Pagini WEB cu PHP 4

Mihai Scorțaru, Claudiu Soroiu

Acest articol este dedicat funcțiilor, claselor și obiectelor PHP. Vor fi prezentate detalii referitoare la modul în care pot fi utilizate aceste elemente ale limbajului.

Funcții

În *PHP* 4 funcțiile pot fi definite de către utilizator folosind următoarea sintaxă:

```
function numef ($param1, $param2, ..., $paramN){
  //instrucțiuni
}
```

În *PHP* 4 o funcție poate fi definită oriunde în cadrul *script*-ului și în interiorul unei funcții poate să apară orice secvență validă de cod care include definirea de alte funcții (similar limbajului *Pascal*) și definiții de clase.

Argumentele unei funcții trebuie separate prin virgulă, și, implicit, acestea sunt transmise prin valoare.

Pentru ca funcția să returneze un rezultat se folosește construcția **return** care primește ca parametru o expresie care reprezintă valoarea funcției. În momentul în care este întâlnită construcția **return**, execuția funcției se încheie (similar limbajelor *Java* și C/C++).

Următorul exemplu reprezintă o funcție simplă în *PHP* care calculează pătratul unui număr, iar în figura 1 se poate observa rezultatul executării acestui script:

```
<HTML>
  <HEAD>
    <TITLE>Exemplu funcție simplă</TITLE>
  </HEAD>
  <BODY>
    <?php
      function patrat($n){
        return $n*$n;
      }
      echo "4^2=<B>".patrat(4)."</B>";
    ?>
  </BODY>
</HTML>
```

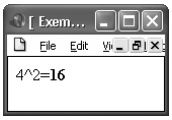

Figura 1

Din exemplul anterior rezultă că o funcție în *PHP* 4 poate primi ca parametru valori constante ca și în cazul majorității limbajelor de programare moderne.

Transmiterea parametrilor prin referință

Pentru a transmite parametri unei funcții prin referință, fapt care implică modificarea valorii parametrilor și păstrarea noii valori după ce execuția funcției s-a încheiat, se folosește operatorul "&" înaintea numelui parametrului formal, în momentul definirii funcției.

Următorul script *PHP* indică modul în care se modifică valoarea unei variabile în interiorul unei funcții și modul în care această modificare este resimțită în exteriorul acesteia:

```
<HTML>
  <HEAD>
    <TITLE>
      Exemplu modificare parametri
    </TITLE>
  </HEAD>
  <BODY>
    <?php
    function modificInt($s){
      $s.=" prima funcție.";
      echo "<B>În modificInt:</B> ".$s."<BR>";
    }
    function modificExt(&$s){
      $s.=" a doua funcție.";
      echo "<B>În modificExt:</B> ".$s."<BR>";
    }

    $s="Ieșire din";
    echo "<B>În script:</B> ".$s."<BR>";
    modificInt($s);
    echo "<B>În script:</B> ".$s."<BR>";
    modificExt($s);
    echo "<B>În
            script:</B>"
            .$s."<BR>";
    ?>
  </BODY>
</HTML>
```

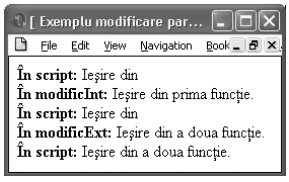

Figura 2

În figura 2 se poate observa efectul executării acestui *script PHP* 4.

Nu există posibilitatea de supraîncărcare a unei funcții, de redefinire a ei după ce aceasta a fost definită în cadrul scriptului respectiv și nu există nici posibilitatea de anulare a unei funcții. O funcție poate fi apelată înainte de definirea ei.

Parametri cu valori implicite

În *PHP* parametrii formali pot avea valori implicite, și, în cazul în care parametrul actual lipsește, atunci se va considera că are valoarea implicită.

