Pagini WEB cu PHP 4

Mihai Scorțaru, Claudiu Soroiu

În cadrul acestui articol vom prezenta operatorii pe care îi avem la dispoziție în PHP, precum și cele mai importante structuri de constrol pe care le putem folosi.

Operatori în PHP

Interpretorul *PHP* permite folosirea a nouă tipuri diferite de operatori. Aceștia operează asupra unor expresii (una, două sau trei) și furnizează ca rezultat o altă expresie (rezultatul operației corespunzătoare).

Operatori aritmetici

Toți operatorii aritmetici operează asupra a două expresii. (operanzi). Există cinci astfel de operatori:

- adunare (`'+'`);
- scădere (`'-'`);
- înmulțire (`'*'`);
- împărțire (`'/'`);
- rest (`'%'`).

Aceste operații se aplică asupra unor valori care reprezintă tipuri numerice. Dacă unul dintre operanzi nu are tip numeric, atunci el va fi convertit automat la o valoare întreagă.

Dacă cel puțin unul dintre operanzi (după efectuarea eventualelor conversii necesare) este un număr real, atunci rezultatul operației va fi tot un număr real, cu excepția operatorului rest; rezultatul operației de determinare a restului este întotdeauna un număr întreg. Dacă ambii operanzi sunt numere întregi, atunci rezultatul va fi un număr întreg, cu excepția împărțirii al cărei rezultat este întotdeauna un număr real.

Dacă operatorul rest este aplicat asupra unor numere reale, atunci rezultatul este un număr întreg.

Formula de calcul a restului împărțirii celor două numere este $a \% b = \mathbf{sgn}(a) \cdot \mathbf{sgn}(b) \cdot [\{|a| / |b|\} \cdot |b|]$, unde:

- $|x|$ reprezintă modulul valorii x; funcția modul aplicată asupra unei valori x furnizează ca rezultat valoarea x, dacă x este un număr pozitiv și valoarea $-x$, dacă x este un număr negativ;
- $\mathbf{sgn}(x)$ reprezintă semnul valorii x; funcția semn aplicată asupra unei valori x furnizează ca rezultat valoarea 1 dacă x este un număr pozitiv, valoarea -1 dacă x este un număr negativ și valoarea 0 dacă x are valoarea 0;
- $[x]$ reprezintă partea întreagă a valorii x, adică cel mai mare număr întreg care este mai mic sau egal cu x;
- $\{x\}$ reprezintă partea fracționară a valorii x, adică diferența dintre x și partea sa întreagă.

Nu este permisă împărțirea la valoarea 0. Așadar, dacă al doilea operand asupra căruia se aplică unul dintre operatorii `'/'` sau `'%'` are valoarea 0, va fi semnalată o eroare.

Prezentăm în continuare un document *PHP* care exemplifică utilizarea acestor operatori asupra unor numere întregi și asupra unor numere reale. Pagina *Web* generată este prezentată în figura 1.

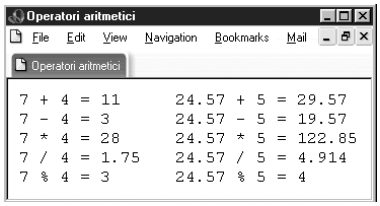

Figura 1

```
<HTML>
  <HEAD>
    <TITLE>
      Operatori
      aritmetici
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      echo "<TABLE><TR><TD width = 150>";
      $a = 7;
      $b = 4;
      echo "<TT>$a + $b = ";
      echo $a + $b;
      echo "<BR>$a - $b = ";
      echo $a - $b;
      echo "<BR>$a * $b = ";
      echo $a * $b;
      echo "<BR>$a / $b = ";
      echo $a / $b;
      echo "<BR>$a % $b = ";
      echo $a % $b;
      echo "</TT></TD><TD>";
      $a = 24.57;
      $b = 5;
      echo "<TT>$a + $b = ";
      echo $a + $b;
      echo "<BR>$a - $b = ";
      echo $a - $b;
      echo "<BR>$a * $b = ";
```

```
        echo $a * $b;
        echo "<BR>$a / $b = ";
        echo $a / $b;
        echo "<BR>$a % $b = ";
        echo $a % $b;
        echo "</TT></TD></TR></TABLE>";
      ?>
    </BODY>
</HTML>
```

Operatori pentru șiruri de caractere

Singurul operator care poate fi aplicat asupra șirurilor de caractere este cel de concatenare (`'.'`). Rezultatul concatenării a două șiruri de caractere este un șir de caractere care conține primul șir, urmat de al doilea.

Un exemplu de utilizare a acestui operator este prezentat în continuare (pagina *Web* generată este cea din figura 2):

```
<HTML>
  <HEAD>
    <TITLE>
      Operatorul de concatenare
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $a = "Hello ";
      $b = "World!";
      echo $a.$b;
    ?>
  </BODY>
</HTML>
```

Figura 2

Operatori pe biți

Interpretorul *PHP* pune la dispoziție șase operatori care operează asupra biților unui număr întreg sau ai unui șir de caractare. Aceștia sunt:

- conjucție (`'&'`) - *ȘI* (*AND*);
- disjuncție (`'|'`) - *SAU* (*OR*);
- disjuncție exclusivă (`'^'`) - *SAU* exclusiv (*XOR*);
- negație (`'~'`) - *NU* (*NOT*);
- deplasare la stânga (`'<<'`);
- deplasare la dreapta (`'>>'`).

Operatorul `'&'` are ca rezultat o valoare întreagă în care un anumit bit este setat pe valoarea 1 dacă ambii biți corespunzători ai celor doi operanzi sunt setați pe valoarea 1. Toți ceilalți biți vor avea valoarea 0.

Operatorul `'|'` are ca rezultat o valoare întreagă în care un anumit bit este setat pe valoarea 1 dacă cel puțin unul dintre biții corespunzători ai celor doi operanzi este setat pe valoarea 1. Toți ceilalți biți vor avea valoarea 0.

Operatorul `'^'` are ca rezultat o valoare întreagă în care un anumit bit este setat pe valoarea 1 dacă exact unul dintre biții corespunzători ai celor doi operanzi este setat pe valoarea 1. Toți ceilalți biți vor avea valoarea 0.

Operatorul `'~'` are ca rezultat o valoare întreagă în care un anumit bit este setat pe valoarea 1 dacă bitul corespunzător operandului este setat pe valoarea 0. Toți ceilalți biți vor avea valoarea 0.

