22. ÓRA

Hibakeresés

E könyv írásakor a PHP 4 semmilyen hibakereső eszközt nem tartalmazott. A fejlesztők ígéretet tettek hibakereső szolgáltatások beépítésére, például hogy a verem tartalmát nyomon követhessük, ezért elképzelhető, hogy mire e könyv az Olvasó kezébe kerül, a legfrissebb kiadás már tartalmaz valamilyen fejlettebb hibakereső eszközt. Ebben a fejezetben a kódban rejlő hibák felderítésének néhány egyszerű módját mutatjuk be.

Az órában a következő témákkal foglalkozunk:

- A PHP beállításainak lekérdezése
- A PHP által automatikusan elérhetővé tett változók
- Hibaüzenetek kiírása naplófájlba
- Az adatok nyomon követése a programban
- A gyakori hibák felfedezése
- Információk a PHP-ről és adott programokról

Ha egy program nem működik megfelelően, érdemes először is a PHP beállításait megvizsgálnunk. Ezután jöhetnek a PHP által létrehozott és a saját változók, és ha még mindig nem találjuk a hibát, akkor megvizsgálhatjuk a forráskódot egy olyan eszközzel, amely színkiemeléssel jelzi a nyelvtani elemeket – így hamar rábukkanhatunk a problémás részre. Ebben a részben két módszert is megvizsgálunk arra, hogyan szerezhetünk információkat a használt PHP-értelmezőről és magáról a futó programról.

A phpinfo()

A `phpinfo()` függvény az egyik leghasznosabb hibakereső eszköz: részletes információkkal szolgál magáról a PHP-ről, a kiszolgálói környezetről és a futó program változóiról. A függvénynek nem kell átadnunk semmilyen paramétert és csak egy logikai értéket ad vissza, viszont egy csinos HTML oldalt küld a böngészőnek. A `phpinfo()` kimenetét a 22.1. ábrán láthatjuk.

22.1. ábra

PHP információk megjelenítése

Az oldal tetején a használt PHP-változatról, a webkiszolgáló típusáról, a rendszerről és a szerzőkről találunk információkat. A következő táblázat részletezi a PHP beállításait – ezeket a `php.ini` fájlban módosíthatjuk. Tegyük fel például, hogy van egy `"felhasznalo"` nevű űrlapmezőnk, de a programban valamilyen okból nem jön létre a `$felhasznalo` változó. Vessünk egy pillantást a következő beállításokra:

```
track_vars    On
register_globals  Off
```

Már meg is találtuk a probléma forrását. A `track_vars` hatására a `GET` változók a `$HTTP_GET_VARS[]` tömbben tárolódnak, a `POST` változók a `$HTTP_POST_VARS[]` tömbben, míg a sütik a `$HTTP_COOKIE_VARS[]` tömbben. Ez eddig rendben is van, a `register_globals` kikapcsolása azonban azt jelenti, hogy a változók nem jönnek létre globális PHP változók formájában. Alapállapotban mindkét lehetőség engedélyezett. Ebben az esetben két lehetőségünk van. Keressük meg a `register_globals` bejegyzést a `php.ini` fájlban és változtassuk `On`-ra. Nem tudjuk, merre keressük a `php.ini` fájlt? Nos, a `phpinfo()` táblázataiban erről is kaphatunk információt. A másik lehetőségünk, hogy a `"felhasznalo"` mező tartalmára a program ne `$felhasznalo`-ként, hanem `$HTTP_POST_VARS["felhasznalo"]`-ként hivatkozzunk.

Egy olyan táblát is találnunk kell, amely a felhasználói munkamenetek kezelésére vonatkozó beállításokat tartalmazza. Ha ez hiányzik, akkor a PHP-változatunkba nem fordítottuk bele a munkamenetek kezelésének támogatását. A táblázatban hasznos információkat találunk a munkamenetek kezelését megvalósító kód hibakereséséhez. Tegyük fel például, hogy olyan munkameneteket szeretnénk létrehozni, amelyek bizonyos ideig fennmaradnak. Ha a munkamenet elvész, amikor a felhasználó bezárja a böngésző ablakát, és a `phpinfo()` a következő beállítást mutatja:

```
session.cookie_lifetime        0
```

már meg is találtuk a probléma forrását. A `session.cookie_lifetime` értéket kell átállítanunk a `php.ini` fájlban, annak megfelelően, hogy hány másodpercig szeretnénk fenntartani a munkameneteket.

