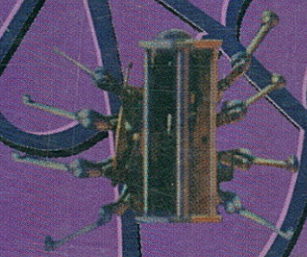
A WEB PROGRAMOZÁSA II.

KRIS JAMSA
SULEIMAN „SAM" LALANI
STEVE WEAKLEY



VISUAL BASIC

TCP/IP

JAVASCRIPT

HTML

HTTP

VRML

VBSCRIPT

JAVA

CGI

J++

C++

A WEB PROGRAMOZÁSA

KRIS JAMSA
SULEIMAN „SAM" LALANI
STEVE WEAKLEY

KOSSUTH KIADÓ 1997

A FORDÍTÁS AZ ALÁBBI KIADÁS ALAPJÁN KÉSZÜLT:
PH. D. KRIS JAMSA, SULEIMAN „SAM" LALANI, STEVE WEAKLEY: WEB-PROGRAMMING
JAMSA PRESS, 2975 S. RAINBOW BLVD., SUITE I.
LAS VEGAS, NV 89 101, USA

FORDÍTOTTA
INOTAI LÁSZLÓ

LEKTORÁLTA
KISS ISTVÁN

ISBN 963 09 3930 4

10. FEJEZET

BEVEZETŐ A VRML NYELVBE

Már most előre látható, hogy a jövőben egyre több Web-helyen megnyílik a harmadik dimenzió is, és a felhasználók a 3-D világba belépve az objektumokat nemcsak sík felületekként, hanem térben is megnézhetik. A Web-helyek tervezői a térbeli objektumok létrehozásának egyik eszközeként a VRML (Virtual Reality Modeling Language, magyarul: látszólagos valóságot modellező nyelv) nyelvet használják. A VRML általános, szöveg alapú nyelv, amelyet azért terveztek, hogy 3-D objektumokat lehessen vele leírni. A HTML-hez hasonlóan a VRML is különböző platformokon használható, beleértve a UNIX, Mac és Windows környezetet. Miután már létrehoztunk kétdimenziós HTML dokumentumokat tartalmazó Web-helyeket, a továbbfejlesztés egyik útja VRML objektumok elkészítése lehet. A VRML nyelvben számos olyan elemet találunk, amelyet a HTML is tartalmaz. Miután megtanultuk, hogy miként készíthetünk objektumokat a VRML nyelv segítségével, olyan Web-oldalakat tervezhetünk, amelyek 3-D képeket jelenítenek meg, és amelyeket a felhasználók több látószögből is megnézhetnek.

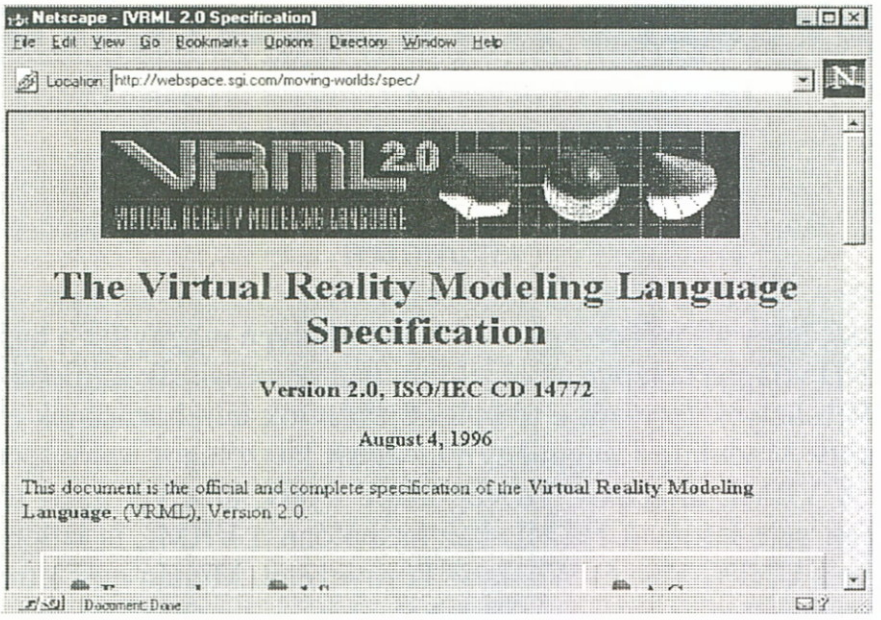
A tervezők elsősorban arra használják a VRML nyelvet, hogy 3-D grafikus képeket hozzanak létre vele. Azonban a VRML segítségével háromdimenziós szövegek is készíthetők. A felhasználók a 3-D képeket és szöveget speciális VRML böngészővel vagy segédprogrammal kiegészített Web-böngészővel nézhetik meg és forgathatják körbe. Ebben a fejezetben megismerkedünk a VRML nyelv használatával, és megtudjuk, hogyan építhetünk fel egy professzionális 3-D világot. A fejezet végére érve érteni fogjuk az alábbiakat:

- A VRML (Virtual Reality Modeling Language) nyelv segítségével 3-D képeket és 3-D szöveget készíthetünk a Web-oldalakon.
- Egy VRML dokumentum ugyanúgy ASCII fájl, mint egy HTML dokumentum.
- VRML dokumentumon belül csomópontokat adunk meg, amelyek meghatározzák azokat az alakzatokat és attribútumait, amelyekből a VRML böngésző a képet visszaállítja.
- A VRML elsősorban nem programozási nyelv (csak kevés programszerkezetet használ), inkább matematikai modellező nyelv, amelynek segítségével alakzatokat és objektumokat állíthatunk elő.

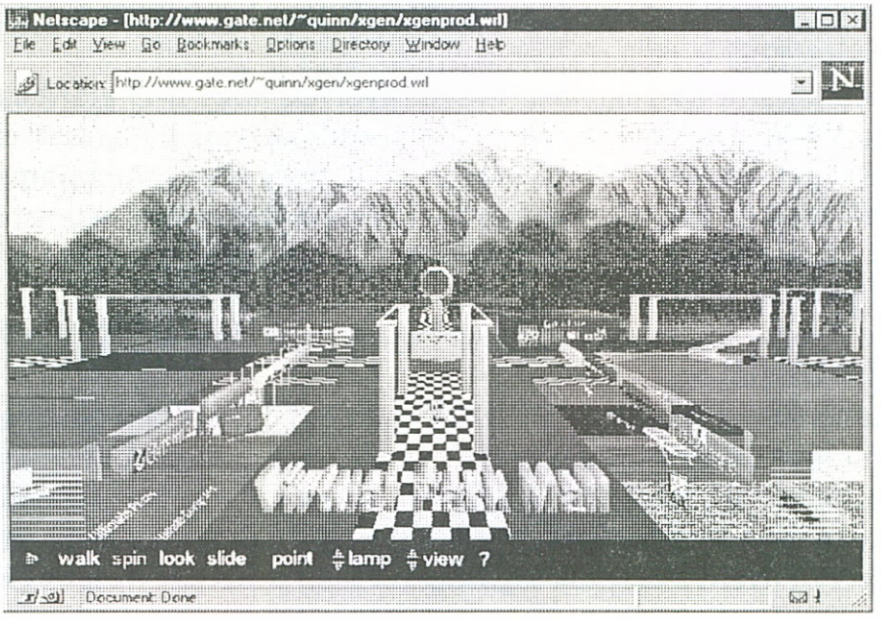
RÖVIDEN A VRML NYELVRŐL

A VRML (Virtual Reality Modeling Language) olyan modellező nyelv, amely lehetővé teszi 3-D képek és 3-D szöveg megjelenítését. A VRML 2.0 specifikációját a *http://vrml.sgi.com/moving-worlds/spec* címen olvashatjuk el (lásd a 10.1. ábrát).

Önálló VRML böngésző vagy egy megfelelő segédprogrammal kiegészített normál Web-böngésző segítségével jeleníthetünk meg egy 3-D jelenetsort, más néven „világot" (angol kifejezéssel: *scene graph*, *world*). Valamely VRML jelenet egyszerűen egy VRML böngészővel megjelenített 3-D kép adatait jelenti. A jelenet, például egy ház tervezője, lehetővé teheti, hogy „bejárjuk" a házat és mindegyik helyiségéről háromdimenziós képet kapjunk. A 10.2. ábra egy VRML 3-D világot szemléltet.



10.1. ábra. Információk a VRML 2.0 specifikációról



10.2. ábra. Egy VRML 3-D világ látványa

A HTML ÉS A VRML KÖZÖTTI KAPCSOLAT

Mint említettük, sok a közös vonás a HTML és a VRML nyelvben. A VRML a HTML-hez hasonlóan használja a hiperhivatkozásokat. (Egyik VRML helyről egy másikra történő átlépésre az angol nyelvű szakirodalom a teleporting kifejezést használja.) VRML böngésző segítségével letölthetünk és megjeleníthetünk VRML dokumentumot. Mind a HTML, mind a VRML dokumentumok alkalmazástól független, szöveg alapú (ASCII) dokumentumok. A böngészők a HTML és a VRML dokumentumokat is HTTP protokoll segítségével töltik le a Webről. A sok hasonlóság mellett azonban lényeges különbségek is vannak a HTML és a VRML dokumentumok között.

A VRML mind 3-D képek, mind 3-D szöveg megjelenítését támogatja. A VRML dokumentumokhoz használt MIME típus az *x-world* típus, a MIME altípus pedig az *x-vrml* altípus. (Ha VRML segédprogramot használunk ahhoz, hogy a Netscape-böngészővel jelenítsünk meg VRML dokumentumot, akkor a böngészőt úgy kell beállítanunk, hogy kezelni tudja ezt a MIME típust/altípust.) A HTML dokumentumok nem ezt a MIME típust/altípust használják, kiterjesztésük tipikusan *htm* vagy *html*. A VRML dokumentumok kiterjesztése *wrl* (a world szóból rövidítve). A VRML doku-

mentumok készítéséhez használt szintaxis és struktúra alapvetően különbözik a HTML dokumentumokban használatos címkéktől (kódjelektől) és struktúráktól. A VRML objektumorientált nyelv, amely *csomópontokat (nodes)* használ objektumokként. A csomópontok jelentik azokat az elsődleges objektumokat, amelyeket egy VRML jelenetben létrehozunk és manipulálunk. A HTML nem képes az ilyen csomópontok értelmezésére.

VRML DOKUMENTUM KÉSZÍTÉSE

Egy VRML dokumentum közönséges ASCII dokumentum, amelyet bármilyen egyszerű ASCII szövegszerkesztővel, például a DOS EDIT segédprogramjával vagy a Windows Jegyzettömbjével előállíthatunk. Ha valamilyen bonyolultabb, formázásokat is végző szövegszerkesztővel írjuk meg a VRML dokumentumunkat, akkor azt ASCII formátumú TXT fájlként kell mentenünk. Ha nem így tennénk, akkor a szövegszerkesztő rejtett formázókaraktereket helyezhetne el a dokumentumunkban, amelyeket a VRML böngésző nem ért meg, és emiatt hibát jelezne.

A VRML DOKUMENTUMOK TÁROLÁSI HELYE

Valamely VRML dokumentumhoz úgy férhetünk hozzá, és úgy jeleníthetjük meg azt, hogy a böngészőnkkel rákapcsolódunk arra a Web-kiszolgálóra, amelyik a kívánt VRML dokumentumot tárolja. A VRML dokumentumokat a HTML dokumentumokhoz hasonlóan a kiszolgálót futtató gépek tárolják különböző könyvtárakban. A kiszolgáló VRML és HTML dokumentumokat is tárolhat. Egy VRML dokumentum olyan *csomópontokra* (VRML objektumokra) hivatkozhat, amelyek más kiszolgálókon találhatók. Ha egy böngészőnek valamely VRML jelenet megjelenítéséhez ilyen csomópontokra van szüksége, akkor a kiszolgáló automatikusan letölti a böngészőre az ezeket a csomópontokat tartalmazó fájlokat.

Megjegyzés: A kiszolgálók egy speciális, konfigurációs fájlt tartalmaznak, amely meghatározza azokat a MIME-típusokat, amelyeket a kiszolgálók a böngészőknek elküldhetnek. Ha egy kiszolgáló konfigurációs fájljában létezik az x-world/x-vrml MIME típus (content type), akkor a kiszolgáló úgy van beállítva, hogy képes VRML fájlok küldésére.

A VRML RÖVID TÖRTÉNETE

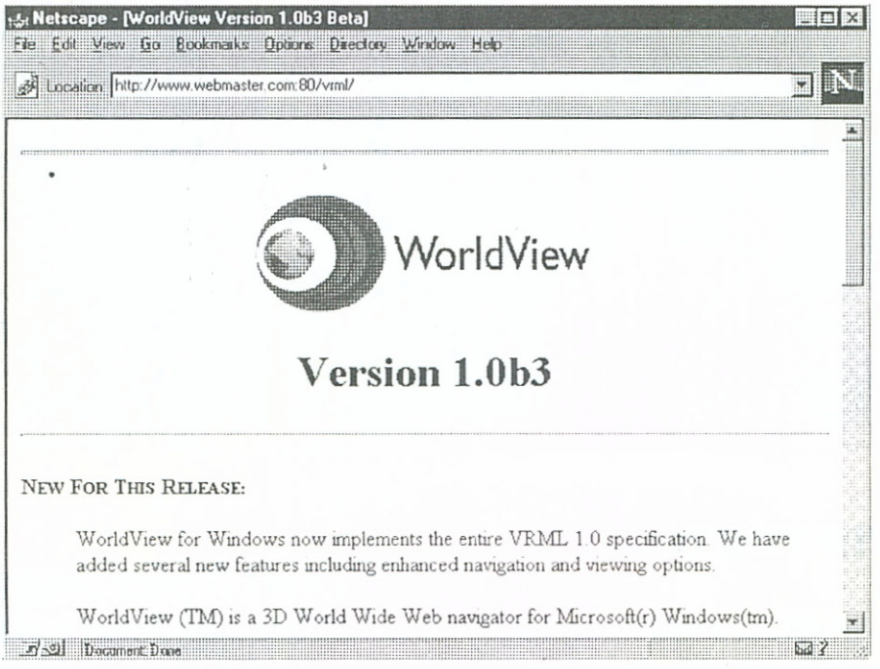
A VRML nyelvet Mark Pesce és Anthony (Tony) Parisi alkotta meg Gavin Bell és Brian Behlendorf segítségével. Mark és Tony 1993. decemberében találkoztak. Együtt dolgozták ki a Web 3-D felületének terveit. Tudták, hogy a VRML nyelvnek el kell térnie a HTML-től, de nem akartak teljesen új nyelvet kifejleszteni. Miután tanulmányoztak néhány meglévő eljárást, körülbelül egy hónapos mérlegelést követően a Silicon Graphics, Inc. (SGI) cég Open Inventor ASCII fájlformátumára esett a választásuk. Elsősorban azért választották az SGI Open Inventorát, mert ennek a fájlformátuma támogatta a 3-D jelenetsorok teljes egészében sokszögekkel történő leírását, és a sokszögeket objektumokként állította vissza. Az SGI Open Inventora továbbá a fényhatásokat, az anyagszerű megjelenítést és más, valósághű hatásokat is támogatta. Így az SGI Open Inventor fájlformátumának egy részhalmaza, kiegészítve a hálózati bővítésekkel, alkotja a VRML alapját.

1994. február közepére elkészítettek egy programot, amely egy banán 3-D képét hívta le és állította vissza a képernyőjükön. Ennek a programnak a Labyrinth nevet adták. Egy hónappal később Mark Pesce meghívást kapott a genfi székhelyű CERN-től a WWW első nemzetközi kon-

ferenciájára. Miután bemutatta a VRML nyelv fejlesztésében elért eredményeit, találkozott Brian Behlendorffal, és közösen elkészítettek egy Web-helyet, ahol leírták a VRML céljait. Ettől a pillanattól kezdve az egész Web-közösség lázba jött. Ezerszámra kapták az érdeklődőktől az elektronikus leveleket, és ez az időpont tekinthető a VRML-korszak kezdetének. A VRML hivatalos, nyilvános bemutatására 1995. április 3-án került sor, és még ugyanezen év májusában közzétették a VRML 1.0-s verzióját. A VRML 1.0-s specifikációját Gavin Bell, Anthoy Parisi és Mark Pesce dolgozta ki. Azok a jelenetsorok, amelyeket ebben a fejezetben a VRML segítségével elkészítünk, a VRML 1.0-s specifikációját követik. A VRML legújabb, 2.0-s verziója animált, virtuális világokat is támogat. Mielőtt azonban megvizsgálnánk a VRML 2.0-s verzióját, tisztában kell lennünk az 1.0-s verzió főbb elgondolásaival. A Weben jelenleg található VRML dokumentumok többsége az 1.0-s verzió szerint készült. Ismerve azonban azokat az előnyöket, amelyeket a 2.0-s verzió a tervezők és programozók számára nyújt, rövidesen minden bizonnyal igen sok VRML 2.0-s verziójú dokumentummal találkozhatunk a Weben.