Operatorul `'<<'` are ca rezultat o valoare întreagă în care biții primului operand sunt deplasați la stânga cu numărul de poziții indicat de al doilea operand. Pozițiile de la sfârșitul reprezentării sunt completate cu zerouri.

Operatorul `'>>'` are ca rezultat o valoare întreagă în care biții primului operand sunt deplasați la dreapta cu numărul de poziții indicat de al doilea operand. Pozițiile de la începutul reprezentării sunt completate cu valoarea bitului semn.

Pentru operatorii de deplasare, dacă al doilea operand are valoarea mai mare decât numărul de biți folosiți pentru reprezentarea numerelor întregi, atunci numărul pozițiilor cu care sunt deplasați biții primului operand va fi dat de restul împărțirii valorii celui de-al doilea operand la numărul biților folosiți pentru reprezentare. Dacă valoarea celui de-al doilea operand este negativă, atunci numărul de poziții va fi dat de restul împărțirii valorii celui de-al doilea operand la numărul biților folosiți pentru reprezentare la care se adună numărul biților folosiți pentru reprezentare.

Pentru a clarifica modul în care se calculează numărul pozițiilor, vom prezenta câteva exemple. Menționăm că în acest moment în versiunea *PHP* 4.3.0 pentru *Windows* (cea folosită pentru testarea documentelor *PHP* prezentate în cadrul acestui articol) sunt folosiți 32 de biți pentru reprezentarea numerelor întregi, dintre care unul este bitul semn.

- expresia `$a << 36` este echivalentă cu expresia `$a << 4`, deoarece `36 % 32 = 4`;
- expresia `$a << -52` este echivalentă cu expresia `$a << 12`, deoarece `-52 % 32 + 32 = -20 + 32 = 12`.

Dacă nu au loc depășiri de reprezentare, o deplasare la stânga cu p poziții este echivalentă cu o înmulțire cu valoarea 2^p, iar o deplasare la dreapta cu p poziții este echivalentă cu o împărțire întreagă la valoarea 2^p.

Dacă un operand asupra căruia acționează un operator pe biți este număr real, acesta este convertit la o valoare întreagă.

În cele ce urmează prezentăm un document *PHP* în care sunt utilizați cei șase operatori pe biți prezentați. Pagina *Web* generată este ilustrată în figura 3.

```
<HTML>
  <HEAD>
    <TITLE>
      Operatori pe biti
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $a = 14;
      $b = 3;
```

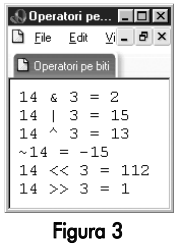

Figura 3

```
        echo "<TT>$a & $b = ";
        echo $a & $b;
        echo "<BR>$a | $b = ";
        echo $a | $b;
        echo "<BR>$a ^ $b = ";
        echo $a ^ $b;
        echo "<BR>~$a = ";
        echo ~$a;
        echo "<BR>$a << $b = ";
        echo $a << $b;
        echo "<BR>$a >> $b = ";
        echo $a >> $b;
        echo "</TT>";
    ?>
  </BODY>
</HTML>
```

Dacă operanzii asupra cărora se aplică operatori logici sunt șiruri de caractere, atunci operațiile sunt efectuate asupra codurilor *ASCII* ale fiecărui caracter în parte. Dacă în documentul *PHP* anterior înlocuim instrucțiunile

```
$a = 14;
$b = 3;
```

cu instrucțiunile

```
$a = "Hello";
$b = "World";
```

atunci pagina *Web* generată va fi cea din figura 4.

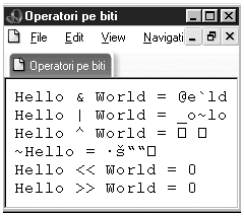

Figura 4

Operatori de comparare

Interpretorul *PHP* pune la dispoziție mai mulți operatori care pot fi folosiți pentru compararea a două valori. Expresiile în care apar astfel de operatori au ca rezultat valori logice (**true** sau **false**).

- `'=='` - rezultatul este **true** dacă cele două expresii au aceeași valoare;
- `'==='` - rezultatul este **true** dacă cele două expresii au aceeași valoare și au același tip;
- `'!='` sau `'<>'` - rezultatul este **true** dacă cele două expresii au valori diferite;
- `'!=='` - rezultatul este **true** dacă cele două expresii au valori diferite sau au tipuri diferite;
- `'<'` - rezultatul este **true** dacă valoarea primei expresii este mai mică decât valoarea celei de-a doua expresii;
- `'>'` - rezultatul este **true** dacă valoarea primei expresii este mai mare decât valoarea celei de-a doua expresii;
- `'<='` - rezultatul este **true** dacă valoarea primei expresii este mai mică sau egală cu valoarea celei de-a doua expresii;
- `'>='` - rezultatul este **true** dacă valoarea primei expresii este mai mare sau egală cu valoarea celei de-a doua expresii.

Pe lângă aceștia există și operatorul condițional `"?"` care are forma *expresie*1`?`*expresie*2`:`*expresie*3 și are ca rezultat valoarea expresiei *expresie*2 dacă valoarea expresiei *expresie*1 este **true** (eventual după conversia la tipul `boolean`) sau valoarea expresiei *expresie*3 în caz contrar.

Operatori de atribuire

Cel mai des utilizat operator de atribuire este `'='`. Acesta este folosit pentru a atribui unei variabile valoarea unei expresii. În urma operației de atribuire, tipul variabilei va deveni același cu tipul expresiei. Este obligatoriu ca primul operand să fie o variabilă. Acest operator a fost utilizat în toate exemplele anterioare.

Pe lângă acest operator, avem la dispoziție mai mulți alți operatori de atribuire. Aceștia sunt de fapt o combinație dintre operatorul `'='` și un operator aritmetic, pe biți sau pentru șiruri de caractere.

Un astfel de operator de atribuire este format din caracterul sau caracterele corespunzătoare operatorului aritmetic, pe biți sau pentru șiruri, după care urmează operatorul egal `'='`. Singurul operator pe biți care nu poate apărea într-o astfel de construcție este negația logică (`'~'`).

Primul operand asupra căruia este aplicat un astfel de operator de atribuire trebuie să fie o variabilă, iar al doilea poate fi o expresie.

În cazul folosirii unui astfel de operator de atribuire se execută operația corespunzătoare operatorului obținut prin eliminarea caracterului `'='` între valoarea variabilei care reprezintă primul operand și valoarea expresiei care reprezintă cel de-al doilea operand, iar rezultatul expresiei este atribuit variabilei care reprezintă primul operand.

