5. ÓRA

Vezérlési szerkezetek

Az előző órában létrehozott programok minden futtatáskor ugyanazt az eredményt adták, mindig ugyanazok az utasítások hajtódtak végre ugyanabban a sorrendben. Ez nem biztosít túl nagy teret a rugalmasságnak.

Ebben az órában néhány olyan szerkezettel ismerkedünk meg, melyek segítségével programunk alkalmazkodhat a körülményekhez. A következőket tanuljuk meg:

- Hogyan használjuk az `if` szerkezetet arra, hogy bizonyos sorok csak adott feltételek teljesülése mellett hajtódjanak végre?
- Hogyan adhatunk meg csak bizonyos feltételek nem teljesülése esetén végrehajtandó műveleteket?
- Hogyan használjuk a `switch` utasítást, hogy egy kifejezés értékétől függően hajtsunk végre utasításokat?
- Hogyan ismételjük egy kódrészlet végrehajtását a `while` utasítás segítségével?
- Hogyan készíthetünk elegánsabb ciklusokat a `for` utasítás segítségével?
- Hogyan lépjünk ki a ciklusokból?
- Hogyan ágyazzuk egymásba a ciklusokat?

Elágazások

A legtöbb program feltételeket értékel ki és azok eredményének megfelelően változtatja viselkedését. A lehetőséget, hogy a PHP oldalak tartalma dinamikussá váljék, az teszi lehetővé, hogy a programok kimenete bizonyos feltételektől függhet. A legtöbb programozási nyelvhez hasonlóan a PHP 4-es változata is biztosítja erre a célra az `if` utasítást.

Az if utasítás

Az `if` utasítás kiértékeli a zárójelek közötti kifejezést. Ha a kifejezés értéke igaz, az utasításhoz tartozó programrész végrehajtódik. Ha a kifejezés hamis, a blokk egyszerűen figyelmen kívül marad. Ez teszi lehetővé a programoknak, hogy döntéseket hozzanak.

```
if ( kifejezés )
    {
    // ha a kifejezés értéke igaz,
    // ez a blokk végrehajtódik
    }
```

Az 5.1. példaprogramban csak akkor hajtódik végre az `if` utáni rész, ha a kifejezés értéke igaz

5.1. program Az if utasítás

```
 1: <html>
 2: <head>
 3: <title>5.1. program Az if utasítás</title>
 4: </head>
 5: <body>
 6: <?php
 7: $hangulat = "boldog";
 8: if ( $hangulat == "boldog" )
 9:     {
10:     print "Hurrá, jó kedvem van!";
11:     }
12: ?>
13: </body>
14: </html>
```

A $hangulat változó értékét a "boldog"-gal az összehasonlító műveletjel (==) segítségével hasonlítottuk össze. Ha egyeznek, a kifejezés értéke igaz, így az if kifejezéshez tartozó programblokk végrehajtódik. Bár az előbbi programban a print utasítást kapcsos zárójelek közé zártuk, ez csak akkor szükséges, ha az if kifejezéstől függően több utasítást akarunk végrehajtani. A következő két sor így elfogadható:

```
if ( $hangulat == "boldog" )
     print "Hurrá, jó kedvem van!";
```

Ha a $hangulat értékét "szomorú"-ra változtatjuk és újból lefuttatjuk programunkat, az if kifejezés értéke hamis lesz, így a print utasítás nem hajtódik végre és programunk nem ad kimenetet.

Az if utasítás else ága

Amikor az if utasítást használjuk, gyakran szeretnénk, hogy legyen egy olyan alternatív programrész, amely akkor hajtódik végre, ha a vizsgált feltétel nem igaz. Ezt úgy érhetjük el, hogy az if utasítás programrésze után kiírjuk az else kulcsszót, majd az alternatív programrészt. A séma a következő:

```
if (feltétel)
   {
   // itt következik az a programrész, amely akkor kerül
   // végrehajtásra, ha a feltétel értéke igaz
   }
else
   {
   // itt pedig az a programrész található, amely akkor fut le,
   // ha a feltétel értéke hamis
   }
```

Az 5.2. példaprogram az 5.1. példaprogram kiegészített változata. Egy olyan programblokkot tartalmaz, amely akkor hajtódik végre, ha a $hangulat értéke nem "boldog".

