1* << *op2*)
op1 >>= *op2*	*op1* = (*op1* >> *op2*)
op1 ~= *op2*	*op1* = ~*op2*

41.8 Strutture di controllo di flusso

«

Il linguaggio PHP gestisce praticamente tutte le strutture di controllo di flusso degli altri linguaggi di programmazione, compreso *go-to* che comunque è sempre meglio non utilizzare.

Le strutture di controllo permettono di sottoporre l'esecuzione di una parte di codice alla verifica di una condizione, oppure permettono di eseguire dei cicli, sempre sotto il controllo di una condizione. La parte di codice che viene sottoposta a questo controllo, può essere una singola istruzione, oppure un gruppo di istruzioni (precisamente si chiamerebbe istruzione composta). Nel secondo caso, è necessario delimitare questo gruppo attraverso l'uso delle parentesi graffe.

Dal momento che è comunque consentito di realizzare un gruppo di istruzioni che in realtà ne contiene una sola, probabilmente è meglio utilizzare sempre le parentesi graffe, in modo da evitare equivoci nella lettura del codice. Dato che le parentesi graffe sono usate nel codice PHP, se queste appaiono nei modelli sintattici indicati, significa che fanno parte delle istruzioni e non della sintassi.

Il linguaggio PHP offre due modi alternativi di rappresentare le strutture di controllo, ma qui si mostra esclusivamente quello conforme al linguaggio C e anche al linguaggio Perl. Tuttavia è necessario essere a conoscenza del fatto che esiste una seconda modalità, per non trovarsi impreparati quando si legge del codice PHP scritto diversamente da come si è abituati.

41.8.1 Struttura condizionale: «if»

La struttura condizionale è il sistema di controllo fondamentale dell'andamento del flusso delle istruzioni.

```
if (condizione) istruzione
```

```
if (condizione) istruzione else istruzione
```

Se la condizione si verifica, viene eseguita l'istruzione o il gruppo di istruzioni che segue; quindi il controllo passa alle istruzioni successive alla struttura. Se viene utilizzata la sotto-struttura che si articola a partire dalla parola chiave '**else**', nel caso non si verifichi la condizione, viene eseguita l'istruzione che ne dipende. Sotto vengono mostrati alcuni esempi.

```
$importo;
…
if ($importo > 10000000) echo ("L'offerta è vantaggiosa\n");
```

```
$importo;
$memorizza;
…
if ($importo > 10000000)
  {
    $memorizza = $importo;
    echo ("L'offerta è vantaggiosa\n");
  }
else
  {
    echo ("Lascia perdere\n");
  }
```

L'esempio successivo, in particolare, mostra un modo grazioso per allineare le sottocondizioni, senza eccedere negli annidamenti:

```
$importo;
$memorizza;
…
if ($importo > 10000000)
  {
    $memorizza = $importo;
    printf ("L'offerta è vantaggiosa\n");
  }
else if ($importo > 5000000)
  {
    $memorizza = $importo;
    printf ("L'offerta è accettabile\n");
  }
else
  {
    printf ("Lascia perdere\n");
  }
```

Va osservato che il PHP consente di fondere assieme le parole '**`else`**' e '**`if`**' in un'unica parola: '**`elseif`**'. Ma per uniformità con il linguaggio C sarebbe meglio evitare di avvalersi di questa forma contratta.

41.8.2 Struttura di selezione: «switch»

«

La struttura di selezione che si attua con l'istruzione '**`switch`**', è un po' troppo complessa per essere rappresentata facilmente attraverso uno schema sintattico. In generale, questa struttura permette di **saltare** a una certa posizione della struttura, in base al risultato

di un'espressione. L'esempio seguente mostra la visualizzazione del nome del mese, in base al valore di una variabile intera.

```
$mese;
…
switch ($mese)
  {
    case 1: echo ("gennaio\n"); break;
    case 2: echo ("febbraio\n"); break;
    case 3: echo ("marzo\n"); break;
    case 4: echo ("aprile\n"); break;
    case 5: echo ("maggio\n"); break;
    case 6: echo ("giugno\n"); break;
    case 7: echo ("luglio\n"); break;
    case 8: echo ("agosto\n"); break;
    case 9: echo ("settembre\n"); break;
    case 10: echo ("ottobre\n"); break;
    case 11: echo ("novembre\n"); break;
    case 12: echo ("dicembre\n"); break;
  }
```

Come si vede, dopo l'istruzione con cui si emette il nome del mese attraverso lo standard output, viene richiesta l'interruzione esplicita dell'analisi della struttura, attraverso l'istruzione '**`break`**', perché altrimenti verrebbero eseguite le istruzioni del caso successivo, se presente. Infatti, un gruppo di casi può essere raggruppato assieme, quando si vuole che ognuno di questi esegua lo stesso insieme di istruzioni.

```
$anno;
$mese;
$giorni;
…
switch ($mese)
```

```
  {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        $giorni = 31;
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        $giorni = 30;
        break;
    case 2:
        if (((($anno % 4 == 0) && !($anno % 100 == 0)) ||
                ($anno % 400 == 0))
            $giorni = 29;
        else
            $giorni = 28;
        break;
  }
```

È anche possibile dichiarare un caso predefinito che si verifichi quando nessuno degli altri si avvera.

```
$mese;
…
switch ($mese)
  {
    case 1: echo ("gennaio\n"); break;
    case 2: echo ("febbraio\n"); break;
    …
    case 11: echo ("novembre\n"); break;
    case 12: echo ("dicembre\n"); break;
    default: echo ("mese non corretto\n"); break;
  }
```

Va osservato che l'espressione oggetto di valutazione può essere di qualunque tipo «scalare» secondo il linguaggio. Pertanto, avrebbe potuto trattarsi anche di una stringa:

```
$mese;
…
switch ($mese)
  {
    case "gennaio": echo ("gennaio\n"); break;
    case "febbraio": echo ("febbraio\n"); break;
    …
    case "novembre": echo ("novembre\n"); break;
    case "dicembre": echo ("dicembre\n"); break;
    default: echo ("mese non corretto\n"); break;
  }
```

41.8.3 Iterazione con condizione di uscita iniziale: «while»

L'iterazione si ottiene normalmente in PHP attraverso l'istruzione '**`while`**', la quale esegue un'istruzione, o un gruppo di queste, finché la condizione continua a restituire il valore *Vero*. La condizione viene

valutata prima di eseguire il gruppo di istruzioni e poi ogni volta che termina un ciclo, prima dell'esecuzione del successivo.

```
while (condizione) istruzione
```

L'esempio seguente fa apparire per 10 volte la lettera «x».

```
$i = 0;
while ($i < 10)
  {
    $i++;
    echo ("x");
  }
```

Nel blocco di istruzioni di un ciclo '**while**', ne possono apparire alcune particolari:

- '**break**', che serve a uscire definitivamente dalla struttura del ciclo;
- '**continue**', che serve a interrompere l'esecuzione del gruppo di istruzioni, riprendendo immediatamente con il ciclo successivo (a partire dalla valutazione della condizione).