De exemplu, expresia `$a += 2` are ca efect creșterea cu `2` a valorii variabilei `$a`, operația fiind echivalentă cu `$a = $a + 2`.

Rezultatul unei expresii de atribuire întotdeauna este valoarea atribuită variabilei care reprezintă primul operand.

Prezentăm în continuare toți operatorii de atribuire de acest tip, precum și expresiile echivalente:

- adunare (`'+='`): `$a += 2` ⇔ `$a = $a + 2`;
- scădere (`'-='`): `$a -= 2` ⇔ `$a = $a - 2`;
- înmulțire (`'*='`): `$a *= 2` ⇔ `$a = $a * 2`;
- împărțire (`'/='`): `$a /= 2` ⇔ `$a = $a / 2`;
- rest (`'%='`): `$a %= 2` ⇔ `$a = $a % 2`;
- conjuncție (`'&='`): `$a &= 2` ⇔ `$a = $a & 2`;
- disjuncție (`'|='`): `$a |= 2` ⇔ `$a = $a | 2`;
- disjuncție exclusivă (`'^='`): `$a ^= 2` ⇔ `$a = $a ^ 2`;
- deplasare la stânga (`'<<='`): `$a <<= 2` ⇔ `$a = $a << 2`;
- deplasare la dreapta (`'>>='`): `$a >>= 2` ⇔ `$a = $a >> 2`;
- concatenare (`'.='`): `$a .= "2"` ⇔ `$a = $a . "2"`.

Documentul *PHP* următor ilustrează modul de utilizare al tuturor operatorilor de atribuire prezentați. Pagina *Web* generată este cea din figura 5.

```
<HTML>
  <HEAD>
    <TITLE>
      Operatori de atribuire
    </TITLE>
  </HEAD>
```

```
<BODY>
  <?PHP
    $b = 24;
    $a = 57;
    echo "<TT>$a += $b = ";
    echo $a += $b;
    echo "<BR>$a -= $b = ";
    echo $a -= $b;
    echo "<BR>$a *= $b = ";
    echo $a *= $b;
    echo "<BR>$a /= $b = ";
    echo $a /= $b;
    echo "<BR>$a %= $b = ";
    echo $a %= $b;
    echo "<BR>$a &= $b = ";
    echo $a &= $b;
    echo "<BR>$a |= $b = ";
    echo $a |= $b;
    echo "<BR>$a ^= $b = ";
    echo $a ^= $b;
    echo "<BR>$a <<= $b = ";
    echo $a <<= $b;
    echo "<BR>$a >>= $b = ";
    echo $a >>= $b;
    echo "<BR>$a .= $b = ";
    echo $a .= $b;
    echo "</TT>";
  ?>
 </BODY>
</HTML>
```

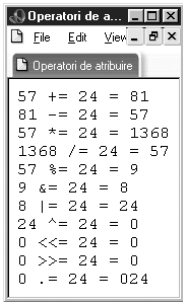

Figura 5

Operatori de incrementare și decrementare

Operatorul de incrementare (`'++'`) are ca efect creșterea cu `1` a valorii unei variable, iar cel de decrementare (`'--'`) are ca efect scăderea cu `1` a valorii variabilei. Tipul variabilei poate fi întreg sau real.

Dacă operatorul precede variabila, atunci rezultatul expresiei este valoarea obținută după incrementare sau decrementare. Dacă variabila precede operatorul, atunci rezultaul expresiei este valoarea variabilei înaintea incrementării sau decrementării.

Cu alte cuvinte, dacă operatorul precede variabila, atunci valoarea variabilei este mai întâi modificată și apoi utilizată, iar dacă variabila precede operatorul, atunci valoarea ei este mai întâi utilizată și apoi modificată.

Acești operatori pot fi utilizați și pentru variabile care conțin șiruri de caractere. Operatorul de incrementare duce la creșterea cu `1` a codului *ASCII* a ultimului caracter din șir dacă acesta este o literă sau o cifră. În cazul în care cifra este `'9'` sau litera este `'z'`, respectiv `'Z'`, atunci ea devine `'0'`, `'a'` respectiv `'A'` și se încearcă incrementarea penultimului caracter. Dacă acesta este tot `'9'`, `'z'` sau `'Z'` se aplică același procedeu și se trece la antepenultimul caracter. Procedeul continuă până în momentul în care se ajunge la un caracter care nu este `'9'`, `'z'` sau `'Z'` sau se ajunge la începutul șirului.

În acest ultim caz, la începutul șirului se adaugă caracterul `'1'`, `'a'` sau `'A'`. Incrementarea nu are nici un efect pentru caracterele care nu sunt cifre sau litere.

Decrementarea șirurilor de caractere nu are nici un efect, valorile variabilelor rămânând nemodificate. Există o singură excepție și anume șirurile care reprezintă numere întregi sau reale. Acestea sunt incrementate și decrementate potrivit regulilor pentru valori numerice.