5.2. program Az else ággal kiegészített if utasítás

```
1: <html>
2: <head>
3: <title>5.2. program Az else ággal kiegészített if
   utasítás</title>
4: </head>
```

5.2. program (folytatás)

```
 5: <body>
 6: <?php
 7: $hangulat = "szomorú";
 8: if ( $hangulat == "boldog" )
 9:     {
10:     print "Hurrá, jó kedvem van!";
11:     }
12: else
13:     {
14:     print "$hangulat vagyok, nem boldog.";
15:     }
16: ?>
17: </body>
18: </html>
```

A példában a `$hangulat` értéke `"szomorú"`, ami persze nem `"boldog"`, így az `if` utasítás feltétele hamis lesz, vagyis az első programrész nem kerül végrehajtásra. Az `else`-et követő programblokk azonban lefut és a böngészőbe a `"szomorú vagyok, nem boldog."` szöveg kerül.

Az `if` utasítás `else` ágának segítségével programunkban kifinomultabb döntéseket hozhatunk, de egy feltétel eredménye alapján még így is csak kétféle dolgot tehetünk. A PHP 4 azonban többre képes: több kifejezés értéke alapján sokféleképp reagálhatunk.

Az if utasítás elseif ága

Mielőtt az `else` ágban alternatív kódrészt adnánk meg, több kifejezés értékétől függően – az `if - elseif - else` szerkezet segítségével – a programmal mást és mást végeztethetünk. A használandó utasításforma a következő:

```
if ( feltétel )
   {
   // ez a rész akkor fut le, ha a feltétel igaz
   }
elseif ( másik feltétel )
   {
   // ez a rész akkor fut le, ha a másik feltétel igaz,
   // és minden előző feltétel hamis
   }
   // itt még tetszőleges számú elseif rész következhet
```

```
else
   {
   // ez a rész akkor kerül végrehajtásra, ha egyik
   // feltétel sem volt igaz
   }
```

A Perl nyelvben gyakorlattal rendelkezők figyeljenek rá, hogy a kulcsszót itt elseif-nek hívják!

Ha az első feltétel értéke hamis, a hozzá tartozó programrész nem kerül végrehajtásra. Ezután a PHP megvizsgálja az `elseif` kifejezés értékét. Ha a kifejezés igaz, a hozzá tartozó programblokk fut le. Végül, ha egyik feltétel sem igaz, az `else` utáni rész kerül végrehajtásra. Annyi `elseif`-et írhatunk a programba, amennyi csak jólesik. Sőt, ha nincs szükségünk else ágra, vagyis olyan programrészre, amely akkor hajtódik végre, ha egyik feltétel sem igaz, akár el is hagyhatjuk.

Az 5.3. példaprogram az előzőeket egészíti ki egy `elseif` ággal.

5.3. program Egy else és elseif ággal bővített if utasítás

```
 1: <html>
 2: <head>
 3: <title>5.3. program Egy else és elseif ággal bővített if utasítás</title>
 4: </head>
 5: <body>
 6: <?php
 7: $hangulat = "szomorú";
 8: if ( $hangulat == "boldog" )
 9:      {
10:      print "Hurrá, jó kedvem van!";
11:      }
12: elseif ( $hangulat == "szomorú" )
13:      {
14:      print "Szomorú vagyok.";
15:      }
16: else
17:      {
18:      print "Sem boldog, sem szomorú nem vagyok, hanem $hangulat.";
```

5.3. program (folytatás)

```
19:       }
20: ?>
21: </body>
22: </html>
```

A `$hangulat` értéke itt `"szomorú"`. Ez nem azonos a `"boldog"`-gal, ezért az első blokk nem kerül végrehajtásra. Az `elseif` kifejezés a `$hangulat` változó értékét hasonlítja össze a `"szomorú"` szöveggel. Mivel az egyenlőség fennáll, ehhez a feltételhez tartozó programrész hajtódik végre.

