17. ÓRA

Karakterláncok kezelése

A Világháló valójában szöveges fájlokra épülő környezet, és igazából nem számít, mivel gazdagodik a jövőben a tartalma, a mélyén mindig szöveges állományokat fogunk találni. Így nem meglepő, hogy a PHP 4 sok olyan függvényt biztosít, amellyel szövegműveletek végezhetők.

Az óra során a következőket tanuljuk meg:

- Hogyan formázzunk karakterláncokat?
- Hogyan határozzuk meg a karakterláncok hosszát?
- Hogyan találjunk meg egy karakterláncon belüli karakterláncot?
- Hogyan bontsuk szét a karakterláncot alkotóelemeire?
- Hogyan távolítsuk el a szóközöket a karakterláncok végéről vagy elejéről?
- Hogyan cseréljünk le karakterlánc-részleteket?
- Hogyan változtassuk egy karakterláncban a kisbetűket nagybetűre és fordítva?

Karakterláncok formázása

A megjeleníteni kívánt karakterláncot eddig egyszerűen kiírattuk a böngészőbe. A PHP két olyan függvényt tartalmaz, amely lehetővé teszi az előzetes formázást, függetlenül attól, hogy tizedestörteket kell valahány tizedes pontosságra kerekíteni, egy mezőn belül kell jobbra vagy balra igazítani valamit, vagy egy számot kell különböző számrendszerekben megjeleníteni. Ebben a részben a `printf()` és az `sprintf()` függvények által biztosított formázási lehetőségekkel ismerkedünk meg.

A printf() függvény használata

Ha már dolgoztunk C-vel, biztosan ismerős lesz a `printf()` függvény, amelynek PHP-s változata hasonló, de nem azonos azzal. A függvény bemenete egy karakterlánc, más néven a formátumvezérlő karakterlánc (röviden formázó karakterlánc vagy egyszerűen formázó), emellett további, különböző típusú paraméterek. A formázó karakterlánc ezen további paraméterek megjelenítését határozza meg. A következő kódrészlet például a `printf()` függvényt használja, hogy egy egész számot decimális értékként írjon ki:

```
printf("az én számom az %d", 55 );
// azt írja ki, hogy "az én számom az 55"
```

A formázó karakterláncban (ami az első paraméter) egy különleges kódot helyeztünk el, amely átalakítási meghatározásként ismert.

ÚJDONSÁG Az átalakítási meghatározás százalékjellel (`%`) kezdődik, és azt határozza meg, hogyan kell a `printf()` függvény neki megfelelő paraméterét kezelni. Egyetlen formátumvezérlő karakterláncba annyi átalakítási meghatározást írhatunk, amennyit csak akarunk, feltéve, hogy a `printf()` függvénynek ugyanennyi paramétert adunk át a formázót követően.

A következő kódrészlet két számot ír ki a `printf()` használatával:

```
printf("Az első szám: %d<br>\nA második szám: %d<br>\n",
       55, 66 );
// A kiírt szöveg:
// Az első szám: 55
// A második szám: 66
```

Az első átalakítási meghatározás a `printf()` második paraméterének felel meg, ami ebben az esetben az 55. A következő átalakítási meghatározás a 66-nak felel meg. A százalékjelet követő `d` betű miatt a függvény az adatot decimális egészként értelmezi. A meghatározásnak ez a része típusparaméterként ismeretes.

A printf() és a típusparaméterek

Egy típusparaméterrel már találkoztunk, ez volt a `d`, amely az adatot decimális formátumban jeleníti meg. A többi típusparamétert a 17.1. táblázatban láthatjuk.

17.1. táblázat Típusparaméterek

Paraméter	*Leírás*
`d`	A paramétert decimális számként jeleníti meg.
`b`	Egész számokat bináris számként jelenít meg.
`c`	Egy egész számot annak ASCII megfelelőjeként jelenít meg.
`f`	A paramétert lebegőpontos számként ábrázolja.
`o`	Egy egész számot oktális (8-as számrendszerű) számként jelenít meg.
`s`	A paramétert karakterlánc-állandónak tekinti.
`x`	Egy egész számot kisbetűs hexadecimális (16-os számrendszerű) számként jelenít meg.
`X`	Egy egész számot nagybetűs hexadecimális (16-os számrendszerű) számként jelenít meg

A 17.1. program a `printf()` függvénnyel egy számot a 17.1. táblázat típusparaméterei segítségével jelenít meg.

Vegyük észre, hogy a formázó karakterlánc nem egyszerűen csak átalakítási meghatározásokat tartalmaz, minden további benne szereplő szöveg kiírásra kerül.

