10. ÓRA

Fájlok használata

A programozási nyelvek egyik legfontosabb lehetősége, hogy fájlokat lehet létrehozni, olvasni, írni velük. A PHP-nek szintén megvannak ezek a tulajdonságai. Ebben a fejezetben ezeket a lehetőségeket fogjuk áttekinteni:

- Hogyan ágyazzunk be fájlokat a dokumentumokba?
- Hogyan vizsgáljuk a fájlokat és könyvtárakat?
- Hogyan nyissunk meg egy fájlt?
- Hogyan olvassunk adatokat a fájlokból?
- Hogyan írjunk egy fájlba vagy fűzzünk hozzá?
- Hogyan zárjunk le egy fájlt?
- Hogyan dolgozzunk könyvtárakkal?

Fájlok beágyazása az include() függvénnyel

Az `include()` függvény lehetőséget ad arra, hogy fájlt ágyazzunk be a PHP dokumentumokba. A fájlban szereplő PHP kód úgy hajtódik végre, mintha a fődokumentum része lenne, ami hasznos, ha egy többoldalas programban külső kódokat szeretnénk beágyazni.

Eddig, ha olyan kódot írtunk, amelyet többször is fel akartunk használni, minden egyes helyre be kellett másolnunk azt. Ha hibát találtunk benne vagy lehetőségeit tovább akartuk bővíteni, mindenütt át kellett írnunk. Ezekre a problémákra megoldás az `include()` függvény. Gyakran használt függvényeinket külső fájlba menthetjük és az `include()` függvénnyel futási időben tölthetjük be programunkba. Az `include()` függvény paramétere a fájl elérési útja. A 10.1. példában látható, hogyan ágyazható be egy fájl a PHP kódba az `include()` függvény segítségével.

10.1. program Az include() használata

```
 1: <html>
 2: <head>
 3: <title>10.1. program Az include() használata</title>
 4: </head>
 5: <body>
 6: <?php
 7: include("10.2.program.php");
 8: ?>
 9: </body>
10: </html>
```

10.2. program A 10.1. példában szereplő beillesztett fájl

```
1: Engem ágyaztak be!!
```

A 10.1. példában található kódba az `10.2.program.php` fájl van beágyazva. Amikor lefuttatjuk a 10.1. példában található kódot, az `"Engem ágyaztak be!!"` szöveg jelenik meg, azaz szöveget ágyaztunk be a PHP kódba. Nem okoz ez gondot? A beágyazott fájl tartalma alapértelmezésben HTML formátumban jelenik meg. Ha végrehajtandó PHP kódot szeretnénk beágyazni, akkor azt az eredeti fájlban PHP címkék közé kell zárni. A 10.3. és 10.4. példában a beágyazott fájlban van a megjelenítő kód.

10.3. program Az include() függvény használata PHP kód végrehajtására más fájlban

```
 1: <html>
 2: <head>
 3: <title>10.3. program Az include() függvény használata
           PHP kód végrehajtására más fájlban</title>
 4: </head>
 5: <body>
 6: <?php
 7: include("10.4.program.php");
 8: ?>
 9: </body>
10: </html>
```

10.4. program PHP kódot tartalmazó beágyazott fájl

```
1: <?php
2: print "Engem ágyaztak be!!<br>""
3: print "De most már össze tudok adni ... 4 + 4 = ".(4+4);
4: ?>
```

A PHP 4-ben a beágyazott fájlokban szereplő kódnak ugyanúgy lehet visszatérési értéket adni, mint a függvényeknek. A visszatérési értéket a fájl vége előtt egy `return` utasítás után kell meghatározni. A 10.5. és 10.6. példákban olyan fájlokat ágyazunk be a dokumentumba, amelyeknek visszatérési értékük van.

10.5. program PHP kód végrehajtása és visszatérési érték megadása az include() függvény használatával

```
 1: <html>
 2: <head>
 3: <title>10.5. program PHP kód végrehajtása és
           visszatérési érték megadása az include()
           függvény használatával</title>
 4: </head>
 5: <body>
 6: <?php
 7: $eredmeny = include("10.6.program.php");
 8: print "A visszatérési érték $eredmeny";
 9: ?>
10: </body>
11: </html>
```

10.6. program Visszatérési értékkel rendelkező beágyazott fájl

```
1: <?php
2: $vissza = ( 4 + 4 );
3: return $vissza;
4: ?>
5: Ez a HTML soha nem jelenik meg, mert a visszatérés
   után írtuk!
```

PHP 3 használatakor a beágyazott fájlnak csak akkor lehet visszatérési értéket adni a return utasítással, ha függvényben szerepel. A 10.6. példa a PHP 3-ban hibát okoz.