A switch utasítás

A `switch` utasítás egy lehetséges módja annak, hogy egy kódrészletet egy kifejezés értékétől függően hajtsunk végre. Van azonban néhány különbség az imént tanult `if` és a `switch` között. Az `if`-et az `elseif`-fel használva több kifejezés értékétől tehetjük függővé, hogy mi történjen. A `switch` esetében a program csak egy kifejezést vizsgál meg és annak értékétől függően különböző sorokat futtat. A kifejezésnek egyszerű típusnak kell lennie (szám, karakterlánc vagy logikai érték). Az `if` feltétele csak igaz vagy hamis lehet, a `switch` kifejezését akárhány értékkel összehasonlíthatjuk. De nézzük inkább az általános szerkezetet:

```
switch ( kifejezés )
       {
       case érték1:
        // ez történjen, ha kifejezés értéke érték1
        break;
       case érték2:
        // ez történjen, ha kifejezés értéke érték2
        break;
       default:
       // ez történjen, ha a kifejezés értéke
       // egyik felsorolt értékkel sem egyezett meg
       break;
          // az ördög nem alszik, jobban járunk, ha kitesszük
          // ezt a "felesleges" break utasítást
       }
```

A `switch` utasítás kifejezése gyakran egyszerűen egy változó. A `switch`-hez tartozó programblokkban vannak a `case` címkék. Az utánuk írt érték kerül összehasonlításra a `switch` kifejezésének értékével. Ha értékük megegyezik, a program ott folytatódik, a `break` utasítás pedig azt eredményezi, hogy a program futása a `switch`

blokkja utáni részre kerül. Ha a `break`-et elfelejtjük, a program átlép a következő `case` kifejezéshez tartozó programrészre és azt is végrehajtja. Ha a kifejezés értéke egyik előző értékkel sem egyezik és a `switch` blokkján belül szerepel `default` címke, akkor az utána levő programrész kerül végrehajtásra. Ez sokszor bosszantó, de néha hasznos is lehet. Vegyük a következő kis példát.

```
switch( $hetnapja )
{
       case "Péntek":
            print "Kikapcsolni a vekkert, holnap nem kell
                  dolgozni!<br>";
       case "Hétfő":
       case "Szerda":
            print "Ma délelőtt dolgozom<br>";
       break;
       case "Kedd":
       case "Csütörtök":
            print "Ma délután dolgozom<br>";
       break;
       case "Vasárnap":
            print "Bekapcsolni a vekkert!<br>";
       case "Szombat":
            print "Hurrá, szabadnap!<br>";
       break;
       default:
            print "Azt hiszem ideje lenne egy új programo-
                  zót és/vagy egy jobb naptárat
                  keríteni<br>";
       break;
}
```

A fenti kis program azt közli, mikor kell mennünk dolgozni és az ébresztőóra kezelésében is segít. A programot úgy építettük fel, hogy több esetben is ugyanazt a kódot kelljen végrehajtani, így hasznos, hogy nem adtuk meg minden `case` címkénél a `break` utasítást. Elég gyakori azonban, hogy a kezdő programozó elfelejti megadni a `break` utasítást. Hasonló helyzetekben jusson eszünkbe ez a példa.

A case címkével elkezdett részeket ne felejtsük el break utasításokkal lezárni. Ha ezt nem tesszük meg, a program a következő case részt is végrehajtja és végül a default utáni rész is lefut. A legtöbb esetben nem ez a switch kívánatos viselkedése.