SAJÁT VRML BÖNGÉSZŐ/SEGÍTŐ BESZERZÉSE

Az 1. fejezetben láttuk, hogyan tölthetjük le a WorldView nevű VRML böngészőt (ami az Anthony Parisi által alapított Intervista cég szoftverterméke). A WorldView böngészőt önálló VRML böngészőként és VRML segédprogramként is használhatjuk, amely a normál Web-böngészőt képessé teszi VRML dokumentumok megjelenítésére. Web-böngészőnk segítségével a WorldView VRML böngészőt a *http://www.webmaster.com/vrml/* helyről tölthetjük le (lásd a 10.3. ábrát).



10.3. ábra. A WorldView VRML böngésző letöltése

Mivel a Web-helyek többsége HTML dokumentumokat tartalmaz, és csak néhány helyen találunk 3-D VRML dokumentumokat, a Web-böngészőnket úgy kell beállítanunk, hogy az a WorldView szoftvert csak segédprogramként használja. A következő bekezdésben megismerkedünk azokkal a lépésekkel, amelyeket meg kell tennünk ahhoz, hogy a Netscape Navigator a WorldView-t segédprogramként használja.

A NETSCAPE NAVIGATOR ÉS A MICROSOFT INTERNET EXPLORER KONFIGURÁLÁSA VRML DOKUMENTUM MEGJELENÍTÉSÉHEZ

A Netscape Navigator és a Microsoft Internet Explorer jelenlegi verziói önmagukban nem képesek VRML dokumentumok megjelenítésére.* Ehhez olyan segédprogramra van szükségük, mint például a WorldView. A böngészőkben megadhatjuk azt a segédprogramot, amelyet futtatni akarunk, amikor egy VRML dokumentumra bukkanunk. Mindegyik böngészőben más lépéseket kell tennünk ahhoz, hogy a böngésző egy segédprogramot futtasson. Olvassuk el a segédprogrammal együtt letöltött tájékoztatót. Emellett még meglátogathatjuk a Netscape és a Microsoft Web-helyeit, ahol javaslatokat kaphatunk a VRML segédprogramokkal kapcsolatban.

A VRML MÉRTÉKEGYSÉGEI

Mielőtt megvizsgálnánk a VRML szintaxisát és szerkezetét, meg kell ismerkednünk azokkal a mértékegységekkel, amelyeket a VRML a 3-D térben a hosszúságok és szögek méréséhez használ. Továbbá ismernünk kell azt a koordinátarendszert, amelyben a VRML a 3-D képeket megjeleníti. A VRML-ben a hosszúság szabványos mértékegysége a méter. A VRML a szögeket radiánban méri. A VRML koordinátarendszere a „jobbkéz-szabály"-ra épül, amire még a matematikaórákról emlékezhetünk. A koordinátarendszer könnyebb megértéséhez tekintsük a monitorunk *x*, *y* és *z* tengelyét. Az *x* tengely a monitor bal alsó, felénk eső sarkából indul, és jobbra mutat a pozitív értékek irányába. Az *y* tengely ugyanebből a sarokból indul, és felfelé halad a pozitív értékek irányába. Végül a *z* tengely ugyancsak ebből a pontból indul, és a monitortól felénk halad a pozitív értékek irányába.

A VRML MINT JELENETSOR-FEJLESZTŐ NYELV

A VRML jelenetsor-fejlesztő nyelv, amelynek szintaxisa és szerkezete is van. A *jelenetsor (scene graph)* azoknak a csomópontoknak a rendezett halmaza, amelyek összerakják a 3-D világot. Minden olyan dokumentumnak, amely jelenetsort rak össze és VRML forráskódot tartalmaz, az alábbi címsorral kell kezdődnie:

```
#VRML V1.0 ascii
```

Ezen a soron belül a VRML minden olyan karaktert figyelmen kívül hagy, ami az *ascii* után áll. A sor végén vagy az ASCII újsor karakterének, vagy a kocsi vissza karakternek kell állnia. Ez az utasítás tájékoztatja a böngészőt arról, hogy a fájl VRML dokumentumot tartalmaz.

A VRML *csomópontoknak* nevezett objektumokat használ a 3-D jelenetek létrehozásához. A VRML nem követi az objektumorientált nyelvek tipikus filozófiáját, amely egymástól függetlenül kezeli az egyes objektumokat, ezzel szemben hierarchikus rendben helyezi el a csomópontokat, amelyekben ezek helyét a VRML dokumentumon belüli helyük határozza meg. Más szavakkal ez azt jelenti, hogy amikor létrehozunk egy VRML csomópontot, akkor a csomópont létrehozásának a dokumentumon belüli helye fogja meghatározni azt a sorrendet, amelyben a böngésző a csomópontot meg fogja jeleníteni. A csomópontok ilyen elrendezése könnyebbé teszi a böngésző számára az objektumok megjelenítési módjának vezérlését, amikor egy VRML jelenetsort állít vissza.

* A könyv fordítása idején a Netscape Navigator 3.01 verziója és a Gold 3.0-s verziója már képes VRML dokumentumok megjelenítésére (a használt segédprogram neve Live3D). – A fordító megjegyzése.

A CSOMÓPONT ÉS A JELENETSOR KAPCSOLATA

Mint említettük, a VRML jelenetsornak nevezett hierarchikus szerkezetbe rendezi a csomópontokat, és ez a szerkezet határozza meg azt a sorrendet, ami szerint a böngésző megjeleníti a csomópontokat. Mint majd látni fogjuk, egy csomópont *örökölheti* egy előzőleg definiált csomópont jellemzőit. Más szavakkal ez azt jelenti, hogy a böngésző által korábban megjelenített csomópontok befolyásolhatják a később visszaállításra kerülő csomópontokat.

A VRML *Separator (elválasztó)* csomópontokat használ annak megakadályozására, hogy egy csomópont örökölje más csomópontok tulajdonságait. Az elválasztó csomópont lehetővé teszi, hogy a jelenet tervezői „tiszta lappal" indulva definiálhassanak új csomópontokat (új alakzatokat a VRML-világon belül). Az *elválasztó* csomópontok tehermentesítik a programokat az alól, hogy figyelemmel kísérjék az előző csomópontokhoz adott összes tulajdonságot.

VRML CSOMÓPONTOK

Amint említettük, a VRML csomópont egy objektumot jelent. Fordítva is igaz: egy VRML jelenetsorban minden objektum egy csomópontot jelent, amely adott típusú objektumot ír le. Egy csomópont lehet például *Sphere (gömb), Cone (kúp) Cylinder (henger)* vagy *Cube (hasáb).* Az ilyen típusú csomópontok úgynevezett *egyszerű geometriai csomópontok (Simple Geometry Nodes).*

A VRML más típusú csomópontokat is használ, mint például *Material (anyag)* és *Texture2 (mintázat).* Ezek *tulajdonság (Property)* csomópontok. A VRML jelenetsorok tulajdonság-csomópontok segítségével alkalmaznak valamilyen *anyagfelületet (Material csomópontot)* az őket követő csomópontokra. Ugyancsak tulajdonság-csomópont segítségével határozza meg egy jelenet azt a mintázatot (Texture2 csomópontot), amelyet a böngészőnek a következő alakzatok megjelenítéséhez használnia kell.

A VRML jelenetsorok paraméterek megadásával jelenítenek meg azonos típusú, de különböző méretű, színű stb. csomópontokat. Egyik *Sphere* csomópontnak például más lehet a sugara, mint egy másik *Sphere* csomópontnak. A *Sphere* csomópont sugarát egy mező adja meg. Egy VRML csomópontnak 0 vagy több mezője lehet, és nevet is rendelhetünk a csomóponthoz.

VRML CSOMÓPONT MEZŐINEK ÉRTÉKEI

Ha mezőértéket deklarálunk egy csomóponthoz, akkor meg kell adni a mező nevét, ami után az értéknek kell következnie. A mező nevét és az értékét úgynevezett kitöltő karakterrel (a szóközvagy a TAB-billentyű lenyomásával) kell egymástól elválasztani. Ha ugyanazon mezőnévhez több érték rendelhető, akkor az értékeket vesszővel kell egymástól elválasztani, és az értékek listáját szögletes zárójelekkel kell körülvenni.

VRML CSOMÓPONT NEVE

Rugalmasabb programfejlesztést tesz lehetővé, hogy a VRML-ben nevet rendelhetünk egy csomóponthoz. Azzal, hogy nevet adunk egy csomópontnak, a VRML forrásfájlban vagy más programból (például egy scriptből) manipulálhatjuk azt. A csomópontoknak nem kötelező nevet adnunk, de ha megtesszük (és minden bizonnyal így lesz), akkor a csomópontnak csak egy neve lehet. A neveknek ugyanakkor nem kell egyedieknek lenniük – ugyanazt a nevet több csomópontnak is adhatjuk. (*Megjegyzés*: Ezt a VRML 1.0 specifikáció tartalmazza!) A csomópont neve nem kezdődhet számjeggyel és nem tartalmazhat szóközöket, vezérlőkaraktereket, egyszeres vagy kétszeres idézőjeleket, balra dőlő ferde vonalú (backslash) karaktereket, kapcsos zárójeleket, plusz jelet és pontot.

A CSOMÓPONT GYERMEKEI

Egy VRML jelenetsoron belül mindegyik objektumot csomópontként kell elképzelnünk. Röviden azt is mondhatjuk, hogy mindegyik alakzat egy csomópont. A jelenetsoron belül a csomópontoknak lehetnek gyermekeik. A gyermekcsomópont szintaxisa a szülő szintaxisát követi. Az a szülőcsomópont, amelyiknek egy vagy több gyermekcsomópontja van, *csoportosító csomópontot (group node)* képez. A VRML az objektumok öröklési képességeit és hierarchikus szerkezetét azáltal valósítja meg, hogy a szülőcsomópontoknak lehetővé teszi, hogy gyermekcsomópontokat tartalmazzanak. A VRML dokumentum egy szülőcsomóponton belül színeket, mintázatokat, transzformációkat stb. definiálhat. Gyermekcsomópont létrehozásakor a csomópont örökölni fogja a szülőcsomópont ezen beállításait.

A CSOMÓPONTOK SZINTAXISA

Egy VRML dokumentumon belül bárhol deklarálhatunk csomópontot. Csomópontot az alábbi utasítással deklarálhatunk:

```
[DEF obejktum_név] objektum_típus { [mezõk] [gyermek] }
```

A deklarációban a szögletes zárójelek ([]) közötti tételek megadása nem kötelező. Az alábbi utasítás például egy *Cube (hasáb)* csomópontot deklarál:

```
Cube { }
```

A VRML egy *Cube* csomópontot definíciószerűen a koordinátatengelyekkel párhuzamosan helyez el úgy, hogy a hasáb közepét a (0, 0, 0) koordinátapontba helyezi. A *Cube* csomópont egyéb alapbeállítás szerinti értékei: az élek mérete mindegyik dimenzióban 2 egység a -1 és a +1 koordinátapont között, mindegyik dimenzió örökli az aktuális koordináta-transzformációt, a mintázatot és az anyagot. A fejezet későbbi részében még visszatérünk a koordinátákra, mintázatokra és anyagokra.

Említettük, hogy amikor deklarálunk egy csomópontot, akkor megadhatunk egy nevet, és mezőértékeket definiálhatunk. Az alábbi utasítás például egy *Gömb* nevű *Sphere* (gömb) csomópontot hoz létre, amelynek 5 egység a sugara:

```
DEF Gömb Sphere {
  radius 5
}
```

A VRML 1.0-s specifikációja 36 csomópontot definiál. A VRML jelenetsorokat ezekből a csomópontokból építhetjük fel. „A VRML 1.0-s verzió 36 csomópontja" címmel részletesen foglalkozunk e csomópontok mindegyikével. A VRML 2.0-s verziójának specifikációja további csomópontokat definiál:

- geometriai vezérlőelemek és színtulajdonságok alakzatokhoz;
- geometriai érzékelők, amelyek jobban támogatják a vidd és dobd műveleteket;
- csomópontok csoportosítása, amelyek támogatják az „ütközés észlelése" objektumot;
- fényhatások jellemzői, mint például köd;
- jelenet-információk a böngészők számára;
- hangforrások támogatása.

Emellett a VRML 2.0 specifikáció lecseréli néhány csomópont nevét, így például az *AsciiText* csomópont neve az új specifikációban *Text* névre változott.

VRML MEGJEGYZÉSEK

A HTML-hez hasonlóan a VRML nyelvben is a „létra" (#, hashmark) karakterrel indul egy megjegyzés. A VRML böngésző a # karakter után álló karaktereket figyelmen kívül hagyja. Ha a megjegyzés egynél hosszabb sort vesz igénybe, akkor mindegyik sort a # karakterrel kell kezdeni.

Megjegyzés: Előfordulhat, hogy egy kiszolgáló nem őrzi meg a VRML dokumentumban lévő megjegyzéseket és kitöltő karaktereket. Lehetséges, hogy a kiszolgáló kihúzza a megjegyzéseket és a felesleges kitöltő karaktereket a VRML fájlból, mielőtt továbbítaná a dokumentumot egy böngészőnek.

A VRML ÁLTALÁNOS SZINTAXISA

A szóközök, a tabulátor karakterek, az ASCII újsor és kocsi vissza karakterek úgynevezett *kitöltő karakterek*. Egy VRML dokumentumban ezen kitöltő karakterek közül egyet vagy többet is használhatunk a szintaktikai egységek egymástól való elválasztásához. Egy csomópont mezőnevét például szóköz vagy tabulátor karakterrel választhatjuk el a mező értékétől.

A kötelezően megadandó címsort követően a VRML fájl egyetlenegy VRML csomópontot tartalmazhat. Ez a csomópont azonban lehet csoportosító csomópont (és általában az is), amely akárhány további csomópontot tartalmazhat.

A VRML 1.0-S VERZIÓ 36 CSOMÓPONTJA

A VRML a csomópontok különböző osztályát definiálja. A csomópontok többsége a következő négy osztály valamelyikébe sorolható be: egyszerű geometriai csomópontok *(simple geometry nodes),* tulajdonság-csomópontok *(property nodes),* csoportosító csomópontok *(group nodes)* és Web-csomópontok *(Web nodes).*

Az *egyszerű geometriai csomópontok* a jelenet geometriáját definiálják. Lényegében ezek az egyedüli olyan csomópontok, amelyek rajzoláshoz használhatók. A *tulajdonság-csomópontok* azt a módot befolyásolják, ahogyan a böngésző megjeleníti az egyszerű geometriai csomópontokat. A tulajdonságok anyagszerű vagy mintázattal borított megjelenést, fényhatásokat stb. kölcsönöznek az alakzatoknak. A *tulajdonság-csomópontok* önmagukban nem tudnak megjelenni (nincsen „testük"). A *csoportosító csomópontok* más csomópontokat fognak össze, amelyeket a dokumentum így egyetlen objektumként tud kezelni. A *Web-csomópontok* a kiszolgálót segítik abban, hogy megtaláljon egy gyermekcsomópontot valahol a Weben.

Mint olvastunk róla, egy VRML csomópont 0 vagy több mezőt tartalmazhat. Minden egyes csomóponttípus minden egyes mezőjéhez típust, nevet és alapértelmezett értéket definiál. Ha egy csomópont deklarációja nem definiál egy mezőértéket, akkor a VRML a mező alapértelmezett értékét használja. Egy csomóponton belül a mezők tetszés szerinti sorrendben helyezhetők el. Az alábbi két deklaráció például egyenértékű egymással:

```
Cube {
   height 4
   depth 6
   width 2
}
```

```
Cube {
   depth 6
   width 2
   height 4
}
```

(A mezők a hasáb szélességét, magasságát és mélységét határozzák meg.)

Egy VRML dokumentumon belül tulajdonság-csomópontokba csoportosíthatjuk az egyszerű geometriai csomópontok olyan tulajdonságait (mezőit), amelyek az objektum megvilágítását, transzformálását stb. befolyásolják.