Ha az `include()` függvényt alkalmazzuk, megadhatunk feltételeket is. A megadott fájlt csak akkor ágyazza be az `include()`, ha a fájlra teljesülnek a feltételek. A következő kódban található beágyazás soha nem fog végrehajtódni:

```
$proba = false;
if ( $proba )
    {
    include( "egy_fajl.txt" ); //nem ágyazza be
    }
```

Ha az `include()` függvényt ciklusban használjuk, az `include()`-ban megadott fájl tartalma minden ismétlésnél bemásolódik és a fájlban lévő kód minden hívásnál végrehajtódik. A 10.7. példában található kódban az `include()` függvény egy `for` ciklus belsejében szerepel. Az `include()` paramétere minden ismétlésnél más fájlra mutat.

10.7. program Az include() használata cikluson belül

```
1: <html>
2: <head>
3: <title>10.7. program Az include() használata ciklu-
son belül</title>
4: </head>
5: <body>
6: <?php
7: for ( $x = 1; $x<=3; $x++ )
8:     {
9:     $incfajl = "incfajl$x.txt";
```

10.7. program (folytatás)

```
10:     print "Megpróbálom beágyazni az $incfajl -t";
11:     include( "$incfajl" );
12:     print "<p>";
13:     }
14: ?>
15: </body>
16: </html>
```


A 10.7. programban található kód futás közben három különböző fájlt ágyaz be, ezek az `"incfajl1.txt"`, `"incfajl2.txt"` és `"incfajl3.txt"`. Mindegyik fájl tartalmazza a nevét és azt a szöveget, hogy beágyazták:

```
Megpróbálom beágyazni az incfajl1.txt -t
Ez a fájl az incfajl1.txt
Megpróbálom beágyazni az incfajl2.txt -t
Ez a fájl az incfajl2.txt
Megpróbálom beágyazni az incfajl3.txt -t
Ez a fájl az incfajl3.txt
```

Rendelkezésre áll egy másik PHP függvény is, a require(), amely hasonlóan működik az include()-hoz. Ciklusban azonban a require() nem használható úgy, mint az include(), ugyanis a require() a program futásának kezdetekor helyettesítődik be. Még egy fontos különbség a két függvény között, hogy a require()-nak nem lehet visszatérési értéke a PHP 4-esben.

Fájlok vizsgálata

Mielőtt elkezdünk ismerkedni a fájlokkal és könyvtárakkal, fussunk át néhány velük kapcsolatos tudnivalót. A PHP 4 igen sok lehetőséget ad arra, hogy a rendszerünkben található fájlokon különböző műveleteket hajtsunk végre. A következő bekezdésekben ezeket a függvényeket tekintjük át.

Fájl létezésének ellenőrzése a file_exists() függvénnyel

Fájl létezését a `file_exists()` függvénnyel vizsgálhatjuk. A `file_exists()` paramétere egy karakterlánc, amely a kérdéses fájl elérési útját és nevét tartalmazza. A függvény visszatérési értéke `true`, ha a fájl létezik, egyébként `false`.

```
if ( file_exists("proba.txt") )
     print("A fájl létezik!");
```

Fájl vagy könyvtár?

Az `is_file()` függvény azt dönti el, hogy paramétere fájl-e. A paraméterben meg kell adnunk egy elérési utat. Ha a megadott elérési út fájlt jelöl, a függvény visszatérési értéke `true`, egyébként `false` lesz.

```
if ( is_file( "proba.txt" ) )
   print("a proba.txt egy fájl");
```

Ellenőrizhetjük azt is, hogy az elérési út könyvtárat jelöl-e. Az ellenőrzést az `is_dir()` függvény végzi. Ha a függvény paraméterében megadott elérési út könyvtárat határoz meg, a visszatérési érték `true`, egyébként `false` lesz.

```
if ( is_dir( "/tmp" ) )
   print "a /tmp egy könyvtár";
```

Fájl állapotának lekérdezése

Miután megtudtuk, hogy egy fájl létezik és valóban fájl, különféle dolgokat végezhetünk vele. Általában írunk bele, olvassuk vagy végrehajtjuk. A PHP-ben különböző függvények állnak rendelkezésre ezen lehetőségek lekérdezésére.