Az 5.4. példaprogram a korábbi, `if`-fel megoldott példát alakítja át a `switch` segítségével.

5.4. program A switch utasítás

```
 1: <html>
 2: <head>
 3: <title>5.4. program A switch utasítás</title>
 4: </head>
 5: <body>
 6: <?php
 7: $hangulat = "szomorú";
 8: switch ( $hangulat )
 9:        {
10:        case "boldog":
11:             print "Hurrá, jó kedvem van!";
12:             break;
13:        case "szomorú":
14:             print "Szomorú vagyok.";
15:             break;
16:        default:
17:             prin "Sem boldog, sem szomorú nem
                vagyok, hanem $hangulat.";
18:        }
19: ?>
20: </body>
21: </html>
```

A `$hangulat` változónak a `"szomorú"` értéket adtuk. A `switch` utasítás kifejezése ez a változó lesz. Az első `case` címke a `"boldog"` szöveggel való egyezést vizsgálja. Mivel nincs egyezés, a program a következő `case` címkére ugrik. A `"szomorú"` szöveg megegyezik a `$hangulat` változó pillanatnyi értékével, így az ehhez tartozó programrész fog lefutni. A programrész végét a `break` utasítás jelzi.

A ?: műveletjel

A `?:` ternális (háromoperandusú) műveletjel egy olyan `if` utasításhoz hasonlít, amely értéket is képes visszaadni. A visszaadott értéket a vizsgált feltétel határozza meg:

```
( feltétel ) ? érték_ha_a_feltétel_igaz : érték_ha_a_feltétel_hamis ;
```

Ha a vizsgált feltétel igaz, a `?` és a `:` közti kifejezés értékét adja, ha hamis, akkor a `:` utánit. Az 5.5. példaprogram ezt a műveletet használja, hogy egy változó értékét a `$hangulat` változó értékétől függően állítsa be.

5.5. program A ?: műveletjel használata

```
 1: <html>
 2: <head>
 3: <title>5.5. program A ?: műveletjel
           használata</title>
 4: </head>
 5: <body>
 6: <?php
 7: $hangulat = "szomorú";
 8: $szoveg = ( $hangulat=="boldog" ) ? “Hurrá,
    jó kedvem van!" : "$hangulat vagyok, nem boldog.";
 9: print "$szoveg";
10: ?>
11: </body>
12: </html>
```

A `$hangulat`-ot `"szomorú"`-ra állítottuk, majd megnéztük, hogy értéke `"boldog"`-e. Mivel ez nem igaz, a `$szoveg` változó értéke a `:` utáni szöveg lesz. Az e műveletet használó programokat eleinte nehéz megérteni, de a művelet hasznos lehet, ha csak két lehetőség közül lehet választani és szeretünk tömör programot írni.

Ciklusok

Most már láttuk, hogyan kell döntések eredményétől függően különböző programrészleteket futtatni. PHP programokkal arra is képesek vagyunk, hogy megmondjuk, hányszor kell lefuttatni egy adott programrészt. Erre valók a ciklusok. Segítségükkel elérhetjük, hogy egyes programrészletek ismétlődjenek. Szinte kivétel nélkül igaz, hogy egy ciklus addig fut, amíg egy feltétel teljesül, vagy meg nem mondjuk, hogy fejeződjön be az ismétlés.

A while ciklus

A `while` ciklus szerkezete rendkívül hasonlít az `if` elágazáséhoz:

```
while ( feltétel )
      {
      // valamilyen tevékenység
      }
```

Amíg a `while` feltétele igaz, a hozzá tartozó programrész újból és újból végrehajtódik. A programrészen belül általában megváltoztatunk valamit, ami hatással lesz a `while` feltételére; ha ezt nem tesszük meg, a ciklusunk a végtelenségig futni fog. Az 5.6. példaprogram a `while` ciklus segítségével írja ki a kettes szorzótáblát.