L'esempio seguente è una variante del calcolo di visualizzazione mostrato sopra, modificato in modo da vedere il funzionamento dell'istruzione '**break**'. All'inizio della struttura, '**while (TRUE)**' equivale a stabilire che il ciclo è senza fine, perché la condizione è sempre vera. In questo modo, solo la richiesta esplicita di interruzione dell'esecuzione della struttura (attraverso l'istruzione '**break**') permette l'uscita da questa.

```
$i = 0;
while (TRUE)
```

```
  {
    if ($i >= 10)
      {
        break;
      }
    $i++;
    echo ("x");
  }
```

41.8.4 Iterazione con condizione di uscita finale: «do-while»

Una variante del ciclo '**while**', in cui l'analisi della condizione di uscita avviene dopo l'esecuzione del blocco di istruzioni che viene iterato, è definito dall'istruzione '**do**'.

```
do blocco_di_istruzioni while (condizione);
```

In questo caso, si esegue un gruppo di istruzioni una volta, poi se ne ripete l'esecuzione finché la condizione restituisce il valore *Vero*.

```
$i = 0;
do
  {
    $i++;
    echo ("x");
  }
while ($i < 10);
```

L'esempio mostrato è quello già usato nella sezione precedente, con l'adattamento necessario a utilizzare questa struttura di controllo.

41.8.5 Ciclo enumerativo: «for»

«

In presenza di iterazioni in cui si deve incrementare o decrementare una variabile a ogni ciclo, si usa preferibilmente la struttura '**for**', che in PHP, come in C, permetterebbe un utilizzo più ampio di quello comune:

```
for ([espressione1]; [espressione2]; [espressione3]) istruzione
```

La forma tipica di un'istruzione '**for**' è quella per cui la prima espressione corrisponde all'assegnamento iniziale di una variabile, la seconda a una condizione che deve verificarsi fino a che si vuole che sia eseguita l'istruzione (o il gruppo di istruzioni) e la terza all'incremento o decremento della variabile inizializzata con la prima espressione. In pratica, l'utilizzo normale del ciclo '**for**' potrebbe esprimersi nella sintassi seguente:

```
for (var = n; condizione; var++) istruzione
```

Il ciclo '**for**' potrebbe essere definito anche in maniera differente, più generale: la prima espressione viene eseguita una volta sola all'inizio del ciclo; la seconda viene valutata all'inizio di ogni ciclo e il gruppo di istruzioni viene eseguito solo se il risultato è *Vero*; l'ultima viene eseguita alla fine dell'esecuzione del gruppo di istruzioni, prima che si ricominci con l'analisi della condizione.

L'esempio già visto, in cui viene visualizzata per 10 volte una «x», potrebbe tradursi nel modo seguente, attraverso l'uso di un ciclo '**for**':

```
$i;
for ($i = 0; $i < 10; $i++)
  {
    echo ("x");
  }
```

Anche nelle istruzioni controllate da un ciclo '**for**' si possono collocare istruzioni '**break**' e '**continue**', con lo stesso significato visto per il ciclo '**while**' e '**do**...**while**'.

Sfruttando la possibilità di inserire più espressioni in una singola istruzione, si possono realizzare dei cicli '**for**' molto più complessi, anche se questo è sconsigliabile per evitare di scrivere codice troppo difficile da interpretare. In questo modo, l'esempio precedente potrebbe essere ridotto a quello che segue, dove si usa un punto e virgola solitario per rappresentare un'istruzione nulla:

```
$i;
for ($i = 0; $i < 10; echo ("x"), $i++)
  {
    ;
  }
```

Se si utilizzano istruzioni multiple, separate con la virgola, occorre tenere presente che **l'espressione che esprime la condizione deve rimanere singola** (se per la condizione si usasse un'espressione multipla, conterebbe solo la valutazione dell'ultima). Naturalmente, nel caso della condizione, si possono costruire condizioni complesse con l'ausilio degli operatori logici, ma rimane il fatto che l'operatore virgola ('`,`') non dovrebbe avere senso lì.

Nel modello sintattico iniziale si vede che le tre espressioni sono opzionali e rimane solo l'obbligo di mettere i punti e virgola relativi. L'esempio seguente mostra un ciclo senza fine che viene interrotto

attraverso un'istruzione '**break**':

```
$i = 0;
for (;;)
  {
    if ($i >= 10)
      {
        break;
      }
    echo ("x");
    $++;
  }
```

41.8.6 Ciclo di scansione degli array: «foreach»

«

Il linguaggio PHP gestisce gli array in modo molto «semplice», consentendo di usare indifferentemente array tradizionali con un indice numerico e array associativi con un indice costituito da un valore scalare qualsiasi. La scansione di un array con la struttura '**for**' può avvenire solo in presenza di un array tradizionale a indice numerico; diversamente l'operazione diventerebbe troppo difficile.

```
foreach (array as valore) istruzione
```

```
foreach (array as indice => valore) istruzione
```

La sintassi per la struttura '**foreach**' è di tue tipi; nel primo caso, la scansione attribuisce alla variabile ***valore***, di volta in volta, una copia del contenuto dell'elemento in corso di scansione. Va osservato che ***valore*** deve essere una variabile e che questa va poi utilizzata solo per leggere tale informazione, perché modificandola **non** si otterrebbe l'aggiornamento dell'elemento corrispondente nell'array.

```
$arr = array (1, 1, 2, 3, 5, 8);
$v;
foreach ($arr as $v)
  {
    echo ("$v, ");
  }
```

Come si vede nell'esempio appena apparso, si scandisce l'array ***$arr*** e si visualizza il suo contenuto, a partire dal primo elemento, fino all'ultimo presente.

La seconda forma sintattica del ciclo '**foreach**' consente di conoscere l'indice utile per accedere all'elemento scandito:

```
$arr = array (1, 1, 2, 3, 5, 8);
$i;
$v;
foreach ($arr as $i => $v)
  {
    echo ($arr[$i] . ", ");
  }
```

L'esempio produce lo stesso risultato di quello precedente, con la differenza che l'elemento scandito viene individuato attraverso l'indice, qualunque esso sia in quel momento, consentendo eventualmente di modificare il contenuto dell'elemento relativo.

Anche nelle istruzioni controllate da un ciclo '**foreach**' si possono collocare istruzioni '**break**' e '**continue**', con lo stesso significato visto per il ciclo '**while**', '**do**...**while**' e '**for**'.

41.9 Funzioni

«

Le funzioni del linguaggio PHP si dichiarano in modo analogo a quello del linguaggio C. Nella situazione più comune si usa una sintassi come quella seguente:

```
function nome ([par_1, [par_2 [, ...]]])
{
    istruzioni
    ...
}
```

Va osservato che la dichiarazione della funzione non specifica il tipo che questa restituisce, ammesso che restituisca qualcosa, e nemmeno il tipo dei parametri della chiamata. A ogni modo, come nel C e come in altri linguaggi, si restituisce un valore con l'istruzione '**return**', ma ciò che può essere restituito non è vincolato a dei tipi particolari e può essere anche un array.

```
function mia ($a, $b, $c)
{
  return ($a+$b+$c);
}
```

L'esempio mostra la funzione ***mia()*** che accetta tre argomenti, di cui inizialmente non si conosce il tipo. La funzione prende i tre argomenti e ne restituisce la somma, ammesso che questi corrispondano a dati numerici che possano essere sommati. Il tipo restituito dalla funzione dipende dal tipo generato dalla somma. Nell'esempio successivo, viene chiamata la funzione ***mia()*** con alcuni valori di cui si vuole ottenere la somma; ciò che la variabile ***$d*** ottiene è il numero

6:

```
$d = mia (1, 2, 3);
```

Il linguaggi PHP consente di stabilire un valore predefinito dei parametri previsti:

```
function mia ($a = 1, $b = 2, $c = 3)
{
  return ($a+$b+$c);
}
```

La funzione ***mia()*** del nuovo esempio agisce come nella dichiarazione precedente, con la differenza che nella chiamata si possono omettere dei dati, se il valore predefinito è valido:

```
$d = mia (5, 6);
```

In questo caso, nella variabile ***$d*** si ottiene il valore 14 (5+6+3), perché il terzo argomento mancante è costituito implicitamente dal valore 3.