Mint látni fogjuk, a *csoportosító csomópontokkal* csoportokba foghatunk össze csomópontokat (és használhatjuk az öröklésből adódó előnyöket), elválaszthatunk egymástól csomópontokat (amivel időlegesen felfüggeszthetjük az öröklést), megadhatjuk, hogy mely gyermekcsomópontok kapják meg az állapotinformációkat, részleges állapotinformációkat használhatunk egy csomóponton (minden tulajdonság öröklése a *Transform csomópont-tulajdonság* kivételével) vagy a Weben lévő bármely más objektumhoz köthetjük a csomópontot. Látni fogjuk, hogy a *csoportosító csomópontok* az egyedüli olyan csomópontok, amelyeknek gyermekcsomópontjai lehetnek.

A VRML MEZŐI

Amikor közelebbről szemügyre vesszük a csomópontokat, látni fogjuk, hogy az egyes mezőnevek és alapértelmezett értékük mellett jobbra megjegyzések állnak, amelyek a mező típusát adják meg. A VRML kétféle mezőtípust használ: olyanokat, amelyek *egyetlen értéket*, és olyanokat, amelyek *több értéket* tartalmaznak. Az egyetlen értéket tartalmazó mező számot, vektort vagy képet foglalhat magába. Az ilyen mezőtípusok neve mindig az SF betűkettőssel kezdődik (single field). A fejezetben bemutatásra kerülő csomópontokat megvizsgálva láthatjuk, hogy a mezőnevek az alapértelmezett értékek bal oldalán állnak.

A több értékű mezők több, egymástól vesszővel elválasztott értéket tartalmaznak. A vesszőkkel elválasztott értéklistát mindig szögletes zárójelek ([]) közé kell tenni, ha a mező történetesen nemcsak egyetlen értéket tartalmaz. Ha a mező egyetlen értéket sem tartalmaz, akkor csak a szögletes zárójeleket kell beírni. Többértékű mező utolsó értéke után kitehető vessző, de nem kötelező. Az ilyen mezőtípusok neve mindig az MF betűkettőssel kezdődik (multiple field). A VRML mezőkről (mind az egy-, mind a többértékűekről) a fejezet későbbi részében, „A VRML mezők áttekintése" címmel részletesen olvashatunk.

A VRML CSOMÓPONTOK KÖZELEBBRŐL

Mivel egy jelenetsorban objektumokként csak a VRML csomópontok használhatók, alaposan meg kell ismernünk ezeket, hogy tudjuk, milyen esetben melyiket célszerű használnunk. A következőkben részletesen bemutatjuk mind a 36 csomópontot.

Megjegyzés: A 36 csomópontot bemutató részben valamennyi mezőérték az alapbeállítás szerinti érték, amit maga a VRML nyelv rendel a mezőkhöz.

AZ ASCIITEXT CSOMÓPONT

A VRML dokumentumok az *AsciiText* csomópontot használják arra, hogy 3-D szöveget jelenítsenek meg egy VRML jelenetben. A böngésző az ASCII karakterkészlet karaktereit képes megjelení-

teni. Az alábbi utasítások az *AsciiText* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
AsciiText {
   justification        LEFT                    # SFEnum
   spacing              1                       # SFFloat
   string               ""                      # MFString
   width                0                       # MFFloat
}
```

Megjegyzés: A VRML 2.0 az AsciiText csomópont nevét a Text névre cseréli le. Ha azt tapasztaljuk, hogy a böngészőnk nem képes az AsciiText csomópontok visszaállítására, akkor az AsciiText nevet próbáljuk Text névre cserélni.

A böngésző által megjelenítendő karaktereket a *string* (karakterlánc) mezőhöz kell rendelnünk. Az elsőként megjelenített karakterlánc a (0, 0, 0) koordinátákkal megadott ponton kezdődik. Az ezt követő karakterláncok az *y* tengely mentén haladva jelennek meg a következő képlet szerinti helyen: *-1*(méret*térköz)*. A *méret (size)* értéke a *FontSize* csomópontban adható meg. A *térköz (spacing)* mező a szomszédos karakterláncok közötti függőleges térközt adja meg. A *justification (igazítás)* mező három értéket használhat: LEFT (balra, ez az alapbeállítás), CENTER (középre) és RIGHT (jobbra). A LEFT érték beállításakor a karakterlánc bal széle az *x = 0* pozíción kezdődik. A CENTER beállításakor a karakterlánc közepe kezdődik az *x = 0* pozíción. A RIGHT beállítást használva a karakterlánc jobb széle kezdődik az *x = 0* pozíción. A böngésző a karaktereket jobbról balra, felülről lefelé jeleníti meg a *FontStyle* csomópontban megadott betűtípussal. A *width (szélesség)* a javasolt szélességkorlátot adja meg az egyes mezőkre. Az alapbeállítás szerinti 0 érték a karakterlánc természetes szélességét állítja be. A böngésző a szöveget az aktuális halmozott (kumulatív) transzformációt használva fogja transzformálni, és az aktuális anyagot és mintázatot alkalmazza rá.

Megjegyzés: A transzformációról bővebben a fejezet későbbi részében, „A Transform csomópont" címmel lesz szó. Az anyagokról „A Material csomópont", a mintázatokról pedig „A Texture2 csomópont" címmel olvashatunk részletesen.

A CONE CSOMÓPONT

A VRML dokumentumok a *Cone* csomópont segítségével jelenítenek meg 3-D kúpot egy VRML jelenetben. Az alábbi utasítások a *Cone* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
Cone {
   bottomRadius         1                       # SFFloat
   height               2                       # SFFloat
   parts                ALL                     # SFBitMask
}
```

A *parts (részek)* mező a kúp részeit adja meg, és a következő értékeket veheti fel: SIDES, BOTTOM, ALL (oldalak, alja, mind). A SIDES érték a kúp oldalát (palástját) jelenti. A BOTTOM érték a kúp alsó, kör alakú felületét, míg az ALL a kúp teljes felületét jelenti. A *bottomRadius* az alsó kör sugarát, a *height* mező a kúp magasságát adja meg.

A *Cone* csomópont segítségével egyszerű kúp rajzolható meg, amelynek a tengelye az *y* tengellyel párhuzamos. Alapbeállítás szerint a böngésző a kúp közepét a (0, 0, 0) koordinátapontban

jeleníti meg, és a méreteit a -1 és +1 pontokkal adja meg mindhárom irányban (a középponttól számítva). Minden kúp két részből áll: az oldalából (palást) és az aljából. A böngésző a kúpot az aktuális halmozott (kumulatív) transzformációjával transzformálja (helyezi át), és az aktuális anyagot és mintázatot alkalmazza rá.

Ha a VRML dokumentum mintázatot használ a kúphoz, akkor a böngésző másként alkalmazza a mintázatot a kúp palástján, és másként az alján. A mintázat a kúp palástját az óramutató járásával ellenkező irányban haladva borítja be (a csúcsponttól lefelé) úgy, hogy a mintázat a kúp hátánál kezdődik. A mintázatot a kúp hátánál ferde varrat fogja össze, amely metszi az *x*-*z* síkot. Ha felfelé fordítjuk a kúp alját, láthatjuk, hogy a böngésző egy kör alakú darabot vág ki a mintázat négyzetéből, és ezt alkalmazza a kúp alsó felületére. Ha a kúp csúcsát a -*z* tengely irányába forgatjuk, akkor a mintázat teteje jelenik meg felül.

A COORDINATE3 CSOMÓPONT

VRML dokumentumban ennek a csomópontnak a segítségével definiálhatjuk azoknak a pontoknak a 3-D koordinátarendszerbeli koordinátáit, amelyeket egy jelenet ezt követően *IndexedFaceSet, IndexedLineSet* vagy *PointSet* csomóponthoz használhat. Az alábbi utasítások a *Coordinate3* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
Coordinate3 {
  point 0 0 0                                    # MFVec3f
}
```

A point mező egy számhármassal adja meg egy 3-D térbeli pont koordinátáit.

Megjegyzés: Az **IndexedFaceSet, IndexedLineSet** *és a* **PointSet** *csomópontokról a fejezet későbbi részében olvashatunk.*

A CUBE CSOMÓPONT

VRML dokumentumokban a *Cube (hasáb)* csomópont segítségével jeleníthetünk meg 3-D hasábokat. Az alábbi utasítások a *Cube* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
Cube {
  depth       2                                  # SFFloat
  height      2                                  # SFFloat
  width       2                                  # SFFloat
}
```

A *width, height* és *depth* (szélesség, magasság, mélység) a hasáb éleinek méreteit adják meg. A csomópont angol neve némileg félrevezető, ugyanis a cube valójában négyzethasábot (kockát) jelent, míg ezzel a csomóponttal olyan hasábok is megvalósíthatók, amelyeknek nem egyenlő a *szélességük, magasságuk és mélységük.*

Alapbeállítás szerint a böngésző a hasáb középpontját a (0, 0, 0) koordinátapontba helyezi, és a hasáb mindegyik élének 2 egység a hossza a -1 és +1 koordinátapontok között. A böngésző a hasábot a jelenet aktuális kumulatív transzformációját használva transzformálja (mozgatja), és az aktuális anyagot és mintázatot alkalmazza.

Cube csomópont definiálásakor a hasáb mindegyik oldalához más és más mintázatot rendelhetünk. A böngésző a teljes mintázatot ráhelyezi az oldalakra. A böngésző a mintázatot az elülső,

hátsó, bal és jobb oldali oldalra normál, álló helyzetben, a felső és alsó oldalra pedig a következőképpen helyezi el: ha a hasáb felső oldalát magunk felé fordítjuk (a *kamera* nézőpontja felé), akkor a mintázatot normál, álló helyzetben látjuk. Hasonlóképpen, ha a hasáb felső oldalát a -*z* tengely irányába fordítjuk (a nézőpontunktól távolítva), akkor az alsó oldalon ugyancsak normál, álló helyzetben jelenik meg a mintázat.

Megjegyzés: A VRML kamerákról „Az OrtographicCamera csomópont" és „A PerspectiveCamera csomópont" címmel, a mintázatokról pedig „A Texture2 csomópont" címmel olvashatunk a fejezet későbbi részében.

A CYLINDER CSOMÓPONT

VRML dokumentumokban a *Cylinder (henger)* csomópont segítségével jeleníthetünk meg 3-D hengereket. Az alábbi utasítások a *Cylinder* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
Cylinder {
  height      2                                    # SFFloat
  parts       ALL                                  # SFBitMask
  radius      1                                    # SFFloat
}
```

A *radius (sugár)* és a *height (magasság)* a henger méreteit adják meg. A *parts (részek)* mező a henger részfelületeit írja le, és a következő értékeket veheti fel: SIDES, BOTTOM, TOP, ALL. A SIDES a henger palástját, a TOP a henger felső, kör alakú oldalát, a BOTTOM az alsó, kör alakú oldalát, míg az ALL a henger valamennyi felületét jelenti.

A *Cylinder* csomópont egy egyszerű, zárt hengert hoz létre az *y* tengely körül. Alapbeállítás szerint a böngésző a henger középpontját a (0, 0, 0) koordinátapontba helyezi. A henger alapbeállítás szerinti méreteit a -1 és +1 koordinátapontok határozzák meg mind a három dimenzióban. A henger felülete három részből tevődik össze: a palástból, a felső körlapból (ahol az $y = +1$) és az alsó körlapból (ahol az $y = -1$). A *sugár* és a *magasság* változtatásával megváltoztathatjuk a henger méretét.

A böngésző a hengert a jelenet aktuális kumulatív transzformációját használva transzformálja (mozgatja), és az aktuális anyagot és mintázatot alkalmazza. Ha a VRML dokumentum mintázatot rendel a hengerhez, akkor a böngésző másként jeleníti meg a mintázatot a paláston, és másként a felső és az alsó körlapon. A mintázat a henger palástját az óramutató járásával ellenkező irányban borítja be a csúcsponttól lefelé úgy, hogy a mintázat a kúp hátánál kezdődik. A mintázatot a henger hátánál függőleges varrat fogja össze, amely metszi az *x*-*z* síkot. Ha felfelé fordítjuk a henger alját, láthatjuk, hogy a böngésző egy kör alakú darabot vág ki a mintázat négyzetéből és ezt alkalmazza a henger alsó felületére. Ha a henger tetejét a -*z* tengely irányába (magunktól távolítva) forgatjuk, akkor a henger alján a mintázat alja kerül alulra. Ha a henger tetejét a +*z* tengely irányába (magunk felé) fordítjuk, akkor a mintázat a henger tetején is így jelenik meg.

A DIRECTIONALLIGHT CSOMÓPONT

VRML dokumentumokban a *DirectionalLight* (párhuzamos fénysugarak) csomópont segítségével olyan fényforrást hozhatunk létre, amely gyakorlatilag párhuzamos sugarakkal világít meg egy objektumot egy VRML jelenetsorban (a Nap sugaraihoz hasonlóan). A sugarak iránya egy 3-D vektorral adható meg. Az alábbi utasítások a *DirectionalLight* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
DirectionalLight {
  color       1 1 1                                    # SFColor
  direction   0 0 -1                                   # SFVer3f
  intensity   1                                        # SFFloat
  on          TRUE                                     # SFBool
}
```

A *color (szín)* mező a fény piros, zöld és kék színösszetevőit határozza meg. A *color* mező értékeinek mindegyike a 0.0 – 1.0 tartományban lehet. A *color* mező 1 0 0 értékei például a piros színt definiálják. Ehhez hasonlóan a 0 1 0 beállítás a zöld színt jelenti. A *direction (irány)* mező értéke egy 3-D vektort ír le, amely a fény irányát adja meg. A fénysugarak iránya párhuzamos lesz a *direction* mező értéke által meghatározott 3-D vektorral. Az *intensity* (fényesség) mező értéke a 0.0 – 1.0 tartományban lehet (lebegőpontos szám), ahol az 1.0 jelenti a legnagyobb fényességet. Az irányított fényt az *on* mező TRUE értéke kapcsolja be, a FALSE értéke pedig ki.

A *DirectionalLight* csomópont olyan fényforrást határoz meg, amely az aktuális fénystílustól függően egy jeleneten belül a soron következő *egyszerű geometriai csomópontokat* (Simple Geometry Nodes) is befolyásolhatja. A böngésző az aktuális transzformációt alkalmazza a *DirectionalLight* fényforrásra. Egy elválasztó csomóponton belüli *DirectionLight* csomópont nem befolyásolja azokat az objektumokat, amelyek az elválasztó csomópont hatókörén kívül vannak.