Az `is_readable()` függvénnyel megtudhatjuk, hogy az adott fájl olvasható-e számunkra. UNIX rendszerben lehet, hogy látjuk a fájlt a könyvtárszerkezetben, de nincs jogosultságunk annak olvasására. Ha az `is_readable()` paraméterében elérési úttal megadott fájl olvasható, visszatérési értéke `true`, egyébként `false` lesz.

```
if ( is_readable( "proba.txt" ) )
   print "a proba.txt olvasható";
```

Az `is_writeable()` függvénnyel megtudhatjuk, hogy az adott fájl írható-e számunkra. Ha az `is_writeable()` paraméterében elérési úttal megadott fájl írható, visszatérési értéke `true`, egyébként `false` lesz.

```
if ( is_writeable( "proba.txt" ) )
   print "a proba.txt írható";
```

Az `is_executable()` függvénnyel azt tudhatjuk meg, hogy az adott fájl futtatható-e. A visszatérési érték jogosultságunktól és a fájl kiterjesztésétől függ. Ha az `is_executable()` paraméterében elérési úttal megadott fájl futtatható, a visszatérési érték `true`, egyébként `false` lesz.

```
if ( is_executable( "proba.txt" )
   print "a proba.txt futtatható";
```

Fájl méretének lekérdezése a filesize() függvénnyel

A `filesize()` függvény a paraméterében elérési úttal megadott fájl bájtban számított méretével tér vissza. A végeredmény `false`, ha bármilyen hiba történik.


```
print "A proba.txt mérete: ";
print filesize( "proba.txt" );
```

Különféle fájlinformációk

Szükségünk lehet rá, hogy tudjuk, mikor hozták létre a fájlt vagy mikor használták utoljára. A PHP-vel ezeket az információkat is megszerezhetjük.

A fájl utolsó megnyitásának dátumát a `fileatime()` függvény segítségével kaphatjuk meg. A függvény paraméterében meg kell adni a fájl elérési útját. Visszatérési értékként az utolsó megnyitás dátumát kapjuk meg. A megnyitás jelenthet írást vagy olvasást is. Ezen függvények visszatérési értéke UNIX időbélyeg formátumú, azaz mindig az 1970 január elseje óta eltelt másodpercek száma. A következő példában ezt a számot a `date()` függvény segítségével olvasható formára alakítjuk. A `date()` függvény leírása a tizenötödik órában szerepel.

```
$atime = fileatime( "proba.txt" );
print "A proba.txt legutolsó megnyitásának dátuma:";
print date("Y.m.d. H:i", $atime);
//Egy minta végeredmény: 2000.01.23. 14:26
```

A `filemtime()` függvénnyel a fájl utolsó módosításának idejét kaphatjuk meg. A függvény paramétere a fájl elérési útja, visszatérési értéke pedig UNIX időbélyeg formátumú. A módosítás azt jelenti, hogy a fájl tartalma valamilyen módon megváltozott.

```
$mtime = filemtime( "proba.txt" );
print "A proba.txt utolsó módosításának dátuma:";
print date("Y.m.d. H:i", $mtime);
//Egy minta végeredmény: 2000.01.23. 14:26
```

A `filectime()` a fájl utolsó változásának időpontját adja meg. UNIX rendszerben a változás alatt azt értjük, amikor a fájl tartalma valamilyen módon módosul vagy a jogosultságok, illetve a tulajdonos megváltozik. Más rendszerekben a `filectime()` visszatérési értéke a fájl létrehozásának dátuma.

```
$ctime = filectime( "proba.txt" );
print "A proba.txt utolsó változásának dátuma:";
print date("Y.m.d. H:i", $ctime);
//Egy minta végeredmény: 2000.01.23. 14:26
```

Több fájltulajdonságot egyszerre megadó függvény

A 10.8. példában olyan függvényt hozunk létre, amely az előzőekben látott lehetőségeket fogja össze egyetlen függvényben.

10.8. program Egyszerre több fájltulajdonságot megadó függvény

```
 1: <html>
 2: <head>
 3: <title>10.8. program Egyszerre több fájltulajdonságot
          megadó függvény</title>
 4: </head>
 5: <body>
 6: <?php
 7: $fajl = "proba.txt";
 8: fileInformaciok( $fajl );
 9: function fileInformaciok( $f )
10:    {
11:    if ( ! file_exists( $f ) )
12:       {
13:       print "$f nem létezik <BR>";
14:       return;
15:       }
16:       print "$f ".(is_file( $f )?"":"nem ")."fájl<br>"
17:       print "$f ".(is_dir( $f )?"":"nem  ").
                "könyvtár<br>"
18:       print "$f ".(is_readable( $f )?"":"nem ").
                "olvasható<br>"
19:       print "$f ".(is_writeable( $f )?"":"nem ").
                "írható<br>"
20:       print "$f ".(is_executable( $f )?"":"nem ").
                "futtatható<br>"
21:       print "$f ".(filesize( $f )."bájt méretű<br>"
22:       print "$f utolsó megnyitásának dátuma: ".
                date( "Y.m.d. H:i", fileatime( $f ) ).
                "<br>";
23:       print "$f utolsó módosításának dátuma: ".
                date( "Y.m.d. H:i", filemtime( $f ) ).
                "<br>";
```