Ha a `php.ini` állomány a következő sort tartalmazza:

```
session.use_cookies        0
```

a sütik nem engedélyezettek a munkamenetek kezelése során. Ebben az esetben az azonosítás során a lekérdező karakterláncra kell hagyatkoznunk vagy módosítanunk kell a beállítást a `php.ini`-ben.

A `phpinfo()` a webkiszolgálóról is rengeteg hasznos információval szolgál, különösen akkor, ha Apache fut a gépünkön. Láthatjuk például a programmal kapcsolatban forgalmazott összes kérés- és válaszfejlécet, illetve a kiszolgáló környezeti változóit is (például `HTTP_REFERER`).

Ha a PHP-t adatbázis-támogatással fordítottuk, az erre vonatkozó beállításokat is láthatjuk, például az alapértelmezett felhasználót, IP címet és kaput.

Az egyik legfontosabb információforrásunk lehet az a tábla, amelyben a PHP által létrehozott globális változók vannak felsorolva az értékeikkel együtt. Lássunk erre egy példát. A 22.1. példa egy egyszerű programot tartalmaz, amely létrehoz egy HTML űrlapot és beállít egy sütit.

22.1. program A phpinfo() függvény kipróbálása

```
 1: <?php
 2: setcookie( "azonosito", "2344353463433",
                time()+3600, "/" );
 3: ?>
 4: <html>
 5: <head>
 6: <title>22.1. program A phpinfo() függvény
           kipróbálása</title>
 7: </head>
 8: <body>
 9: <form action="<?php print "$PHP_SELF" ?>"
                 METHOD="get"">
10: <input type="text" name="felhasznalo">
11: <br>
12: <select name="termekek[]" multiple>
13: <option>Krumpli
14: <option>Sajt
15: <option>Kenyér
16: <option>Csirke
17: </select>
18: <br>
19: <input type="submit" value="Lássuk!">
20: </form>
21: <p></p>
22: <hr>
23: <p></p>
24: <?php
25: phpinfo();
26: ?>
27: </body>
28: </html>
```

Ha a „Lássuk!” gombra kattintunk, a program megkapja a felhasználó által megadott adatokat és a süti is beállítódik. Ha meghívjuk a `phpinfo()` függvényt, látni fogjuk ezeket a változókat – a kimenet lényeges részét a 22.2. ábrán láthatjuk.

22.2. ábra

Globális változók elérése

Látható, hogy a süti és a `$HTTP_GET_VARS` változó elérhető. Az űrlap tartalmazott egy olyan listát is, amelyből több elemet is kiválaszthattunk – a változók között a teljes tömb megjelenik.

Nagyobb lélegzetű feladatoknál gyakran gondot okoz az űrlapváltozók és a sütik nyomon követése, ilyenkor a `phpinfo()` felbecsülhetetlen segítséget nyújthat.

A forráskód megjelenítése színkiemeléssel

Ha nem találjuk meg a probléma forrását a `phpinfo()` függvény segítségével, talán nem is a beállításokkal van baj. Jó ötlet lehet egy újabb pillantást vetni a forráskódra. A PHP segítségével megtekinthetjük a program forráskódját, ráadásul a kulcsszavakat, a karakterláncokat, a megjegyzéseket és a HTML kódot színkiemeléssel is megjeleníthetjük.

Ha Apache kiszolgálót használunk, a beállítófájlhoz (többnyire `httpd.conf`) kell hozzáadnunk a következő sort:

```
AddType application/x-httpd-php-source .phps
```

Ezután minden `.phps` kiterjesztésű fájl színkiemeléssel fog megjelenni a böngészőablakban. Ha nincs jogunk megváltoztatni a kiszolgáló beállítóállományát, használjuk a `show_source()` függvényt, amely a paraméterül megadott fájlt színkiemeléssel jeleníti meg a böngészőablakban.

A 22.2. példaprogram segítségével megtekinthetjük programjaink forráskódját.