Perché il meccanismo degli argomenti predefiniti possa essere efficace, è necessario che i parametri rispettivi siano messi per ultimi nella dichiarazione della funzione, secondo un ordine di importanza. Nel caso della funzione ***mia()***, è conveniente supporre che se non si specifica il secondo argomento (parametro ***$b***), non abbia alcun senso specificare invece il terzo (parametro ***$c***), perché diversamente sarebbe scomodo saltare l'argomento centrale. Quindi, con una funzione strutturata così, si intende implicitamente che sia conveniente avere chiamate senza argomenti, con i primi due argomenti o con tutti e tre gli argomenti.

Con la dichiarazione dei parametri di una funzione (nell'esempio della funzione ***mia()*** si tratta delle variabili ***$a***, ***$b*** e ***$c***), si ha im-

plicitamente la loro dichiarazione in qualità di variabili locali. Ciò comporta che la modifica del contenuto di queste variabili non si trasmette ai dati di origine, anche se si trattasse di un array. Tuttavia, è possibile dichiarare espressamente un parametro in modo tale che faccia riferimento a una variabile nella chiamata:

```
function tua ($e, $f, &$g)
{
  $g = $e+$f;
}
//
$h = 10;
tua (1, 2, $h);
```

Nell'esempio si vede la funzione ***tua()***, nella quale l'ultimo parametro (***$g***) è preceduto dalla e-commerciale, '**&**'. In tal modo, per il linguaggio PHP, si intende specificare che la variabile locale corrispondente viene trattata come riferimento a una variabile usata nella chiamata. Nel caso della funzione dell'esempio, si vede che si va a modificare quella variabile con la somma degli altri due argomenti. Nell'esempio si vede poi che si dichiara una variabile ***$h*** con un certo valore di partenza, quindi si chiama la funzione ***tua()***, utilizzando la variabile ***$h*** come ultimo argomento. Dopo la chiamata, la variabile ***$h*** contiene il valore 3, pari alla somma degli altri due argomenti.

Quando una funzione prevede dei parametri trasmessi per riferimento, la chiamata di tale funzione deve mettere, in corrispondenza di quei parametri, delle variabili. Se nel caso dell'esempio, nella chiamata della funzione ***tua()***, se il terzo argomento fosse una costante, l'interprete PHP produrrebbe un errore irreversibile.

Così come è possibile consentire l'uso di funzioni la cui chiamata sia, totalmente o parzialmente, per riferimento, è possibile anche che una funzione restituisca una propria variabile per riferimento, in modo da consentirne la modifica al di fuori della funzione stessa:

```
function &sua ()
{
  static $i = 0;
  $i++;
  return ($i);
}
//
$j = &sua ();
$j += 10;
sua ();
```

L'esempio mostra la funzione ***sua()*** che non dichiara parametri, ma al suo interno mette una variabile statica, ***$i***, che poi viene restituita. La funzione viene dichiarata con l'operatore '**&**' per indicare che quanto viene restituito è (deve essere) il riferimento a una variabile. Poi si vede la variabile ***$j*** che diviene un riferimento alternativo alla variabile restituita dalla chiamata alla funzione ***sua()*** e inizialmente si trova a contenere il valore 1. Poi ***$j*** viene incrementata di 10 unità, passando a ***11***, quindi viene chiamata nuovamente la funzione ***sua()***, la quale incrementa ulteriormente la propria variabile ***$i*** che però corrisponde sempre alla variabile ***$j*** e ora contiene ***12***.

41.10 Suddivisione del programma in più file

«

Il codice PHP può essere distribuito su più file, specialmente se più programmi condividono l'uso di certe funzioni o di certe dichiarazioni. I programmi che si avvalgono di altri file usano delle istruzioni di inclusione, per far sì che in un certo punto del proprio codice si inserisca quello di un altro file. In generale è opportuno che le inserzioni di file diversi avvengano al di fuori delle funzioni e contengano codice adatto per collocarsi al livello del campo di azione globale.

```
include (file_da_includere)
```

```
include_once (file_da_includere)
```

```
require (file_da_includere)
```

```
require_once (file_da_includere)
```

I modelli sintattici mostrano quattro istruzioni alternative per l'inclusione di codice esterno (hanno l'apparenza di funzioni, ma in realtà le parentesi tonde possono essere omesse). Queste istruzioni hanno in comune l'argomento richiesto, costituito da una stringa che indica il percorso di un file da includere. In teoria il percorso di tale file potrebbe essere espresso come URI, per raggiungere un file remoto, ma è sicuramente meglio evitare di dipendere da file remoti e disporre tutto nello stesso file system del programma principale.

Le istruzioni il cui nome inizia per ***include***, si limitano a generare un avvertimento nel caso il file non risulti accessibile, senza però compromettere l'esecuzione del programma; al contrario, le istruzioni ***require***, nel caso non riuscissero a caricare il file richiesto, produrrebbero un errore irreversibile e l'arresto del programma. In pratica, vanno usate le funzioni ***require*** se l'inclusione è indispensabile.

Le istruzioni che finiscono per ***once***, hanno in comune il fatto di caricare il file soltanto se questo non risulta già essere stato caricato.