10.8. program (folytatás)

```
24:        print "$f utolsó változásának dátuma: ".
                 date( "Y.m.d. H:i", filectime( $f ) ).
                 "<br>";
25:     }
26:
27: ?>
28: </body>
29: </html>
```


Az egyszerűbb leírás kedvéért ? : műveleteket használtunk. Nézzük meg ezek közül az egyiket:

```
print "$f ".(is_file( $f )?"":"nem ")."fájl<br>"
```

A műveletjel bal oldalára került az `is_file()` függvény. Ha ennek eredménye `true`, akkor az operátor visszatérési értéke üres karakterlánc, egyébként a `"nem "` szöveg. A visszatérési érték után a szövegösszefűzés jelet követő karakterlánc is kiíródik. Az előbbi kifejezés kevésbé tömör formában a következőképpen írható le:

```
$igen_vagy_nem = is_file( $f ) ? "" : "nem ";
print "$f $igen_vagy_nem"."fájl<br>"
```

Az `if` vezérlési szerkezettel még világosabb kódot írhatunk, ekkor azonban a program mérete jelentősen nő.

```
if ( is_file( $f ) )
   print "$f fájl<br>";
else
   print "$f nem fájl<br>";
```

Mivel az előző példák ugyanannak a feladatnak a megoldásai, szabadon választhatjuk ki a nekünk tetszőt.

Fájlok létrehozása és törlése

Ha egy fájl nem létezik, a `touch()` függvény segítségével hozhatjuk létre. A függvény paraméterében meg kell adni egy elérési utat. Ezen az elérési úton próbál meg létrehozni a `touch()` egy üres fájlt. Ha a fájl már létezik, tartalma nem változik meg, de a módosítás dátuma a `touch()` függvény végrehajtási idejére módosul.

```
touch("sajat_fajl.txt");
```

Létező fájlt törölni az `unlink()` függvénnyel lehet. Az `unlink()` paramétere a fájl elérési útja:

```
unlink("sajat_fajl.txt");
```

A létrehozás, törlés, írás, olvasás, módosítás csak akkor lehetséges egy fájlon, ha a megfelelő jogosultságokkal rendelkezünk.

Fájl megnyitása írásra, olvasásra, hozzáfűzésre

Mielőtt elkezdünk dolgozni egy fájllal, meg kell nyitnunk írásra vagy olvasásra. Ezt az `fopen()` függvénnyel tehetjük meg. A függvény paraméterében meg kell adni a fájl elérési útját és a megnyitási módot. A legtöbbször használt módok az olvasás (`"r"`), írás (`"w"`), hozzáfűzés (`"a"`). A függvény visszatérési értéke egy egész szám. Ez az egész szám az úgynevezett fájlazonosító, amelyet változóként tárolhatunk. Ezzel hivatkozhatunk később a fájlra. Fájl olvasásra való megnyitásához a következőt kell beírnunk:

```
$fa = fopen( "proba.txt", 'r' );
```

Az írásra való megnyitáshoz a következőt:

```
$fa = fopen( "proba.txt", 'w' );
```

Ha a fájlt hozzáfűzésre akarjuk megnyitni, tehát nem kívánjuk felülírni a tartalmát, csak a végéhez szeretnénk fűzni valamit, a következőt kell tennünk:

```
$fa = fopen( "proba.txt", 'a' );
```

Az `fopen()` false értékkel tér vissza, ha valamiért nem tudta megnyitni a fájlt, ezért érdemes mindig ellenőrizni, hogy sikeres volt-e a megnyitás. Ezt például az `if` vezérlési szerkezettel tehetjük meg:

```
if ( $fa = fopen( "proba.txt", "w" ) )
    {
    // $fa-val csinálunk valamit
    }
```

Esetleg használhatunk logikai műveletet, hogy megszakítsuk a végrehajtást, ha a fájl nem létezik:

```
( $fa = fopen( "proba.txt", "w" ) ) or die
 ➥ ("A fájl sajnos nem nyitható meg!");
```

Ha az `fopen()` `true` értékkel tér vissza, a `die()` nem hajtódik végre, különben a kifejezés jobb oldalán a `die()` kiírja a paraméterében szereplő karakterláncot és megszakítja a program futását.

Amikor befejeztük a munkát egy fájllal, mindig be kell zárnunk azt. Ezt az `fclose()` függvénnyel tehetjük meg, amelynek paraméterében azt a fájlazonosítót kell megadnunk, amelyet egy sikeres `fopen()` végrehajtása során kaptunk:


```
fclose( $fa );
```

Olvasás fájlból

A PHP-ben egy fájlból különböző függvények segítségével bájtonként, soronként vagy karakterenként olvashatunk.