22.2. program Dokumentum forrásának megjelenítése

```
 1: <html>
 2: <head>
 3: <title>22.2. program Dokumentum forrásának
          megjelenítése</title>
 4: </head>
 5: <body>
 6: <form action="<?php print "$PHP_SELF" ?>"
     method="get">
 7: Kérem a fájl nevét:
 8: <input type="text" name="file" value="<?php print
     $file; ?>">
 9: <p></p><hr><p></p>
10: <?php
11: if ( isset( $file ) )
12:      show_source( $file ) or print "nem lehet
                 megnyitni a következő fájlt: \"$file\"";
13: ?>
14: </body>
15: </html>
```

A 22.3. ábra a 22.2. példaprogramot mutatja működés közben.

22.3. ábra

Színkiemelés használata

Miért hasznos ez a lehetőség? Hiszen megnézhetnénk a kódot megszokott szövegszerkesztőnkkel is. A legnagyobb előny a színkiemelésben rejlik. Igen könnyű észrevenni például az elgépeléseket, hiszen ezeket nem kulcsszóként értelmezi a megjelenítő, ezért más színnel jelöli.

Egy másik gyakori probléma, hogy elég nehéz nyomon követni egy félig nyitott idézőjel-párt. Mivel a PHP az idézőjel utáni szöveget karakterláncként értelmezi, gyakran rossz helyen jelzi a hibát. Az idézőjelben lévő karakterláncok szintén más színnel jelöltek, így ez a típusú hiba nagyon könnyen észrevehető.

Ne tegyük ezt a programot elérhetővé webhelyünkön, mert így bárki belenézhet forrásainkba. Csak fejlesztéskor használjuk ezt a kódot!

A 22.1. táblázat a színkiemelések jelentését tartalmazza.

22.1. táblázat Színkiemelések

php.ini bejegyzés	*Alapértelmezett szín*	*Kód*	*Jelentés*
`highlight.string`	Vörös	`#DD0000`	Idézőjelek és karakterláncok
`highlight.comment`	Narancs	`#FF8000`	PHP megjegyzések
`highlight.keyword`	Zöld	`#007700`	Műveletjelek, nyelvi elemek és a legtöbb beépített függvény
`highlight.default`	Kék	`#0000BB`	Minden egyéb PHP kód
`highlight.html`	Fekete	`#000000`	HTML kód

PHP hibaüzenetek

Miközben e könyv segítségével elsajátítottuk a PHP programozás alapjait, bizonyára előfordult, hogy hibaüzeneteket kaptunk a várt eredmény helyett. Például elfelejtettünk zárójelbe tenni egy kifejezést vagy elgépeltük egy függvény nevét. A hibaüzenetek nagyon fontosak a hibakeresés során. Állítsuk a `php.ini` fájlban a `display_errors` bejegyzést `"On"`-ra, ezzel biztosíthatjuk, hogy a PHP elküldje a hibaüzeneteket a böngészőnek. Ne feledjük, hogy a `phpinfo()` függvény segítségével megnézhetjük, hogy be van-e kapcsolva ez a lehetőség.

Miután meggyőződtünk arról, hogy programunk tudatni fogja velünk az esetleges hibákat, meg kell adnunk, hogy mennyire legyenek „szigorúak” az üzenetek. Ha be akarunk állítani egy alapértelmezett szintet, rendeljünk a `php.ini` fájlban az `error_reporting` bejegyzéshez egy számot. Szerencsére nem kell fejben

tartanunk az összes számot, mivel rendelkezésünkre állnak azok az állandók, amelyeket a hibakezelés szintjének beállításához használhatunk. A különböző értékeket a 22.2. táblázat tartalmazza.

22.2. táblázat Hibakezelési szintek

Állandó	*Név*	*Leírás*	*Mi történik?*
`E_ALL`	Mind	Mindenféle hiba	Függ a hiba típusától
`E_ERROR`	Hibák	Végzetes hibák (például memória-kiosztási problémák)	Felhasználó értesítése és a program futásának megállítása
`E_WARNING`	Figyelmeztetések	Nem végzetes hibák (például nem szabályos paraméterátadás)	Értesíti a felhasználót, de nem szakítja meg a program futását
`E_PARSE`	Értelmező hiba	Az értelmező nem érti az utasítást	Felhasználó értesítése és a program futásának megállítása
`E_NOTICE`	Megjegyzések	Lehetséges problémaforrás (például egy kezdeti értékkel el nem látott változó)	Értesíti a felhasználót és folytatja a program futtatását
`E_CORE_ERROR`	Belső hiba az értelmező indításakor	Az értelmező indításakor fellépő végzetes hibák	Megszakítja az értelmező indítását
`E_CORE_WARNING`	Figyelmeztető üzenet az értelmező indításakor	Az értelmező indításakor fellépő nem végzetes hibák	Értesíti a felhasználót és folytatja a program futtatását