Următorul document *PHP* ilustrează modul de utilizare al acestor operatori, precum și efectul acestora în funcție de poziția lor.

```
<HTML>
  <HEAD>
    <TITLE>
      Incrementare si decrementare
    </TITLE>
  </HEAD>
  <BODY>
    <?php
      $a = 7;
      echo "<TT>a = $a";
      echo ": </TT><I>Rezultatul operatiei
           </I> <TT>a++</TT> <I>este</I> <TT>";
      echo $a++;
      echo "</TT>. <I>Noua valoare a variabilei
          </I> <TT>a</TT> <I>este</I> <TT>$a.";
      echo "<BR>a = $a";
      echo ": </TT><I>Rezultatul operatiei
           </I> <TT>++a</TT> <I>este</I> <TT>";
      echo ++$a;
      echo "</TT>. <I>Noua valoare a variabilei
          </I> <TT>a</TT> <I>este</I> <TT>$a.";
      echo "<BR>a = $a";
      echo ": </TT><I>Rezultatul operatiei
           </I> <TT>a--</TT> <I>este</I> <TT>";
      echo $a--;
      echo "</TT>. <I>Noua valoare a variabilei
          </I> <TT>a</TT> <I>este</I> <TT>$a.";
      echo "<BR>a = $a";
      echo ": </TT><I>Rezultatul operatiei
           </I> <TT>--a</TT> <I>este</I> <TT>";
      echo --$a;
      echo "</TT>. <I>Noua valoare a variabilei
          </I> <TT>a</TT> <I>este</I> <TT>$a.";
      $a = 7.5;
      echo "<BR>a = $a";
      echo ": </TT><I>Rezultatul operatiei
           </I> <TT>a++</TT> <I>este</I> <TT>";
      echo $a++;
      echo "</TT>. <I>Noua valoare a variabilei
          </I> <TT>a</TT> <I>este</I> <TT>$a.";
      echo "<BR>a = $a";
      echo ": </TT><I>Rezultatul operatiei
           </I> <TT>++a</TT> <I>este</I> <TT>";
      echo ++$a;
```

```
        echo "</TT>. <I>Noua valoare a variabilei
            </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
             </I> <TT>a--</TT> <I>este</I> <TT>";
        echo $a--;
        echo "</TT>. <I>Noua valoare a variabilei
            </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
             </I> <TT>--a</TT> <I>este</I> <TT>";
        echo --$a;
        echo "</TT>. <I>Noua valoare a variabilei
            </I> <TT>a</TT> <I>este</I> <TT>$a.";
        $a = "PHP";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
             </I> <TT>a++</TT> <I>este</I> <TT>";
        echo $a++;
        echo "</TT>. <I>Noua valoare a variabilei
            </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
             </I> <TT>++a</TT> <I>este</I> <TT>";
        echo ++$a;
        echo "</TT>. <I>Noua valoare a variabilei
            </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
             </I> <TT>a--</TT> <I>este</I> <TT>";
        echo $a--;
        echo "</TT>. <I>Noua valoare a variabilei
            </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "<BR>a = $a";
        echo ": </TT><I>Rezultatul operatiei
             </I> <TT>--a</TT> <I>este</I> <TT>";
        echo --$a;
        echo "</TT>. <I>Noua valoare a variabilei
            </I> <TT>a</TT> <I>este</I> <TT>$a.";
        echo "</TT>";
    ?>
  </BODY>
</HTML>
```

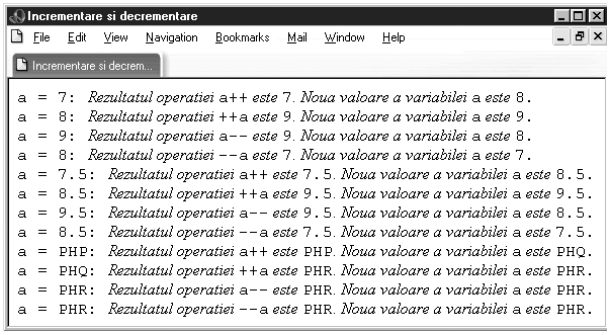

Figura 6

Pagina *Web* generată de acest document *PHP* este prezentată în figura 6. Prezentăm în continuare un document *PHP* în care se exemplifică modul în care sunt realizate incrementările și decrementările pentru șirurile de caractere. Efectul acestor operații este prezentat în figura 7.

```
<HTML>
  <HEAD>
    <TITLE>
      Siruri de caractere
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      echo "<TABLE width = 100%>";
      $a = "7";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
      $a = "7.5";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
      $a = "7.5z";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
      $a = "7.9z";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
      $a = "7zz";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
      $a = "az99zz";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
      $a = "zz99zz";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
      $a = "99Z99Z";
      echo "<TR><TD width = 50%>Inainte de
       incrementare: <TT>a = $a</TT>. </TD><TD>
       Dupa incrementare: <TT>a = ".++$a."</TT>
                                <BR></TD></TR>";
      $a = "+";
```

```
echo "<TR><TD width = 50%>Inainte de
 incrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa incrementare: <TT>a = ".++$a."</TT>
                          <BR></TD></TR>";
$a = "+9";
echo "<TR><TD width = 50%>Inainte de
 incrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa incrementare: <TT>a = ".++$a."</TT>
                          <BR></TD></TR>";
$a = "a+9";
echo "<TR><TD width = 50%>Inainte de
 incrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa incrementare: <TT>a = ".++$a."</TT>
                          <BR></TD></TR>";
$a = "++9";
echo "<TR><TD width = 50%>Inainte de
 incrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa incrementare: <TT>a = ".++$a."</TT>
                          <BR></TD></TR>";
$a = "-9";
echo "<TR><TD width = 50%>Inainte de
 incrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa incrementare: <TT>a = ".++$a."</TT>
                          <BR></TD></TR>";
$a = "-9z";
echo "<TR><TD width = 50%>Inainte de
 incrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa incrementare: <TT>a = ".++$a."</TT>
                          <BR></TD></TR>";
$a = "--9";
echo "<TR><TD width = 50%>Inainte de
 incrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa incrementare: <TT>a = ".++$a."</TT>
                          <BR></TD></TR>";
$a = "+-99";
echo "<TR><TD width = 50%>Inainte de
 incrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa incrementare: <TT>a = ".++$a."</TT>
                          <BR></TD></TR>";
$a = "-9";
echo "<TR><TD width = 50%>Inainte de
 decrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa decrementare: <TT>a = ".--$a."</TT>
                          <BR></TD></TR>";
$a = "-9.5";
echo "<TR><TD width = 50%>Inainte de
 decrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa decrementare: <TT>a = ".--$a."</TT>
                          <BR></TD></TR>";
$a = "-9.5b";
echo "<TR><TD width = 50%>Inainte de
 decrementare: <TT>a = $a</TT>. </TD><TD>
 Dupa decrementare: <TT>a = ".--$a."</TT>
                    <BR></TD></TR></TABLE>";?>
  </BODY>
</HTML>
```

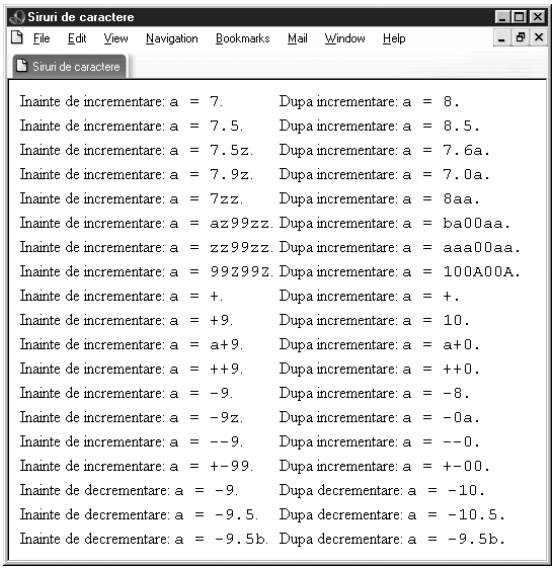

Figura 7

Operatori logici

Există patru tipuri de operatori care pot fi folosiți pentru construirea expresiilor logice. Operanzii trebuie să aibă tipul `boolean`; dacă operatorii au alt tip, valoarea este convertită la o valoare booleană. Operatorii logici sunt:

- conjuncție logică (`'&&'` sau `'and'`) - *ȘI* (*AND*);
- disjuncție logică (`'||'` sau `'or'`) - *SAU* (*OR*);
- disjuncție exclusivă logică (`'xor'`) - *SAU* exclusiv (*XOR*);
- negație logică (`'!'`) - *NU* (*NOT*).