Sorok olvasása fájlból az fgets() és feof() függvényekkel

Miután megnyitottunk egy fájlt, beolvashatjuk a tartalmát sorról sorra, az `fgets()` függvénnyel. A függvény paraméterében meg kell adni az olvasandó fájl azonosítóját (amit az `fopen()` függvény ad a megnyitáskor), továbbá második paraméterként kell adnunk egy egész számot is, amely meghatározza, hogy legfeljebb hány bájtot olvasson ki a PHP, amíg sorvége vagy fájlvége jelet nem talál. Az `fgets()` függvény addig olvas a fájlból, amíg újsor karakterhez (`"\n"`) nem ér, a megadott bájtnyi adatot ki nem olvassa vagy a fájl végét el nem éri.

```
$sor = fgets( $fa, 1024 ); // ahol az $fa az fopen() által
 ➥ visszaadott fájlazonosító
```

Tudnunk kell tehát, mikor érünk a fájl végére. Ezt az `feof()` függvény adja meg, melynek visszatérési értéke `true`, ha a fájl végére értünk, egyébként `false`.
A függvény paraméterében egy fájlazonosítót kell megadni.

```
feof( $fa ); // ahol az $fa az fopen() által visszaadott
 ➥ fájlazonosító
```

A 10.9. példában láthatjuk, hogyan kell egy fájlt sorról sorra beolvasni.

10.9. program Fájl megnyitása és sorról sorra olvasása

```
 1: <html>
 2: <head>
 3: <title>10.9. program Fájl megnyitása és sorról sorra
          olvasása</title>
 4: </head>
 5: <body>
 6: <?php
 7: $fajlnev = "proba.txt";
 8: $fa = fopen( $fajlnev, "r" ) or die("$fajlnev nem
     nyitható meg");
 9: while ( ! feof( $fa ) )
10:    {
11:    $sor = fgets( $fa, 1024 );
12:    print "$sor<br>";
13:    }
14: fclose($fa);
15: ?>
16: </body>
17: </html>
```

Az `fopen()` függvénnyel próbáljuk olvasásra megnyitni a fájlt. Ha a kísérlet sikertelen, a `die()` függvénnyel megszakítjuk a program futását. Ez legtöbbször akkor történik meg, ha a fájl nem létezik vagy (UNIX rendszerben) nincs megfelelő jogosultságunk a megnyitásra. Az olvasás a `while` ciklusban történik. A `while` ciklus minden ismétlés alkalmával megnézi az `feof()` függvénnyel, hogy az olvasás elért-e a fájl végéhez, tehát a ciklus addig olvas, amíg el nem éri a fájl végét. Az `fgets()` minden ismétlésnél kiolvas egy sort vagy 1024 bájtot. A kiolvasott karaktersorozatot a `$sor` változóba mentjük és kiírjuk a böngészőbe. Minden kiírásnál megadunk egy `<br>` címkét, hogy a szöveg olvashatóbb legyen, végül bezárjuk a megnyitott állományt.

Tetszőleges mennyiségű adat olvasása fájlból

A fájlokból nem csak soronként, hanem előre meghatározott méretű darabokat is olvashatunk. Az `fread()` függvény paraméterében meg kell adni egy fájlazonosítót és az egyszerre kiolvasandó adatok mennyiségét, bájtban. A függvény visszatérési értéke a megadott mennyiségű adat lesz, kivéve, ha elérte a fájl végét.

```
$reszlet = fread( $fa, 16 );
```

A 10.10. példaprogram 16 bájtonként olvassa be a fájlt.

10.10. program Fájl beolvasása az fread() függvénnyel

```
 1: <html>
 2: <head>
 3: <title>10.10. program Fájl beolvasása az fread()
          függvénnyel</title>
 4: </head>
 5: <body>
 6: <?php
 7: $fajlnev = "proba.txt";
 8: $fa = fopen( $fajlnev,  "r" ) or die("$fajlnev nem
     nyitható meg");
 9: while ( ! feof( $fa ) )
10:    {
11:    $reszlet = fread( $fa, 16 );
12:    print "$reszlet<br>";
13:    }
14: fclose($fa);
15: ?>
16: </body>
17: </html>
```

Bár az `fread()` függvénynek megadjuk, mennyi adatot olvasson ki, azt nem tudjuk meghatározni, hogy honnan kezdje az olvasást. Erre az `fseek()` ad lehetőséget, ezzel a függvénnyel állítható be az olvasás helye. Paraméterként egy fájlazonosítót és egy egész számot kell megadnunk, amely a fájl elejétől bájtban mérve meghatározza az új olvasási helyet:

```
fseek( $fa, 64 );
```

A 10.11. példában az `fseek()` és `fread()` függvényekkel a fájl tartalmának második felét jelenítjük meg a böngészőben.