Ezekkel a beállításokkal nem változtathatjuk meg, mi történjen a hiba felbukkanásakor, csak abba van beleszólásunk, hogy a hibaüzenetet el akarjuk-e küldeni a böngészőnek vagy sem.

Természetesen egyszerre több hibakezelési szintet is engedélyezhetünk, ekkor az egyes szinteket a VAGY (`|`) jellel elválasztva kell megadnunk. A következő sor például engedélyezi a hibaüzenetek és a figyelmeztetések megjelenítését is:

```
error_reporting = E_ERROR|E_WARNING
```

Ha a hibakezelést az összes hibatípusra ki akarjuk terjeszteni, használjuk az E_ALL állandót. Mi a teendő akkor, ha az összes típust engedélyezni szeretnénk, kivéve egyet? A következő sor éppen ezt valósítja meg:

```
error_reporting = E_ALL & ~E_NOTICE
```

Az értelmező a megjegyzéseken kívül minden felmerülő üzenetet elküld a böngészőnek.

Az E_ERROR|E_WARNING és az E_ALL&~E_NOTICE tulajdonképpen kettes számrendszerbeli aritmetikai műveletek, melyek eredménye egy új hibakezelési szintet megadó szám. A kettes számrendszerbeli aritmetikával ebben a könyvben nem foglalkozunk, de a módszer ettől még remélhetőleg érthető marad.

A php.ini beállítását az error_reporting() függvénnyel bírálhatjuk felül, melynek bemenő paramétere a hibakezelési szintet megadó egész szám, visszatérési értéke pedig a megelőző hibakezelési beállítás. Természetesen ennél a függvénynél is használhatjuk az előzőekben megismert állandókat. Lássunk egy példát egy olyan esetre, ahol a hibakezelési szint módosítása a segítségünkre lehet. Lássuk, észrevesszük-e a szándékolt hibát a 22.3. programban.

22.3. program Egy szándékos hiba

```
1: <?php
2: error_reporting( E_ERROR|E_WARNING|E_PARSE );
3: $flag = 45;
4: if ( $flg == 45 ) {
5:     print "Tudom, hogy a \$flag változó értéke 45";
6: } else {
7:     print "Tudom, hogy a \$flag változó értéke NEM 45";
8: };
9: ?>
```


Mint látható, a $flag változó értékét akarjuk vizsgálni, de elgépeltük. Nincs végzetes hiba a programban, így minden további nélkül lefut és az E_ERROR|E_WARNING|E_PARSE hibakezelési beállítás mellett még csak üzenetet sem küld a böngészőnek. A kód bekerül a programba és a lehető legrosszabb pillanatban működésbe lép. Ha azonban az error_reporting() függvénynek az E_ERROR|E_WARNING|E_PARSE|E_NOTICE értéket adnánk át (ez magában foglalja a megjegyzések elküldését is), a program a következő kimenetet küldené:

```
Warning: Undefined variable: flg in /home/httpd/htdocs/
 ➥ peldak/22.3.program.php on line 4
Tudom, hogy a $flag változó értéke NEM 45
```

Azaz:

```
Figyelem: Nem meghatározott változó: flg a /home/httpd/htdocs/
 ➥ peldak/22.3.program.php fájlban a 4. sorban
Tudom, hogy a $flag változó értéke NEM 45
```

Az üzenetből nem csak az derül ki számunkra, hogy valami nincs rendben, de még a problémás sor számát is megtudtuk. Ugyanezt a hatást úgy is elérhetjük, ha az `error_reporting()` függvénynek az `E_ALL` állandót adjuk át paraméterként, ennek azonban lehetnek nem kívánt mellékhatásai is. Elképzelhető az is, hogy szándékosan alkalmazunk nem meghatározott változókat. Vegyük például a következő kódrészletet:

```
<INPUT TYPE="text" NAME="felhasznalo" VALUE="<?
 ➥ print $felhasznalo; ?>">
```

Itt azt használjuk ki, hogy ha egy nem meghatározott változó értékét íratjuk ki, akkor annak hasonló a hatása ahhoz, mintha egy üres karakterláncot jelenítenénk meg (tulajdonképpen semmilyen hatása nincs). A `"felhasznalo"` mező egy előzőleg elküldött értéket vagy semmit sem tartalmaz. Ha a megjegyzések elküldését is engedélyeztük, hibaüzenetet kapunk.