Expresia în care apar operatorii `'&&'` și `'and'` va avea valoarea `true` dacă ambii operanzi au valoarea **`true`**.

Expresia în care apar operatorii `'||'` și `'or'` va avea valoarea **`true`** dacă cel puțin unul dintre operanzi are valoarea **`true`**.

Expresia în care apare operatorul `'xor'` va avea valoarea **`true`** dacă exact unul dintre operanzi are valoarea **`true`**.

Expresia în care apare operatorul `'!'` va avea valoarea **`true`** dacă operandul are valoarea **`false`**.

În celelalte cazuri valorile acestor tipuri de expresii vor fi **`false`**.

Operatori pentru controlul erorilor

În *PHP* există un operator (`'@'`) care permite ignorarea erorilor. Dacă este aplicat asupra unei expresii care ar duce la afișarea unui mesaj de eroare, atunci mesajul respectiv nu va mai fi afișat.

Operatori de execuție

Interpretorul *PHP* permite executarea unor comenzi sistem prin intermediul operatorului `"``"`. Comanda respectivă este cuprinsă între apostroafele inverse, iar rezultatul acestei comenzi este un șir de caractere care reprezintă și rezultatul expresiei.

Următorul *script PHP* determină execuția pe *server* a comenzii `help /?` și afișarea rezultatului în fereastra programului de navigare. Efectul este ilustrat în figura 8.

```
<HTML>
  <HEAD>
    <TITLE>
      Executarea unei comenzi
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      echo "<PRE>".`help /?`."</PRE>";
    ?>
  </BODY>
</HTML>
```

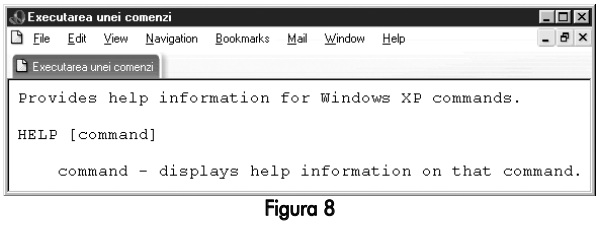

Figura 8

Operatori pentru vectori

Singurul operator care poate fi aplicat asupra vectorilor este cel de concatenare (`'+'`). Rezultatul concatenării a doi vectori este un vector care conține elementele din cei doi vectori.

Dacă cei doi vectori conțin elemente cu aceeași cheie, atunci este păstrat doar elementul din vectorul care reprezintă primul operand.

Un exemplu de utilizare a acestui operator este prezentat în continuare (pagina *Web* generată este cea din figura 9):

Figura 9

```
<HTML>
  <HEAD>
    <TITLE>
      Concatenarea vectorilor
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $a[2] = 2;
      $a[4] = 4;
      $a[5] = 5;
      $a[7] = 7;
      $b[1] = 100;
      $b[2] = 200;
      $b[3] = 300;
      $b[4] = 400;
      $b[5] = 500;
      $c = $a + $b;
      echo "$c[0] $c[1] $c[2] $c[3] $c[4] $c[5]
                                          $c[7]";
    ?>
  </BODY>
</HTML>
```

Ordinea operațiilor

Evident, o expresie poate conține mai mulți operatori, din categorii diferite. Pentru a indica ordinea în care trebuie efectuate calculele, trebuie stabilită o ordine a operațiilor. Pentru aceasta a fost definită ***precedența*** și ***asociativitatea*** operatorilor.

Valoarea expresiilor corespunzătoare operatorilor cu precedență mai mare va fi calculată înaintea valorilor expresiilor cu o precedență mai mică. De exemplu, pentru expresia `1 + 2 * 3` va fi efectuată mai întâi înmulțirea `2 * 3`, datorită faptului că operatorul `'*'` are precedența mai mare decât cea a operatorului `'+'`.

Dacă o expresia sau o subexpresie conține numai operatori cu aceeași precedență, atunci operațiile se vor efectua în ordinea determinată de asociativitatea acestor operatori care poate fi de la dreapta la stânga sau de la stânga la dreapta. Toți operatorii cu aceeași precedență au același tip de asociativitate.

Dacă asociativitatea este de la dreapta la stânga, atunci operațiile se vor efectua începând din "partea dreaptă" a expresiei, iar dacă asociativitatea este de la stânga la dreapta, vor fi efectuate mai întâi operațiile aflate în "partea stângă" a expresiei.

Există și operatori care nu sunt asociativi, deci nu pot exista expresii în care apar doi astfel de operatori.

Pe lângă operatorii amintiți există și alții care au funcții speciale. Pe unii i-am prezentat în cadrul articolului din numărul anterior al revistei (de exemplu cei folosiți pentru realizarea conversiilor de tip), iar pe ceilalți îi vom prezenta atunci când îi vom folosi pentru a realiza diferite operații în care aceștia pot apărea.

Evident, pentru a modifica ordinea implicită de efectuare a operațiilor pot fi folosite paranteze pentru a indica faptul că o anumită operație trebuie efectuată înaintea altora, chiar dacă acestea din urmă au o precedență mai mare.

Trebuie observat faptul că, datorită asociativității de la dreapta spre stânga a operatorilor de atribuire, expresiile de tipul `$a = $b = 1` sunt valide. Mai întâi se va efectua atribuirea `$b = 1`; variabila `$b` va primi valoarea `1` și, datorită faptului că rezultatul unei operații de atribuire este valoarea atribuită variabilei care reprezintă primul operand, în continuare se va efectua atribuirea `$a = 1`, deci și variabila `$a` va primi tot valoarea `1`.

Așadar, în urma executării unei expresii de tipul `$v1 = $v2 = ... = $vn = val`, toate cele *n* variabile vor avea valoarea *val*.

Tabelul 1 conține o listă cu majoritatea opratorilor din *PHP*. Liniile tabelului indică precedența, în sensul că un operator are aceeași precedență cu operatorii de pe linie în care apare și o precedență mai mare decât oricare operator aflat pe o linie anterioară.