Următorul exemplu ilustrează modul de folosire al funcțiilor când acestea au parametri formali cu valori implicite:

```
<HTML>
  <HEAD>
     <TITLE>Exemplu valori implicite</TITLE>
</HEAD>
  <BODY>
    <?php
      function comanda($s="cafea"){
        return "Ați comandat ".$s.".";
      }
      echo comanda();
      echo "<BR>";
      echo comanda("suc");
    ?>
  </BODY>
</HTML>
```

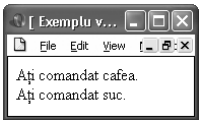

Figura 3

În figura 3 se poate observa rezultatul executării acestui script *PHP*.

În cazul în care se folosesc parametri cu valori implicite este necesar ca orice parametru care are o valoare implicită să se afle în partea dreaptă a tuturor parametrilor pentru care nu se folosesc valori implicite, în caz contrar interpretorul *PHP* nu poate să decidă cărui parametru să-i atribuie valoarea de pe o anumită poziție din lista cu parametri.

De exemplu, dacă avem o funcție a cărei declarație este **function** `transform($baza=10, $nr)` și care returnează rezultatul transformării lui `$nr` din baza 16 în baza `$baza`, a cărei valoare implicită este 10, dacă se apelează `transform(50)`, interpretorul nu atribuie valoarea 50 parametrului `$nr`, ci parametrului `$baza` și generează o eroare deoarece lipsește valoarea parametrului `$nr`.

Funcții cu număr variabil de parametri

O altă facilitate importantă a limbajului *PHP* este aceea că oferă programatorului posibilitatea de a utiliza funcții care au un număr nedeterminat de parametri. Funcțiile care folosesc un număr variabil de parametri nu au nici o particularitate în ceea ce privește definirea lor. Aceste funcții se definesc la fel ca cele prezentate anterior, dar pentru a putea accesa parametri se vor folosi următoarele funcții predefinite:

- `func_num_args()` - această funcție returnează numărul parametrilor funcției care a apelat-o; dacă această funcție este apelată din exteriorul unei funcții definite de utilizator se va genera un mesaj de avertizare;
- `func_get_arg(arg_num)` - returnează valoarea parametrului care se află pe poziția `arg_num` în lista de parametri; primul parametru are numărul de ordine 0; dacă este apelată din exteriorul unei funcții definite de utilizator se va genera un mesaj de avertizare;
- `func_get_args()` - returnează un tablou unidimensional care conține valorile parametrilor pe care funcția apelantă i-a primit; dacă această funcție este apelată din exteriorul unei funcții definite de utilizator se va genera un mesaj de avertizare.

În continuare vom prezenta exemple de utilizare a acestor funcții.

Primul exemplu afișează lista parametrilor funcției folosind funcțiile `func_num_args` și `func_get_arg`, iar al doilea exemplu afișează aceeași listă folosind numai funcția `func_get_args`.

```
<HTML>
  <HEAD>
    <TITLE>
      Exemplu parametri multipli
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      function lista_parametri(){
        for($i=0;$i<func_num_args();$i++){
          print_r(func_get_arg($i));
          echo "<BR>";
        }
      }
      echo lista_parametri("Comanda:",1,
        "calculator", 2, "procesoare",
        "configurație", array("local",2,3));
    ?>
  </BODY>
</HTML>
```

```
<HTML>
  <HEAD>
    <TITLE>
      Exemplu parametri multipli
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      function lista_parametri(){
        foreach(func_get_args() as $i){
          print_r($i);
```

```
            echo "<BR>";
          }
        }
        echo lista_parametri("Comanda:",1,
          "calculator", 2, "procesoare",
          "configurație", array("local",2,3));
    ?>
  </BODY>
</HTML>
```

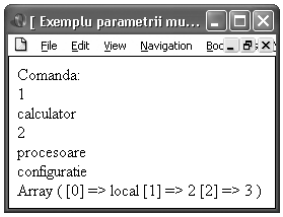

Figura 4

În figura alăturată se poate observa rezultatul executării celor două scripturi *PHP*.

Valorile returnate de funcții

Rezultatul obținut după apelarea unei funcții poate avea orice tip. O funcție poate să returneze chiar și liste sau obiecte.