Hibaüzenetek kiírása naplófájlba

Azok a hibák, amelyekre eddig vadásztunk, nagyobb részben azonnali, fejlesztés közben keletkező hibák voltak. Vannak azonban olyan problémák is, amelyek később jönnek elő, például azért, mert megváltozik a futtató környezet. Tegyük fel például, hogy szükségünk van egy programra, amely a felhasználó által kitöltött űrlap adatait írja fájlba. Elkészítjük a programot, kipróbáljuk, mérjük a teljesítményét éles körülmények között, majd magára hagyjuk, hogy végezze a feladatát. Néhány héttel később azonban véletlenül töröljük azt a könyvtárat, amely az adott fájlt tartalmazza.

A PHP `error_log()` függvénye pontosan az ilyen esetekre kifejlesztett hibakereső eszköz. Ez egy beépített függvény, amellyel a kiszolgáló naplófájljába vagy egy általunk megadott fájlba naplózhatjuk a felmerülő hibákat. Az `error_log()` függvénynek két bemenő paramétere van: egy karakterlánc, amely a hiba szövegét tartalmazza és egy egész szám, amely a hiba típusát írja le. A hiba típusától függően egy további paramétert is meg kell adnunk az `error_log()` függvénynek: egy elérési utat vagy egy e-mail címet.

A 22.3. táblázat az elérhető hibatípusokat sorolja fel.

22.3. táblázat Az error_log() függvény hibatípusai

Érték	*Leírás*
0	A hibaüzenetet a `php.ini` fájl `error_log` bejegyzésénél megadott fájlba kell kiírni.
1	A hibaüzenetet a harmadik paraméterben megadott e-mail címre kell küldeni.
3	A hibaüzenetet a harmadik paraméterben megadott fájlba kell kiírni.

Ha hibanaplózást szeretnénk, adjunk meg egy naplófájlt a `php.ini` `error_log` bejegyzésénél. Természetesen a `phpinfo()` függvénnyel megnézhetjük, van-e már ilyen beállítás a `php.ini` fájlban. Ha nincs, adjuk hozzá a fájlhoz például a következő bejegyzést:

```
error_log = /home/httpd/logs/php_errors
```

Ha NT vagy Unix operációs rendszert használunk, a `"syslog"` karakterláncot is hozzárendelhetjük az `error_log` bejegyzéshez. Ekkor a `0`-ás hibatípussal meghívott `error_log()` hibaüzenetek a rendszernaplóba (Unix), illetve az eseménynaplóba (NT) kerülnek. A következő kódrészlet egy hibaüzenetet hoz létre, ha nem tudunk megnyitni egy fájlt:

```
fopen( "./fontos.txt", "a" )
       or error_log( __FILE__.", ".__LINE__.". sor:
 ➥ Nem lehet megnyitni a következő fájlt: ./fontos.txt", 0 );
```

A hibaüzenet a `__FILE__` és `__LINE__` állandókat használja, amelyek az éppen futó program elérési útját, illetve az aktuális sor számát tartalmazzák. A `0`-ás hibatípussal azt adjuk meg, hogy a hibaüzenet a `php.ini` fájlban megadott helyre kerüljön. Ezek alapján valami ilyesmi bejegyzés kerül a naplófájlba:

```
/home/httpd/htdocs/proba5.php, 4. sor:
 ➥ Nem lehet megnyitni a következő fájlt: ./fontos.txt
```

Ha a hibaüzenetet adott fájlba akarjuk kiküldeni, használjuk a `3`-as hibatípus paramétert:

```
if ( ! $megvan = mysql_connect( "localhost", "jozsi",
     "valami" ) )
   {
   error_log( date("d/m/Y H I")." Nem lehet csatlakozni
                                  az adatbázishoz",
              3, "dbhiba.txt" );
   return false;
   }
```

Ha `3`-as típusú hibaüzenetet hozunk létre, egy harmadik paraméterben azt is meg kell adnunk az `error_log()` függvénynek, hogy hová kerüljön a hibaüzenet.