41.11 Input di dati

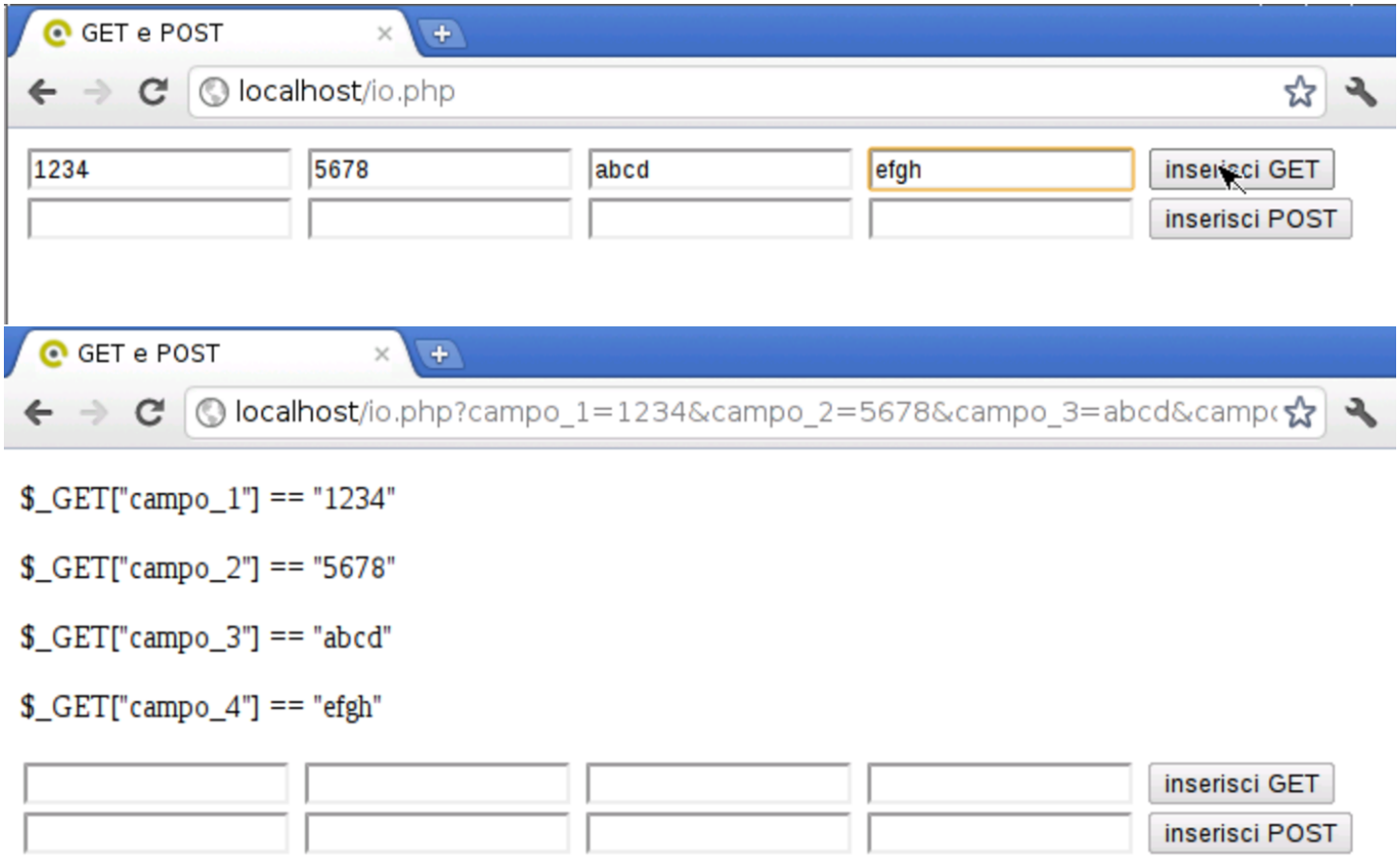
Quando una pagina PHP riceve dati attraverso una chiamata GET o POST, queste informazioni sono disponibili al linguaggio PHP tramite gli array superglobali ***$_GET[]*** e ***$_POST[]***. L'indice per raggiungere tali informazioni è costituito dal nome del campo corrispondente, nel formulario realizzato presumibilmente con l'elemento '**FORM**' del HTML:

```
<?php
echo ("<!doctype html>\n");
echo ("<html>\n");
echo ("  <head>\n");
echo ("    <meta charset=\"UTF-8\">\n");
echo ("    <title>GET e POST</title>\n");
echo ("  </head>\n");
echo ("  <body>\n");
$i;
$v;
foreach ($_GET as $i => $v)
  {
   echo ("    <p>\$_GET[\"$i\"] == \"$v\"</p>\n");
  }
foreach ($_POST as $i => $v)
```

```
  {
   echo ("    <p>\$_POST[\"$i\"] == \"$v\"</p>\n");
  }
echo ("    <FORM ACTION=\"io.php\" METHOD=\"GET\">\n");
echo ("    <INPUT NAME=\"campo_1\" SIZE=\"15\">\n");
echo ("    <INPUT NAME=\"campo_2\" SIZE=\"15\">\n");
echo ("    <INPUT NAME=\"campo_3\" SIZE=\"15\">\n");
echo ("    <INPUT NAME=\"campo_4\" SIZE=\"15\">\n");
echo ("    <INPUT TYPE=\"submit\" "
     ."VALUE=\"inserisci GET\">\n");
echo ("    </FORM>\n");
echo ("    <FORM ACTION=\"io.php\" METHOD=\"POST\">\n");
echo ("    <INPUT NAME=\"campo_1\" SIZE=\"15\">\n");
echo ("    <INPUT NAME=\"campo_2\" SIZE=\"15\">\n");
echo ("    <INPUT NAME=\"campo_3\" SIZE=\"15\">\n");
echo ("    <INPUT NAME=\"campo_4\" SIZE=\"15\">\n");
echo ("    <INPUT TYPE=\"submit\" "
     ."VALUE=\"inserisci POST\">\n");
echo ("    </FORM>\n");
echo ("  </body>\n");
echo ("</html>\n");
?>
```

Nell'esempio si vede che, attraverso il codice PHP, viene generata una pagina HTML contenente due elementi '**FORM**', i quali inviano dati al file '`io.php`', rispettivamente secondo il metodo GET e secondo il metodo POST. Il codice PHP, prima di visualizzare gli elementi '**FORM**', scandisce gli array ***$_GET[]*** e ***$_POST[]***, mostrando tutto il loro contenuto. In pratica, ammesso che questo esempio si trovi nel file '`io.php`', la prima volta che lo si visualizza si ottiene solo il formulario, quindi, inviando qualche dato, si vede ciò che era stato inserito in precedenza.

Figura 41.75. Esempio di inserimento di dati nel file ‘`io.php`’ ed esito successivo.

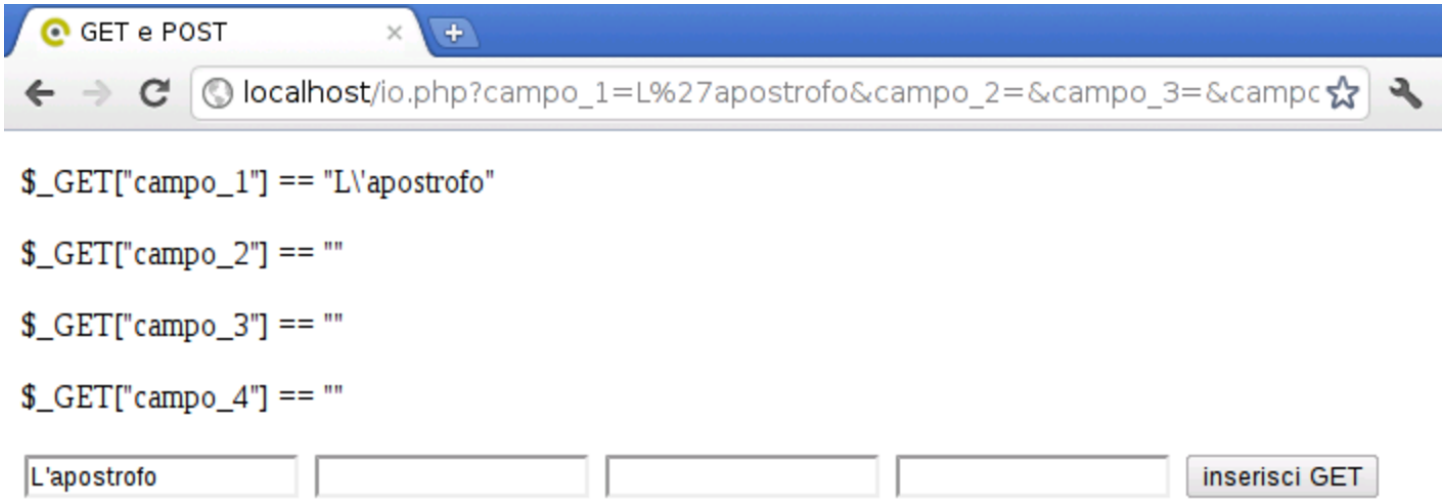


È importante che nella configurazione del PHP (file ‘`php.ini`’) ci sia la direttiva ‘**`magic_quotes_gpc=Off`**’:

```
magic_quotes_gpc = Off
```

Se questa direttiva non c’è o se è impostata in modo differente, quando si inseriscono dati nei campi di un formulario, la lettura degli array ***$_GET[]*** e ***$_POST[]*** produce un effetto spiacevole in corrispondenza dell’apostrofo: viene trasformato in ‘`\'`’.

Figura 41.77. Apostrofo ottenuto senza la direttiva di configurazione '**magic_quotes_gpc=Off**'.