10.11. program Az fseek() használata

```
1: <html>
2: <head>
3: <title>10.11. program Az fseek() használata</title>
4: </head>
```

10.11. program (folytatás)

```
 5: <body>
 6: <?php
 7: $fajlnev = "proba.txt";
 8: $fa = fopen( $fajlnev,  "r" ) or die("$fajlnev nem
     nyitható meg");
 9: $fajlmeret = filesize($fajlnev);
10: $felmeret = (int)( $fajlmeret / 2 );
11: print "A fájl felének mérete: $felmeret <br>\n"
12: fseek( $fa, $felmeret );
13: $reszlet = fread( $fa, ($fajlmeret - $felmeret) );
14: print $reszlet;
15: fclose($fa);
16: ?>
17: </body>
18: </html>
```

A felezőpontot úgy számoltuk ki, hogy a `filesize()` függvénnyel lekértük a fájl méretét és elosztottuk kettővel. Ezt az értéket használtuk az `fseek()` második paramétereként. Végül az `fread()` segítségével kiolvastuk a fájl második felét és kiírtuk a böngészőbe.

Fájl karakterenkénti olvasása az fgetc() függvénnyel

Az `fgetc()` függvény hasonlít az `fgets()` függvényhez, de minden alkalommal, amikor meghívjuk, csak egyetlen karaktert olvas ki a fájlból. Mivel egy karakter mindig egy bájt méretű, más paramétert nem kell megadnunk, csak a már megszokott fájlazonosítót:

```
$karakter = fgetc( $fa );
```

A 10.12. példaprogram egy ciklus segítségével karakterenként kiolvassa a `"proba.txt"` tartalmát és minden karaktert külön sorba ír ki a böngészőbe.

10.12. program Az fgetc() használata

```
1: <html>
2: <head>
3: <title>10.12. program Az fgetc() használata</title>
4: </head>
5: <body>
```

10.12. program (folytatás)

```
 6: <?php
 7: $fajlnev = "program.txt";
 8: $fa = fopen( $fajlnev, "r" ) or die("$fajlnev nem
     nyitható meg");
 9: while ( ! feof( $fa ) )
10:     {
11:     $karakter = fgetc( $fa );
12:     print "$karakter<br>";
13:     }
14: fclose($fa);
15: ?>
16: </body>
17: </html>
```


Fájlba írás és hozzáfűzés

A fájlba írás és a hozzáfűzés nagyon hasonlítanak egymáshoz. Az egyetlen különbség az `fopen()` hívásában rejlik. Amikor írásra nyitjuk meg a fájlt, akkor az `fopen()` második paramétereként a `"w"` karaktert kell megadnunk:

```
$fa = fopen( "proba.txt", "w" );
```

Minden írási próbálkozás a fájl elején történik. Ha a fájl még nem létezne, a rendszer létrehozza azt. Ha a fájl létezik, tartalma felülíródik a megadott adatokkal.

Ha a fájlt hozzáfűzésre nyitjuk meg, az `fopen()` második paramétereként az `"a"` karaktert kell megadnunk:

```
$fa = fopen( "proba.txt", "a" );
```

Hozzáfűzésnél minden adat a fájl végére íródik, megtartva az előző tartalmat.

Fájlba írás az fwrite() és fputs() függvényekkel

Az `fwrite()` függvény paramétereként egy fájlazonosítót és egy karakterláncot kell megadni. A karakterlánc a megadott fájlba íródik. Az `fputs()` ugyanígy működik.

```
fwrite( $fa, "hello világ" );
fputs( $fa, "hello világ" );
```

A 10.13. példában először az `fwrite()` függvény használatával egy fájlba írunk, majd az `fputs()` függvénnyel adatokat fűzünk hozzá.

10.13. program Fájlba írás és hozzáfűzés

```
 1: <html>
 2: <head>
 3: <title>10.11. program Fájlba írás és
           hozzáfűzés</title>
 4: </head>
 5: <body>
 6: <?php
 7: $fajlnev = "proba.txt";
 8: print "$fajlnev fájlba írás";
 9: $fa = fopen( $fajlnev,  "w" ) or die("$fajlnev nem
     nyitható meg");
10: fwrite ( $fa, "Hello világ\n");
11: fclose( $fa );
12: print "$fajlnev fájlhoz hozzáfűzés";
13: $fa = fopen( $fajlnev,  "a" ) or die("$fajlnev nem
     nyitható meg");
14: fputs ( $fa, "És más dolgok");
15: fclose( $fa );
16: ?>
17: </body>
18: </html>
```

Fájlok zárolása az flock() függvénnyel

Az eddig megtanultak igen jól használhatók, ha programjaink csak egyetlen felhasználót szolgálnak ki. A valóságban azonban egy alkalmazás oldalait általában többen is el szeretnék érni egyidőben. Képzeljük el, mi történne, ha egyszerre két felhasználó írna ugyanabba a fájlba. A fájl használhatatlanná válna.