17.1. program Néhány típusparaméter használatának bemutatása

```
1: <html>
2: <head>
3: <title>17.1. program Néhány típusparaméter
          használatának bemutatása</title>
4: </head>
5: <body>
6: <?php
7: $szam = 543;
8: printf("Decimális: %d<br>", $szam );
9: printf("Bináris: %b<br>", $szam );
```

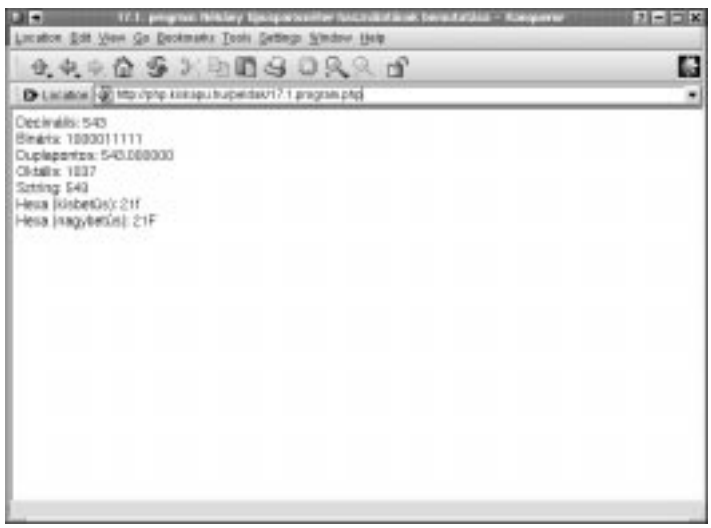
17.1. program (folytatás)

```
10: printf("Kétszeres pontosságú: %f<br>", $szam );
11: printf("Oktális: %o<br>", $szam );
12: printf("Karakterlánc: %s<br>", $szam );
13: printf("Hexa (kisbetűs): %x<br>", $szam );
14: printf("Hexa (nagybetűs): %X<br>", $szam );
15: ?>
16: </body>
17: </html>
```

A 17.1. program kimenetét a 17.1. ábrán láthatjuk. A `printf()` függvénnyel gyorsan tudunk adatokat egyik számrendszerből a másikba átalakítani és az eredményt megjeleníteni.

17.1. ábra

Néhány típusparaméter használatának bemutatása



Ha a HTML-ben hivatkoznunk kell egy színre, három 00 és FF közé eső, a vörös, zöld és kék színt képviselő hexadecimális számot kell megadnunk. A `printf()` függvényt használhatjuk arra, hogy a három 0 és 255 közé eső decimális számot azok hexadecimális megfelelőire alakítsuk át:

```
$piros = 204;
$zold = 204;
$kek = 204;
printf( "#%X%X%X", $piros, $zold, $kek );
// azt írja ki, hogy "#CCCCCC"
```

Bár a típusparaméterrel a decimális számokat hexadecimálissá alakíthatjuk, azt nem tudjuk meghatározni, hogy az egyes paraméterek kimenete hány karaktert foglaljon el. A HTML színkódjában minden hexadecimális számot két karakteresre kell kitölteni, ami problémát okoz, ha például az előző kódrészlet `$piros`, `$zold`, `$kek` változóit úgy módosítjuk, hogy 1-et tartalmazzanak. Kimenetül `"#111"`-et kapnánk. A bevezető nullák használatát egy kitöltő paraméter segítségével biztosíthatjuk.

A kitöltő paraméter

Beállíthatjuk, hogy a kimenet bizonyos karakterekkel megfelelő szélességűre töltődjön ki. A kitöltő paraméter közvetlenül az átalakító paramétert kezdő százalékjelet követi. Ha a kimenetet bevezető nullákkal szeretnénk kitölteni, a kitöltő paraméterben a `0` karaktert az a szám követi, ahány karakteresre szeretnénk a kimenetet bővíteni. Ha a kimenet hossza ennél a számnál kisebb lenne, a különbség nullákkal kerül kitöltésre, ha nagyobb, a kitöltő paraméternek nincs hatása:

```
printf( "%04d", 36 )
// a kimenet "0036" lesz
printf( "%04d", 12345 )
// a kimenet "12345" lesz
```

Ha a kimenetet bevezető szóközökkel szeretnénk kitölteni, a kitöltő paraméternek tartalmaznia kell egy szóköz karaktert, amelyet a kimenet elvárt karakterszáma követ:

```
printf( "% 4d", 36 )
// azt írja ki, hogy "  36"
```

Bár a HTML dokumentumokban egymás után szereplő több szóközt a böngészők nem jelenítik meg, a megjelenítendő szöveg elé és után helyezett <PRE> címkével mégis biztosíthatjuk a szóközök és sortörések megjelenítését.

```
<pre>
<?php
print "A    szóközök    láthatóvá válnak."
?>
</pre>
```

Ha teljes dokumentumot szeretnénk szövegként megformázni, a header() függvényt használhatjuk a dokumentumtípus (Content-Type) fejlécének módosításához.

```
header("Content-type: text/plain");
```

Ne feledjük, hogy programunk nem küldhet semmilyen kimenetet a böngészőnek a header() függvényhívást megelőzően, hogy a megfelelő módon működjön, mivel a kimenet megkezdésekor a válasz fejrészét már elküldtük a böngészőnek.