Quando si devono attendono dati attraverso gli array ***$_GET[]*** e ***$_POST[]***, è necessario accertarsi che questi ci siano effettivamente, per evitare il manifestarsi di errori, anche se di lieve entità:

```
$cognome = "";
$nome = "";
if (isset ($_GET["cognome"])) $cognome = $_GET["cognome"];
if (isset ($_GET["nome"]))    $nome    = $_GET["nome"];
```

In questo caso si usa la funzione ***isset()*** per determinare se l'elemento richiesto dell'array ***$_GET[]*** esiste veramente: se ciò è vero, allora copia il suo contenuto in una variabile scalare apposita.

Oltre al problema di verificare l'esistenza di un dato, è necessario applicare un filtro preliminare alle informazioni ricevute, per evitare di accettare dati inappropriati o pericolosi. Per esempio, se si prevede un campo numerico, conviene eseguire un cast, con il quale tutto ciò che non fosse numerico verrebbe semplicemente scartato:

```
$anni = 0;
if (isset ($_GET["anni"])) $anni (int) $_GET["anni"];
```

Per situazioni più complesse, come il caso dell'inserimento

di un indirizzo di posta elettronica, si possono utilizzare le espressioni regolari, con l'aiuto della funzione ***preg_match()***:

```
$email = "";
if (isset ($_GET["email"])
  && preg_match ("/^[a-zA-Z0-9._-]*@[a-zA-Z0-9._-]*$/", $_GET["email"]))
  {
    $email = $_GET["email"];
  }
```

In verità, il linguaggio PHP offre delle funzioni appropriate per il filtro dei dati in ingresso; tuttavia, il meccanismo standard rischia di creare confusione. A ogni modo, si tratterebbe di sfruttare le funzioni ***filter_...()***.

Il filtro in ingresso ai dati consente di rifiutare dati non validi e di ignorare il superfluo. Tuttavia, all'interno di dati validi si possono nascondere altri problemi, nel momento in cui questi dati devono essere usati. Per esempio, se l'informazione ricevuta serve per popolare una tabella SQL, è necessario trasformare la stringa che rappresenta l'informazione in modo che non si creino interferenze con i simboli usati per la delimitazione nella sintassi SQL.

Tabella 41.81. Funzioni utili per la trasformazione di stringhe, secondo vari criteri relativi all'uso in istruzioni SQL e HTML. La prima versione trasforma la stringa, la seconda, se c'è, la ripristina.

Funzione	Descrizione

`addslashes (`***str***`)`
 	
`stripslashes (`***str***`)`
	La funzione ***addslashes()*** trasforma la stringa ***str***, restituendola con l'aggiunta di simboli '\' davanti agli apostrofi, agli apici doppi e alle barre oblique inverse. Ciò serve a consentire l'uso dell'informazione all'interno di un dato delimitato da apici, singoli o doppi, che preveda questo tipo di sequenza di escape. La funzione ***stripslashes()*** fa il lavoro opposto, togliendo le sequenze di escape.

`mysql_real_escape_string (`***str***`)`	Trasforma la stringa ***str***, restituendola con tutti gli adattamenti necessari all'uso in un'istruzione SQL di MySQL, precisamente in quella parte dell'istruzione che si trova a essere delimitata da apici.
`preg_quote (`***str***`)`	Trasforma la stringa ***str***, restituendola con tutti gli adattamenti necessari a usarla, tale e quale, in un'espressione regolare Perl.

Funzione	Descrizione

`htmlspecialchars (`***str***`)`
 	
`htmlspecialchars_decode (`***str***`)`
	La funzione ***htmlspecialchars()*** trasforma la stringa ***str***, restituendola con la trasformazione di simboli speciali per l'HTML in entità standard. Per esempio, '**&**' viene trasformato in '**&**'; '**<**' e '**>**' vengono trasformati in '**<**' e '**>**'. La funzione ***htmlspecialchars_decode()*** compie la trasformazione opposta.

`htmlentities (`***str***`)`
 	
`html_entity_decode (`***str***`)`
	La funzione ***htmlentities()*** trasforma la stringa ***str***, restituendola con la trasformazione di tutti i simboli possibili in entità standard. La funzione ***html_entity_decode()*** compie la trasformazione opposta.

Funzione	Descrizione
`nl2br (`***str***`)`	Trasforma la stringa ***str***, trasformando il codice di interruzione di riga in '**` `**'. Ciò può essere utile per l'incorporazione in codice HTML.

41.12 Sessione

«

Le sessioni sono il modo con il quale è possibile conservare delle informazioni, nell'ambito di un'applicazione scritta in PHP, attraverso accessi successivi. Questo problema si pone quando è necessario riconoscere che si tratta dello stesso utente che continua ad accedere durante una stessa sessione di lavoro.

Le informazioni relative alla sessione vengono conservate dall'interprete PHP in file temporanei, la cui collocazione è determinata attraverso la direttiva di configurazione ***session.save_path***, nel file '`php.ini`' (può essere modificata anche attraverso la funzione ***session_save_path()***, il cui uso è però sconsigliabile, se si vuole scrivere un programma che non dipenda dalle caratteristiche particolari della piattaforma in cui si trova a funzionare). Generalmente, potrebbe trattarsi della directory '`/var/lib/php…/`', la quale deve consentire l'accesso in lettura e scrittura all'utenza di sistema con cui figura funzionare l'interprete PHP. Eventualmente, in presenza di errori relativi alla gestione delle sessioni, va verificato proprio quale sia il percorso per questi file temporanei e i permessi di accesso esistenti in tale directory.

Naturalmente, perché la sessione possa mantenersi, il programma cliente (il navigatore) deve conservare un'informazione univoca che permetta al PHP di riconoscere che l'accesso fa parte di una certa sessione già attiva. Per questo si usano i *cookie* o informazioni inserite come metodi GET o POST. Il sistema dei *cookie* è quello più efficace e, generalmente, le applicazioni scritte in PHP richiedono che il programma cliente consenta l'uso dei *cookie*.

La sessione inizia formalmente con l'uso della funzione ***session_start()***, la quale genera una nuova sessione o riprende una sessione precedente, se questa risulta già attiva.

Le informazioni relative alla sessione in corso, sono conservate nell'array superglobale ***$_SESSION[]***, ed è in questo array che le informazioni da preservare vanno aggiunte.

Viene mostrato un esempio completo, di un file PHP che, chiamato per la prima volta, richiede di inserire una parola d'ordine; poi, alle chiamate successive, riconoscendo che questa è già stata inserita, consente di incrementare un contatore, fino a quando si richiede espressamente di uscire dalla sessione, azzerando il contatore e la parola d'ordine memorizzata.

```
<?php
session_start ();
$password_attesa = "la mia password";
//
if (!isset ($_SESSION["contatore"]))
  {
    $_SESSION["contatore"] = 0;
  }
if (!isset ($_SESSION["password"]))
  {
    $_SESSION["password"] = "";
```

```
  }
//
if (isset ($_POST["password"]))
  {
    $_SESSION["password"] = $_POST["password"];
  }
if (isset ($_POST["incrementa"])
  && $_POST["incrementa"] == "+"
  && $_SESSION["password"] == $password_attesa)
  {
    $_SESSION["contatore"]++;
  }
if (isset ($_POST["esci"])
  && $_POST["esci"] == "0"
  && ($_SESSION["password"] == ""
    || $_SESSION["password"] == $password_attesa))
  {
    $_SESSION["contatore"] = 0;
    $_SESSION["password"] = "";
  }
//
echo ("<!doctype html>\n");
echo ("<html>\n");
echo ("  <head>\n");
echo ("    <meta charset=\"UTF-8\">\n");
echo ("    <title>Sessione</title>\n");
echo ("  </head>\n");
echo ("  <body>\n");
echo ("    <p>contatore: ".$_SESSION["contatore"]."</p>\n");
//
if ($_SESSION["password"] != $password_attesa)
  {
    echo ("    <form method=\"POST\">\n");
    echo ("    <p>Password: ");
```