Az `flock()` függvény használatával kizárhatjuk ezt a lehetőséget. Az `flock()` függvénnyel zárolt fájlt nem olvashatja vagy írhatja más folyamat, amíg a zárolás érvényben van. Az `flock()` függvény paramétereként egy fájlazonosítót és egy egész számot kell megadni. Ez utóbbi a zárolás típusát határozza meg. A lehetséges zárolásokat a 10.1. táblázat mutatja.

10.1. táblázat Az flock() függvény második paraméterének lehetséges értékei

Egész	*Lezárás típusa*	*Leírás*
1	Megosztott	Más folyamatok olvashatják a fájlt, de nem írhatnak bele (akkor használjuk, amikor olvassuk a fájlt)
2	Kizáró	Más folyamatok nem olvashatják és nem írhatják a fájlt (akkor használjuk, amikor írunk a fájlba)
3	Felszabadítás	A fájl zárolásának megszüntetése

Az `flock()` függvényt a fájl megnyitása után alkalmazzuk zárolásra és az `fclose()` előtt oldjuk fel vele a zárat.

```
$fa = fopen( "proba.txt", "a" );
flock( $fa, 2 ); // kizáró lefoglalás
// fájlba írás
flock( $fa, 3 ); // zárolás feloldása
fclose( $fa );
```

Az `flock()` a PHP zárolási megoldása. Csak azok a programok veszik figyelembe ezt a fajta zárolást, ahol ugyanezt a módszert alkalmazzuk, ezért előfordulhat, hogy a nem-PHP programok megnyitják a fájlunkat, akár írás közben is.

Munka könyvtárakkal

Az előzőekben áttekintettük, hogyan kezelhetjük a fájlokat, most megnézzük, hogyan hozhatunk létre, törölhetünk és olvashatunk könyvtárakat a PHP-ben.

Könyvtár létrehozása az mkdir() függvénnyel

Könyvtárat az `mkdir()` függvénnyel hozhatunk létre. Az `mkdir()` paraméterében meg kell adni egy karakterláncot, amely a létrehozandó könyvtár elérési útja, valamint egy oktális (nyolcas számrendszerű) számot, amely a jogosultságokat határozza meg. Az oktális számok elé mindig 0-t kell írni. A jogosultság megadásának csak UNIX rendszerekben van hatása. A jogosultsági mód három 0 és 7 közé eső számot tartalmaz, amelyek sorban a tulajdonos, a csoport, és mindenki más

jogait adják meg. A függvény visszatérési értéke `true`, ha a könyvtárat sikerült létrehozni, egyébként `false`. A könyvtár létrehozása általában akkor nem sikeres, ha a megadott elérési úton nincs jogosultságunk könyvtárat létrehozni, azaz nincs jogosultságunk írásra.

```
mkdir( "proba_konyvtar", 0777 ); // teljes
 ➥ írás/olvasás/végrehajtás jogok
```

Könyvtár törlése az rmdir() függvénnyel

A rendszerből könyvtárat az `rmdir()` függvénnyel törölhetünk. A sikeres törléshez megfelelő jogosultsággal kell rendelkeznünk és a könyvtárnak üresnek kell lennie. A függvény paraméterében a törlendő könyvtár elérési útját kell megadni.

```
rmdir( "proba_konyvtar" );
```

Könyvtár megnyitása olvasásra

Mielőtt be tudnánk olvasni egy könyvtár tartalmát, szükségünk van egy könyvtárazonosítóra. Ezt az azonosítót az `opendir()` függvény adja meg. A függvény paraméterében annak a könyvtárnak az elérési útját kell átadni, amelyet olvasni szeretnénk. A függvény visszatérési értéke a könyvtár azonosítója, kivéve, ha a könyvtár nem létezik vagy nincs jogosultságunk az olvasására. Ezekben az estekben a visszatérési érték `false`.

```
$kvt = opendir( "proba_konyvtar" );
```

Könyvtár tartalmának olvasása

Ahogy az `fgets()` függvénnyel fájlból olvastunk, ugyanúgy használhatjuk a `readdir()` függvényt, hogy fájl vagy könyvtárnevet olvassunk ki egy megnyitott könyvtárból. A `readdir()` paraméterében meg kell adni az olvasandó könyvtár azonosítóját. A függvény visszatérési értéke a könyvtár következő elemének neve. Ha a könyvtár végére értünk, a visszatérési érték `false`. A `readdir()` csak az elem nevét és nem annak elérési útját adja meg. A 10.4. példában a `readdir()` függvény használati módját láthatjuk.