În acest tabel se observă faptul că, deși operatorii `'&&'` și `'and'`, respectiv `'||'` și `'or'` realizează aceeași operație, ei au precedențe diferite, deci nu sunt echivalenți din toate punctele de vedere.

Tabelul 1

Tip asociativitate	Tip	Operanzi
de la stânga la dreapta	unar	. (selectare)
de la stânga la dreapta	binar	or
de la stânga la dreapta	binar	xor
de la stânga la dreapta	binar	and
de la dreapta la stânga	binar	print
de la stânga la dreapta	binar	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
de la stânga la dreapta	ternar	? :
de la stânga la dreapta	binar	||
de la stânga la dreapta	binar	&&
de la stânga la dreapta	binar	|
de la stânga la dreapta	binar	^
de la stânga la dreapta	binar	&
fără asociativitate	binar	== != === !==
fără asociativitate	binar	< <= > >=
de la stânga la dreapta	binar	<< >>
de la stânga la dreapta	binar	+ - . (concatenare)
de la stânga la dreapta	binar	* / %
de la dreapta la stânga	unar	! ~ ++ -- (int) (float) (string) (array) (object) @
de la dreapta la stânga	unar	[
fără asociativitate	unar	new

Precedența și asociativitatea operatorilor în PHP

În tabel este prezentat și numărul expresiilor asupra cărora acționează un operand. Operatorii unari acționează asupra unei singure expresii, cei binari asupra a două expresii și singurul operator ternar asupra a trei expresii.

Structuri de control în PHP

Așa cum am afirmat într-un articol anterior, instrucțiunile *PHP* sunt separate prin caracterul `';'`. Deseori, suntem nevoiți să grupăm mai multe instrucțiuni pentru a forma un bloc. Astfel, obținem instrucțiunile compuse care sunt formate din mai multe instrucțiuni simple, separate prin caracterul `';'`. În *PHP*, instrucțiunile compuse au următoarea sintaxă:

```
{
instrucțiune #1
instrucțiune #2
...
instrucțiune #n
}
```

Instrucțiunile care formează instrucțiunea compusă pot fi de orice tip: orice structură de control, alte instrucțiuni compuse etc. Așadar un bloc de instrucțiuni (o instrucțiune compusă), în *PHP*, este delimitat de acolade.

Structura if

Una dintre cele mai importante structuri în orice limbaj este cea alternativă. În *PHP* sintaxa acestei structuri este următoarea:

```
if (condiție)
   instrucțiune
```

Folosirea unei astfel de structuri indică faptul că instrucțiunea se va executa dacă și numai dacă valoarea expresiei *condiție* (eventual după conversia la tipul `boolean`) este **`true`**.

Construcția if - else

În foarte multe cazuri dorim să executăm o altă instrucțiune dacă valoarea expresiei *condiție* este **`false`**. În *PHP* putem utiliza o construcție de tipul **`if`** - **`else`** în acest scop. Sintaxa este:

```
if (condiție)
   instrucțiune #1
else
   instrucțiune #2
```

Construcția elseif

Uneori, suntem nevoiți să folosim structuri alternative imbricate în diferite scopuri. Folosind construcții de tipul **`if`** - **`else`** vom obține o secvență de tipul:

```
if (condiție #1)
   instrucțiune #1
else
   if (condiție #2)
      instrucțiune #2
   else
   ...
   else
      if (condiție #n)
         instrucțiune #n
   else
         instrucțiune #n+1
```

Limbajul *PHP* permite folosirea unei prescurtări și anume construcția **elseif**. Practic, aceasta înlocuiește un **else** urmat de un **if**. Folosind această structură, codul anterior poate fi scris astfel:

```
if (condiție #1)
   instrucțiune #1
elseif (condiție #2)
   instrucțiune #2
elseif
...
elseif (condiție #n)
   instrucțiune #n
else
   instrucțiune #n+1
```

Sintaxe alternative

Dacă dorim să se execute mai multe instrucțiuni când este îndeplinită o anumită condiție, atunci ar trebui să folosim instrucțiuni compuse. De exemplu, pentru o structură **if** vom scrie:

```
if (condiție) {
   instrucțiune #1
   instrucțiune #2
   ...
   instrucțiune #n
}
```

Limbajul *PHP* oferă o alternativă care permite indicarea tuturor instrucțiunilor fără a mai folosi acolade. Structura prezentată mai sus poate fi rescrisă astfel:

```
if (condiție) :
   instrucțiune #1
   instrucțiune #2
   ...
   instrucțiune #n
endif
```

Practic, pentru orice structură de control *PHP*, putem înlocui acolada deschisă prin caracterul `':'` și acolada închisă printr-un cuvânt cheie obținut prin adăugarea prefixului **end** la cuvântul cheie care indică structura de control.

Trebuie observat faptul că **else** și **elseif** nu sunt structuri, ele fiind simple construcții folosite în cadrul structurii **if**. Din acest motiv, nu vom avea niciodată **endelse** sau **endelseif**, ci doar **endif**-uri. Așadar, sintaxa alternativă pentru o structură **if** în care se folosesc construcții **else** și **elseif** este următoarea:

```
if (condiție #1)
   grup instrucțiuni #1
elseif (condiție #2)
   grup instrucțiuni #2
elseif
...
elseif (condiție #n)
   grup instrucțiuni #n
else
   grup instrucțiuni #n+1
endif;
```

Structura while

Majoritatea *script*-urilor *PHP* vor conține cicluri. Pentru a folosi o buclă anterior condiționată vom utiliza structura **while**. Sintaxa acestei proceduri este:

```
while (condiție)
   instrucțiune
```

Efectul acestei structuri este executarea instrucțiunii atât timp cât valoarea expresiei *condiție* (eventual după conversia la tipul `boolean`) este **true**. Trebuie observat faptul că este posibil ca instrucțiunea să nu fie executată niciodată. Și pentru această structură avem la dispoziție o sintaxă alternativă și anume:

```
while (condiție) :
   grup instrucțiuni
endwhile;
```

Structura do - while

Uneori, dorim să folosim bucle posterior condiționate. În acest scop, în *PHP* avem la dispoziție structura **do - while** a cărei sintaxă este:

```
do
   instrucțiune
while (condiție);
```

Singura diferență față de structura **while** este faptul că valoarea condiției este determinată, de fiecare dată, după executarea instrucțiunii. Ca urmare, instrucțiunea va fi executată cel puțin o dată. Nu există o sintaxă alternativă pentru această structură.