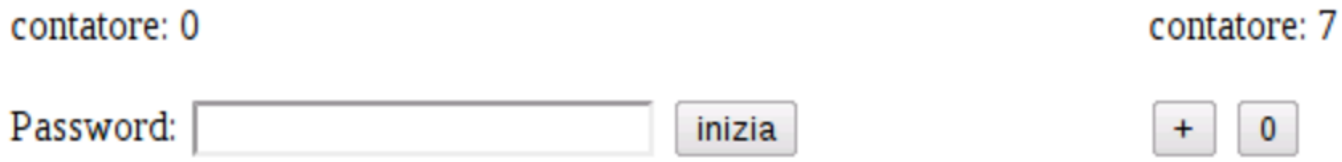
```
    echo ("    <input type=\"password\" name=\"password\" "
         ."value=\"\">\n");
    echo ("    <input type=\"submit\" value=\"inizia\">\n");
    echo ("    </form>\n");
  }
else
  {
    echo ("    <form method=\"POST\">\n");
    echo ("    <input type=\"submit\" name=\"incrementa\" "
         ."value=\"+\">\n");
    echo ("    <input type=\"submit\" name=\"esci\" "
         ."value=\"0\">\n");
    echo ("    </form>\n");
  }
echo ("  </body>\n");
echo ("</html>\n");
?>
```

Come si vede nell'esempio, la sessione conserva due informazioni: ***$_SESSION["password"]***, pari alla parola d'ordine corrente, e ***$_SESSION["contatore"]***, pari al contatore da incrementare. Dal metodo POST si attendono tre campi possibili: ***password***, ***incrementa*** ed ***esci***. Se c'è il campo ***password***, questo viene annotato nella sessione; se c'è il campo ***incrementa*** che contiene correttamente la stringa '`+`', e la parola d'ordine annotata nella sessione è quella attesa, allora il contatore ***$_SESSION["contatore"]*** viene incrementato; se c'è il campo ***esci*** che contiene correttamente la stringa '`0`', e la parola d'ordine annotata nella sessione è quella attesa, oppure non c'è affatto, allora il contatore ***$_SESSION["contatore"]*** viene azzerato e viene azzerata anche la parola d'ordine.

Successivamente, viene predisposta una pagina HTML, dove si mo-

stra un formulario con la richiesta della parola d'ordine se questa non risulta inserita correttamente, oppure con due bottoni, uno per incrementare il contatore e l'altro per azzerarlo e ricominciare.

Figura 41.83. Aspetto del programma di esempio: a sinistra si vede l'inizio della sessione con la richiesta della parola d'ordine; a destra si vede la sessione in corso, con i bottoni di incremento e di uscita dalla sessione.



Prima di creare o di fare riferimento a una sessione, attraverso la funzione ***session_start()***, è possibile indicare il nome di una sessione precisa: ciò consente di tenere distinte sessioni alternative, per qualche scopo. Quando questo nome di sessione non viene specificato, si intende implicitamente quello predefinito, pari a '**`PHPSESSID`**'. Il nome della sessione può essere definito o letto attraverso la funzione ***session_name()***, la quale va usata sempre prima di ***session_start()***:

```
$nome = session_name ("mia_sessione");
session_start ();
```

L'esempio mostra l'uso della funzione ***session_start()***: in questo caso viene salvato il nome della sessione attiva precedentemente nella variabile ***$nome*** e viene selezionata la nuova sessione '**`mia_sessione`**'.

Una sessione può essere chiusa esplicitamente, eliminando quanto memorizzato nell'array ***$_SESSION[]***, con la funzione ***session_destroy()***. Va però osservato che viene eliminata la sessione

attiva: se si vuole eliminare una sessione precisa, la quale potrebbe risultare attiva o meno, va usata prima la funzione ***session_name()*** e, per maggiore sicurezza, anche ***session_start()***:

```
$nome = session_name ("mia_sessione");
session_start ();
session_destroy ();
```

41.13 Accesso ai file

Il linguaggio PHP offre molte funzioni più o meno conformi allo standard della libreria C e POSIX, per l'accesso al file system, ma mette a disposizione anche alcune funzioni particolari che rendono più semplice l'accesso ai file. Queste ultime funzioni sono quelle che si trovano più frequentemente nel codice PHP, proprio per la loro praticità.

```
file (percorso)
```

La funzione ***file()*** richiede l'indicazione di una stringa contenente il percorso necessario a raggiungere un file di testo. La funzione si occupa di leggere il file e di restituire un array di stringhe, contenenti ognuna la riga ***n***-esima del file di origine. Le righe del file originario vengono copiate come sono, complete del codice di interruzione di riga finale. Se la lettura fallisce, la funzione restituisce il valore *Falso*.

```
file_get_contents (percorso)
```

La funzione ***file_get_contents()*** legge un file di qualunque tipo e lo restituisce come se fosse una stringa. Se la lettura fallisce, restituisce

il valore *Falso*.

```
file_put_contents (percorso, contenuto)
```

La funzione ***file_put_contents()*** crea o sovrascrive un file di qualunque tipo, restituendo la quantità di byte scritti con successo. Il primo argomento richiesto è il percorso del file, mentre il secondo rappresenta il contenuto da scrivere, il quale può essere in forma di stringa o di array di stringhe.

```
read_file (percorso)
```

La funzione ***read_file()*** consente di leggere un file e di riprodurlo (visualizzarlo). In pratica svolge il ruolo di ***echo(file_get_contents())***.

Le funzioni di lettura dei file che sono state descritte (***file()***, ***file_get_contents()*** e ***read_file()***) possono essere usate per leggere sia file locali, sia file remoti, raggiungibili con i protocolli HTTP, HTTPS e FTP. In tal caso, invece di indicare un percorso tradizionale a un file locale, si indica un URI, del tipo '`http://`…'.

41.14 Espressioni regolari

«

Il linguaggio PHP utilizza preferibilmente le espressioni regolari secondo la sintassi del Perl come descritto in sintesi nella sezione 24.13. Senza entrare nel dettaglio della sintassi delle espressioni regolari, la quale invece va approfondita nella documentazione originale, va osservato che le stringhe di tali espressioni devono essere raccolte in delimitatori appropriati, corrispondenti di solito alla barra obliqua '`/`', come si fa nel linguaggio Perl, anche se qui, con il

PHP, tali espressioni sono ben individuate da parametri distinti nelle funzioni che le utilizzano.

Le funzioni fondamentali per la gestione delle espressioni regolari secondo la sintassi Perl, sono ***preg_match()*** e ***preg_replace()***.

```
preg_match (regex, stringa)
```

La funzione ***preg_match()*** confronta l'espressione regolare che costituisce il primo parametro, con la stringa che ne costituisce il secondo. La funzione restituisce zero se non si verifica alcuna corrispondenza, oppure uno se una corrispondenza c'è.

```
$n = preg_match ('/href=["\'].*\.css["\'].*$/', $riga);
```

L'esempio mostra la ricerca, all'interno della variabile ***$riga***, di una corrispondenza con l'espressione regolare '**`href=["'].*\.css["'].*$`**', la quale, in qualità di stringa, è delimitata da apici singoli, pertanto, gli apici singoli che appaiono al suo interno sono protetti da una barra obliqua inversa; inoltre, essendo un'espressione regolare, è delimitata ulteriormente, in questo caso dalla barra obliqua normale.