A FONTSTYLE CSOMÓPONT

VRML dokumentumokban a *FontStyle* csomópont segítségével határozhatjuk meg azt az aktuális betűtípust, amelyet a jelenet a soron következő *AsciiText* csomópontokra alkalmazni fog. A böngésző meghatározott betűtípusokat rendel az attribútumok különböző kombinációihoz (a VRML forrásprogramnak nincs közvetlen ráhatása a betűtípus attribútumaira). Az alábbi utasítások a *FontStyle* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
FontStyle {
  family      SERIF                                    # SFEnum
  size        10                                       # SFFloat
  style       NONE                                     # SFBitMask
}
```

A *family* mező a kívánt betűtípus-családot adja meg, mint például SERIF, SANS vagy TYPEWRITER. A *size* mező a betűk magasságát írja elő. A *style* mező a megadott betűcsalád stílusát határozza meg, amelynek a következő értékei lehetnek: NONE (normál, nincs formázás), BOLD (félkövér) és ITALIC (dőlt betű).

A GROUP CSOMÓPONT

VRML dokumentumokban a *Group* csomópont segítségével definiálhatjuk a bázis osztályt valamennyi csoportosító csomópont számára. Az alábbi utasítás egy egyszerű *Group* csomópontot deklarál (ennek a csomópontnak nincsenek mezői, így értelemszerűen alapbeállítás szerinti értékei sincsenek):

```
Group {
    # itt állnak a gyermekcsomópontok definíciói
  }
```

Egy *Group* csomópont egy tároló csomópontot definiál, amely gyermekcsomópontok rendezett listáját tárolja. A *Group* csomópont nem változtatja meg a jelenet *haladási állapotát* (azt a módot, ahogyan a böngésző feldolgozza a csomópontokat és amilyen sorrendben visszaállítja azokat). Mint említettük, a VRML az aktuális állapotot (színek, mintázatok, transzformációk stb.) továbbadja az összes egymást követő gyermekcsomópontnak. Egy *Group* csomópont nem menti és nem állítja vissza *(push and pop)* a haladási állapotot, ellentétben a *Separator* (elválasztó) csomóponttal, amely menti és visszaállítja az állapotot. (A Group csomópont az 1.0-nál későbbi verziókban már nem használható. – A fordító megjegyzése.)

Megjegyzés: A Separator csomópontról további információkat olvashatunk „A Separator csomópont" címmel, a fejezet későbbi részében.

AZ INDEXEDFACESET CSOMÓPONT

VRML dokumentumok az *IndexedFaceSet* csomópontot használják 3-D alakzat megjelenítésére. Egy ilyen testet síkidomokból építünk fel úgy, hogy az aktuális koordinátákon lévő csúcspontokat összekötve síkidomokat készítünk, és ezeket a test oldalaiként egymáshoz illesztjük. Az alábbi utasítások az *IndexedFaceSet* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
IndexedFaceSet {
  coordIndex        0                    # MFLong
  materialIndex     -1                   # MFLong
  normalIndex       -1                   # MFLong
  textureCoordIndex -1                   # MFLong
}
```

A *coordIndex* mező a síkidomok csúcspontjainak megfelelő koordinátahármasok indexértékeit adja meg (0-tól induló indexértékeket használva). Egy nem negatív indexérték egy *(x, y, z)* koordinátájú 3-D pontnak felel meg. A testet alkotó síkidomok mindegyikének definíciószerűen legkevesebb három vagy több élűnek kell lennie. Ezért síkidomonként legkevesebb három koordinátaindexet kell megadni (mindegyik koordinátaindex egy-egy koordinátahármasnak, azaz egy térbeli csúcspontnak felel meg). A *coordIndex* mezőben a -1 értékkel kell jelezni az aktuális síkidom csúcspontjainak végét. Ha az alakzat egynél több síkidomból áll, akkor az egyes síkidomokat a -1 mezőértékkel kell egymástól elválasztani. Az alakzat utolsó síkidomának megadása után is be kell írni a -1 mezőértéket. A *materialIndex* mező azoknak az anyagoknak az indexét adja meg, amelyeket a böngésző az alakzat csúcspontjaira alkalmaz (így érhető el például, hogy egy hasáb oldalainak különbözők legyenek a színei). A *normalIndex* mező azoknak a *normálisoknak* az indexeit adja meg, amelyeket a böngésző az egyszerű geometriai csomópontok csúcspontjaira alkalmaz. (A matematikában normálisnak vagy normál vektornak nevezik a valamely sík felületre merőleges, onnan kifelé mutató vektort. A böngésző *normálisok* segítségével számítja ki azt a fényhatást, ami abból származik, hogy a jelenet fényforrásai merőlegesen visszaverődnek egy objektum felületéről.) A *textureCoordIndex* mező azoknak a mintázatoknak az indexeit adja meg, amelyeket a böngésző az adott koordinátákhoz köt.

A VRML dokumentumok az *IndexedFaceSet* csomópont *coordIndex* mezőértékeit használják egy sokszög valamely oldalának megadásához. A böngésző az aktuális transzformációs mátrixot használja a sokszög csúcspontjainak transzformálásához.

EGY RÖVID PÉLDA:

Az alábbi kód gúlát készít a *Coordinate3* és az *IndexedFaceSet* csomópont segítségével. A gúla négy egybevágó háromszögből és egy négyzet alakú alaplapból tevődik össze. A *Coordinate3* cso-

mópontban megadjuk azoknak a csúcspontoknak a koordinátáit, amelyeket az ezt követő *IndexedFaceSet* csomópontnak a *coordIndex* mezőben felsorolt indexhivatkozásokkal használnia kell a gúla létrehozásához.

```
#VRML V1.0 ascii

DEF Gula Separator {

Material {
diffuseColor 1 0 0 # piros színû legyen
  }

    Coordinate3 {
    point [-1 0 -1, 1 0 -1, 1 0 1, -1 0 1, 0 1 0]      # a gúlát alkotó síkidomok
    # csúcspontjainak koordinátái
      }

        IndexedFaceSet {
        coordIndex [

        0, 4, 1, -1,            # az 1. háromszög csúcspontjainak indexei
        1, 4, 2, -1,            # a 2. háromszög csúcspontjainak indexei
        2, 4, 3, -1,            # a 3. háromszög csúcspontjainak indexei
        3, 4, 0, -1,            # a 4. háromszög csúcspontjainak indexei
        0, 1, 2, 3, -1          # az alapot képezõ négyzet csúcspontjainak indexei
        ]
      }
  }
```

Ha a mintázat koordinátájának kötése PER_VERTEX, akkor kívülről (a *TextureCoordinate2* csomópontban) megadott mintázatkoordinátákat köthetünk egy indexelt, egyszerű geometriai csomóponthoz, egyiket a másik után. Ha viszont PER_VERTEX_INDEXED a kötés, akkor a *textureCoordIndex* mezőben megadott indexeket kell használni a mintázatkoordinátáknak a sokszög csúcspontjaihoz történő kötéséhez.

Valamennyi, csúcspontokkal meghatározott *egyszerű geometriai alakzatra* vonatkozik, hogy ha definiálunk egy mintázatot, de nem definiáljuk a mintázat koordinátáit, akkor a böngésző alapértelmezés szerint egy befoglaló téglalap segítségével képezi le a mintázat koordinátáit. Ennek a befoglaló téglalapnak a méreteit egy S és egy T nevű érték adja meg. A befoglaló téglalap S koordinátáját a nagyobbik, a T koordinátát a kisebbik méret adja. A téglalapot az S koordináta szerint normalizálják, azaz: $0 \leq S \leq 1$.

AZ INDEXEDLINESET CSOMÓPONT

A VRML dokumentumok az *IndexedLineSet* csomópontot is 3-D alakzat megjelenítésére használják. Egy ilyen „testet" azonban nem síkidomokból, hanem az azok körvonalait alkotó sokszögekből építünk fel úgy, hogy az adott koordinátákon lévő csúcspontokból sokszögeket készítünk és ezeket egymáshoz illesztjük. Az így kapott test „üres" lesz (drótváz-modell). Az alábbi utasítások az *IndexedLineSet* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
IndexedLineSet {
  coordIndex          0                                    # MFLong
  materialIndex       -1                                   # MFLong
  normalIndex         -1                                   # MFLong
  textureCoordIndex -1                                     # MFLong
}
```

A *coordIndex* mező a sokszöget alkotó csúcspontok koordinátáinak indexértékeit adja meg (0-tól induló indexértékeket használva). Egy nem negatív indexérték egy *(x, y, z)* koordinátájú 3-D pontnak felel meg. Mivel az alakzatot alkotó sokszögek elméletileg ugyancsak síkidomot határoznak meg, ezek mindegyikének definíciószerűen legkevesebb három vagy több oldalúnak kell lennie. Ezért sokszögenként legkevesebb három *koordinátaindexet* kell megadni (mindegyik koordinátaindex egy-egy koordinátahármasnak, azaz egy térbeli csúcspontnak felel meg). A *coordIndex* mezőben a -1 értékkel kell jelezni az aktuális sokszög csúcspontjainak végét. Ha az alakzat egynél több sokszögből áll, akkor az egyes sokszögeket a -1 mezőértékkel kell egymástól elválasztani. Az utolsó sokszög megadása után is be kell írni a -1 mezőértéket.

A *materialIndex* mező azoknak az anyagoknak az indexét adja meg, amelyet a böngésző a sokszög csúcspontjaira alkalmaz. A *normalIndex* mező azoknak a *normálisoknak* az indexeit adja meg, amelyeket a böngésző az *egyszerű geometriai csomópontok* csúcspontjaira alkalmaz. A *textureCoordIndex* mező azoknak a mintázatoknak az indexeit adja meg, amelyeket a böngésző az adott koordinátákhoz köt.

A VRML dokumentumok az *IndexedLineSet* csomópont *coordIndex* mezőértékeit használják egy sokszög csúcspontjainak megfelelő indexértékek megadásához. A böngésző az aktuális transzformációs mátrixot használja a sokszög csúcspontjainak transzformálásához.

AZ INFO CSOMÓPONT

VRML dokumentumokban az *Info* csomópont segítségével definiálható egy információs csomópont (mint például a Copyright közlemény vagy a szerző neve) a jelenetsorban. Az alábbi utasítás az *Info* csomópont mezőnevét és a hozzá tartozó, alapbeállítás szerinti értékeket mutatja be:

```
Info {
  string "<Az info szövege>"                               # SFString
}
```

A LOD CSOMÓPONT

VRML dokumentumokban a LOD (Level of Detail – részletezési szint) csomópont automatikusan átkapcsolja a VRML böngészőket egy jelenetsor objektumainak különböző megjelenítési módjai között. Ilyen módon a böngésző „közelebb" vagy „távolabb" mehet egy képhez attól függően, hogy milyen távolságra van a felhasználó az objektumtól. Az alábbi utasítások a LOD csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
LOD {
  center 0 0 0                                             # SFVec3f
  range  []                                                # MFFloat
}
```

A *center* mező egy LOD objektumcsoport középpontját adja meg. A böngésző az aktuális transzformációt alkalmazza az objektum középpontjára. A *range* mező egy értékeket tartalmazó tömböt ad meg, amelynek az értékei a felhasználó nézőpontja és a LOD csomóponton belüli objektumcsoport középpontja közötti távolságoknak felelnek meg. A böngésző összehasonlítja az aktuális távolságot a *range* tömbben lévő első értékkel. Ha a távolság kisebb, mint ez az első érték, akkor a böngésző visszaállítja a LOD csoport első gyermekét. Ha a távolság nagyobb, mint az első érték, de kisebb, mint a tömb második értéke, akkor a böngésző a LOD objektum második gyermekét állítja vissza, és így tovább.

A MATERIAL CSOMÓPONT

VRML dokumentumokban a *Material* csomópont segítségével definiálhatjuk az aktuális és az utána következő alakzatok felületének anyagjellemzőit. Ha a *Material* csomópontot egy *MaterialBinding* csomóponttal (lásd később) együtt használjuk, akkor a jelenetsor különböző kinézeteket kölcsönözhet az objektumoknak. Az alábbi utasítások a *Material* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
Material {
  ambientColor     0.2 0.2 0.2                    # MFColor
  diffuseColor     0.8 0.8 0.8                    # MFColor
  emissiveColor    0 0 0                          # MFColor
  shininess        0.2                            # MFFloat
  specularColor    0 0 0                          # MFColor
  transparency     0                              # MFFloat
}
```

Az *ambientColor* (környező fény), *diffuseColor* (szórt fény), *specularColor* (visszatükrözött fény) és az *emissiveColor* (kibocsátott fény) mezők az anyag piros, zöld és kék színkeverékét adják meg. A piros, zöld és kék színek értékei a 0 – 1 tartományban lehetnek. Ha például egy mező színét kékre akarjuk állítani, akkor a mezőhöz a 0 0 1 értéket kell rendelni. Hasonlóképpen, ha egy objektumot sárga színnel akarunk megjeleníteni, akkor a mezőhöz az 1 1 0 értéket kell rendelni (a piros és a zöld keveréke adja a sárga színt).

A *shininess* (fényesség) és a *transparency* (áttetszőség) mezők értékei a 0 – 1 tartományban lehetnek. Ha nagyon fénylően akarunk megjeleníteni egy objektumot, akkor a shininess mezőhöz az 1 értéket rendeljük. Ha egy objektumot teljesen *átlátszóként* szeretnénk megjeleníteni, akkor a *transparency* mezőhöz az 1 értéket rendeljük.

A *Material* csomópont határozza meg az aktuális anyag megjelenését, miközben a böngésző a jelenetsorban halad. Észrevehetjük, hogy különböző alakzatok különbözőképpen jelenítenek meg többféle értékkel rendelkező anyagokat. A VRML dokumentumok a *MaterialBinding* csomópont segítségével kötik az anyagokat az alakzatokhoz (lásd a következő bekezdést).

A MATERIALBINDING CSOMÓPONT

A VRML dokumentumok a *MaterialBinding* csomópont segítségével határozzák meg, hogy a böngészők hogyan kössék az aktuális anyagokat a jelenetsorban következő alakzatokhoz. Az alábbi utasítások a *MaterialBinding* csomópont mezőnevét és alapbeállítás szerinti értékét mutatják be:

```
MaterialBinding {
  value   DEFAULT                                 # SFEnum
}
```

A *value* mező azt írja le, hogy miként kösse a *MaterialBinding* csomópont az anyagot egy objektum részeihez, oldalaihoz vagy csúcspontjaihoz. A mező az alábbi értékeket veheti fel:

- DEFAULT – A böngésző az alapbeállítás szerinti kötését alkalmazza.
- OVERALL – A böngésző ugyanazt a kötést alkalmazza a teljes objektumra.
- PER_FACE – A böngésző meghatározott anyagot alkalmaz az objektum minden egyes oldalára.
- PER_FACE_INDEXED – A böngésző meghatározott anyagot alkalmaz az objektum minden egyes oldalára, az anyag indexértéke alapján.
- PER_PART – A böngésző meghatározott anyagot alkalmaz az objektum minden egyes részére.
- PER_PART_INDEXED – A böngésző meghatározott anyagot alkalmaz az objektum minden egyes részére, az anyag indexértéke alapján.
- PER_VERTEX – A böngésző meghatározott anyagot alkalmaz az objektum minden egyes csúcspontjára.
- PER_VERTEX_INDEXED – A böngésző meghatározott anyagot alkalmaz az objektum minden egyes csúcspontjára, az anyag indexértéke alapján.

Ne feledjük, hogy a böngésző különböző alakzatokra eltérően értelmezheti az anyagkötéseket. Az aktuális anyagnak mindig van egy alapértéke, amelyet az összes anyagmező első értéke határoz meg. Mivel az anyagmezőknek több értékük is lehet, a *MaterialBinding* csomópont határozza meg, hogy a böngésző hogyan alkalmazza (kösse) az anyagokat egy alakzatra. Ha a böngészőnek több anyagot kell kötnie egy objektumhoz, akkor elejétől végéig ciklikusan végighalad az anyagértékeken, a legtöbb értéket tartalmazó anyagösszetevő periódusidejének megfelelően.

A MATRIXTRANSFORM CSOMÓPONT

A VRML dokumentumok a *MatrixTransform* csomópontot használják egy geometrikus 3-D transzformációs mátrix megadásához. A 3-D transzformációhoz egy 4x4 elemű egységmátrixra van szükség. Az alábbi utasítások a *MatrixTransform* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
MatrixTransform {
  matrix       1 0 0 0                          # SFMatrix
               0 1 0 0
               0 0 1 0
               0 0 0 1
}
```

A NORMAL CSOMÓPONT

A VRML dokumentumok a *Normal* csomópont segítségével definiálják azoknak a normál vektoroknak a tömbjét, amelyeket a jelenetsorban a *Normal* csomópontot követő, csúcspontokkal meghatározott geometriai alakzatok használnak. Más szavakkal ez azt jelenti, hogy a *Normal* csomópont segítségével egy vagy több 3-D normál vektort definiálhatunk. Az alábbi utasítások a *Normal* csomópont mezőnevét és az alapbeállítás szerinti értékeket mutatják be:

```
Normal {
  vector    0 0 1                               # MFVec3f
}
```

A NORMALBINDING CSOMÓPONT

VRML dokumentumokban a *NormalBinding* csomópont segítségével határozható meg, hogy a böngésző hogyan kösse az aktuális normálisokat a jelenetsorban a csomópontot követő alakzatok-

hoz. Az alábbi utasítások a *NormalBinding* csomópont mezőnevét és az alapbeállítás szerinti értékeit mutatják be:

```
NormalBinding {
  value      DEFAULT                                    # SFEnum
}
```

A *value* mező azt írja le, miként kösse a böngésző a normálisokat a jelenetsor objektumaihoz. A böngésző különböző alakzatokra eltérően értelmezheti a kötéseket. A *value* mező az alábbi értékeket veheti fel:

- DEFAULT – A böngésző az alapbeállítás szerinti normál kötést alkalmazza.
- OVERALL – A böngésző ugyanazt a kötést alkalmazza a teljes objektumra.
- PER_FACE – A böngésző meghatározott normálist alkalmaz az objektum minden egyes oldalára.
- PER_FACE_INDEXED – A böngésző meghatározott normálist alkalmaz az objektum minden egyes oldalára, az anyag indexértéke alapján.
- PER_PART – A böngésző meghatározott normálist alkalmaz az objektum minden egyes részére.
- PER_PART_INDEXED – A böngésző meghatározott normálist alkalmaz az objektum minden egyes részére, az anyag indexértéke alapján.
- PER_VERTEX – A böngésző meghatározott normálist alkalmaz az objektum minden egyes csúcspontjára.
- PER_VERTEX_INDEXED – A böngésző meghatározott normálist alkalmaz az objektum minden egyes csúcspontjára, az anyag indexértéke alapján.