Ha szóközön vagy nullán kívül más karaktert szeretnénk a kitöltéshez használni, a kitöltő paraméteren belül a kitöltő karakter elé írjunk egyszeres idézőjelet:

```
printf( "%'x4d", 36 )
// azt írja ki "xx36"
```

Most már rendelkezésünkre állnak azok az eszközök, melyekkel a korábbi HTML kódot kiegészíthetjük. Eddig ugyan már át tudtuk alakítani a három számot, de nem tudtuk bevezető nullákkal kitölteni azokat:

```
$piros = 1;
$zold = 1;
$kek = 1;
printf( "#%02X%02X%02X", $piros, $zold, $kek );
// azt írja ki, hogy "#010101"
```

Most már minden változó hexadecimális számként fog megjelenni. Ha a kimenet két karakternél rövidebb, a hiányt bevezető nullák pótolják.

A mezőszélesség meghatározása

Meghatározhatjuk a kimenet által elfoglalt mező szélességét is. A mezőszélesség paramétere egy olyan egész szám, amely a százalékjel után következik az átalakító paraméterben (feltéve, hogy nem használunk helykitöltő karaktereket). A következő kódrészlet kimenete egy négyelemű felsorolás, amelynek mezőszélessége 20 karakternyi. A szóközök láthatóvá tételéhez a kimenetet egy `PRE` elembe ágyazzuk.

```
print "<pre>";
printf("%20s\n", "Könyvek");
printf("%20s\n", "CDk");
printf("%20s\n", "Játékok");
printf("%20s\n", "Magazinok");
print "</pre>";
```

A 17.2. ábrán a fenti kódrészlet eredményét láthatjuk.

17.2. ábra

Igazítás a mezőszélességhez

A kimenet alapértelmezés szerint a mezőn belül jobbra igazodik. A balra igazítást a mezőszélesség paramétere elé tett mínuszjellel (`-`) érhetjük el.

```
printf ("%-20s|<- eddig tart a balra zárás\n", "Balra zárt");
```

Fontos megjegyezni, hogy ha lebegőpontos számot írunk ki, az igazítás csak a kimenetben levő szám (ponttól jobbra levő) tizedesrészére vonatkozik. Más szóval jobbra igazításkor a lebegőpontos számnak a tizedesponttól balra eső része a mező bal oldali, túlsó végén marad.

A pontosság meghatározása

Ha az adatot lebegőpontos formában szeretnénk megjeleníteni, meghatározhatjuk a kerekítés pontosságát. Ez főleg a pénznem átváltásakor szokott fontos lenni. A pontosságot meghatározó paraméter egy pontból és egy számból áll, és közvetlenül a típusparaméter elé kell írni. A megadott szám határozza meg, hány tizedesre szeretnénk kerekíteni. Ez a paraméter csak akkor érvényes, ha a kimenet `f` típusparaméterű:

```
printf ("%.2f\n", 5.333333);
// azt írja ki, hogy "5.33"
```

A C nyelv printf() függvényében akkor is lehetőségünk van a pontosság megadására, ha decimális kimenet esetén kérünk kitöltést. A PHP 4-ben a pontosság paraméterének nincs semmilyen hatása a decimális kimenetre. Egész számok nullákkal való bevezetéséhez a kitöltő paramétert kell használnunk.

Átalakító paraméterek (Ismétlés)

A 17.2. táblázatban az átalakító paramétereket foglaljuk össze. Megjegyzendő, hogy a két (kitöltő és mezőszélesség) paraméter együttes használata bonyolult, így azt tanácsoljuk, egyszerre csak az egyiket használjuk.

17.2. táblázat Az átalakítás lépései

Név	*Leírás*	*Példa*
Kitöltő paraméter	A kimenet által elfoglalandó karakterszámot és a kitöltésre használt karaktert adja meg.	`' 4'`
Mezőszélesség paraméter	A formázandó karakterlánc méretét adja meg.	`'20'`
Pontosság paraméter	Meghatározza, hogy a kétszeres pontosságú számokat hány tizedesre kell kerekíteni.	`'.4'`
Típusparaméter	Meghatározza az eredmény adattípusát.	`'d'`

A 17.2. program a `printf()` használatával egy termékárlistát hoz létre.

17.2. program Termékárlista formázása a printf() függvénnyel

```
 1: <html>
 2: <head>
 3: <title>17.2. program Termékárlista formázása
           a printf() függvénnyel</title>
 4: </head>
 5: <body>
 6: <?php
 7: $termekek = Array("Zöld karosszék"=>"222.4",
 8:                   "Gyertyatartó" => "4",
 9:                   "Kávézóasztal" => "80.6"
10:                  );
11: print "<pre>";
12: printf("%-20s%23s\n", "Név", "Ár");
13: printf("%'-43s\n", "");
14: foreach ( $termekek as $kulcs=>$ertek )
15:       printf( "%-20s%20.2f\n", $kulcs, $ertek );
16: print "</pre>";
17: ?>
18: </body>
19: </html>
```

Először a termékek nevét és árát tartalmazó tömböt adjuk meg. Egy `PRE` elemmel jelezzük a böngészőnek, hogy a szóközöket és a soremeléseket értelmezze.
A `printf()` első meghívása a következő formázó karakterláncot adja meg:

```
"%-20s%23s\n"
```

Az első átalakító meghatározás (`"%-20s"`) a mezőszélességet balra zárt 20 karakteresre állítja. Ebben a mezőben egy karakterlánc típusú paramétert helyezünk el.
A második meghatározás (`"%23s"`) egy jobbra zárt mezőszélességet ad meg.
A `printf()` függvénynek ez a meghívása hozza létre leendő táblázatunk fejlécét.