Az `error_log()` függvénynek azt is megadhatjuk, hogy a hibaüzenetet egy e-mail címre postázza. Ehhez az `1`-es típuskódot kell használnunk:

```
if ( ! file_exists( "kritikus.txt" ) )
   or error_log( "Ó, nem! kritikus.txt-nek vége van!",
➥ 1, "veszeset@kritikus.hu" );
```

Ha `1`-es típusú hibaüzenetet küldünk, az `error_log()` harmadik paraméterében meg kell adnunk az e-mail címet is.

A hibaüzenet elfogása

Ha hibaüzeneteket írunk egy fájlba vagy elektronikus levélben küldjük el azokat, hasznos lehet, ha hozzáférünk a PHP által előállított hibaüzenethez.
Ehhez a `php.ini` fájlban a `track_errors` bejegyzést `"On"`-ra kell állítanunk – ennek hatására a legutolsó PHP hibaüzenet bekerül a `$php_errormsg` változóba (hasonlóan a `Perl $!` változójához).

A `$php_errormsg` változó felhasználásával a hibaüzeneteket és naplóbejegyzéseket beszédesebbé tehetjük.

A következő kódrészlet által létrehozott hibaüzenet már sokkal használhatóbb, mint az előzőek:

```
fopen( "./fontos.txt", "a" )
       or error_log( __FILE__.", ".__LINE__.". sor:
                     ".$php_errormsg, 0 );
```

Az üzenet így fog festeni:

```
/home/httpd/htdocs/proba.php, 21. sor:
fopen("./fontos.txt","a") - Permission denied
```

Kézi hibakeresés

Mindig nagyon nehéz azonban az olyan hibák kijavítása, amelyek nem váltanak ki hibaüzenetet.

A hibakeresés egyik kézenfekvő módja, ha megtűzdeljük a kódunkat `print` utasításokkal és kiíratjuk a fontosabb változók értékét. A futás során a változók ugyanis gyakran nem várt értékeket vesznek fel

Egy jellemző nyomkövető sor:

```
print "<p>\$valtozo: $valtozo</p>";
```

Ez egy gyors próba, amit beszúrhatunk a programba, a `$valtozo` értékének ellenőrzésére.

Természetesen a fent alkalmazott módszer nem kötelező érvényű – mindenki kialakíthatja a saját módszerét, például:

```
print "<p>".__LINE__.":  \$teszt értéke: $teszt</p>";
```

Talán a legkényelmesebb módszer, ha írunk egy függvényt a változók értékének nyomon követésére és az `include()` használatával elérhetővé tesszük a fejlesztési szakaszban. A 22.4. példa egy olyan függvényt tartalmaz, amely lehetővé teszi a fejlesztő által megadott formátumú nyomkövetési üzenetek megjelenítését.

22.4. program Formázott nyomkövetési üzenetek megjelenítése

```
 1: <?php
 2: function debug( $sor, $uzenet)
 3:      {
 4:      static $hivasok = 1;
 5:      print "<P><HR>\n";
 6:      print "DEBUG $hivasok: $sor. sor: $uzenet<br>";
 7:      $argumentumok = func_get_args();
 8:      if (  count( $argumentumok ) % 2 )
 9:          print "Páratlan számú paraméter!<BR>";
10:      else
11:          {
12:          for ( $x=2; $x< count($argumentumok); $x += 2 )
13:              {
14:              print "   \$$argumentumok[$x]: "
                        .$argumentumok[$x+1];
```