5.6. program A while ciklus

```
 1: <html>
 2: <head>
 3: <title>5.6. program A while ciklus</title>
 4: </head>
 5: <body>
 6: <?php
 7: $szamlalo = 1;
 8: while ( $szamlalo <= 12 )
 9:      {
10:      print "$szamlalo kétszerese ".($szamlalo*2)."<br>";
11:      $szamlalo++;
12:      }
13: ?>
14: </body>
15: </html>
```

A példában létrehoztuk a `$szamlalo` nevű változót. A `while` kifejezés feltétele megvizsgálja, hogy ez a változó mekkora. Ha az érték nem nagyobb, mint 12, a ciklus folytatódik (vagy elkezdődik). A ciklusban a `$szamlalo` értéke és annak kétszerese kiírásra kerül, majd a `$szamlalo` értéke eggyel nő. Ez az utasítás rendkívül fontos, mert ha elfelejtjük, a `while` feltétele soha nem lesz hamis, ezért végtelen ciklusba kerülünk.

A do..while ciklus

A `do..while` ciklus kicsit hasonlít a `while`-hoz. A lényegi különbség abban van, hogy ebben a szerkezetben először hajtódik végre a kód és csak azután értékelődik ki a feltétel:

```
do
      {
      // végrehajtandó programrész
      } while ( feltétel );
```

A do..while ciklus feltételét tartalmazó zárójel után mindig ki kell tenni a pontosvesszőt.

Ez a ciklus akkor lehet hasznos, ha mindenképpen szeretnénk, hogy a ciklushoz tartozó programrész még akkor is legalább egyszer lefusson, ha a feltétel már az első végrehajtáskor hamis. Az 5.7. példaprogram a `do..while` szerkezet egy alkalmazását mutatja be. A programrész mindig legalább egyszer lefut.

5.7. program A do..while ciklus

```
 1: <html>
 2: <head>
 3: <title>5.7. program A do..while ciklus</title>
 4: </head>
 5: <body>
 6: <?php
 7: $szam = 1;
 8: do
 9:     {
10:     print "Végrehajtások száma: $szam<br>\n";
11:     $szam++;
12:     } while ( $szam > 200 && $szam < 400 );
13: ?>
14: </body>
15: </html>
```

A `do..while` ciklus megnézi, hogy a `$szam` változó értéke 200 és 400 között van-e. Mivel a `$szam` változónak az 1 kezdeti értéket adtuk, így a feltétel hamis. Ennek ellenére a programblokk egyszer végrehajtódik, mégpedig a feltétel kiértékelése előtt, ezért a böngészőben egy sor kiírásra kerül.

A for ciklus

A `for` semmi újat nem nyújt a `while` ciklushoz képest. A `for` ciklus használatával azonban sokszor takarosabb, biztonságosabb módon közelíthetjük meg ugyanazt a problémát. Korábban – az 5.6. példaprogramban – létrehoztunk egy változót a `while` cikluson kívül, majd a `while` kifejezése megvizsgálta ennek a változónak az értékét. Végül a változó értékét a ciklus végén eggyel növeltük. A `for` ciklus mindezt egyetlen sorban teszi lehetővé. Ez tömörebb programot eredményez és ritkábban fordulhat elő, hogy a változót elfelejtjük növelni, ami végtelen ciklust okoz.

```
for ( változó_hozzárendelése; feltétel; számláló_növelése)
    {
    // a végrehajtandó programblokk
    }
```

Az ezt megvalósító egyenértékű while:

```
változó_hozzárendelése;
while ( feltétel )
      {
      // a végrehajtandó programblokk
      számláló_növelése;
      }
```

A zárójelekben levő kifejezéseket pontosvesszővel kell elválasztanunk egymástól. Az első kifejezés rendszerint egy számlálónak ad kezdeti értékét, a második egy feltétel, ami alapján eldől, hogy folytatódik-e a ciklus; a harmadik egy számlálót növelő utasítás. Az 5.8. példaprogram az 5.6. példaprogram `for` ciklusos megoldása.