În *PHP* există un caz special de rezultat numit referință. Pentru ca o funcție să poată returna o referință, aceasta trebuie declarată folosindu-se operatorul '&' înaintea numelui funcției. Acest operator trebuie să apară înaintea numelui funcției și în mometul când o variabilă primește ca valoare referința rezultată din apelul funcției.

În exemplul următor se definește o funcție al cărui rezultat îl constituie o referință:

```
<HTML>
  <HEAD>
    <TITLE>
      Exemplu rezultat de tip referință
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      function &refer(){
        global $s;
        return $s;
      }
      $s = "Acesta este conținutul variabilei
          referite cu ajutorul funcției.";
      $z = &refer();
      echo $z;
    ?>
  </BODY>
</HTML>
```

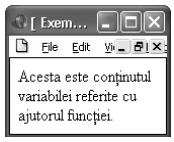

Figura 5

În figura 5 se poate observa rezultatul execuției acestui script *PHP*.

Spre deosebire de majoritatea limbajelor de programare moderne, o funcție *PHP* poate să returneze o referință la o variabilă care a fost declarată în interiorul funcției, însă acest lucru nu este indicat deoarece, în anumite cazuri, poate duce la efecte neașteptate ale executării unui script *PHP*. În alte limbaje de programare efectele devin uneori fatale.

Următorul script *PHP* arată unul dintre cazurile în care variabila a cărei referință este returnată este declarată în interiorul funcției:

```
<HTML>
  <HEAD>
    <TITLE>
      Exemplu #2 rezultat de tip referință
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      function &refer(){
        $s = "Acesta este conținutul variabilei
             referite cu ajutorul funcției.";
        return $s;
      }

      $z = &refer();
      echo $z;
    ?>
  </BODY>
</HTML>
```

Rezultatul executării acestui script este același cu cel din figura 5.

Variabile de tip funcție

O altă facilitate a limbajului *PHP* în ceea ce privește funcțiile este aceea că suportă variabilele de tip funcție. Acest lucru este util atunci când se folosesc liste de funcții pentru prelucrarea anumitor tipuri de date. Pentru a atribui un nume de funcție unei variabile în *PHP* se folosește aceeași construcție ca în cazul atribuirii unui șir de caractere, și anume, o variabilă va primi ca valoare numele funcției scris între ghilimele simple sau duble. În cazul în care interpretorul *PHP* găsește un nume de variabilă urmată de o listă de parametri, acesta caută funcția pe care variabila o referă și în cazul în care există, o execută. Variabilele de tip funcție nu funcționează cu construcții ale limbajului ca **echo**, **unset**, **isset**, **empty**, **include** etc.

Următorul script *PHP* ilustrează modul de lucru cu variabile de tip funcție, iar în figura 6 se poate observa rezultatul executării acestuia:

```
<HTML>
  <HEAD>
    <TITLE>
      Variabile de tip funcție
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      function produs($a, $b){
        return $a*$b;
      }
```

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

      $operatie = 'produs';
      $rez = $operatie(4, 5);
      echo "4 * 5 = <B>".$rez."</B><BR>";
      $operatie = 'suma';
      $rez = $operatie(4, 5);
      echo "4 + 5 = <B>".$rez."</B><BR>";
    ?>
  </BODY>
</HTML>
```

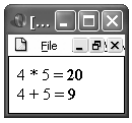

Figura 6

Clase și obiecte

În continuare vom presupune că paradigmele programării orientate pe obiecte sunt cunoscute, așadar nu vom prezenta aspectele teoretice referitoare la clase și obiecte, ci doar vom descrie modul în care acestea pot fi utilizate în *PHP*. Pe scurt, o clasă este o colecție de variabile și funcții care operează asupra variabilelor respective. Sintaxa folosită pentru declararea unei clase în *PHP*:

```
class nume_clasă{
  // date membre
  var nume_variabilă_1
  ...
  var nume_variabilă_m
  // metode
  function nume_funcție_1(parametri){
    ... // definiția funcției
  }
  ...
  function nume_funcție_n(parametri){
    ... // definiția funcției
  }
}
```

Pentru numele unei clase poate fi utilizat orice identificator permis în *PHP* cu o singură excepție: `stdclass`; acest identificator este folosit de *PHP* în scopuri interne.

În *PHP* funcțiile ale căror identificatori încep cu '__' sunt considerate funcții magice și utilizarea acestora nu este recomandată.

În *PHP*, datele membre nu pot fi inițializate decât cu valori constante. Pentru a inițializa variabilele cu valori care nu sunt constante trebuie folosit un constructor.

Așadar, nici una dintre următoarele inițializări nu este corectă:

```
class Nepermis{
  var $data = date("Y-m-d");
  var $nume = $prenume;
  var $dest = 'Mihai ' . 'Claudiu';
  var $obiecte = array("minge", "pantof");
}
```

Obiecte

În *PHP* clasele sunt considerate a fi tipuri de date; ele pot fi privite ca fiind "amprentele" variabilelor propriu-zise. Pentru a crea o variabilă al cărei tip este o clasă trebuie utilizat operatorul **new**.

În continuare vom defini o clasă `Aritmetica` cu două date membre `x` și `y` care sunt numere întregi și două metode care realizează adunarea, respectiv înmulțirea lor.

```
class Aritmetica{
  var x = 2;
  var y = 3;

  function Suma(){
    return $this->x + $this->y;
  }
  function Produs(){
    return $this->x * $this->y;
  }
}
```

Pentru a crea un obiect de tipul `Aritmetica` vom utiliza o instrucțiune de tipul:

```
$aritm = new Aritmetica;.
```

Acum putem utiliza metodele clasei; pentru a afișa suma sau produsul celor două numere vom putea apela cele două metode astfel:

```
echo $aritm->Suma();
echo $aritm->Produs();
```

Vom obține rezultatele `5`, respectiv `6`. Valorile datelor membre pot fi și ele modificate prin instrucțiuni de tipul:

```
$aritm->x = 5;
$aritm->y = 4;
```

Dacă, în urma modificării apelăm din nou metodele `Suma()` și `Produs()`, rezultatele vor fi `9`, respectiv `20`.

Vom prezenta în continuare un document *PHP* integral în care se definește o clasă `HelloWorld` care are ca date membre un șir de caractere care conține mesajul `Hello World!` și o metodă care returnează acest mesaj. Efectul *script*-ului este ilustrat în figura 7.

```
<HTML>
  <HEAD>
    <TITLE>Hello World!</TITLE>
  </HEAD>
  <BODY>
    <?PHP
      class HelloWorld{
        var $hello = "Hello World!";
        function Hello(){
          return $this->hello;
        }
      }
```

Figura 7

```
      $helloWorld = new HelloWorld;
      echo $helloWorld->hello()."<BR>";
    ?>
  </BODY>
</HTML>
```

În acest exemplu a fost utilizată pseudo-variabila **$this**. Aceasta este folosită pentru a indica faptul că se operează asupra unei date membre a obiectului curent.

Extinderea claselor

Deseori este necesară definirea unor clase cu proprietăți (date membre) și metode asemănătoare. Pentru a ușura definirea unor astfel de clase a fost introdus conceptul de *extindere* (derivare) a claselor.

O clasă derivată va păstra toate proprietățile și metodele clasei pe care o extinde și poate conține diferite proprietăți și metode noi.

Păstrarea de către clasa extinsă a metodelor și proprietăților clasei din care este derivată poartă denumirea de *moștenire*.

Clasa pe care o extinde o clasă derivată poartă denumirea de *clasă de bază* sau *clasă părinte*. Nu există nici o posibilitate de a elimina din clasa derivată anumite proprietăți sau metode ale clasei de bază.

O anumită clasă poate avea o singură clasă părinte; așadar, în *PHP* nu este permisă moștenirea multiplă.

Pentru a extinde o anumită clasă se utilizează cuvântul cheie **extends**.

În următorul exemplu vom extinde clasa `Aritmetica`; vom adăuga încă o variabilă și vom crea două noi funcții: una pentru calculul sumei celor trei variabile și una pentru calcularea produsului lor.