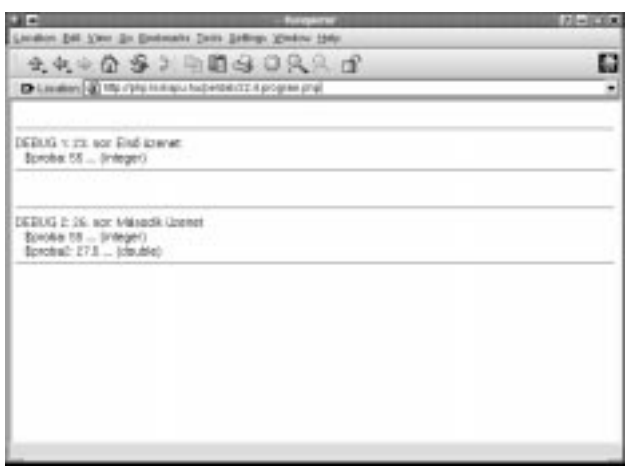
22.4. program (folytatás)

```
15:              print " .... (".gettype(
                         $argumentumok[$x+1] ).")<BR>\n";
16:              }
17:          }
18:      print "<hr><p></p>\n";
19:      $hivasok++;
20:      }
21: $proba = 55;
22: debug( __LINE__, "Első üzenet:", "proba", $proba );
23: $teszt = 66;
24: $proba2 = $proba/2;
25: debug( __LINE__, "Második üzenet", "proba", $proba,
           "proba2", $proba2 );
26: ?>
```

A `debug()` függvénynek egy sorszámot, egy üzenetet, illetve tetszőleges számú név–érték párt adhatunk át. A sor számát és az üzenetet kiírni a feladat könnyebbik része. Ezután ki kell használnunk a PHP 4 azon tulajdonságát, hogy egy függvényt változó számú paraméterrel is meghívhatunk. A `func_get_args()` által visszaadott tömb függvényünk összes paraméterét tartalmazza – ezeket a változókat töltjük be az `$argumentumok` tömbbe. Mivel név–érték párokat várunk, hibaüzenetet kell adnunk, ha páratlan számú elem érkezik a tömbben. Egyébként végiglépkedünk a tömb összes elemén (a harmadikkal kezdve) és kiíratjuk az összes párt a változó adattípusával együtt. A 22.4. ábra a 22.4. program kimenetét mutatja.

22.4. ábra

A debugO függvény használata



Gyakori hibák

Mindannyian követünk el hibákat, különösen akkor, ha vészesen közeledik a leadási határidő. Vannak olyan csapdák, amelyekbe előbb-utóbb minden programozó beleesik.

Észrevesszük a hibát a következő kódrészletben?

```
$valtozo=0;
while ( $valtozo < 42 );
       {
       print "$valtozo<BR>";
       $valtozo++;
       }
```

A kapkodás sokszor vezet hibához – például az utasítássort jelölő pontosvesszőt gyakran kitesszük olyan helyeken is, ahol nem kellene. Így kerülhetett a fenti `while` kifejezés mögé is pontosvessző, ami azt jelzi az értelmezőnek, hogy a ciklus törzse üres – a `$valtozo` értéke soha nem fog nőni, ezért a program végtelen ciklusba kerül.

Lássunk még egy példát:

```
$ertek = 1;
while ( $ertek != 50 )
       {
       print $ertek;
       $ertek+=2;
       }
```

Az elöltesztelő ismétléses vezérlési szerkezet ciklustörzsének végrehajtása a zárójelben található kifejezés logikai értékétől függ. A fenti példában ez a kifejezés csak akkor lesz hamis, ha az `$ertek` változó értéke `50`. Vagyis a ciklustörzs addig ismétlődik, amíg az `$ertek` értéke nem `50`. Csakhogy ez soha nem fog bekövetkezni, mivel a változó értéke `1`-ről indul és minden lépésben kettővel növekszik – vagyis csak páratlan értékeket vesz fel. Ismét sikerült végtelen ciklusba ejtenünk a PHP-t.

A következő kódrészletben rejlő hiba elég alattomos:

```
if ( $proba = 4 )
    {
    print "<P>A \$proba értéke 4</P>\n";
    }
```

A végrehajtást vezérlő logikai kifejezésben nyilvánvalóan ellenőrizni akartuk a `$proba` változó értékét. A kód azonban ehelyett `4`-et rendel a `$proba`-hoz. Mivel a hozzárendelés mint kifejezés értéke a jobb oldali operandussal egyenlő (jelen esetben 4-gyel), a logikai kifejezés értéke a `$proba`-tól függetlenül mindig igaz lesz. Ezt a típusú hibát azért nehéz felfedezni, mert semmilyen üzenetet nem hoz létre, a program egyszerűen nem úgy fog működni, ahogyan azt elvárjuk.