10.14. program Könyvtár tartalmának kiíratása

```
1: <html>
2: <head>
3: <title>10.14. program Könyvtár tartalmának
          kiíratása</title>
4: </head>
```

10.14. program (folytatás)

```
 5: <body>
 6: <?php
 7: $kvtnev = "proba_konyvtar";
 8: $kvt = opendir( $kvtnev );
 9: while ( gettype( $fajl = readdir( $kvt )) != boolean )
10:    {
11:    if ( is_dir( "$kvtnev/$fajl" ) )
12:       print "(D)";
13:    print "$fajl<br>";
14:    }
15: closedir( $kvt );
16: ?>
17: </body>
18: </html>
```


A könyvtárat megnyitjuk az `opendir()` függvénnyel, majd egy `while` ciklussal végiglépkedünk annak összes elemén. A `while` ciklus feltételes kifejezésében meghívjuk a `readdir()` függvényt és a visszaadott értéket hozzárendeljük a `$fajl` változóhoz. A ciklus törzsében az `is_dir()` függvénnyel vizsgáljuk, hogy a `$kvtnev` és a `$fajl` változókból készített elérési út könyvtárat jelöl-e. Ha igen, neve elé teszünk egy `"(D)"` jelet. Így kiírjuk a böngészőbe a könyvtár tartalmát.

A `while` ciklus feltételes kifejezésének megírásakor igen elővigyázatosak voltunk. Sok PHP programozó a következőt használta volna:

```
while ( $fajl = readdir( $kvt ) )
       {
       print "$fajl<br>\n";
       }
```

Itt a `readdir()` visszatérési értékét vizsgáljuk. Minden `"0"`-tól különböző karakterlánc `true`-ként viselkedik, tehát ebből nem lehet probléma. Mi történik, ha könyvtárunk négy fájlt tartalmaz, melyek nevei `"0"`, `"1"`, `"2"`, `"3"`. Az előbbi kód a következő végeredményt adja:

```
.
..
```

Amikor a ciklus eléri a `"0"` nevű fájlt, a `readdir() false` értéknek veszi és a ciklus leáll. A 10.14. példában a `readdir()` függvény visszatérési értékének típusát vizsgáljuk és ezzel oldjuk meg a problémát.

Összefoglalás

Ebben az órában megtanultuk, hogyan ágyazhatunk be a dokumentumokba külső fájlban tárolt PHP kódot. Áttekintettük a fájlok különböző tulajdonságainak ellenőrzését, megnéztünk, hogyan olvashatunk fájlokat sorról sorra, karakterenként, vagy meghatározott részletekben. Megtanultuk, hogyan írhatunk fájlba és hogyan fűzhetünk karakterláncokat hozzá. Végül áttekintettük, hogyan hozhatunk létre, törölhetünk, vagy listázhatunk ki könyvtárakat.

Kérdések és válaszok

Az `include()` függvény lassítja a programok futását?
Mivel a beágyazott fájlt meg kell nyitni és a tartalmát be kell olvasni, ezért azt kell mondjuk, hogy igen, a lassulás azonban nem számottevő.

Mindig le kell állítani a program futását, ha egy fájlt nem sikerült megnyitni írásra vagy olvasásra?
Ha a program futásához elengedhetetlen a fájl, akkor a `die()` függvénnyel érdemes leállítani. Más esetben a program futását nem fontos megszakítani, de érdemes figyelmeztetni a felhasználót a hibára vagy feljegyezni a sikertelen megnyitási kísérletet egy naplóállományba. A huszonkettedik órában többet olvashatunk erről a megoldásról.

Műhely

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

Kvíz

1. Melyik függvénnyel adható programunkhoz külső kódot tartalmazó fájl?

2. Melyik függvénnyel tudjuk meg, hogy rendszerünkben megtalálható-e egy fájl?

3. Hogyan kapható meg egy fájl mérete?

4. Melyik függvénnyel nyitható meg egy fájl írásra vagy olvasásra?

5. Melyik függvénnyel olvashatunk ki egy sort egy fájlból?

6. Honnan tudhatjuk meg, hogy elértük a fájl végét?

7. Melyik függvényt használjuk, hogy egy sort írjunk egy fájlba?

8. Hogyan nyitunk meg egy könyvtárt olvasásra?

9. Melyik függvényt használjuk, hogy egy könyvtár elemeinek nevét megkapjuk, miután megnyitottuk azt?

Feladatok

1. Készítsünk egy oldalt, amely bekéri a felhasználó vezeték- és keresztnevét. Készítsünk programot, amely ezen adatokat fájlba menti.

2. Készítsünk programot, amely az előbbi feladatban mentett adatokat kiolvassa a fájlból. Írjuk ki azokat a böngészőbe (minden sor végén használjunk `<BR>` címkét). Írjuk ki, hány sort tartalmaz a fájl, illetve a fájl méretét.