A `printf()` második meghívásával egy 43 karakter hosszú, mínuszjelekből (-) álló vonalat húzunk. Ezt úgy érjük el, hogy egy üres karakterláncot kitöltő paraméterrel jelenítünk meg.

A `printf()` legutolsó meghívása annak a `foreach` utasításnak a része, amely végiglépked a termékek tömbjén. Két átalakító meghatározást használunk.

Az első (`"%-20s"`) a termék nevét írja ki egy 20 karakteres mezőbe, balra igazítva. A másik (`"%20.2f"`) a mezőszélesség paraméterrel azt biztosítja, hogy az eredmény egy 20 karakteres mezőben jobbra igazítva jelenjen meg, a pontossági paraméterrel pedig azt, hogy a megjelenített kétszeres pontosságú érték két tizedesre legyen kerekítve.

A 17.3. ábrán a 17.2. program eredményét láthatjuk.

17.3. ábra

Termékárlista formázása a printf() függvénnyel

Formázott karakterlánc tárolása

A `printf()` az adatokat a böngészőben jeleníti meg, ami azzal jár, hogy eredménye programunk számára nem elérhető. Használhatjuk azonban a `sprintf()` függvényt is, amelynek használata megegyezik a `printf()` függvényével, viszont az eredményét egy karakterláncban adja vissza, amelyet aztán későbbi használatra egy változóba helyezhetünk. A következő kódrészlet a `sprintf()` függvény segítségével egy lebegőpontos értéket két tizedesre kerekít és az eredményt a `$kerek` változóban tárolja:

```
$kerek = sprintf("%.2f", 23.34454);
print "Még $kerek forintot költhetsz";
```

A `sprintf()` függvény egy lehetséges felhasználása, hogy vele a megformázott adatot fájlban lehet tárolni. Ennek az a módja, hogy először meghívjuk a `sprintf()` függvényt, a visszaadott értékét egy változóban tároljuk, majd az `fputs()` függvénnyel fájlba írjuk.

17

Részletesebben a karakterláncokról

Nem minden esetben tudunk mindent azokról az adatokról, amelyekkel dolgozunk. A karakterláncok többféle forrásból is érkezhetnek, beviheti a felhasználó, érkezhetnek adatbázisokból, fájlokból és weboldalakról. A PHP 4 több függvénye segítségünkre lehet abban, hogy ezekről a külső forrásból érkező adatokról többet tudjunk meg.

Szövegek indexelése

A karakterláncokkal kapcsolatban gyakran használjuk az indexelés szót, de a tömböknél még gyakrabban találkozhatunk vele. Valójában a karakterláncok és a tömbök nem állnak olyan messze egymástól, mint azt gondolnánk. Egy karakterláncot elképzelhetünk egy karakterekből álló tömbként is. Ennek megfelelően a karakterláncok egyes karaktereihez ugyanúgy férhetünk hozzá, mint egy tömb elemeihez:

```
$proba = "gazfickó";
print $proba[0]; // azt írja ki, hogy "g"
print $proba[2]; // azt írja ki, hogy "z"
```

Ne felejtsük el, hogy amikor egy karakterláncon belüli karakter indexéről vagy helyéről beszélünk, akkor a karaktereket – ugyanúgy, mint a tömb elemeit – `0`-tól kezdve számozzuk. A tömbökkel kapcsolatos félreértések elkerülése érdekében a PHP 4 bevezette a `$proba{0}` formát is erre a célra.

Szöveg hosszának megállapítása az strlen() függvénnyel

Az `strlen()` függvény segítségével megállapíthatjuk egy karakterlánc hosszát. A függvény bemenete egy karakterlánc, visszatérési értéke pedig egy egész szám, amely a függvénynek átadott változó karaktereinek száma. Ez a függvény például kimondottan jól jöhet a felhasználó által megadott bemenet hosszának ellenőrzésére. A következő kódrészlettel ellenőrizhetjük, hogy a tagnyilvántartó azonosító négykarakteres-e:

```
if ( strlen( $tagazonosito ) == 4)
     print "Köszönöm!";
else
     print "Az azonosítónak négykarakteresnek kell lennie<P>";
```

Ha a `$tagazonosito` globális változó értéke négykarakteres, megköszönjük a felhasználónak, hogy rendelkezésünkre bocsátotta, más esetben hibaüzenetet jelenítünk meg.