A karakterláncok kezelése során szintén elkövethetünk típushibákat. Az alábbi elég gyakori:

```
$datum = "A mai dátum: . date('d/m/Y H I');
print $datum;
```

Mivel nem zártuk le az idézőjelet, az értelmezőnek fogalma sincs, hol végződik a karakterlánc. Ez szinte mindig hibajelzéssel jár, de nehézkes visszakeresni, mivel gyakran rossz sorszámot kapunk. A zárójelpárok hibás kezelése szintén hasonló jelenséghez vezet.

A következő hibát viszont elég könnyű megtalálni:

```
print "Ön a $tartomany-ról "$felhasznalo"-kent
 ➥ jelentkezett be";
```

Ha idézőjelet akarunk kiíratni, jeleznünk kell az értelmezőnek, hogy itt nem különleges karakterként kell kezelnie. Az előbbi kódrészlet a következőképpen fest helyesen:

```
print "Ön a $tartomany-ról \"$felhasznalo\"-kent
 ➥ jelentkezett be";
```

Összefoglalva, ebben a részben a következő gyakori hibatípusokkal foglalkoztunk:

- Üres ciklustörzs létrehozása hibásan alkalmazott pontosvesszővel.
- Végtelen ciklus létrehozása hibás vezérlőfeltétellel.
- Hozzárendelés használata az „egyenlő” logikai műveletjel helyett.
- Levédetlen idézőjelek használata miatt elcsúszott karakterlánc-határok.

Összefoglalás

A fejlesztés talán egyik leghálátlanabb része a hibakeresés. A megfelelő módszerekkel azonban megkönnyíthetjük az életünket.

Ebben az órában tanultunk arról, hogy a `phpinfo()` függvény segítségével hogyan nyerhetünk információkat a rendszer beállításairól és a környezeti változókról. Az `error_log()` függvénnyel hibaüzeneteket írtunk ki fájlba és küldtünk el elektronikus levélben. Tanultunk a kézi hibakeresésről a fontosabb változók értékének kiíratásával és írtunk egy függvényt a feladat megkönnyítésére. Ha ismerjük az adatokat, amelyekkel programunk dolgozik, viszonylag könnyen és gyorsan megtalálhatjuk a hibákat. Végül megnéztünk néhány olyan típushibát, amelyek még a tapasztaltabb programozók életét is megkeserítik.

Kérdések és válaszok

Létezik módszer, amellyel csökkenthetjük a kódba kerülő hibákat?
Legyünk szemfülesek! Tulajdonképpen szinte lehetetlen egy olyan nagyobb lélegzetű programot létrehozni, amely elsőre tökéletesen működik. Használjunk moduláris tervezést, hogy gyorsan elkülöníthessük a hibát tartalmazó kódrészletet. Olyan gyakran ellenőrizzük a kódot, amilyen gyakran csak lehetséges. Rossz megközelítés, ha megírjuk a programot, elindítjuk, azután várjuk, mi történik. Oldjunk meg egy részfeladatot (lehetőleg függvények formájában), majd próbáljuk ki az elkészült kódot. Ha tökéletesen működik, mehetünk a következő részfeladatra. Próbáljunk „bolondbiztos” kódot készíteni és ellenőrizzük működését szélsőséges körülmények között. Ha az egyik leglényegesebb lépés például az adatok kiírása egy megadott fájlba, nézzük meg, mi történik, ha a fájl hiányzik.

Műhely

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

Kvíz

1. Melyik függvénnyel kaphatunk hasznos információkat a PHP-ről és a rendszerünkről?

2. Melyik függvénnyel jeleníthetjük meg a színekkel kiemelt forráskódot a böngészőablakban?

3. Melyik `php.ini` bejegyzéssel szabályozhatjuk a hibaüzenetek kijelzésének szintjét?

4. Melyik függvénnyel bírálhatjuk felül ezt a beállítást?

5. Melyik függvénnyel naplózhatjuk a hibákat?

6. Melyik beépített változó tartalmazza a hibaüzenetet, ha a `track_errors` bejegyzés engedélyezett a `php.ini` fájlban?

Feladatok

1. Vizsgáljuk meg eddig elkészült programjaink kódját és felépítését az ebben az órában tanult módszerek segítségével.