AZ ORTOGRAPHICCAMERA CSOMÓPONT

A VRML dokumentumok az *OrtographicCamera* csomópont segítségével definiálják a valamely nézőpontból történő párhuzamos leképzést (vetítést). A jelenetsorok tervezői kamerák segítségével szabályozhatják azt a szöget (vagy perspektívát), amelyben a felhasználó megtekintheti a VRML jelenetsort. Az ortografikus kamera a *perspektivikus kamerával* ellentétben (lásd később) a távolság növekedésével nem tünteti el az objektumokat. Ha ortografikus kamerával történő leképzést használunk, akkor amikor a *felhasználó távolodik* a színhelytől, az objektumok úgy húzódnak vissza a háttérbe, hogy nem zsugorodnak össze egy eltűnő pontba (a látómező közepén lévő pontba, amely felé az objektumok konvergálnak, majd eltűnnek). Az alábbi utasítások az *OrtographicCamera* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
OrtographicCamera {
  focalDistance   5                                     # SFFloat
  height          2                                     # SFFloat
  orientation     0 0 1 0                               # SFRotation
  position        0 0 1                                 # SFVec3f
}
```

A *focalDistance* mező az ortografikus kamera és az objektumok közötti távolságot adja meg. A *height* mező a látótér teljes magasságát adja meg (hosszú, négyszögletes mező, amelyen belül az objektumok megjelennek). A position mező az ortografikus kamera *x*, *y* és *z* koordinátáját adja meg. Az orientation mező az ortografikus kamerát forgatja. Alapbeállítás szerint az ortografikus kamera helye a (0, 0, 1) koordinátapont, a tekintetünk pedig a -*z* tengely irányába néz (a monitor belseje felé).

A böngésző az aktuális transzformációt használja egy ortografikus kamerára. Úgy pozicionálhatunk egy kamerát, hogy a színhelyen az *OrtographicCamera* csomópont elé elhelyezünk egy *Transform* csomópontot.

A PERSPECTIVECAMERA CSOMÓPONT

A VRML dokumentumok a *PerspectiveCamera* csomópont segítségével definiálják a valamely nézőpontból történő perspektivikus leképzést (vetítést). A jelenetsorok tervezői kamerák segítségével szabályozhatják azt a szöget (vagy perspektívát), amelyben a felhasználó megtekintheti a VRML jelenetsort. A perspektivikus kamera az ortografikus kamerával ellentétben a távolság növekedésével eltünteti az objektumokat. Ha *perspektivikus kamerával* történő leképzést használunk, akkor amikor a felhasználó távolodik a színhelytől, az objektumok egy *eltűnő ponthoz* közelítenek (a látómező közepén lévő ponthoz, amely felé az objektumok konvergálnak, majd eltűnnek). Az alábbi utasítások a *PerspectiveCamera* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
PerspectiveCamera {
  focalDistance  5                                  # SFFloat
  heightAngle    0.785398                           # SFFloat
  orientation    0 0 1 0                            # SFRotation
  position       0 0 1                              # SFVec3f
}
```

A *focalDistance* mező a perspektivikus kamera és az objektumok közötti távolságot adja meg. A *heightAngle* mező a látótér teljes függőleges szögét adja meg. A perspektivikus kamera látótere csonkagúla alakú. A *position* mező a perspektivikus kamera *x*, *y* és *z* koordinátáját adja meg. Az *orientation* mező a perspektivikus kamerát forgatja. Alapbeállítás szerint a perspektivikus kamera helye a (0, 0, 1) koordinátapont, a tekintetünk pedig a -*z* tengely irányába néz (a monitor belseje felé).

A POINTLIGHT CSOMÓPONT

A böngészők a *PointLight* csomópont segítségével tudnak egy fényforrást adott térbeli helyre elhelyezni. A pontszerű fényforrás minden irányban egyforma megvilágítást ad. A *PointLight* csomópont olyan fényforrást definiál, amely a jelenetsor további alakzataira is hatással lehet, attól függően, hogy milyen az aktuális fény stílusa. Az alábbi utasítások a *PointLight* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
PointLight {
  color          1 1 1                              # SFColor
  intensity      1                                  # SFFloat
  location       0 0 1                              # SFVec3f
  on             TRUE                               # SFBool
}
```

A *color* (szín) mező a fény piros, zöld és kék színösszetevőit határozza meg. A *color* mező értékeinek mindegyike a 0 – 1 tartományban lehet. A *location* mező a fénypont helyét adja meg a jelenetsoron belül. Az *intensity* (fényesség) mező a fénypont *minden irányban* azonos fényerősségét állítja be. Az *intensity* mező értéke a 0 – 1 tartományban lehet, ahol az 1 jelenti a legnagyobb

fényességet. Az *on* mező TRUE értéke kapcsolja be, a FALSE értéke pedig a pontszerű fényforrást kapcsolja ki.

A POINTSET CSOMÓPONT

A VRML dokumentumok a *PointSet* csomópont segítségével jelenítik meg az aktuális koordinátákon elhelyezett pontokat. A *PointSet* csomópontot egy értékeket tartalmazó tömbnek is tekinthetjük. Az alábbi utasítások a *PointSet* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
PointSet {
  numPoints       -1                                    # SFLong
  startIndex      0                                     # SFLong
}
```

A *numPoints* mező a készletben lévő pontok számát adja meg, és a böngésző sorrendben megjeleníti a *PointSet* csomópont elemeit. A *numPoints* mező -1 értéke azt jelzi, hogy a böngészőnek az aktuális koordinátákon lévő valamennyi fennmaradó értéket pontokként kell megjelenítenie. A *startIndex* a kezdő 3-D pontnak megfelelő indexet adja meg. A böngésző az aktuális kumulatív transzformációt alkalmazza a pontkészletben lévő koordinátákra. Továbbá a böngésző az aktuális anyagot és szövetet használva rajzolja meg a pontokat.

A ROTATION CSOMÓPONT

A VRML dokumentumok a *Rotation* csomópont segítségével határoznak meg egy háromdimenziós forgatást az *x*, az *y* és a *z* tengely körül. Az alábbi utasítások a *Rotation* csomópont mezőnevét és az alapbeállítás szerinti értékeit mutatják be:

```
Rotation {
  rotation        0 0 1 0                               # SFRotation
}
```

A *rotation* mező első három értéke az *x*, az *y* és a *z* tengely körüli relatív forgatásnak felel meg, míg a negyedik érték a forgatás szögét adja meg radiánban. A böngésző az aktuális transzformációt alkalmazza a *Rotation* csomópontra.

A SCALE CSOMÓPONT

A VRML dokumentumok a *Scale* csomópontot használják arra, hogy meghatározzák az eredeti méretek három dimenzióban történő megváltoztatását. A *Scale* csomópont segítségével a jelenetsor tervezői növelhetik vagy csökkenthetik egy objektum *x*, *y* és *z* tengelyű méreteit. A csomópont az objektum méreteit a három tengelyen különböző mértékben is megváltoztathatja. Az alábbi utasítások a *Scale* csomópont mezőnevét és az alapbeállítás szerinti értékeit mutatják be:

```
Scale {
  scaleFactor     1 1 1                                 # SFVec3f
}
```

A *scaleFactor* mező azt a mértéket adja meg, amellyel a böngészőnek meg kell változtatnia az objektum méreteit a *x*, az *y* és a *z* tengely mentén.

A SEPARATOR CSOMÓPONT

A VRML dokumentumok a *Separator* (elválasztó) csomópontot használják arra, hogy az elválasztó csomópont gyermekcsomópontjait elkülönítsék a jelenetsor egyéb részeitől. A *Separator* csomópont menti (kiveszi) a haladási állapotot (azt a módot, ahogyan a böngésző feldolgozza a csomópontokat, és amilyen sorrendben visszaállítja azokat), mielőtt áthaladna a gyermekcsomópontjain. Miután a *Separator* csomópont az utolsó gyermekcsomóponton is áthaladt, megjeleníti a haladási állapotot. Az alábbi utasítások a *Separator* csomópont mezőnevét és az alapbeállítás szerinti értékét mutatják be:

```
Separator {
  renderCulling  AUTO                                    # SFEnum
}
```

A *renderCulling* mező az úgynevezett „objektumkiselejtezést" (object culling) állítja be a jelenetsorban. (A valós világban a környezetünknek mindig csak azt a részét látjuk, ami a látókörünkbe esik. A látókörünkön kívüli tárgyak nem láthatók, így ezeket a böngészőnek nem kell megjelenítenie. Azt az eljárást, amely megállapítja, hogy egy adott jelenetsorban mely objektumok megjelenítése hagyható el, objektumkiselejtezésnek nevezik.)

A *Separator* csomópont fényeket, kamerákat, koordinátákat, normálisokat, kötéseket és egyéb csomópont-tulajdonságokat foglalhat magában. A *Separator* csomópont objektumok kiselejtezését is elvégzi, amivel a böngésző automatikusan átlépi a *Separator* gyermekcsomópontjait, ha azokat nem jeleníti meg.

Azt, hogy egy gyermekcsomópontot meg kell-e jeleníteni, a *Separator* csomópont befoglaló hasábjának és az aktuális látótér összehasonlításának eredménye határozza meg. Ha a *renderCulling* mező az AUTO értékre van állítva (ami az alapbeállítás), akkor a VRML implementáció dönti el, hogy szükség van-e a selejtezésre. Ha a mező értéke OFF, akkor nem történik meg a selejtezés, ha viszont a mező értéke ON, akkor a böngésző minden esetben megkísérli a selejtezést.

A SHAPEHINTS CSOMÓPONT

A VRML dokumentumok a *ShapeHints* csomópont segítségével határozzák meg, hogy az *IndexedFaceSet* csomópontok alakzatai tömörek-e, sorba rendezettek-e a csúcspontjai, vagy hogy tartalmaznak-e domború (konvex) felületeket. A VRML dokumentumok a *ShapeHints* csomópont révén segítik a VRML implementációkat bizonyos megjelenítési műveletek optimalizálásában. Ilyen optimalizálás például a hátoldalak megjelenítésének elhagyása (egy kocka hat lapjából egyszerre mindig csak maximum három látható) és a kétoldali megvilágítás tiltása. Ha például egy jelenetsorban egy objektum tömör és a csúcspontjai sorba rendezettek, akkor a VRML implementáció bekapcsolhatja a hátoldalak elhagyását, és kikapcsolhatja a kétoldali megvilágítást. Ehhez hasonlóan, ha egy objektum nem tömör, de a csúcspontjai sorba rendezettek, akkor a VRML implementáció kikapcsolhatja a hátoldalak elhagyását, és bekapcsolhatja a kétoldali megvilágítást. Az alábbi utasítások a *ShapeHints* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
ShapeHints {
  creaseAngle     0.5                                  # SFFloat
  faceType        CONVEX                               # SFEnum
  shapeType       UNKNOWN_SHAPE_TYPE                   # SFEnum
  vertexOrdering  UNKNOWN_ORDERING                     # SFEnum
}
```

(A VRML 1.1 verziójától kezdődően a shapeType alapbeállítás szerinti értéke SOLID, a vertexOrdering értéke pedig COUNTERCLOCKWISE. – A fordító megjegyzése.)

A *creaseAngle (hajlási szög)* mező két szomszédos, egymással egy él mentén találkozó síkidom *normálisainak* egymással bezárt szögét állítja be. Ha egy *IndexedFaceSet* csomópontnak az általa megadott síkidomokra az alapbeállítás szerinti *normálisokat* kell előállítania, akkor a *creaseAngle* mező segítségével határozza meg, hogy a böngészőnek mely találkozási éleket kell lágyabb, és melyeket élesebb átmenetként megjelenítenie. (Egy félbehajtott, majd majdnem teljesen síkba kiterített papírlap hajtási éle sokkal kevésbé látható, mintha derékszögben hajlítanánk meg a papírlapot.) A hajlási szög a szomszédos síkidomok felszíni normálisai által bezárt szög. Így például egy 0,5 radián hajlási szög (ez az alapértelmezés szerinti érték) azt jelzi, hogy két szomszédos síkidom találkozási éle lágy átmenetű lesz, ha a két felület normálisai 0,5 radiánnál kisebb szöget (megközelítőleg 30 fok) zárnak be. Ellenkező esetben a két felület találkozási éle hangsúlyosabb lesz.

A *faceType* mező értéke azt adja meg, hogy az alakzatnak vagy mindegyik oldala domború (a CONVEX érték), vagy hogy a dokumentumnak nincsen információja az alakzat oldalairól (az UNKNOWN_FACE_TYPE érték). A *vertexOrdering* mező az objektumok csúcspontjainak rendezését állítja be a következő értékek egyikére: UNKNOWN_ORDERING (ismeretlen rendezés), CLOCKWISE (óramutató járásával egyező irányú rendezés) vagy COUNTERCLOCKWISE (óramutató járásával ellentétes irányú rendezés). A *shapeType* mező azt jelzi, hogy az alakzat térfogatot zár közre (SOLID), vagy azt, hogy a dokumentumnak nincs információja az alakzatról (UNKNOWN_SHAPE_TYPE).

A SPHERE CSOMÓPONT

VRML dokumentumokban a *Sphere* csomóponttal jeleníthető meg gömb. Az alábbi utasítások a *Sphere* csomópont mezőnevét és az alapbeállítás szerinti értékét mutatják be:

```
Sphere {
  radius        1                                  # SFFloat
}
```

A *radius* (sugár) mező határozza meg a gömb sugarát. Alapbeállítás szerint a gömb közepe a (0, 0, 0) koordinátapontban van. A böngésző az aktuális kumulatív transzformációt alkalmazza a *Sphere* csomópontra, és az aktuális anyagot és szövetet használja.

Egy gömbnek nincsenek oldalai vagy részei. Ezért a *Sphere* csomópontok nem fogadják el anyagok és *normálisok* kötéseit, hanem az első anyagot használják a gömb teljes felületének beborítására, és saját normálisaikat használják. Amikor a böngésző valamilyen mintázatot alkalmaz egy *Sphere* csomópontra, akkor a gömb teljes felületét beborítja a mintázattal úgy, hogy azt a gömb hátoldalánál kezdve az óramutató járásával ellentétes irányban helyezi fel. A felhelyezett mintázatot egy varrat fogja össze a gömb hátoldalán, az *y-z* síkon.

A SPOTLIGHT CSOMÓPONT

VRML dokumentumokban a *SpotLight* csomópont segítségével jeleníthető meg egy fényszóró egy objektumon. A *SpotLight* csomópont olyan fényforrást definiál, amely az aktuális fénystílustól függően a jelenetsor további alakzatait is befolyásolhatja. Az alábbi utasítások a *SpotLight* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
SpotLight {
  color          1 1 1                              # SFVec3f
  cutOffAngle    0,785398                           # SFFloat
```

```
  direction       0 0 -1                          # SFVec3f
  dropOffRate     0                               # SFFloat
  intensity       1                               # SFFloat
  location        0 0 1                           # SFVec3f
  on              TRUE                            # SFBool
}
```

A *color* mező a fényszóró színét határozza meg, ami a mező piros, zöld és kék színértékeiből keverhető ki. Ha például piros színt szeretnénk, akkor a *color* mezőhöz az 1 0 0 értékeket kell rendelnünk. A böngésző a fénysugarakat kúp alakban sugározza szét. A kúp tengelyét a *direction* mező által megadott 3-D vektor határozza meg. A fény intenzitása exponenciálisan csökken, ahogyan egyre jobban távolodik a kúp középvonalától. A *cutOffAngle* mező a kúp szögét állítja be, és ezzel befolyásolja a fényforrás fényének csökkenését.

A *location* mező a fényforrásnak a jelenetsoron belüli koordinátáit határozza meg. A *direction* mező az adott vektorral előírt irányba fordítja a fényszórót. A *dropOffRate* mező azt adja meg, hogy a távolsággal arányosan milyen gyorsan csökkenjen a fényszóró fénye (az érték a 0 – 1 tartományban lehet, ahol a 0 jelenti a leglassabb csökkenést).

Az *on* mező kapcsolja be (TRUE) és ki (FALSE) a fényszóró fényét. Az *intensity* mező a fényszóró fényességét állítja be (az érték a 0 – 1 tartományban lehet; a 0 a legkisebb, az 1 a legnagyobb fényerőt jelenti). A böngésző az aktuális transzformációt alkalmazza a *SpotLight* csomópontra. Egy *Separator* csomóponton belüli *SpotLight* csomópont egyetlen olyan csomópontra sincs hatással, amelyik a *Separator* csomóponton kívül van.