Szövegrész megkeresése az strstr() függvénnyel

Ezzel a függvénnyel azt állapíthatjuk meg, hogy egy karakterlánc megtalálható-e beágyazva egy másik karakterláncban. Az `strstr()` függvény két paramétert kap bemenetéül: a keresendő szöveget és a forrásláncot, azaz azt a karakterláncot, amelyben keresnie kell. Ha a keresett karakterlánc nem található a szövegben, a visszatérési érték `false` (hamis) lesz, ellenkező esetben a függvény a forrásláncból a keresett karakterlánccal kezdődő részt adja vissza. A következő példában megkülönböztetetten kezeljük azokat a tagazonosítókat, amelyekben megtalálható az `AB` karakterlánc:

```
$tagazonosito = "pAB7";
if ( strstr( $tagazonosito, "AB" ) )
     print "Köszönöm. Ne feledje, hogy tagsága hamarosan
            lejár!";
else
     print "Köszönöm!";
```

Mivel a `$tagazonosito` változó tartalmazza az AB karakterláncot, az `strstr()` az `AB7` karakterláncot adja vissza. Ez `true` (igaz) eredménynek számít, így egy különleges üzenetet ír ki. Mi történik akkor, ha a felhasználó a `"pab7"` karakterláncot adja meg? Az `strstr()` megkülönbözteti a kis- és nagybetűket, így nem találja meg az `AB` karakterláncot. Az `if` utasítás feltétele nem válik igazzá, így a szokványos üzenet kerül a böngészőhöz. Ha vagy `AB`-t, vagy `ab`-t szeretnénk kerestetni a szövegben, használjuk az `stristr()` függvényt, melynek működése azonos az `strstr()`-évél, viszont nem különbözteti meg a kis- és nagybetűket.

Részlánc elhelyezkedésének meghatározása az strpos() függvénnyel

Az `strpos()` függvény segítségével kideríthetjük, hogy egy szöveg megtalálható-e egy másik szöveg részeként, és ha igen, hol. Bemenetét két paraméter képezi, a forráslánc, amelyben keresünk, és a karakterlánc, amelyet keresünk. Ezek mellett létezik még egy harmadik, nem kötelező paraméter, mégpedig azt az indexet megadó egész szám, amelytől kezdve keresni szeretnénk. Ha a keresett karakterlánc nem található, a függvény a `false` (hamis) értéket adja vissza, ellenkező esetben azt az egész számot, mely indextől a keresett szöveg kezdődik. A következő programrészletben az `strpos()` függvénnyel győződünk meg arról, hogy egy karakterlánc az `mz` karakterekkel kezdődik-e:

```
$tagazonosito = "mz00xyz";
if ( strpos($tagazonosito, "mz") === 0 )
     print "Üdv, mz.";
```

Vegyük szemügyre azt a trükköt, amellyel a kívánt eredményt kaptuk. Az `strpos()` megtalálja az `mz` karaktersort a lánc elején. Ez azt jelenti, hogy `0` értéket ad vissza, ami viszont a kifejezés kiértékelése során hamis értéket eredményezne. Hogy ezt elkerüljük, a PHP 4 új azonosság műveleti jelét (`===`) alkalmazzuk, amely akkor ad vissza `true` (igaz) értéket, ha bal és jobb oldali operandusa egyenlő értékű és azonos típusú is egyben.

Szövegrészlet kinyerése a substr() függvénnyel

A `substr()` függvény egy kezdőindextől kezdve meghatározott hosszúságú karakterláncot ad vissza. Bemenetét két paraméter képezi, az egyik a forráslánc, a másik a kezdőindex. A függvény az összes karaktert visszaadja a kezdőindextől a forráslánc végéig. Harmadik, nem kötelező paramétere egy egész szám, amely a visszaadandó szöveg hosszát jelenti. Ha ezt is megadjuk, a függvény csak a meghatározott mennyiségű karaktert adja vissza a kezdőindextől számítva.

```
$proba = "gazfickó";
print substr($proba,3); // azt írja ki "fickó"
print substr($proba,3,4); // azt írja ki "fick"
```

Ha kezdőindexként negatív számot adunk meg, a függvény nem a karakterlánc elejétől számolja a karaktereket, hanem a végétől. A következő kódrészlet meghatározott üzenetet ír ki azoknak, akiknek e-mail címe `.hu`-val végződik.

```
$cim = "felhasznalo@szolgaltato.hu"
if ( substr( $cim, -3 ) == ".hu" )
     print "Ne felejtse el magyar vásárlóinknak járó
            speciális ajánlatainkat!";
else
     print "Üdvözöljük üzletünkben!";
```

Karakterlánc elemekre bontása az strtok() függvénnyel

Az `strtok()` függvénnyel karakterláncok szintaktikai elemzését végezhetjük. A függvény legelső meghívásakor két paramétert vár, az elemzendő karakterláncot és egy határolójelet, amely alapján a karakterláncot felbontja. A határolójel tetszőleges számú karakterből állhat. A függvény első meghívásakor átmenetileg a memóriába helyezi a teljes forrásláncot, így a további meghívások alkalmával már csak a határolójelet kell megadnunk. Az `strtok()` minden meghívásakor a következő megtalált elemet adja vissza, a karakterlánc végére érkezést a `false` (hamis) érték visszaadásával jelzi. Ezt a függvényt legtöbbször cikluson belül használjuk. A 17.3. program egy URL-t bont elemeire: először leválasztja a gazdagépet és az elérési útvonalat a karakterláncról, majd a változó–érték párokat bontja szét. A 17.3. program eredményét a 17.3. ábrán láthatjuk.