Structura for

O alternativă cu o funcționalitate mai ridicată pentru utilizarea buclelor este structura repetitivă **for**. Sintaxa este foarte asemănătoare cu cea din limbajele *C/C++* și *Java* și anume:

```
for (expresie #1; expresie #2; expresie #3)
   instrucțiune
```

Prima expresie este evaluată o singură dată, înainte de începerea execuției ciclului. Instrucțiunea este executată cât timp cea de-a doua expresie are valoarea **true**. De fiecare dată, după executarea instrucțiunii este evaluată cea de-a treia expresie. Oricare dintre cele trei expresii poate lipsi;

în cazul în care o expresie lipsește, se consideră că ea are valoarea **true**. Pentru structura **for** poate fi folosită și următoarea sintaxă alternativă:

```
for (expresie #1; expresie #2; expresie #3):
   instrucțiune
endfor;
```

Exemple

Vom prezenta în continuare un exemplu care afișează numerele cuprinse între 1 și 15. Numerele pare vor fi scrise cu roșu, iar cele impare care sunt divizibile cu 3 cu verde, iar celelalte cu albastru.

Vom folosi o structură repetitivă pentru a afișa numerele și o structură alternativă pentru a determina culoarea cu care vor fi afișate. Putem utiliza oricare dintre cele trei structuri repetitive; codul *PHP* pentru cele trei variante sunt prezentate în caseta de mai jos, iar pagina generată este cea din figura 10, indiferent de varianta utilizată.

Figura 10

Structura foreach

Această structură poate fi folosită pentru a realiza o iterație printre toate elementele unui vector. Așadar, ea nu poate fi folosită decât împreună cu vectori; utilizarea sa asupra unei variabile de alt tip duce la apariția unei erori.

Există două sintaxe acceptate pentru această structură și anume:

```
foreach (expresie_vectorială as $valoare)
   instrucțiune
foreach (expresie_vectorială as $cheie => $valoare)
   instrucțiune
```

Dacă se utilizează prima variantă, atunci la fiecare iterație valoarea elementului curent este atribuită variabilei *$valoare* și apoi se trece la elementul următor (a cărui valoare va fi atribuită variabilei la următoarea iterație). Execuția ciclului se încheie în momentul în care nu mai există alte elemente în vector.

Singura diferență care apare în cazul utlizării celei de a doua variante este faptul că la fiecare iterație valoarea cheii elementului curent este atribuită variabilei *$cheie.*

Prezentăm în continuare un document *PHP* în care sunt utilizate cele două sintaxe ale structurii **foreach**. Efectul este ilustrat în figura 11.

```
<HTML>
  <HEAD>
    <TITLE>
      Structura foreach
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      echo "<TABLE width = 100%> <TR>
                              <TD width = 50%>";
      // un vector obisnuit
      $a = array (1, 2, 3, 10);
      echo "<B>Parcurgerea unui vector:
                                  </B><BR><OL>";
      foreach ($a as $v)
        echo "<LI><I>Valoarea curenta este
                    </I>: <TT>$v</TT>\n<BR>";
      // parcurgerea vectorului, contorizarea
      // elementelor si afisarea numarului de
      // ordine al elementului curent
      $i = 0;
      echo "</OL><B>O alta parcurgere:
                                      </B><UL>";
```

Structuri alternative și repetitive

Bucla for

```
<HTML>
  <HEAD>
    <TITLE>
      Structuri alternative si
      repetitive
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      for ($i=1;$i<=15;$i++):
        echo "<B><FONT size = 5 ";
        echo "color = ";
        if (!($i % 2))
          echo "red";
        elseif (!($i % 3))
          echo "green";
        else
          echo "blue";
        echo "> $i </FONT></B>";
      endfor;
    ?>
  </BODY>
</HTML>
```

Bucla while

```
<HTML>
  <HEAD>
    <TITLE>
      Structuri alternative si
      repetitive
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $i = 1;
      while ($i<=15){
        echo "<B><FONT size = 5 ";
        echo "color = ";
        if (!($i % 2))
          echo "red";
        elseif (!($i % 3))
          echo "green";
        else
          echo "blue";
        echo "> $i </FONT></B>";
        $i++;
      }
    ?>
  </BODY>
</HTML>
```

Bucla do - while

```
<HTML>
  <HEAD>
    <TITLE>
      Structuri alternative si
      repetitive
    </TITLE>
  </HEAD>
  <BODY>
    <?PHP
      $i = 1;
      do{
        echo "<B><FONT size = 5 ";
        echo "color = ";
        if (!($i % 2))
          echo "red";
        elseif (!($i % 3))
          echo "green";
        else
          echo "blue";
        echo "> $i </FONT></B>";
        $i++;
      } while ($i<=15);
    ?>
  </BODY>
</HTML>
```

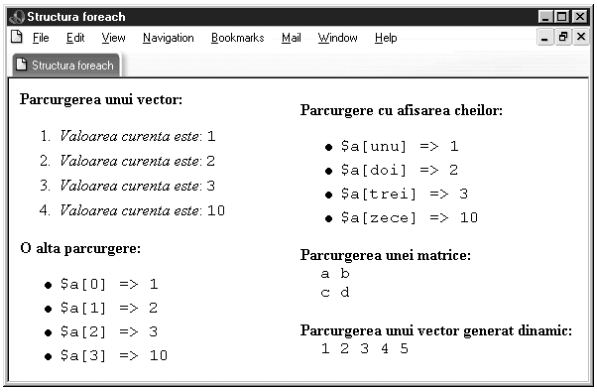

Figura 11

```
      foreach ($a as $v){
        echo "<LI><TT>\$a[$i] => $v</TT>\n
                                        <BR>";
        $i++;
      }
      // un vector ale carui elemente sunt
      // identificate prin chei
      $a = array (
        "unu" => 1,
        "doi" => 2,
        "trei" => 3,
        "zece" => 10
      );
      echo "</UL><TD width = 50%><B>
            Parcurgere cu afisarea cheilor:
                                    </B><UL>";
      foreach ($a as $k => $v) {
        echo "<LI><TT>\$a[$k] => $v</TT>\n";
      }
      // o matrice
      $b[0][0] = "a";
      $b[0][1] = "b";
      $b[1][0] = "c";
      $b[1][1] = "d";
      echo "</UL><B>Parcurgerea unei
                        matrice:</B><BR><TT>";
      foreach ($b as $v1) {
        echo "  ";
        foreach ($v1 as $v2)
          echo "$v2\n";
        echo "<BR>";
      }
      echo "</TT><BR><B>Parcurgerea unui
          vector generat dinamic:</B><BR><TT>
                                  ";
      // un vector generat dinamic
      foreach (array(1, 2, 3, 4, 5) as $v)
        echo "$v\n";
      echo "</TT></TD></TR></TABLE>";
    ?>
  </BODY>
</HTML>
```

Structura switch

În anumite cazuri trebuie să efectuăm anumite operații în funcție de anumite valori ale unei expresii. O soluție ar fi folosirea unor structuri **if** imbricate sau a uneia singure în care apar mai multe construcții **elseif**.