A feltétel, a számláló_növelése és a változó_hozzárendelése paraméterek helyére bármilyen érvényes PHP állítás írható. A kezdők bánjanak óvatosan ezzel a lehetőséggel.

5.8. program A for ciklus használata

```
1: <html>
2: <head>
3: <title>5.8. program A for ciklus használata</title>
4: </head>
5: <body>
6: <?php
```

5.8. program (folytatás)

```
 7: for ( $szamlalo = 1; $szamlalo <= 12; $szamlalo++ )
 8:     {
 9:     print "$szamlalo kétszerese ".( $szamlalo * 2
            )."<br>";
10:     }
11: ?>
12: </body>
13: </html>
```

Az 5.6. és az 5.8. példaprogram kimenete teljesen azonos. A `for` ciklus használata a programot összefogottabbá tette. Mivel a `$szamlalo` nevű változó létrehozása és módosítása egy sorban van, így a ciklus egészének logikája első látásra világos. A `for` zárójelében az első utasítás a `$szamlalo` változót egyre állítja. A feltételes kifejezés megnézi, hogy a `$szamlalo` értéke nem nagyobb-e, mint 12. Az utolsó kifejezés a `$szamlalo`-t eggyel növeli.

Amikor a program a `for` ciklushoz ér, megtörténik az értékadás, majd rögtön utána a feltétel kiértékelése. Ha a feltétel igaz, a ciklus végrehajtódik, majd a `$szamlalo` értéke eggyel nő és a feltétel kiértékelése újból végrehajtódik. A folyamat addig folytatódik, amíg a feltétel hamissá nem válik.

Ciklus elhagyása a break utasítás segítségével

A `while` és `for` ciklusok lehetőséget biztosítanak arra, hogy egy beépített feltételes kifejezés segítségével kilépjünk belőlük. A `break` utasítás lehetővé teszi, hogy más feltételektől függően megszakítsuk egy ciklus futását. Ez jó lehet például hibakezeléskor vagy hibák megelőzésekor. Az 5.9. példaprogram egy egyszerű `for` ciklusból áll, amely egy nagy számot egy egyre növekvő számmal oszt el és a művelet eredményét meg is jeleníti.

5.9. program Egy for ciklus, amely a 4000-et tíz, egyre növekvő számmal osztja el

```
1: <html>
2: <head>
3: <title>5.9. program A for ciklus használata
   2.</title>
4: </head>
5: <body>
6: <?php
```

5.9. program (folytatás)

```
 7: for ( $szamlalo=1; $szamlalo <= 10, $szamlalo++ )
 8:      {
 9:      $seged = 4000/$szamlalo;
10:      print "4000 $szamlalo részre osztva $seged.<br>";
11:  }
12: ?>
13: </body>
14: </html>
```

A példában a `$szamlalo` nevű változónak az 1 kezdeti értéket adjuk. A `for` utasítás feltételes kifejezése ellenőrzi, hogy a `$szamlalo` nem nagyobb-e, mint 10. A ciklus belsejében a 4000-et elosztjuk a `$szamlalo`-val és az eredményt kiírjuk a böngészőbe.

Ez elég célratörőnek tűnik. De mi a helyzet akkor, ha a `$szamlalo` értéke a felhasználótól származik? A változó értéke lehet negatív szám vagy akár szöveg is. Vegyük az első esetet: változtassuk meg a `$szamlalo` kezdőértékét 1-ről -4-re. Így a ciklusmag ötödik futtatása során nullával kellene osztani, ami nem elfogadható. Az 5.10. példaprogram ezt azzal védi ki, hogy a ciklus futását befejezi, ha a `$szamlalo` változóban a nulla érték van.