A SWITCH CSOMÓPONT

A VRML dokumentumok a *Switch* csomópontot használják különböző tulajdonságok be-, illetve kikapcsolására. A *Switch* csomópont *Group* csomópont, amely gyermekcsomópontjai közül egyen, mindegyiken, vagy egyiken sem halad át. A csomópont mezőértékétől függően a *Switch* csoport hasonlóan vagy pontosan úgy viselkedhet, mint egy *Group* csomópont. Az alábbi utasítások a *Switch* csomópont mezőnevét és az alapbeállítás szerinti értékét mutatják be:

```
Switch {
  whichChild    -1                                # SFLong
}
```

A *whichChild* mező annak a gyermeknek az indexét adja meg, amelyiken át kell haladni. Az első gyermeket a 0 index jelenti. Ha a *whichChild* mező értéke -1 (ez az alapértelmezés szerinti érték), akkor a *Switch* csomópont arra utasítja a böngészőt, hogy ne haladjon át egyik gyermekcsomóponton sem. Ha a *whichChild* mező értéke -3, akkor ez azt jelenti, hogy valamennyi gyermekcsomóponton át kell haladni, vagyis ekkor a *Switch* csomópont pontosan úgy viselkedik, mint egy normál *Group* csomópont.

(A VRML 1.1 verziótól kezdődően a -3 beállítás nem használható. – A fordító megjegyzése.)

A TEXTURE2 CSOMÓPONT

A VRML dokumentumok a *Texture2* csomóponttal definiálnak mintázatot és a mintázat paramétereit. A böngészők az így definiált mintázattal borítják be az utána következő alakzatokat, amikor megjelenítik azokat. Az alábbi utasítások a *Texture2* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
Texture2 {
  filename        ""                                    # SFString
  image           0 0 0                                 # SFImage
  wrapS           REPEAT                                # SFEnum
  wrapT           REPEAT                                # SFEnum
}
```

A *filename* mező a mintázatot tartalmazó fájl nevét adja meg. A fájl lehet helyi vagy lehet a Weben is – utóbbi esetben a fájl URL azonosítóját kell megadni. Ha a *filename* mező üres karakterláncot tartalmaz, akkor a böngésző kikapcsolja a mintázat megjelenítését.

Az *image* mező egy soron belüli (inline) képet ad meg, amelyet a böngészőnek a mintázat leképezéséhez használnia kell. A *wrapS* és a *wrapT* mező az S és a T tengely körüli burkolás módját határozza meg (ezek a tengelyek a vízszintes és a függőleges tengelynek felelnek meg). A mintázat méretei mind függőleges, mind vízszintes irányban egy 0.0 – 1.0 közötti értéktartományba esnek. Ha a dokumentum a REPEAT értéket rendeli a wrapS és a wrapT mezőhöz, akkor a böngésző az ezen a 0.0 – 1.0 tartományon kívül is megismétli a mintázatot úgy, hogy a mintázatokat mozaikszerűen egymás mellé vagy fölé helyezi. Ha a mezők értéke CLAMP, akkor a böngésző a 0 – 1 tartományon belüli koordinátákra alkalmazza a mintázatot.

A TEXTURE2TRANSFORM CSOMÓPONT

A VRML dokumentumok a *Texture2Transform* csomóponttal definiálnak egy 2-D transzformációt, amelyet aztán a böngésző a mintázat koordinátáira alkalmaz. A *Texture2Transform* befolyásolja azt a módot, ahogyan a böngésző a mintázatot a jelenetsorban egymás után következő alakzatok felületén alkalmazza. Az alábbi utasítások a *Texture2Transform* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
Texture2Transform {
  center          0 0                                   # SFVec2f
  rotation        0                                     # SFFloat
  scaleFactor     1 1                                   # SFVec2f
  translation     0 0                                   # SFVec2f
}
```

A *Texture2Transform* csomópont négy mezőt kombinál a transzformálás elvégzéséhez. A *translation* mező nem azonos mértékű méretváltoztatást definiál egy tetszőleges középpont körül, a *rotation* mező értéke ugyanezen pont körüli elforgatást határozza meg. Ez a két mezőérték együttesen határozza meg azt a 2-D transzformációt, amit a böngésző az ezt követő alakzatok mintázataira alkalmaz.

A TEXTURECOORDINATE2 CSOMÓPONT

A VRML dokumentumok a *Texture2Transform* csomópont segítségével definiálják azokat a 2-D koordinátapárokat, amelyeket a böngésző a mintázatoknak a jelenetsorban következő *PointSet*, *IndexedLineSet* vagy *IndexedFaceSet* objektumok csúcspontjaira történő leképzéséhez használ. A *TextureCoordinate2* csomópont segítségével a VRML dokumentumok lecserélik az aktuális mintázatkoordinátákat. Az alábbi utasítások a *TextureCoordinate2* csomópont mezőnevét és az alapbeállítás szerinti értékét mutatják be:

```
TextureCoordinate2 {
  point          0 0                                    # MFVec2f
}
```

A *point* mező egy koordinátapárt ad meg (a koordinátapár két, 0 és 1 közötti számból áll), amely a mintázat egy meghatározott pontját hozzáilleszti a megfelelő indexkészlethez. A mintázatkoordináták a mintázat 0 és 1 értéktartományán belül helyezkednek el. A 0 érték az adott tengely kezdete, az 1 érték pedig a mintázat legtávolabbi pontja az illető tengelyen. Először a vízszintes, majd a függőleges koordinátát kell megadni.

A TRANSFORM CSOMÓPONT

A VRML dokumentumok a *Transform* csomópont segítségével definiálnak egy geometriai 3-D transzformációt, amely a következő műveletekből áll: (esetleges) nem azonos mértékű méretváltoztatás egy tetszőleges ponthoz képest, forgatás egy tetszőleges pont és tengely körül, valamint eltolás (translation, objektum mozgatása három dimenzióban). Az alábbi utasítások a *Transform* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
Transform {
  center           0 0 0                                # SFVec3f
  rotation         0 0 1 0                              # SFRotation
  scaleFactor      1 1 1                                # SFVec3f
  scaleOrientation 0 0 1 0                              # SFRotation
  translation      0 0 0                                # SFVec3f
}
```

A *center* mező az objektumnak a transzformálás során használt viszonyítási (kiindulási) pontját adja meg. A *rotation* mező azt a szöget adja meg, amellyel a böngészőnek el kell forgatnia az objektumot – a *Rotation* csomóponthoz hasonlóan. A *scaleFactor* mező azt írja elő, hogyan változtassa meg a böngésző az objektum méretét – a Scale csomóponthoz hasonlóan. A *scaleOrientation* mező az irányt adja meg a *scaleFactor* mező számára, mellyel lehetővé teszi, hogy a böngésző irányonként eltérő módon változtassa meg a méreteket. A *translation* mező azt határozza meg, hogy hová helyezzen át a böngésző egy objektumot. A *Transform* csomópont egy vagy több *Rotation*, *Scale* vagy *Translation* csomópontot tartalmazhat.

A TRANSFORMSEPARATOR CSOMÓPONT

A VRML dokumentumok a *Separator* csomóponthoz hasonlóan használják a *TransformSeparator* csomópontot. Mindkét csomópont megőrzi a jelenetsor állapotát, mielőtt végighaladnának gyermekcsomópontjaikon. Az áthaladást követően mindkét csomópont vissza is állítja az állapotot. A *TransformSeparator* csomópont viszont csak az aktuális transzformációt őrzi meg; az összes többi állapotérték őrizetlen marad. Az alábbi utasítás a *TransformSeparator* csomópontot szemlélteti:

```
TransformSeparator {
    # Ide jön a többi csomópont
  }
```

A VRML dokumentumok a *TransformSeparator* csomópontot egy kamera elhelyezéséhez használják, mert a kamerára vonatkozó transzformációk nem befolyásolják a jelenet további részét. A

VRML dokumentumok továbbá arra is használhatják ezt a csomópontot, hogy elszigeteljék a fényforrásokra és egyéb objektumokra vonatkozó transzformációkat.

(A VRML 1.1 verziótól kezdődően a specifikációk már nem tartalmazzák ezt a csomópontot. – A fordító megjegyzése.)

A TRANSLATION CSOMÓPONT

A VRML dokumentumok arra használják a *Translation* csomópontot, hogy egy 3-D vektor segítségével három dimenzióban mozgassanak egy objektumot. A *Translation* csomópont azokat a távolságokat határozza meg, amelyekkel a böngészőnek az objektumot az *x*, *y* és *z* tengely mentén el kell tolnia. Az alábbi utasítás a *Translation* csomópont mezőnevét és az alapbeállítás szerinti értékeit mutatja be:

```
Translation {
  translation     0 0 0                                  # SFVec3f
}
```

A *translation* mező az eltolás mértékét és irányát előíró 3-D vektort adja meg.

A WWWANCHOR CSOMÓPONT

A VRML dokumentumok arra használják a *WWWAnchor* csomópontot, hogy új jelenetet töltsenek be a böngészőbe, amikor a felhasználó egy gyermekcsomópontot választ. A *WWWAnchor* csomópont annyiban hasonlít a *Separator* csomóponthoz, hogy a gyermekcsomópontjain történő végighaladása előtt megőrzi az áthaladási állapotot, majd az áthaladást követően visszaállítja azt. Az alábbi utasítások a *WWWAnchor* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
WWWAnchor {
  description   ""                                       # SFString
  map    NONE                                            # SFEnum
  name   ""                                              # SFString
}
```

A *description* mező felszólítást ad meg, amit a böngésző a name mezőben lévő URL alternatívájaként jeleníthet meg. A *map* mező azt határozza meg, hogy elküldje-e a böngésző a pont koordinátáit (az objektum területéről) az URL-lel együtt, amikor a felhasználó ezt a horgonyt (anchor) választja. A *map* mező értéke NONE (ez az alapértelmezés szerinti) vagy POINT lehet. Ha a *map* mező értéke POINT, akkor a felhasználó által az objektumon kiválasztott pont koordinátái „?x,y,z" alakban hozzáadódnak a name mezőben lévő URL-hez. A *name* mező egy új jelenetet tartalmazó helyi vagy Web fájl nevét adja meg.

A WWWINLINE CSOMÓPONT

A VRML dokumentumok a *WWWInline* csomópont segítségével adják meg azon gyermekcsomópontok helyét, amelyeket a böngésző dinamikusan beolvashat a Webről. Az alábbi utasítások a *WWWInline* csomópont mezőneveit és a hozzájuk tartozó, alapbeállítás szerinti értékeket mutatják be:

```
WWWInline {
   bboxCenter   0 0 0                                    # SFVec3f
   bboxSize   0 0 0                                      # SFVec3f
   name   ""                                             # SFString
}
```

A *bboxSize* és a *bboxCenter* mező a befoglaló hasábot határozzák meg, ha a sorban lévő (inline) objektum alakzat. A *name* mező egy sorban lévő (inline) URL-t ad meg, amely egy gyermekcsomópont fájljának felel meg.

A VRML MEZŐK ÁTTEKINTÉSE

Mint már említettük, a VRML nyelvben a mezők két alapvető osztályba sorolhatók: az egyik osztályba azok tartoznak, amelyek csak egyetlen értéket tartalmazhatnak (ami szám, vektor vagy akár kép is lehet), a másik osztályba pedig azok, amelyek több értéket tartalmazhatnak. A fejezet következő részében áttekintjük a VRML nyelv teljes mezőkészletét, amelyben 12 mező egyetlen értéket, 4 mező pedig több értéket tartalmazhat.

A VRML mezők lehetővé teszik a dokumentumok számára, hogy azok egész (integer), valós (real), logikai (Boolean), vektor, karakterlánc (string) stb. értékeket használjanak. A VRML mezői segítségével azt írjuk le (matematikai kifejezésekkel), hogy miként szeretnénk egy böngészővel 3-D objektumokat megjeleníteni. Amint emlékezhetünk rá, az egyetlen értéket tartalmazó mezők nevei az SF betűkkel (Single-value Field = egyetlen értéket tartalmazó mező), míg a több értéket tartalmazó mezők nevei az MF betűkkel (Multiple-value Field = több értéket tartalmazó mező) kezdődnek.

A VRML 2.0 specifikáció több új mezővel bővül, többek között olyanokkal, amelyek csomópontokat támogatnak (csomópontokat azonosító mutatók helyett), animációkat támogató időmezőkkel és másokkal. A VRML 2.0 mezőtípusait a VRML 2.0 specifikáció tartalmazza.

SFBITMASK

Az *SFBitMask* mező egyetlen értéket tartalmazó mező. A mező jelzőbitekből álló maszkot tartalmaz, amelyet a VRML dokumentumok arra használnak, hogy lehetővé tegyék a böngésző számára különböző jelenetsor-állapotok átugrását felsorolt nevek segítségével. A VRML dokumentumok ezt a mezőt olyan csomópontokkal használják, amelyek mnemonikokat definiálnak a jelzőbitekhez.

SFBOOL

Az *SFBool* egyetlen értéket tartalmazó mező, amelyet a VRML dokumentumok arra használnak, hogy karakterlánc vagy numerikus érték alakjában írják le a logikai értékeket. A 0 numerikus érték egyenlő a „FALSE" karakterlánccal, az 1 numerikus érték pedig a „TRUE" karakterlánccal.

SFCOLOR

Az *SFColor* egyetlen színértéket tartalmazó mező. A VRML dokumentumok egy *SFColor* értéket az RGB (red, green, blue, magyarul: piros, zöld, kék) számhármassal írnak le (szabványos tudományos jelölésmódban megadott három lebegőpontos számmal a 0.0 és az 1.0 tartományban). Az (1, 0, 0) RGB számhármas például a tiszta piros színt jelenti (100% piros, 0% zöld és 0% kék).

SFENUM

Az *SFEnum* egyetlen értéket tartalmazó mező, amely felsorolásos típusú, és tipikusan egy érték mnemonikját definiálja.

SFFLOAT

Az *SFFloat* egyetlen értéket tartalmazó mező, amely egyszeres pontosságú lebegőpontos számot ad meg. A VRML dokumentumok tudományos jelölésmódban írják le az SFFloat mezőhöz rendelendő értékeket, mint például 0.125.

SFIMAGE

Az *SFImage* egyetlen értéket tartalmazó mező, amely egy bitképes képet tárol. Egy *SFImage* mező tömörítetlen, kétdimenziós, színes vagy szürkeárnyalatos képet tartalmazhat. A VRML dokumentumok az *SFImage* mezőt három egész értékkel definiálják, amelyek a kép (képpontokban mért) szélességét, magasságát és a komponensek számát adják meg. Ezt a számhármast az egyes képpontokat megadó hexadecimális értékek követik (szélesség-magasság sorrendben), egymástól kitöltő karakterekkel elválasztva.

Az egykomponensű kép egybájtos hexadecimális értékekkel ábrázolja a képpontok fényességét (0xFF maximális fényesség, 0x00 nincs fényesség). A kétkomponensű kép a fényerősséget az első (felső) bájton, az átlátszóságot pedig a második (alsó) bájton tárolja. A háromkomponensű képek képpontjait három bájt ábrázolja: az első (felső) bájt a piros, a középső a zöld, a harmadik pedig a kék szín értékét tárolja (például a 0xFF0000 bájthármas a piros színnek felel meg). A négykomponensű képek az átlátszósági bájtot a piros, zöld és kék érték után tárolják.

Megjegyzés: Mivel mindegyik képpontnak egyetlen, előjel nélküli szám felel meg, egy „0x0000FF" értékű, háromkomponensű képpont a „0xFF" vagy a „255" (decimális) alakban is megadható. A képpontok megadása balról jobbra és alulról felfelé történik. Az első hexadecimális érték a kép bal alsó sarkában lévő képpontnak, az utolsó hexadecimális érték pedig a kép jobb felső sarkában lévő képpontnak felel meg.

SFLONG

Az *SFLong* egyetlen értéket tartalmazó mező, amely egy hosszú (32 bites) egész számot ad meg. A VRML dokumentumok decimális, hexadecimális („0x" kezdetű) vagy oktális („0" kezdetű) alakban rendelhetnek értéket egy *SFLong* típusú mezőhöz.

SFMATRIX

Az *SFMatrix* egyetlen értéket tartalmazó mező, amely egy transzformációs mátrixot tartalmaz. A VRML dokumentumok a soronkénti felsorolást használva 16, egymástól kitöltő karakterrel elválasztott lebegőpontos számként rendelnek értékeket az *SFMatrix* mezőhöz.

SFROTATION

Az *SFRotation* egyetlen értéket tartalmazó mező, amely egy elforgatást definiál. Az *SFRotation* mező 4, egymástól kitöltő karakterrel elválasztott lebegőpontos értéket használ. A négy érték közül az első három az elforgatás tengelyét, a negyedik az adott tengely körüli jobbkezes elforgatás mértékét adja meg radiánban.