Să presupunem că o anumită variabilă poate avea cinci valori distincte și pentru fiecare valoare trebuie executată o anumită instrucțiune. Dacă folosim structura **if**, secvența de cod va fi următoarea:

```
if ($variabila == valoare #1)
  instrucțiune #1
elseif ($variabila == valoare #2)
  instrucțiune #2
elseif ($variabila == valoare #3)
  instrucțiune #3
elseif ($variabila == valoare #4)
  instrucțiune #4
else
  instrucțiune #5
```

Folosind structura **switch**, secvența echivalentă este:

```
switch ($variabila){
  case valoare #1: instrucțiune #1
                   break;
  case valoare #2: instrucțiune #2
                   break;
  case valoare #3: instrucțiune #3
                   break;
  case valoare #4: instrucțiune #4
                   break;
  case valoare #5: instrucțiune #5
                   break;
}
```

Prezentăm în continuare sintaxa generală a structurii **switch**:

```
switch (expresie){
  case valoare #1 : grup instrucțiuni #1
  case valoare #2 : grup instrucțiuni #2
  ...
  case valoare #n : grup instrucțiuni #n
  [default: grup instrucțiuni #n + 1]
}
```

Pentru început se determină valoarea expresiei și apoi se verifică dacă aceasta apare ca valoare pentru una dintre construcțiile **case**.

În caz afirmativ, toate instrucțiunile începând cu cea corespunzătoare valorii respective (până la sfârșitul blocului **switch**) vor fi executate.

Execuția acestor instrucțiuni poate fi întreruptă dacă este folosită instrucțiunea **break**, care va fi prezentată în cadrul acestui articol.

Dacă valoarea expresiei nu corespunde nici uneia dintre valorile corespunzătoare construcțiilor **case**, se execută instrucțiunile corespunzătoare construcției **default**. Dacă aceasta lipsește, atunci nu se execută nici o instrucțiune.

Trebuie precizat faptul că valorile corespunzătoare construcțiilor **case** pot fi numere întregi, numere reale sau șiruri de caractere; nu pot fi utilizate obiecte sau vectori.

Instrucțiunea break

Această instrucțiune poate fi folosită pentru a întrerupe forțat execuția unui ciclu sau a secvenței de instrucțiuni corespunzătoare unei structuri **switch**.

Spre deosebire de limbajele *Pascal* sau *C/C++*, instrucțiunea poate fi urmată de un argument care indică numărul de structuri imbricate a căror execuție se încheie. Valoarea implicită este 1, deci se întrerupe execuția unei singure structuri.

Următoarea secvență de cod *PHP* realizează parcurgerea elementelor unui vector de numere întregi până în momentul în care se întâlnește un număr negativ.

```
foreach ($a as $v)
  if ($v < 0)
    break;
```

Vom exemplifica acum și cazul în care este întreruptă execuția mai multor cicluri; vom considera că parcurgem elementele unei matrice pătratice cu n elemente și n coloane până în momentul în care întâlnim o valoare nulă.

```
for ($i = 0; $i < $n; $i++)
  for ($j = 0; $j < $n; $j++)
    if (!$a[$i][$j])
      break 2;
```

Instrucțiunea **break** poate fi utilizată pentru întreruperea execuției secvențelor de instrucțiuni corespunzătoare structurilor **for**, **foreach**, **while**, **do** - **while** și **switch**.

Instrucțiunea continue

Această instrucțiune este folosită pentru a întrerupe execuția secvenței de instrucțiuni din interiorul unui ciclu și trecerea la următoarea iterație. În cazul instrucțiunii **for**, înainte de următoarea iterație se evaluează (execută) expresia de incrementare (*expresia* #3 din sintaxa generală).

La fel ca și în cazul instrucțiunii **break**, poate apărea un argument care indică numărul structurilor imbricate asupra cărora are efect.

Următoarea secvență *PHP* realizează afișarea elementelor unui șir de numere întregi care sunt mai mari decât 1000.

```
foreach ($a as $v){
  if ($v <= 1000)
    continue;
  echo $v;
}
```

Următorul exemplu ilustrează efectul folosirii argumentelor pentru instrucțiunea **continue**.

```
$i = 0;
while ($i++ < 5){
  echo "Ciclul #1<br>\n";
  while (1) {
    echo "  Ciclul #2<BR>\n";
    while (1) {
      echo "  Ciclul #3<BR>\n";
      continue 3;
    }
    echo "Acest mesaj nu va fi afisat
                              niciodata.<BR>\n";
  }
  echo "Nici acest mesaj nu va fi afisat
                              niciodata.<BR>\n";
}
```

Alte structuri PHP

Există mai multe alte structuri *PHP* care pot fi utilizate în anumite scopuri. Vom aminti acum câteva dintre ele și le vom prezenta detaliat în momentul în care va fi necesară utilizarea lor în diferite scopuri.

Structurile **include**, **require**, **include_once** și **require_once** pot fi utilizate pentru a "insera" anumite instrucțiuni care sunt păstrate într-un alt fișier (document). Interpretorul *PHP* consideră că secvența din fișierul inserat se află în fișierul din care s-a "comandat" inserarea în poziția în care apare structura de inserare.

O altă structură este **declare** care permite crearea unor directive de execuție.

Funcțiile *PHP* trebuie să utilizeze instrucțiunea **return** pentru a furniza un rezultat. Detalii despre utilizarea acestei instrucțiuni vor fi prezentate în cadrul următorului articol dedicat limbajului *PHP*.

În numărul următor

Vom continua prezentarea interpretorului *PHP* cu descrierea modului în care pot fi declarate și utilizate funcții.

De asemenea, în următorul articol vom prezenta detalii referitoare la clasele și obiectele *PHP*. Nu vom insista asupra paradigmelor programării orientate obiect, ci vom prezenta doar modul în care pot fi utilizate clasele și obiectele.

În plus, articol din numărul următor va conține detalii despre modul în care pot fi utilizate referințele în *PHP* pentru a accesa conținutul unei singure variabile folosind mai multe nume (identificatori).

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.