17.3. program Karakterlánc elemekre bontása az strtok() függvénnyel

```
 1: <html>
 2: <head>
 3: <title>17.3. program Karakterlánc elemekre bontása
 4:        az strtok() függvénnyel</title>
 5: </head>
 6: <body>
 7: <?php
 8: $proba = "http://www.deja.com/qs.xp?
 9: OP=dnquery.xp&ST=MS&DBS=2&QRY=developer+php";
10: $hatarolo = "?&";
11: $szo = strtok( $proba, $hatarolo );
12: while ( is_string( $szo ) )
13:    {
14:    if ( $szo )
15:        print "$szo<br>";
16:    $szo = strtok( $hatarolo );
17:    }
18: ?>
19: </body>
20: </html>
```

Az `strtok()` függvény önmagában nem sok mindent támogat, így csak különféle trükkökkel bírhatjuk igazán hasznos munkára. Először a `$hatarolo` változóban tároljuk a határolójelet. Meghívjuk az `strtok()` függvényt, átadjuk neki az elemzendő URL-t és a `$hatarolo` karakterláncot. Az első eredményt a `$szo` változóba helyezzük. A `while` ciklus feltételében azt ellenőrizzük, hogy a `$szo` karakterlánc-e. Ha nem az, tudhatjuk, hogy elértük az URL végét és a feladat befejeződött.

Azért ellenőrizzük a visszaadott érték típusát, mert ha egy karakterlánc egy sorában két határolójel van, az `strtok()` az első határolójel elérésekor üres karakterláncot ad vissza. Így, ha a `$szo` egy üres karakterlánc, az alábbi példában látható szokványosabb próbálkozás sikertelen lesz, még akkor is, ha a függvény még nem érte el a forráslánc végét, mivel a `while` feltétele hamissá válik:

```
while ( $szo )
      {
      $szo = strtok( $hatarolo );
      }
```

Ha meggyőződtünk róla, hogy a `$szo` változó karakterláncot tartalmaz, elkezdhetünk dolgozni vele. Ha a `$szo` nem üres karakterlánc, megjelenítjük a böngészőben. Aztán újra meg kell hívnunk a `strtok()` függvényt, hogy a `$szo` változót újabb karakterlánccal töltse fel a következő kiértékeléshez. Vegyük észre, hogy a második alkalommal nem adtuk át a függvénynek a forrásláncot. Ha ezt mégis megtennénk, újra a forráslánc első szavát találná meg, így végtelen ciklusba kerülnénk.

A karakterláncok kezelése

A PHP 4 a karakterlánc paraméterek kisebb-nagyobb átalakításához számos függvényt biztosít.

Szöveg tisztogatása a trim() típusú függvényekkel

Ha egy felhasználótól vagy fájlból kapunk információt, sohasem lehetünk biztosak benne, hogy az adat előtt vagy után nincs egy vagy több fölösleges elválasztó karakter. A `trim()` függvény ezeket az elválasztó karaktereket (soremelés, tabulátorjel, szóköz stb.) hagyja el a karakterlánc elejéről és végéről. Bemenete a megtisztítandó szöveg, kimenete pedig a megtisztított.

```
$szoveg = "\t\t\teléggé levegős ez a szöveg         ";
$szoveg = trim( $szoveg );
print $szoveg
// azt írja ki, hogy "eléggé levegős ez a szöveg"
```

Persze lehet, hogy ez a túlbuzgó függvény nem a legmegfelelőbb számunkra. Elképzelhető, hogy a szöveg elején levő elválasztó karaktereket meg szeretnénk tartani és csak a szöveg végéről akarjuk eltávolítani azokat. Pontosan erre a feladatra találták ki a `chop()` függvényt. Vigyázzunk, ha Perl ismeretekkel is rendelkezünk, mivel ott a `chop()` függvény egy kicsit más jelentéssel bír. A probléma áthidalására a PHP fejlesztői ugyanezen szolgáltatás eléréséhez megadták az `rtim()` függvénynevet is.

```
$szoveg = "\t\t\teléggé levegős ez a szöveg         ";
$szoveg = chop( $szoveg );
print $szoveg
// azt írja ki, hogy "        eléggé levegős ez a szöveg"
```

Lehetőségünk van az `ltrim()` függvény használatára is, amely az elválasztó karaktereket csak a karakterlánc elejéről távolítja el. Ahogy az előzőeknél is, bemenete az átalakítandó szöveg, kimenete pedig az elválasztó karakterektől csupán a bal oldalán mentes karakterlánc:

```
$szoveg = "\t\t\teléggé levegős ez a szöveg         ";
$szoveg = ltrim( $szoveg );
print $szoveg
// azt írja ki, hogy "eléggé levegős ez a szöveg         "
```

Karakterlánc részének lecserélése a substr_replace() függvénnyel

A `substr_replace()` hasonlóan működik, mint a `substr()`, a különbség abban rejlik, hogy itt lehetőség nyílik a kivonatolt karakterlánc-részlet lecserélésére is. A függvény három paramétert vár: az átalakítandó karakterláncot, a csereszöveget és a kezdőindexet. Ezek mellett létezik egy nem kötelező, negyedik hosszúság paraméter is. A `substr_replace()` függvény megtalálja a kezdőindex és a hosszúság paraméter által meghatározott részláncot, lecseréli azt a csereszövegre, és a teljes átalakított karakterláncot adja vissza.