Le difficoltà maggiori nell'uso delle espressioni regolari in PHP, riguardano la protezione dei caratteri a causa del modo in cui si delimitano le stringhe, dal momento che questo comporta l'uso di sequenze di escape da inserire nelle espressioni regolari. Pertanto, quando si usano funzioni PHP per le espressioni regolari, prima vanno scritte le espressioni, poi vanno rielaborate in funzione dei delimitatori di stringa utilizzati.

```
preg_replace (regex, rimpiazzo, oggetto)
```

La funzione ***preg_replace()*** interviene su una stringa o su un array di stringhe (l'ultimo parametro), restituendo un risultato dello stesso tipo (stringa o array di stringhe), eseguendo una trasformazione in corrispondenza delle occorrenze dell'espressione regolare che costituisce il primo parametro, utilizzando come rimpiazzo il secondo parametro.

```
$nuova = preg_replace ('/(href=["\'])(.*\.css["\'].*)$/',
                       '${1}http://mio.dominio.it${2}',
                       $riga);
```

Nell'esempio si vede che l'espressione regolare indicata come primo argomento, contiene delle parentesi tonde, con le quali si delimitano due porzioni. Nella stringa di rimpiazzo, si fa riferimento alla corrispondenza delle due porzioni con delle metavariabili (relative all'espressione regolare), indicate come '**${1}**' e '**${2}**'. Lo scopo dell'esempio è quello di rimpiazzare il percorso di un file che si presume relativo, con l'aggiunta del protocollo e del nome a dominio.

```
preg_grep (regex, array)
```

La funzione ***preg_grep()*** scandisce l'array di stringhe fornito come secondo argomento, alla ricerca della corrispondenza con l'espressione regolare indicata come primo argomento, restituendo un array di stringhe che contiene gli elementi del primo che hanno una corrispondenza positiva.

```
preg_split (regex, stringa)
```

La funzione ***preg_split()*** restituisce un array di stringhe, ottenuto spezzando la stringa fornita come secondo argomento, dove l'espressione regolare fornita come primo argomento trova una corrispondenza.

41.15 Accesso a basi di dati MySQL

Con il linguaggio PHP è possibile accedere a diversi tipi di DBMS, ma quello a cui il PHP è stato abbinato storicamente è MySQL e generalmente tutte le configurazioni comuni dell'interprete PHP hanno la disponibilità di almeno un accesso a una base di dati MySQL. Pertanto, anche se ci possono essere ragioni importanti per preferire DBMS diversi, sul piano tecnico, sul piano della licenza o su quello della fiducia nei confronti di chi ne detiene i diritti, MySQL rimane la prima scelta per il PHP.

Per poter accedere a una base di dati è necessario che sia instaurata una connessione con il servente MySQL, attraverso la funzione ***mysql_connect()***, la quale va usata preferibilmente con gli argomenti di questo modello:

```
mysql_connect (nodo_e_porta, utente, parola_d'ordine);
```

La funzione restituisce un valore che serve a identificare la connessione instaurata, oppure il valore *Falso* in caso di fallimento dell'operazione.

```
$link = mysql_connect ("127.0.0.1:3306", "tizio",
                       "miapassword");
if (!$link)
  {
    echo ("<p>Non riesco a connettermi al DBMS!</p>\n");
  }
```

L'esempio mostra un tentativo di collegamento a un servente MySQL presso l'elaboratore locale, in ascolto alla porta 3306, la quale dovrebbe essere quella predefinita, in qualità di utente '**`tizio`**' (utente del DBMS), con la parola d'ordine '**`miapassword`**'. Se il collegamento fallisce si produce un avvertimento.

Il nodo a cui ci si deve connettere può essere indicato anche per nome, se esiste un nome a dominio valido; inoltre il numero di porta può essere omesso (in tal caso si tolgono anche i due punti separatori).

La funzione ***mysql_connect()*** ha di buono che può essere richiamata quante volte si vuole, ma se gli argomenti della chiamata sono gli stessi (oppure se sono omessi), queste chiamate ridondanti non vanno a creare connessioni ulteriori, in quanto si limitano a confermare quella già in essere.

Dopo la connessione al DBMS si deve pensare alla selezione della base di dati, con la funzione ***mysql_select_db()***:

```
mysql_select_db (nome_db [, connessione]);
```

Come si vede dal modello sintattico, è necessario indicare il nome della base di dati a cui ci si vuole collegare, mentre è possibile indicare il riferimento alla connessione (il DBMS) a cui si fa riferimento.

In mancanza dell'indicazione esplicita della connessione, si intende fare riferimento all'ultima connessione attivata.

```
$result = FALSE;
$link = mysql_connect ("127.0.0.1:3306", "tizio",
                       "miapassword");
if ($link)
  {
    $result = mysql_select_db ("db_1");
    if (!$result)
      {
        echo ("<p>Non riesco ad accedere "
             ."alla base di dati!</p>\n");
      }
  }
```

La funzione ***mysql_select_db()*** restituisce un valore logico, pari a *Vero* se tutto è andato bene, o pari a *Falso* in caso di problemi. L'esempio appena apparso mette in evidenza questo fatto.

La fase successiva consiste nello scrivere un comando SQL, da impartire attraverso la funzione ***mysql_query()***:

```
mysql_query (interrogazione);
```

```
$result = FALSE;
$result = mysql_query ("SELECT * FROM Articoli "
                       ."WHERE Listino >= 1");
```

L'esempio mostra una situazione molto semplice, con la quale si esegue il comando SQL '**`SELECT * FROM Articoli WHERE Listino >= 1;`**'. L'esito di questa interrogazione viene raccolto dalla variabile ***$result***, la quale contiene il valore *Falso* se l'operazione fallisce. In questo caso, il comando SQL dovrebbe produr-

re le tuple della tabella '**Articoli**' che corrispondono alla condizione posta, ma per leggere questi dati, occorre una fase successiva. Entrano in gioco, a questo punto, due funzioni importanti: ***mysql_num_rows()*** e ***mysql_fetch_assoc()***.

```
mysql_num_rows (risorsa);
```

La funzione ***mysql_num_rows()*** riceve come argomento l'esito di un'interrogazione SQL, prodotto attraverso la funzione ***mysql_query()***, restituendo la quantità di righe ottenute:

```
$result = FALSE;
$righe = 0;
$result = mysql_query ("SELECT * FROM Articoli "
                       ."WHERE listino >= 1");
if (!$result)
  {
    echo ("<p>La lettura della tabella è fallita!</p>\n");
  }
else
  {
    $righe = mysql_num_rows ($result);
    echo ("<p>Ho letto $righe righe.</p>\n");
  }
```

Come si vede nell'esempio, dopo l'interrogazione SQL si valuta se l'esito è valido; se lo è, si verifica la quantità di righe ottenute che viene inserita nella variabile ***$righe***.

```
mysql_fetch_assoc (risorsa);
```

La funzione ***mysql_fetch_assoc()*** permette di leggere, una riga al-

la volta, quanto ottenuto attraverso un'interrogazione SQL eseguita con la funzione ***mysql_query()***. La riga letta viene resa in forma di array associativo, in cui l'indice di accesso è costituito dal nome della colonna. Quando la lettura termina, la funzione restituisce il valore *Falso*.

```
$result = FALSE;
$righe  = 0;
$riga   = "";
$result = mysql_query ("SELECT * FROM Articoli "
                       ."WHERE listino >= 1");
if (!$result)
  {
    echo ("<p>La lettura della tabella è fallita!</p>\n");
  }
else
  {
    $righe = mysql_num_rows ($result);
    echo ("<p>Ho letto $righe righe.</p>\n");
    while ($riga = mysql_fetch_assoc ($result))
      {
        echo ("<p>articolo ".$riga["codice"]
              ." "
              .$riga["descrizione"]
              .", prezzo: "
              .$riga["prezzo"]
              ."</p>");
      }
  }
```

Come si vede, la funzione ***mysql_fetch_assoc()*** viene usata in un ciclo, fino a quando restituisce un'informazione valida. Si presume che la tabella che è stata oggetto dell'interrogazione contenga le colonna ***codice***, ***descrizione*** e ***prezzo*** (oltre a ***listino*** che viene usata

per la condizione di selezione delle tuple).