```
class Aritmetica3 extends Aritmetica{
  var z = 4;

  function Suma3(){
    return $this->x + $this->y + $this->z;
  }
  function Produs3(){
    return $this->x * $this->y * + $this->z;
  }
}
```

Dacă definim un obiect prin intermediul unei instrucțiuni de genul:

```
$aritm3 = new Aritmetica3;
```

atunci pentru acest obiect vom putea utiliza atât metodele definite în cadrul clasei `Aritmetica3`: `Suma3()` și `Produs3()`, cât și metodele definite în cadrul clasei de bază `Aritmetica`: `Suma()` și `Produs()`.

În continuare vom prezenta întregul document *PHP* care ilustrează modul în care pot fi create și utilizate clasele derivate. Efectul execuției acestui script este prezentat în figura 8.

```
<HEAD>
  <BODY>
    <?PHP
      class Aritmetica{
        var $x = 2;
        var $y = 3;
        function Suma(){
          return
            $this->x +
            $this->y;
        }
        function Produs(){
          return $this->x * $this->y;
        }
      }

      class Aritmetica3 extends Aritmetica{
        var $z = 4;
        function Suma3(){
          return $this->x + $this->y +
                                  $this->z;
        }
        function Produs3(){
          return $this->x * $this->y *
                                  $this->z;
        }
      }

      $aritm3 = new Aritmetica3;
      echo "<B>Înainte de modificare:</B>";
      echo "<BR>";
      echo "Suma primelor două numere: ";
      echo $aritm3->Suma()."<BR>";
      echo "Produsul primelor două numere: ";
      echo $aritm3->Produs()."<BR>";
      echo "Suma celor trei numere: ";
      echo $aritm3->Suma()."<BR>";
      echo "Produsul celor trei numere: ";
      echo $aritm3->Produs()."<BR>";
      $aritm3->x = 5;
      $aritm3->y = 4;
      $aritm3->z = 3;
      echo "<BR><BR>";
      echo "<B>După modificare:</B><BR>";
      echo "<BR>";
      echo "Suma primelor două numere: ";
      echo $aritm3->Suma3()."<BR>";
      echo "Produsul primelor două numere: ";
      echo $aritm3->Produs3()."<BR>";
      echo "Suma celor trei numere: ";
      echo $aritm3->Suma3()."<BR>";
      echo "Produsul celor trei numere: "
      echo $aritm3->Produs3()."<BR>";
    ?>
  </BODY>
</HTML>
```

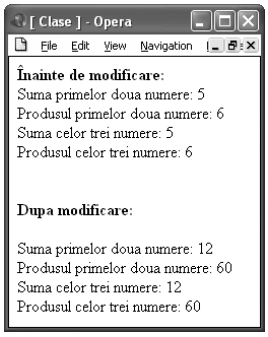

Figura 8

Relația dintre o clasă derivată și clasa pe care o extinde poartă denumirea de ***relație părinte - fiu***. În *PHP* clasele trebuie definite înaintea utilizării lor; așadar clasa părinte va fi definită întotdeauna înaintea clasei fiu.

Constructori

Un constructor este o metodă (funcție) a unei clase care este apelată automat în momentul în care este creată o nouă instanță a clasei (cu ajutorul operatorului **new**).

În *PHP* este considerată ca fiind un constructor orice funcție care are același nume cu clasa în interiorul căreia este definită.

Constructorii pot fi folosiți pentru inițializarea datelor membre cu valori care nu sunt constante. Ei pot avea argumente, iar acestea pot fi opționale. Pentru a putea utiliza clasa fără a specifica nici un parametru în momentul creării unui obiect, se recomandă stabilirea unor valori implicite pentru toate argumentele constructorului.

În cazul în care nu este definit un constructor pentru o anumită clasă, se utilizează constructorul clasei de bază, dacă acesta există.

De exemplu, pentru următoarea secvență de cod, în momentul creării obiectului corespunzător variabilei `$b`, va fi apelat constructorul clasei `A`.