A következő kódrészletben egy felhasználó tagazonosítójának megújításához le kell cserélnünk annak harmadik és negyedik karakterét:

```
<?
$tagazonosito = "mz99xyz";
$tagazonosito = substr_replace( $tagazonosito, "00", 2, 2 );
print "Az új tagnyilvántartó azonosító: $tagazonosito<p>";
// azt írja ki, hogy "Az új tagnyilvántartó azonosító:
                      mz00xyz"
?>
```

Az összes részlánc lecserélése az str_replace() függvénnyel

Az `str_replace()` függvény a keresett karakterlánc-rész összes előfordulását lecseréli egy másik karakterláncra. Bemenetének három paramétere a lecserélendő karakterlánc, a csereszöveg és a forrásszöveg. A függvény kimenete az átalakított karakterlánc. A következő példában egy karakterláncban az 1999 összes előfordulását 2000-re cseréljük:

```
$karakterlanc = "Ezt az oldal 1999-ben szerzői jog által
                 védett";
$karakterlanc .= "Felsőoktatási tájékoztató 1999";
print str_replace("1999","2000",$karakterlanc);
```

Kis- és nagybetűk közti váltás

A PHP több függvénnyel is segítségünkre van a kis- és nagybetűk cseréjében. Amikor felhasználók által beírt adattal dolgozunk, fontos lehet mindent csupa nagybetűsre vagy csupa kisbetűsre alakítani, hogy aztán könnyebben összehasonlíthatóak legyenek. Az `strtoupper()` függvény segítségével egy karakterláncot csupa nagybetűsre alakíthatunk. A függvény egyetlen bemenete az átalakítandó szöveg, visszatérési értéke pedig a csupa nagybetűs karakterlánc:

```
$tagazonosito = "mz00xyz";
$tagazonosito = strtoupper( $tagazonosito );
print "$tagazonosito<P>"; // azt írja ki, hogy "MZ00XYZ"
```

Karakterláncunk csupa kisbetűssé való alakításához használjuk az `strtolower()` függvényt. Ennek egyetlen bemenete az átalakítandó szöveg és a csupa kisbetűs karakterláncot adja vissza:

```
$honlap_url = "WWW.KISKAPU.HU";
$honlap_url = strtolower( $honlap_url );
if ( ! ( strpos ( $honlap_url, "http://" ) === 0 ) )
   $honlap_url = "http://$honlap_url";
print $honlap_url; // azt írja ki, hogy
                      "http://www.kiskapu.hu"
```

A PHP-nek van egy nagyszerű, „tüneti kezelést” biztosító függvénye, az `ucwords()`. Ez a függvény egy karakterlánc minden szavának első betűjét teszi nagybetűssé. A következő programrészletben a felhasználó által beírt karakterláncban a szavak első betűjét nagybetűre cseréljük:

```
$teljes_nev = "vitéz tinódi lantos sebestyén";
$teljes_nev = ucwords( $teljes_nev );
print $teljes_nev; // azt írja ki, hogy "Vitéz Tinódi
                                        Lantos Sebestyén"
```

A függvény kizárólag a szavak első betűjét cseréli le, így ha a felhasználónak nehézségei vannak a SHIFT billentyűvel és azt írta be, hogy `"ViTÉz tINóDi laNtos sEBeSTyéN"`, ez a függvény nem sokban lesz segítségére, hiszen a tüneti kezelés eredménye `"ViTÉz TINóDi LaNtos SEBeSTyéN"` lesz. Ezen úgy segíthetünk, hogy az `ucwords()` meghívása előtt az `strtolower()` függvénnyel először csupa kisbetűssé alakítjuk a karakterláncot:

```
$teljes_nev = "ViTÉz tINóDi laNtos sEBeSTyéN";
$teljes_nev = ucwords( strtolower($teljes_nev) );
print $teljes_nev; // azt írja ki, hogy "Vitéz Tinódi
                                        Lantos Sebestyén"
```

Fel kell, hogy hívjuk a kedves olvasó figyelmét arra, hogy a magyar szövegekkel az ékezetes betűk miatt alapbeállításban problémáink akadhatnak. A nemzeti beállítások testreszabására használható `setlocale()` függvényt kell alkalmaznunk, hogy a kívánt eredményt elérjük.