41.16 Il problema dell'iniezione di codice SQL

«

Il PHP è un linguaggio interpretato che consente di espandere le variabili all'interno delle stringhe; per esempio, consente di scrivere codice di questo tipo:

```
$tvb = "ti voglio bene";
echo ("Ma lo sai che $tvb?");
```

Si comprende che l'esito della funzione ***echo()*** è la frase completa: «Ma lo sai che ti voglio bene?». In generale questo è un fatto positivo, ma diventa un problema quando si lavora con la funzione ***mysql_query()***, quando il comando SQL viene costruito a partire da dati immessi dagli utenti.

```
$comando = "";
$result = FALSE;
$codice = "q123";
$comando = "SELECT * FROM Articoli WHERE codice = '$codice'";
$result = mysql_query ($comando);
```

In questo esempio, alla fine viene eseguito il comando SQL '**`SELECT * FROM Articoli WHERE codice = 'q123';`**', senza alcun problema particolare. Tuttavia, se il codice che si cerca provenisse dall'esterno, si potrebbe produrre qualcosa di non desiderabile:

```
// La variabile $comando contiene la stringa seguente:
//
//     %' AND descrizione='d%
//
$comando = "SELECT * FROM Articoli WHERE codice = '$codice'";
$result = mysql_query ($comando);
```

In questo caso, il comando che viene dato effettivamente diventa '**SELECT * FROM Articoli WHERE codice = '%' AND descrizione='d%';**'. In pratica, una ricerca che era intesa da svolgersi con il riferimento al codice, diventa una ricerca basata sulla descrizione. L'esempio in sé non mostra nulla di così pericoloso, ma serve a far capire che c'è sempre il rischio che i comandi SQL vengano trasformati in qualcosa di non desiderabile. Per evitare questo problema, occorre produrre la codifica in modo appropriato.

D'altra parte, anche senza voler considerare la malizia umana, occorre considerare che i comandi SQL sono scritti secondo una sintassi che prevede la delimitazione di alcune stringhe e la protezione di caratteri che altrimenti verrebbero interpretati con significati particolari. Per esempio, il codice articolo cercato, potrebbe contenere il carattere apostrofo:

```
$codice = "q'123";
$comando = "SELECT * FROM Articoli WHERE codice = '$codice'";
$result = mysql_query ($comando);
```

In questo caso, il comando SQL risulterebbe errato, perché il codice avrebbe dovuto essere scritto come '**q\'123**'.

Per prima cosa è bene evitare l'espansione delle variabili nelle stringhe che servono a costruire i comandi SQL. Per questo si può usare il concatenamento di stringa:

```
$comando = "SELECT * FROM Articoli ".
           "WHERE codice = '".$codice."'";
```

Oppure, si può usare la funzione ***sprintf()*** (equivalente a quella con lo stesso nome dello standard C) che rende il procedimento ancora più chiaro:

```
$comando = sprintf ("SELECT * FROM Articoli "
                    ."WHERE codice = '%s'",
                     $codice);
```

Poi occorre trattare i dati da immettere in un comando SQL in modo che ottengano la protezione dei caratteri che non possono essere rappresentati, tali e quali, nelle stringhe SQL:

```
$codice = "q'123";
$comando = sprintf ("SELECT * FROM Articoli "
                    ."WHERE codice = '%s'",
                     mysql_real_escape_string ($codice));
```

La funzione ***mysql_real_escape_string()*** ha quindi lo scopo di trasformare la stringa ricevuta come argomento, in modo da poter essere inserita all'interno della delimitazione con apici singoli dei comandi SQL.

41.17 GWADM

«

GWADM è un servizio per la didattica, attraverso il quale è possibile esercitarsi nella realizzazione di applicazioni in PHP, senza dover installare nulla in un elaboratore locale.

Il servizio riconosce gli accessi in base al sistema Openid di Google, pertanto, lo si può utilizzare solo se si dispone di un'utenza Google. Tuttavia, una volta entrati nella gestione di GWADM, i programmi che si realizzano in PHP potrebbero interferire con tutto il servizio, sia con quanto fatto da altre persone, sia con il programma che costituisce GWADM, perché i privilegi efficaci sono gli stessi per tutti.

Per la debolezza descritta, si tratta di un servizio puramente didattico, dove chi lo utilizza deve avere l'accortezza e il rispetto necessari, nei confronti di tutti gli utilizzatori; ma va anche considerato il rischio di perdere il lavoro a causa di un'aggressione al sistema stesso.

GWADM può essere installato in un proprio servente HTTP+PHP, prelevando il pacchetto da *https://docs.google.com/open?id=0B7kc1cYTL1pjOWs1U1E3NTN5MjA* [2].

GWADM si mostra come un pannello che elenca il contenuto di una cartella. La prima cartella che viene mostrata è quella principale dell'utente. I vari componenti evidenziati nella figura sono:

1. nominativo e indirizzo di posta elettronica dell'utente;
2. spazio disponibile e spazio utilizzato attualmente dall'utente;

3. riferimento ipertestuale per visualizzare la pagina principale dell'utente (*home page*);
4. riferimento ipertestuale per visualizzare l'elenco dei registri degli accessi;
5. bottone per visualizzare la configurazione di PHP;
6. bottone per richiedere l'uscita dalla sessione di lavoro;
7. barra per indicare manualmente la cartella nella quale si vuole operare (deve trovarsi all'interno del percorso a cui è abbinato l'utente);
8. barra per la selezione e il caricamento di un file;
9. barra per la creazione di un file o di una cartella;
10. elenco del contenuto della cartella corrente (quella indicata nel punto 7);
11. tendina con le azioni disponibili per i file e le cartelle selezionate eventualmente dall'elenco;
12. bottone ▷ per procedere con il comando relativo ai file selezionati dall'elenco;
13. permessi di accesso di file e cartelle (per modificare un permesso basta un clic sullo stesso);
14. tendina con le azioni disponibili per una singola voce dell'elenco;
15. bottone ▷ per procedere con il comando relativo al file selezionato o alla cartella selezionata;

Nell'elenco, la dimensione che appare a fianco delle cartelle, rappresenta lo spazio utilizzato complessivamente al loro interno.

Si può osservare che il servizio è fatto prevalentemente per creare e modificare file, direttamente, senza l'ausilio di un'applicazione locale. Pertanto, il caricamento dei file è ammesso solo singolarmente, mentre è possibile scaricare gruppi di file e di cartelle, impacchettati in un archivio ZIP.

41.18 Riferimenti

«

- *PHP documentation*, *http://php.net/doc.php*
- Gianluca Giusti, *Programmare in PHP*, 2003, *http://www.urcanet.it/brdp/php_manual/*

[1] **PHP** PHP license

[2] Se questo riferimento non dovesse funzionare, si veda la pagina *http://informaticalibera.net* .