```
class A{
  function A(){
    echo "Constructorul clasei A<BR>";
  }
  function B(){
    echo "O funcție obișnuită a clasei A.<BR>";
  }
}

class B extends A{
  function C(){
    echo "O funcție obișnuită a clasei B.<BR>";
  }
}

$b = new B;
```

În mai multe limbaje de programare constructorul unei clase derivate apelează automat constructorul clasei de bază. În *PHP* apelul constructorului clasei de bază trebuie să fie explicit dacă este necesară executarea operațiilor corespunzătoare.

În majoritatea limbajelor de programare există funcții speciale numite destructori care sunt apelate automat în momentul "distrugerii" unui obiect. În *PHP* nu există destructori.

Operatorul ::

Uneori este utilă folosirea unor metode sau variabile ale clasei de bază sau ale unei clase care nu a fost instanțiată încă. În acest scop a fost introdus operatorul `::`.

Pentru a descrie modul de utilizare al acestui operator vom prezenta mai întâi un exemplu:

```
class A{
  function exemplu(){
    echo "Funcția clasei de bază.<BR>";
  }
}

class B extends A{
  function exemplu(){
    echo "Funcția redefinită<BR>\n";
    A::exemplu();
  }
}

A::exemplu();
$b = new B;
$b->exemplu();
```

Prin intermediul instrucțiunii `A::exemplu();` este apelată metoda `exemplu()` a clasei `A`, așadar se afișează mesajul `Funcția clasei de bază` cu toate că nu există nici un obiect care este o instanță a acestei clase, deci nu putem scrie o instrucțiune de tipul `$a->exemplu();`. În schimb apelăm metoda `$b->exemplu();` ca "o funcție a clasei" și nu ca "o funcție a unui obiect".

Putem avea funcții ale claselor, dar nu putem avea variabile ale claselor. De fapt, în momentul unui astfel de apel nu se creează nici un obiect care este instanță a clasei respective. Ca urmare, o funcție a unei clase nu poate opera asupra unor proprietăți ale clasei, dar poate utiliza variabile locale sau globale. În plus, o astfel de funcție nu poate utiliza pseudo-variabila **`$this`**.

În exemplul anterior, în cadrul clasei `B` este redefinită funcția `exemplu()`. Așadar, definiția "originală" (din cadrul clasei `A`) nu poate fi accesată în interiorul clasei `B` decât dacă ne referim la ea explicit prin intermediul operatorului `::`.

Accesarea clasei de bază

În exemplul anterior am utilizat o funcție a clasei de bază. În locul utilizării denumirii clasei de bază poate fi folosită denumirea specială `parent` care este o referință la clasa de bază definită în cadrul construcției **`extends`**.

Folosirea denumirii speciale este utilă în cazul în care ierarhia de clase se modifică. În acest caz este suficientă o singură modificare în cadrul construcției **`extends`**, fără a mai fi necesare modificări în interiorul clasei derivate.

Așadar, definiția clasei `B` poate fi rescrisă astfel:

```
class B extends A{
  function exemplu(){
    echo "Funcția redefinită<BR>\n";
    parent::exemplu();
  }
}
```

Serializarea obiectelor

Prin serializare se înțelege crearea unui șir de octeți care conține reprezentarea internă (binară) a variabilei respective. Așadar, serializarea permite "salvarea" valorilor unei variabilei. Dacă este serializat un obiect sunt salvate doar proprietățile acestuia (variabilele membre) și numele clasei din care face parte, nu și metodele deoarece funcțiile nu reprezintă valori. Pentru a serializa un obiect este utilizată funcția `serialize()` care returnează șirul de octeți care conține reprezentarea binară.

Pentru a deserializa un obiect se folosește funcția pereche `unserialize()`. Pentru ca o astfel de operație să funcționeze corect este necesară definirea clasei din care face parte obiectul respectiv.

Funcția returnează valoarea variabilei serializate. În continuare vom prezenta un exemplu care ilustrează modul în care poate fi serializat și deserializat un obiect. Șirul de octeți obținut în urma serializării va fi scris într-un fișier și va fi citit din fișierul respectiv pentru efectuarea deserializării. De obicei serializarea și deserializarea sunt realizate în documente *PHP* diferite deoarece aceste operații nu au aproape nici o utilitate dacă sunt folosite în cadrul aceluiași document.

Documentul *PHP* în care se realizează serializarea trebuie să conțină o secvență asemănătoare cu următoarea:

```
class A{
  var $msg = "Hello World!";
  function scrie(){
    echo $this->msg;
  }
}
$a = new A;
$s = serialize($a);
//salvarea șirului într-un fișier
$fp = fopen("fisier", "w");
fputs($fp, $s);
fclose($fp);
```

Pentru deserializare al doilea document va conține următoarea secvență:

```
class A{
  var $msg = "Hello World!";
  function scrie(){
    echo $this->msg;
  }
}
//citirea șirului din fișier
$s = implode("", @file("fisier"));
$a = unserialize($s);
//după deserializare obiectul poate fi folosit
$a->show_one();
```

Referințe în PHP

Referințele pot fi utilizate pentru a accesa conținutul unei variabile folosind mai multe nume. Spre deosebire de limbajul *C*, în *PHP* referințele nu sunt pointeri, ci *alias*-uri într-o tabelă de simboluri.

În *PHP* denumirile variabilelor și conținutul acestora nu sunt unul și același lucru. Așadar este posibil ca același conținut să aibă denumiri diferite.

Utilizarea referințelor

Referințele *PHP* permit unor variabile cu denumiri diferite să corespundă unui același conținut. Cu alte cuvinte, instrucțiunea `$a = &$b` are ca efect faptul că `$a` și `$b` referă aceeași variabilă.

În această situație `$a` și `$b` au același statut. Nu se poate spune că `$a` referă `$b` sau invers.

O altă posibilitate de utilizare a referințelor este transmiterea prin referință a parametrilor unei funcții. Efectul unei astfel de transmisii este crearea unei variabile locale care referă spre același conținut ca variabila din contextul apelant.

Să considerăm următorul exemplu:

```
function inc(&$var){
  $var++;
}
$a=5;
inc($a);
```

Inițial valoarea variabilei `$a` este `5`. După apel variabila locală `$var` și variabila din contextul apelant `$a` indică spre același conținut. Valoarea păstrată în locația de memorie respectivă este incrementată (devine `6`) prin intermediul instrucțiunii `$var++;`. Datorită faptului că cele două variabile au același conținut, valoarea variabilei `$a` va fi `6` după executarea funcției.

Un parametru transmis prin referință poate fi:

- o variabilă;
- o instrucțiune **new**;
- o referință returnată de o funcție.

Dacă unei astfel de funcții i se transmite ca parametru un alt tip de expresie rezultatul este nedefinit. Așadar, pentru o funcție care are un parametru transmis prin referință nu se poate folosi o constantă în momentul apelului. De exemplu, pentru funcția `inc()` prezentată anterior nu este permis un apel de forma `inc(5)`.

Referințe globale

În momentul declarării unei variabile globale (printr-o instrucțiune de tipul **`global`** `$var`) se creează de fapt o referință spre o variabilă globală.

Cu alte cuvinte, această instrucțiune este echivalentă cu `$var = &$GLOBALS["var"];`.

Referința $this

În cadrul unei metode a unui obiect **`$this`** este întotdeauna o referință spre obiectul care utilizează funcția (obiectul curent).

Mihai Scorțaru este redactor-șef GInfo și poate fi contactat prin e-mail la adresa skortzy@yahoo.com. *Claudiu Soroiu este redactor GInfo și poate fi contactat prin e-mail la adresa* csoroiu@yahoo.com.