5.10. program A break utasítás használata

```
 1: <html>
 2: <head>
 3: <title>5.10. program A break utasítás
           használata</title>
 4: </head>
 5: <body>
 6: <?php
 7: $szamlalo = -4;
 8: for ( ; $szamlalo <= 10; $szamlalo++ )
 9:      {
10:      if ( $szamlalo == 0 )
11:           break;
12:      $seged = 4000/$szamlalo;
13:      print "4000 $szamlalo részre osztva $seged.<br>";
14:      }
15: ?>
16: </body>
17: </html>
```

A nullával való osztás nem eredményez végzetes hibát a PHP 4-ben, csak egy figyelmeztető üzenet jelenik meg a böngészőben. A program futása folytatódik.

Egy `if` utasítással megvizsgáljuk a `$szamlalo` értékét. Ha nullával egyenlő, azonnal kilépünk a ciklusból; a program ekkor a `for` ciklus utáni sorokat kezdi feldolgozni. Figyeljük meg, hogy a `$szamlalo`-t a cikluson kívül hoztuk létre, hogy olyan helyzetet utánozzunk, amikor a `$szamlalo` értéke „kívülről" származik, például egy űrlapról vagy egy adatbázisból.

A for ciklus fejéből bármelyik kifejezés elhagyható, de figyelnünk kell arra, hogy a pontosvesszőket mindig kiírjuk.

Következő ismétlés azonnali elkezdése a continue utasítás segítségével

A `continue` utasítás segítségével az éppen folyó ismétlést befejezhetjük, mégpedig úgy, hogy ez ne eredményezze az egész ciklusból való kilépést, csak a következő ismétlés kezdetét jelentse. Az 5.10. példában a `break` utasítás használata kicsit drasztikus volt. Az 5.11. példaprogramban a nullával való osztást úgy kerüljük el, hogy közben programunk nem lép ki az egész ciklusból.

5.11. program A continue utasítás használata

```
 1: <html>
 2: <head>
 3: <title>5.11. program A continue utasítás
    használata</title>
 4: </head>
 5: <body>
 6: <?php
 7: $szamlalo = -4;
 8: for ( ; $szamlalo <= 10; $szamlalo++ )
 9:     {
10:     if ( $szamlalo == 0 )
11:          continue;
12:     $seged = 4000/$szamlalo;
13:     print "4000 $szamlalo részre osztva $seged.<br>";
14:     }
```

5.11. program (folytatás)

```
15: ?>
16: </body>
17: </html>
```

Itt a `break` utasítást `continue`-ra cseréltük. Ha a `$szamlalo` értéke nullával egyenlő, az éppen folyó ismétlés véget ér, a végrehajtás pedig rögtön a következőre kerül.

A break és continue utasítások a ciklus logikáját bonyolultabbá, a programot pedig nehezebben olvashatóvá teszik, így furcsa programhibákat idézhetnek elő, ezért óvatosan használandók.

Egymásba ágyazott ciklusok

A ciklusok törzsében is lehetnek ciklusok. Ez a lehetőség különösen hasznos, ha futási időben előállított HTML táblázatokkal dolgozunk. Az 5.12. példaprogramban két egymásba ágyazott for ciklus segítségével egy szorzótáblát írunk ki a böngészőbe.

5.12. program Két for ciklus egymásba ágyazása

```
 1: <html>
 2: <head>
 3: <title>5.12. program Két for ciklus egymásba
           ágyazása</title>
 4: </head>
 5: <body>
 6: <?php
 7: print "<table border=1>\n"; // HTML táblázat kezdete
 8: for ( $y=1; $y<=12; $y++ )
 9:      {
10:      print "<tr>\n"; // sor kezdete a HTML táblázatban
11:      for ( $x=1; $x<=12; $x++ )
12:          {
13:          print "\t<td>"; // cella kezdete
14:          print ($x*$y);
15:          print "</td>\n"; // cella vége
16:          }
17:      print "</tr>\n"; // sor vége
18:      }
```