SFSTRING

Az *SFString* egyetlen értéket tartalmazó mező, amely egy ASCII karakterláncot ad meg. A VRML dokumentumok kettős idézőjelek között megadott ASCII karakterek sorozatával rendelnek értéket egy *SFString* mezőhöz. (Ha a karakterlánc nem tartalmaz kitöltő karaktert, akkor elhagyhatók az idézőjelek.) Az idézőjelek között bármilyen karakter – újsor karakter – is előfordulhat. Ha kettős idézőjelet kell elhelyezni a karakterláncba, akkor a kettős idézőjel karaktere elé egy hátrafelé dőlő ferde vonal (backslash) karaktert kell beírni.

SFVEC2F

Az *SFVec2f* egyetlen értéket tartalmazó mező, amely egy 2-D vektort tartalmaz. A VRML dokumentumok kitöltő karakterrel elválasztott, lebegőpontos számpár megadásával rendelnek értéket egy *SFVec2f* mezőhöz.

SFVEC3F

Az *SFVec3f* egyetlen értéket tartalmazó mező, amely egy 3-D vektort tartalmaz. A VRML dokumentumok kitöltő karakterrel elválasztott, lebegőpontos számhármas megadásával rendelnek értéket egy *SFVec3f* mezőhöz.

A TÖBB ÉRTÉKET TARTALMAZÓ MEZŐK ÁTTEKINTÉSE

A VRML nyelv négy többértékű mezőt használ, amelyek segítségével előírja, hogy miként jelenítsenek meg a böngészők 3-D objektumokat. Az egyértékű és a többértékű mezők között az a különbség, hogy a több értéket tartalmazók egynél több csomópontra gyakorolhatnak hatást, míg az egyetlen értéket tartalmazók csak egyetlen csomópontra.

MFCOLOR

Az *MFColor* több értéket tartalmazó mező, amely tetszőleges számú RGB színt tartalmazhat. A VRML dokumentumok egy *SFColor* értéket az RGB számhármassal írnak le (amelyek szabványos tudományos jelölésmódban megadott három lebegőpontos számot tartalmaznak). Egynél több érték megadásakor az értékeket vesszőkkel kell egymástól elválasztani, az értéksort pedig szögletes zárójelek közé kell tenni.

MFLONG

Az *MFLong* több értéket tartalmazó mező, amely tetszőleges számú hosszú (32 bites) egész számot tartalmazhat. A VRML dokumentumok egy vagy több, decimális, hexadecimális vagy oktális formátumú egész szám megadásával rendelnek értékeket egy *MFLong* mezőhöz. Ha egynél több értéket rendelünk egy *MFLong* mezőhöz, akkor az értékeket vesszőkkel kell egymástól elválasztani, az értéksort pedig szögletes zárójelek közé kell tenni.

MFVEC2F

Az *MFVec2f* több értéket tartalmazó mező, amely tetszőleges számú 2-D vektort tartalmazhat. A VRML dokumentumok kitöltő karakterrel elválasztott, egy vagy több, lebegőpontos számpár megadásával rendelnek értékeket egy *MFVec2f* mezőhöz. Ha egynél több értéket rendelünk egy *MFVec2f* mezőhöz, akkor az értékeket vesszőkkel kell egymástól elválasztani, az értéksort pedig szögletes zárójelek közé kell tenni.

MFVEC3F

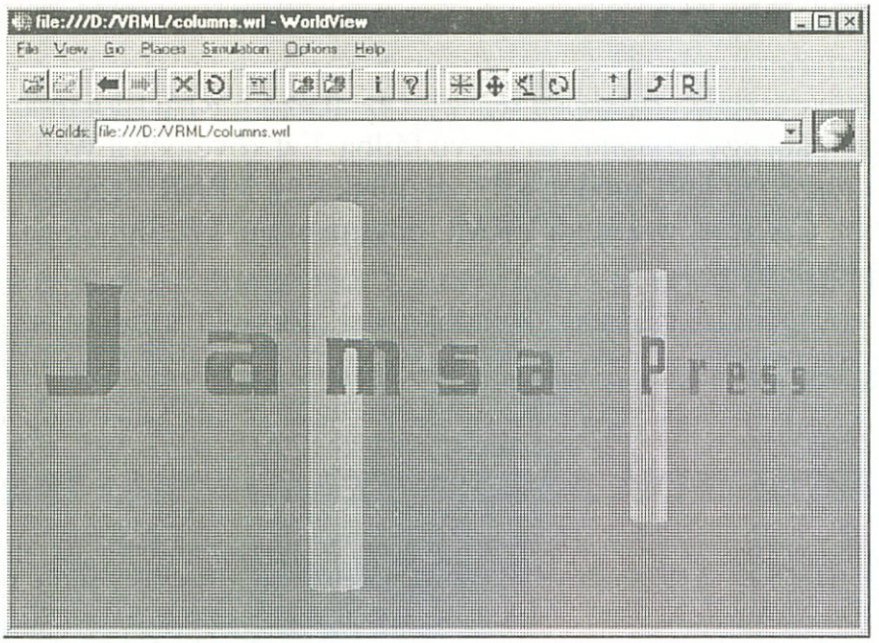
Az *MFVec3f* mező több értéket tartalmazó mező, amely tetszőleges számú 3-D vektort tartalmazhat. A VRML dokumentumok kitöltő karakterrel elválasztott, egy vagy több, lebegőpontos számhármas megadásával rendelnek értékeket egy *MFVec3f* mezőhöz. Ha egynél több értéket rendelünk egy *MFVec2f* mezőhöz, akkor az értékeket vesszőkkel kell egymástól elválasztani, az értéksort pedig szögletes zárójelek közé kell tenni.

NÉHÁNY VRML PÉLDA

A kedvenc ASCII szövegszerkesztőnkben írjuk be az alábbi VRML dokumentumot, és *wrl* kiterjesztéssel mentsük például egy *Columns.wrl* nevű fájlba. Ezt követően a Web-böngészőnkkel (amelyet a fejezet előző részében leírt módon kibővítettünk egy VRML segédprogrammal) jelenítsük meg a fájlt. Az alábbi, *Columns.wrl* nevű VRML dokumentum két függőleges oszlopot és a Jamsa Press kiadó nevét jeleníti meg:

```
#VRML V1.0 ascii

Separator {
    Material {
        diffuseColor 1 1 0                    # legyen sárga szín
      }
    Cylinder {                                # vékony, magas henger
        radius 1
        height 15
      }
    Transform {                               # eltolás 20 egységgel
        translation 20 0 0                    # az x tengely mentén
      }
    Cylinder {                                # ugyanolyan, mint az elõbbi henger
        radius 1
        height 15
      }
    Transform {                               # eltolás 10 egységgel a -x tengelyen
        translation -10 0 5                   # és 5 egységgel a z tengelyen
      }
    Material {                                # legyen piros
        diffuseColor 1 0 0
      }
    AsciiText {                               # Jamsa Press
        string "Jamsa Press"
        justification CENTER
        width 40
      }
  }
```

Amint látható, a VRML dokumentum a *Material* csomópontot használja a színek megadásához, a *Cylinder* csomóponttal két oszlopot készít és az *AsciiText* csomóponttal egy szöveget jelenít meg. Ha egy böngészővel megjelenítjük a dokumentumot, akkor a böngésző a 10.4. ábrán látható módon két oszlopot és egy logót jelenít meg.

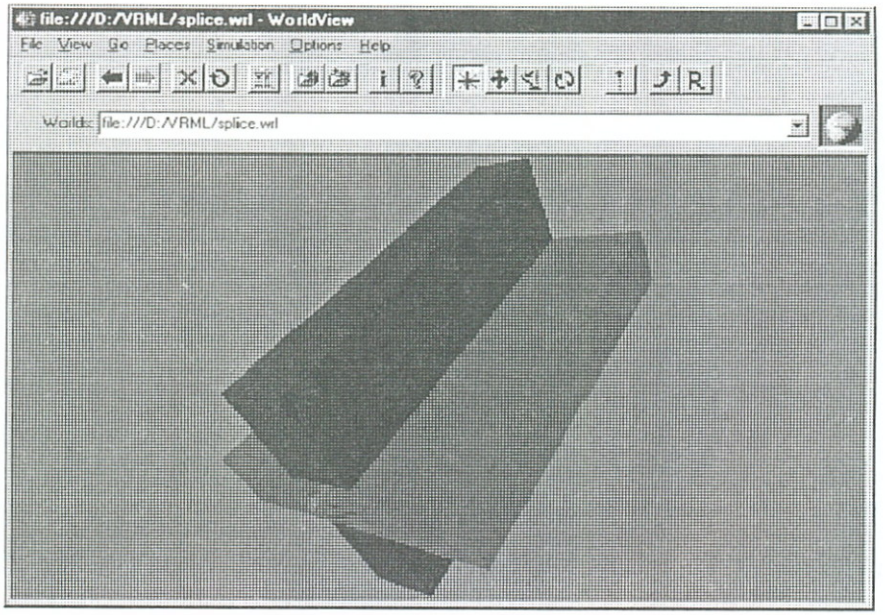


10.4. ábra. A Columns.wrl VRLM dokumentum megjelenítése

A következő, *Splice.wrl* dokumentum két *Cube* csomópont segítségével két, egymáson áthatoló hasábot rajzol meg. A dokumentum a *Rotation* csomópont használatát is bemutatja:

```
#VRML V1.0 ascii

Separator {
    Material {
        diffuseColor 1 0 0                          # legyen piros
      }
    Rotation {
        rotation 0 1 0 1.57079                      # 90 fokos elforgatás
      }                                             # az y tengely körül
    }Cube {
        width 2
        height 3
        depth .5
      }
    Material {
        diffuseColor 0 0 1                          # legyen kék
      }
      Rotation {
        rotation 0 1 0 1.57079                      # 90 fokos elforgatás
      }                                             # az y tengely körül
    Cube {
        width 2
        height 3
        depth .5
      }
}
```

Ha beolvassuk ezt a dokumentumot, akkor a böngésző a 10.5. ábrán láthatóhoz hasonló ábrát fog megjeleníteni az ablakában:



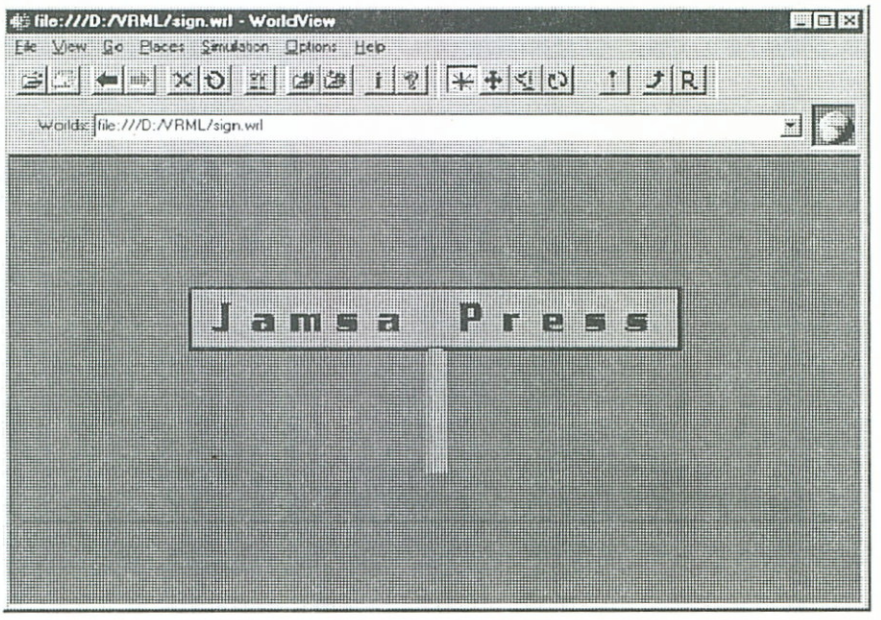
10.5. ábra. A Splice.wrl VRML dokumentum megjelenítése

A következő, *Sign.wrl* nevű VRML dokumentum a Jamsa Press nevét jeleníti meg három dimenziós útvonaljelző táblán. A dokumentum számos csomópontot használ a már megismertek közül:

```
#VRML V1.0 ascii

Material {                                    # élénk kék színû betûk és keret
    diffuseColor 0 0 0
    emissiveColor 0 0 0.5
  }

AsciiText {
    string "Jamsa Press"
    justification CENTER
  }

Translation {
    translation 0 3.5 -1
  }

Cube {
    height 16
    width 120
    depth .9
  }

Material {                                    # legyen sárga
    diffuseColor 1 1 0
  }
```

```
Cube {                                          # az útjelzõ tábla elõlapja
     height 15
     width 118
     depth 1
   }

Translation {
     translation 0 -24 0
   }

Cylinder {                                      # az útjelzõ tábla tartóoszlopa
     height 32
     radius 3
   }

Material {                                      # legyen barna
     diffuseColor 0.5 0.4 0.2
   }

Translation {
     translation 0 -16 0
   }

Cube {                                          # az alap
     width 130
     height 0.2
     depth 100
   }
```

Ha beolvassuk ezt a VRML dokumentumot, akkor a böngészőnk ablakában a 10.6. ábrán láthatóhoz hasonlónak kell megjelennie:



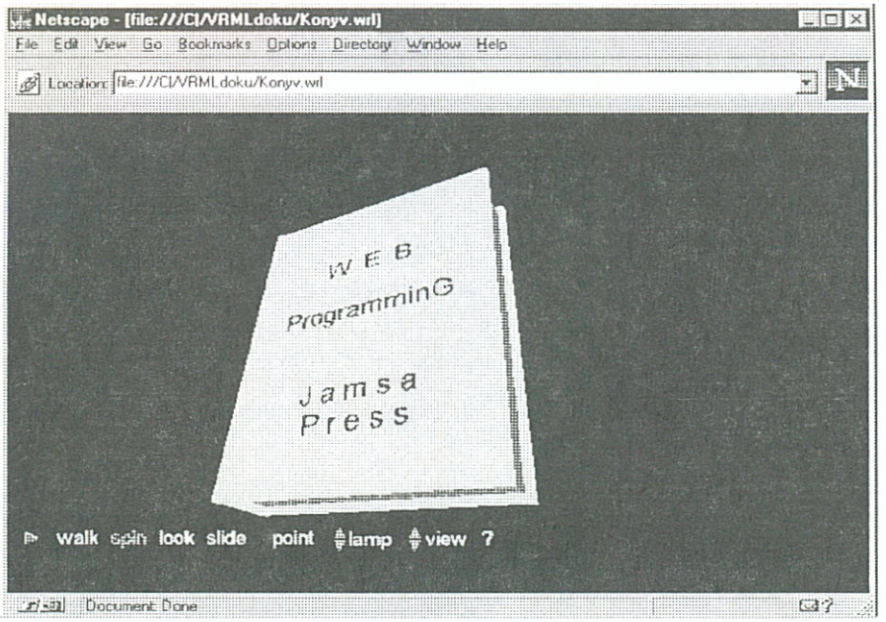
10.6. ábra. A Sign.wrl VRML dokumentum megjelenítése

A most következő, *Book.wrl* nevű VRML dokumentum az egyik *Separator* csomópontot egy könyv megrajzolásához, a másik *Separator* csomópontot pedig a könyv címének megjelenítéséhez használja:

```
#VRML V1.0 ascii

Separator {                             # könyv elkészítése
    Material {                          # legyen kék
        diffuseColor 0 0 1
      }
    Cube {                              # a könyv elõlapja
        height 11.0
        width 8.5
        depth 0.25
      }
    Translation {                       # eltolás az x tengelyen 1/8 egységgel balra,
        translation -.125 0 -1          # a z tengelyen 1 egységgel hátra
      }
    Material {                          # legyen világoskék
        diffuseColor 0 .93 1
      }
    Cube {                              # a könyv lapjai
        height 10.5
        width 8.25
        depth 2.0
      }
    Material {                          # legyen kék
        diffuseColor 0 0 1
      }
    Translation {                       # eltolás az x tengelyen 1/8 egységgel jobbra,
        translation .125 0 -1           # a z tengelyen 1 egységgel hátra
      }
    Cube {                              # a könyv hátlapja
        height 11.0
        width 8.5
        depth 0.25
      }
    Translation {                       # eltolás az x tengelyen 4 3/8 egységgel balra,
        translation -4.375 0 1          # a z tengelyen 1 egységgel elõre
      }
    Cube {                              # a könyv gerince
        height 11.0
        width .25
        depth 2.0
      }
    Translation {                       # eltolás a z tengelyen 1 egységgel elõre
        translation 0 0 1
      }
```

```
    Cylinder {                          # a gerinc egyik oldaléle
        height 11.0
        radius .2
      }
    Translation {                       # eltolás a z tengelyen 2 egységgel hátra
        translation 0 0 -2
      }
    Cylinder {                          # a gerinc másik oldaléle
        height 11
        radius .2
      }
  }

Separator {
    Translation {                       # eltolás az y tengelyen 2 1/4 egységgel fel,
        translation 0 2.25 .15          # a z tengelyen 0.15 egységgel elõre
      }
    Material { # Legyen piros
        diffuseColor 1 0 0
      }

FontStyle {
    size 1
      }
    AsciiText {                         # szöveg formázása és középre állítása
        string [ "W E B","", "ProgramminG", "","",
        "J a m s a", "P r e s s"]
        justification CENTER
        width 8
      }
  }
```