Karakterláncok tömbbé alakítása az explode() függvénnyel

A mókásan „robbantó”-nak nevezett függvény bizonyos mértékben hasonló az `strtok()` függvényhez. Ez a függvény azonban egy karakterláncot tömbbé bont fel, amit aztán tárolhatunk, rendezhetünk, vagy azt tehetünk vele, amit csak szeretnénk. Bemenetét két paraméter alkotja, ez egyik egy határolójel, ami alapján fel szeretnénk bontani a forrásláncot, a másik maga a forráslánc. A határolójel több karakterből is állhat, ezek együtt fogják alkotni a határolójelet. (Ez eltér az `strtok()` függvény működésétől, ahol a megadott karakterlánc minden karaktere egy-egy önálló határolójel lesz.) A következő kódrészlet egy dátumot bont fel részeire és az eredményt egy tömbben tárolja:

```
$kezdet = "2000.12.01";
$datum_tomb = explode(".", $kezdet);
// $datum_tomb[0] == "2000"
// $datum_tomb[1] == "12"
// $datum_tomb[2] == "00"
```

Összefoglalás

A PHP külvilággal való kapcsolattartása és adattárolása leginkább karakterláncokon keresztül valósul meg. Az órán a programjainkban levő karakterláncok kezelésével ismerkedtünk meg.

A `printf()` és az `sprintf()` függvények segítségével megtanultuk formázni a karakterláncokat. Ezt a két függvényt olyan karakterláncok létrehozására használjuk, amelyek egyrészt átalakítják, másrészt el is rendezik az adatokat. Tanultunk olyan függvényekről, amelyek információt árulnak el a karakterláncokról. Meg tudjuk állapítani egy karakterlánc hosszúságát az `strlen()`, egy részlánc jelenlétét az `strpos()` függvénnyel, vagy kiszakíthatunk részláncokat az `strtok()` segítségével.

Végül azokról a függvényekről tanultunk, amelyek karakterláncokat alakítanak át. Most már el tudjuk tüntetni az elválasztó karaktereket a `trim()`, `ltrim()` vagy a `chop()` függvénnyel, válthatunk a kis- és nagybetűk között az `strtoupper()`, az `strtolower()` és az `ucwords()` függvényekkel, valamint egy karakterlánc összes előfordulását lecseréltethetjük az `str_replace()` segítségével.

Ezek után nehéz elhinni, de még mindig nem végeztünk a karakterláncokkal. A PHP ugyanis támogatja a szabályos kifejezéseket, amelyek a karakterlánc-kezelés még hatékonyabb formáját biztosítják. A szabályos kifejezések alkotják a következő óra anyagát.

Kérdések és válaszok

Vannak még egyéb hasznos karakterlánc-függvények?
Vannak. A PHP több mint 60 ilyen függvénnyel rendelkezik! Ezekről a PHP 4 kézikönyvében olvashatunk, amelynek megfelelő része a `http://php.net/manual/ref.strings.php` címen található.

A `printf()` működését bemutató példában a formázást úgy jelenítettük meg, hogy a kimenetet `<PRE>` elemek közé tettük. Ez a legjobb módja a formázott szöveg megjelenítésének a böngészőben?
A `<PRE>` címkék akkor szükségesek, ha a sima szöveg (`plain text`) formázást HTML-es környezetben is meg szeretnénk tartani. Ha viszont a teljes dokumentumot így szeretnénk megjeleníteni, a legokosabb, ha közöljük a böngészővel, hogy sima szövegként formázza meg azt. Ez a `header()` függvénnyel érhető el:

```
Header("Content-type: text/plain");
```

Műhely

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

Kvíz

1. Milyen átalakító paramétert használnánk a `printf()` függvényben egy egész szám lebegőpontos számként való megformázására?

2. Hogyan egészítsük ki az 1. kérdésben átalakított számot nullákkal úgy, hogy a tizedespont előtti (attól balra eső) rész 4 karakter hosszúságú legyen?

3. Hogyan kerekítenénk az előző kérdés lebegőpontos számát két tizedesjegyre?

4. Milyen függvényeket használnánk egy szöveg hosszának kiderítéséhez?

5. Milyen függvényeket használnánk egy részlánc más karakterláncon belüli kezdetének meghatározására?

6. Milyen függvényeket használnánk arra, hogy egy szövegből kivonjuk annak egy darabját?

7. Hogyan távolíthatjuk el az elválasztó karaktereket egy karakterlánc elejéről?

8. Hogyan alakítanánk át egy karakterláncot csupa nagybetűsre?

9. Hogyan bontanánk fel egy határolójelekkel elválasztott karakterláncot tömbökre?

Feladatok

1. Hozzunk létre egy olyan vélemény-visszajelző űrlapot, amely a felhasználó nevét és e-mail címét kéri be. Használjuk a kis- és nagybetűket átalakító függvényeket a nevek első betűjének nagybetűsítésére, majd jelenítsük meg az eredményt a böngészőben. Ellenőrizzük, hogy a cím tartalmaz-e @-jelet, ha nem, figyelmeztessük a felhasználót.

2. Hozzunk létre egy lebegőpontos és egész számokból álló tömböt. Léptessünk végig a tömbön és kerekítsük az összes lebegőpontos számot két tizedesjegyre. Igazítsuk jobbra a kimenetet egy 20 karakter szélességű mezőben.