5.12. program (folytatás)

```
19: print "</table>\n"; // táblázat vége
20: ?>
21: </body>
22: </html>
```

A külső `for` ciklus az `$y` változónak az 1 kezdeti értéket adja. A ciklus addig fog futni, amíg a változó nem nagyobb 12-nél, az utolsó kifejezés pedig biztosítja, hogy `$y` értéke minden ismétlés után eggyel nőjön. A ciklus törzse minden ismétlés során kiír egy `tr` (`table row` – táblázatsor) HTML elemet, majd egy újabb `for` ciklus kezdődik. A belső ciklus az `$x` változó értékét a külső ciklushoz hasonlóan végigfuttatja 1-től 12-ig. A ciklus törzsében egy `td` (`table data` – táblázatcella) elemet ír ki, amelybe az `$x*$y` érték kerül. Az eredmény egy ízlésesen formázott szorzótábla lesz.

Összefoglalás

Ebben az órában a vezérlési szerkezetekről tanultunk és arról, hogyan segítenek minket programjaink változatosabbá és rugalmasabbá tételében. A legtöbb megtanult szerkezet újból és újból meg fog jelenni a könyv hátralevő részében. Megtanultuk, hogyan hozzunk létre `if` utasításokat, hogyan egészítsük ki azokat `elseif` és `else` ágakkal. Hallottunk arról, hogyan használjuk a `switch` utasítást, hogy egy kifejezés adott értékei esetén más és más történjen. Tanultunk a ciklusokról, pontosabban a `while` és a `for` ciklusokról, és arról, hogy a `break` és a `continue` utasítások segítségével hogyan léphetünk ki végleg a ciklusból, illetve hogyan hagyhatunk ki egy-egy ismétlést. Végül megtanultuk, hogyan kell ciklusokat egymásba ágyazni és erre gyakorlati példát is láttunk.

Kérdések és válaszok

A vezérlési szerkezetek feltételes kifejezéseinek mindenképpen logikai értéket kell adniuk?

Végső soron igen, bár a feltételes kifejezéseknél a kiértékelés szempontjából minden, ami nulla, üres karakterlánc vagy nem meghatározott változó, `false`-nak, minden egyéb `true`-nak számít.

A vezérlési szerkezetekhez tartozó programblokkot mindig kapcsos zárójelbe kell tenni?

Ha a programblokk csak egy utasításból áll, a kapcsos zárójel elhagyható, de ezt tenni nem célszerű, mert ha a programblokkot új utasítással egészítjük ki, véletlenül hibákat idézhetünk elő.

Ez az óra bemutatta az összes ciklust?
Nem, a hetedik, tömbökkel foglalkozó órában találkozunk még a `foreach` ciklussal is, melynek segítségével egy tömb elemein haladhatunk végig.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Hogyan használnánk az `if` vezérlési szerkezetet olyan program írására, hogy ha az `$eletkor` változó értéke 18 és 35 között van, az `"Üzenet fiataloknak"` szöveget írja ki? Ha az `$eletkor` értéke bármi más, az `"Általános üzenet"` szöveg jelenjen meg a böngészőben.

2. Hogyan egészíthetnénk ki az első kérdésbeli programunkat úgy, hogy az `"Üzenet gyerekeknek"` jelenjen meg akkor, ha az `$eletkor` változó értéke 1 és 17 között van?

3. Hogyan készítenénk egy `while` ciklust, amely kiírja az 1 és 49 közötti páratlan számokat?

4. Hogyan valósítanánk meg az előző kérdésbeli programot `for` ciklus segítségével?

Feladatok

1. Nézzük végig a vezérlési szerkezetek utasításformáját! Gondoljuk végig, hogyan lehetnek ezek a szerkezetek a segítségünkre!

2. Nézzük meg a `?:` műveletet! Miben különbözik ez a többi vezérlési szerkezettől? Mikor lehet hasznos?