Separator csomópontokat használva a programnak nem kell figyelemmel kísérnie az aktuális beállításokat (mint például a koordinátákat, színeket és mintázatokat), amikor egyik képről átlép egy másikra. Ha beolvassuk ezt a VRML dokumentumot, akkor a böngésző ablakában a 10.7. ábrán láthatóhoz hasonló jelenik meg:



10.7. ábra. A Book.wrl VRML dokumentum megjelenítése

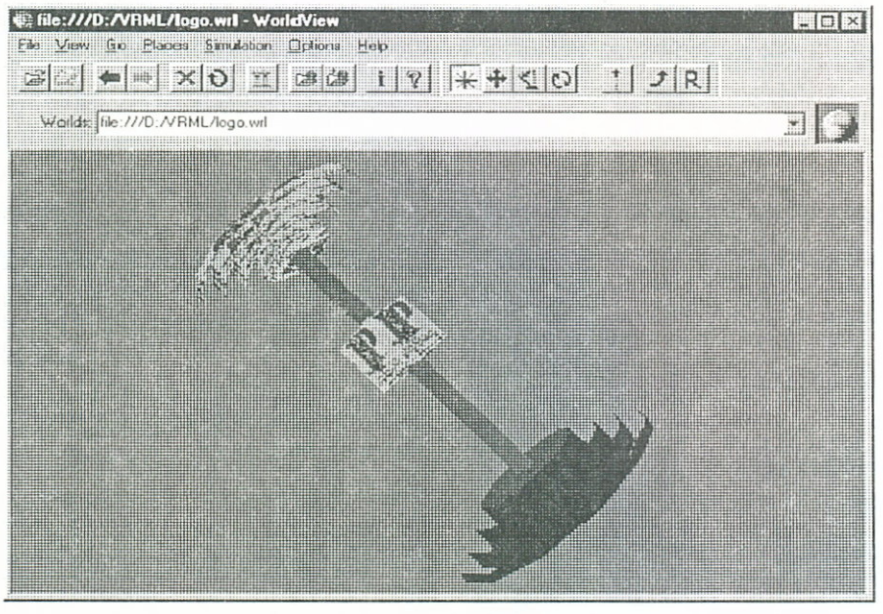
A következő, *Logo.wrl* nevű VRML dokumentum korongokat jelenít meg egy tengely két végén. A tengely közepén egy hasáb van, amelyen a Jamsa Press logója látható. A dokumentum bemutatja, miként lehet egy *Texture2* csomópontban megadni fájlnevet, és miként használható a *Texture2Transform* a kép méretének megváltoztatásához:

```
#VRML V1.0 ascii

Texture2 {
    filename "logo.gif"                 # a fájl a CD-én található
  }
Cube {                                  # tömör hasáb rajzolása,
    height 3                            # mindegyik oldalán a logóval
    width 3
    depth 3
  }

Separator {
    Texture2 {                          # kép törlése ehhez a Separator
        filename ""                     # csomóponthoz
      }
    Material {
        diffuseColor 1 0 0              # legyen piros
      }
    Cylinder {                          # vékony, hosszú tengely rajzolása
        height 20
        radius .5
      }
  }

Separator {
    Translation {                       # felfelé tolni az y tengelyen
```

```
Separator {
    Translation {                          # felfelé tolni az y tengelyen
        translation 0 7.5 0
    }
    Texture2Transform {                    # a kép méretének megváltoztatása
                                           # úgy, hogy 4 kép férjen el a most
                                           # következõ alakzatok szélességében
        scaleFactor 4 1
    }
    Cylinder {                             # tömör korong megrajzolása
        height 2
        radius 2
    }
    Cylinder {                             # felsõ és alsó oldal nélküli
        height 1.5                         # korongpalástok megrajzolása
        radius 3
        parts SIDES
    }
    Cylinder {
        height 1
        radius 4
        parts SIDES
    }
    Cylinder {
        height 0.5
        radius 5
        parts SIDES
    }
}

Separator {                                # talpkorongok megrajzolása
    Texture2 {                             # a logo.gif kép törlése
        filename ""
    }
    Material {                             # legyen kék
        diffuseColor 0 0 1
    }
    Translation {                          # eltolás az y tengelyen
        translation 0 -10 0                # 10 egységgel lefelé
    }
    Cylinder {                             # tömör kék henger megrajzolása
        height 2
        radius 3
    }
    Translation {                          # eltolás az y tengelyen
        translation 0 -.25 0               # 0.25 egységgel lefelé
    }
    Cylinder {                             # felsõ és alsó oldal nélküli
        height 1.5                         # korongpalástok megrajzolása
```

```
        radius 4
        parts SIDES
      }
    Translation {
        translation 0 -.25 0
      }
    Cylinder {
        height 1
        radius 5
        parts SIDES
      }
    Translation {
        translation 0 -.25 0
      }
    Cylinder {
        height 0.5
        radius 6
        parts SIDES
      }
}
```

Amint látható, a VRML dokumentum csak a *Cylinder*, *Cube*, *Material*, *Texture2*, *Texture2Transform* és *Translation* csomópontokat használja, amelyek jó részét már korábbról megismertük. Ha beolvassuk ezt a VRML dokumentumot, akkor a böngésző ablakában a 10.8. ábrán láthatóhoz hasonlónak kell megjelennie.



10.8. ábra. A Logo.wrl VRML dokumentum megjelenítése

ÖSSZEFOGLALÁS

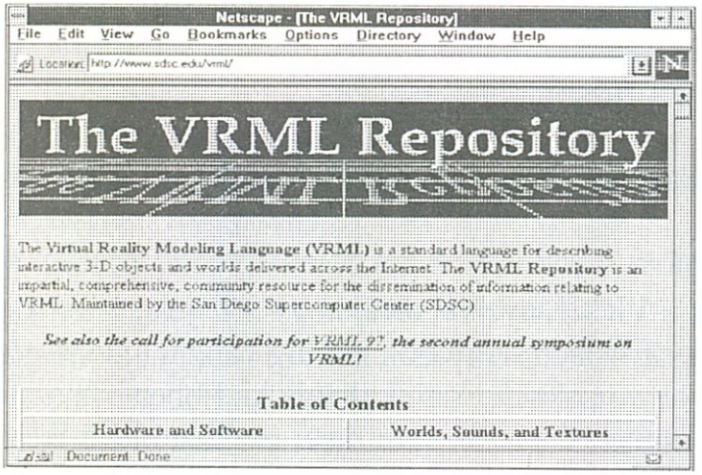
Minden híresztelés ellenére láthattuk, hogy nem olyan nagy ördöngösség VRML dokumentumok elkészítése és látványos 3-D jelenetsorok megtervezése. A Web különböző helyein sokfelé találkozhatunk VRML képekkel. A 11. fejezetben a Web programozásának egy másik eszközével, a CGI *(Common Gateway Interface)* script fájlokkal ismerkedünk meg. A CGI segítségével interaktív űrlapokat hozhatunk létre Web-helyeken. Mielőtt azonban rátérnénk erre a fejezetre, győződjünk meg arról, hogy értjük az alábbiakat:

- Egy VRML dokumentum ASCII dokumentum, amelynek segítségével 3-D képeket és 3-D szövegeket állíthatunk elő a Weben.
- A VRML csomópontok azok az objektumok, amelyek segítségével felépítjük a jelenetsorokat.
- A VRML mezők két osztályba sorolhatók: az egyik osztályba tartozók egyetlen értéket, míg a másik osztályba tartozók több értéket tartalmazhatnak.
- A Group csomópontok megváltoztatják az állapot öröklését, amikor visszaállítjuk a jelenetsorokat.
- VRML dokumentumok megjelenítéséhez különleges böngészőre vagy segédprogramokra (ún. „beépülő" programokra) van szükség – ugyanakkor egy VRML böngésző nem képes HTML dokumentumok megjelenítésére.
- VRML csomópontok segítségével egyszerűen hozhatunk létre különböző alakzatokat (testeket), amelyeket belevehetünk a virtuális világba.
- A VRML anyag, kamera, fény és transzformációs objektumai lehetővé teszik, hogy rendkívül rugalmasan kezeljük és befolyásoljuk az egyszerű geometriai csomópontok helyét és megjelenését.

VRML NYELVVEL ÉS DOKUMENTUMOKKAL FOGLALKOZÓ FONTOSABB HELYEK

A következő Web-helyek segíthetnek abban, hogy részletesen olvashassunk a VRML programozásról, valamint a VRML fejlesztőeszközeiről és erőforrásairól. A helyeket kiindulási pontként is használhatjuk a Web felfedezéséhez.

The VRML Repository

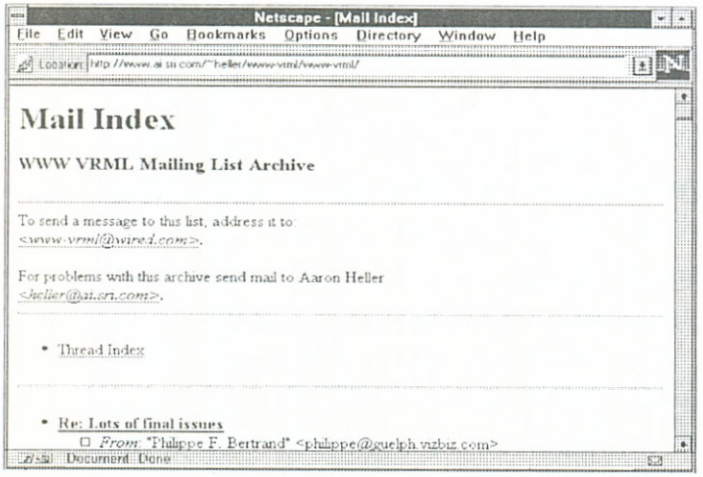


http://www.sdsc.edu/vrml/

UK VR-SIG 3D Object Archive – VRML

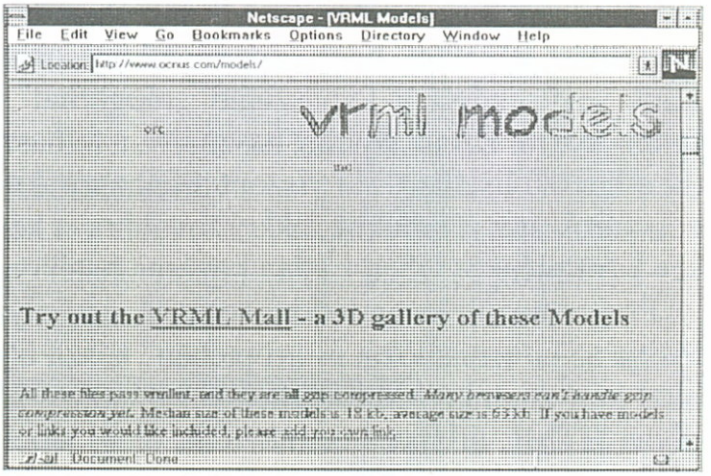
http://www.dcs.ed.ac.uk/generated/package-links/objects/vrml.html

Mail Index



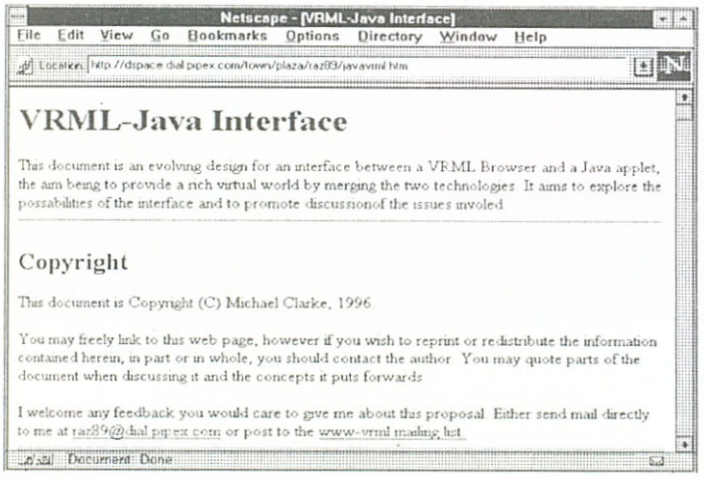
http://www.ai.sri.com/~heller/www-vrml/www-vrml/

VRML Models



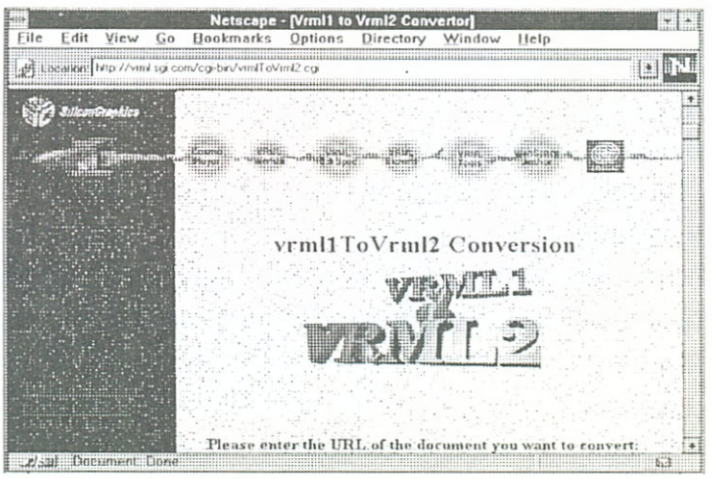
http://www.ocnus.com/models/

VRML-Java Interface



http://dspace.dial.pipex.com/town/plaza/raz89/javavrml.htm

Vrml1 to Vrml2 Convertor



http://vrml.sgi.com/cgi-bin/vrmlToVrml2.cgi

11. FEJEZET

CGI PROGRAMOZÁS

A 10. fejezetben láttuk, hogy miként használhatjuk a VRML nyelvet arra, hogy 3-D jelenetsorokat hozzunk létre a Weben. Nem nehéz elképzelni, hogy a Weben szörfözők mindig valami újdonságot keresnek, és VRML jelenetek megalkotásával nagyon vonzó helyek alakíthatók ki. Ebben a fejezetben megtanuljuk, hogyan használhatjuk a CGI (Common Gateway Interface) eszközt olyan Web-helyek készítéséhez, amelyek interaktív kapcsolatot teremthetnek a felhasználóval. Tegyük fel például, hogy meg szeretnénk jeleníteni a cégünk kínálatában lévő számítógép-alkatrészek árait. Feltételezve, hogy cégünk több száz, vagy akár több ezer különböző alkatrészt forgalmaz, valószínűleg nem az lehet a célunk, hogy egy véget nem érő Web-oldalon felsoroljuk a teljes listát és árjegyzéket, hosszadalmas keresgetésre kényszerítve ügyfeleinket.

Miután már ismerjük a HTML hiperhivatkozásokat, jobb (bár némileg továbbra is korlátozott) megoldást jelenthet, hogy hiperhivatkozásokat helyezünk el a statikus HTML dokumentumban. Ezek segítségével a felhasználó „ráugorhat" az alkatrészek különböző csoportjaira, például a monitorokra. Ezt az eljárást használva a felhasználó gyorsan megtalálhatja a kívánt termékcsoportot a hosszú listában. Ha viszont a felhasználót nemcsak a monitorok, hanem például a mikroprocesszorok árai is érdeklik, akkor továbbra is kénytelen több hely között „ugrálni", ami meglehetősen körülményes lehet, s talán kevésbé hatékony megoldás, mint amit versenytársunk kínál.

Programozók lévén tervbe vehetjük egy olyan program megírását, amely dinamikusan csoportosítja az információkat és „röptében" készít olyan Web-oldalakat, amelyek megjelenítik az ügyfél által kért adatokat. Másképpen fogalmazva: ha egy ügyfél felkeresi a Web-helyünket, akkor ott olyan HTML űrlapot jeleníthetünk meg, amelynek kitöltésével ő maga határozhatja meg, hogy mely alkatrészcsoportok érdeklik. Az ügyfél válaszának birtokában programunk aztán dinamikusan elkészíthet egy Web-oldalt, amelyen megjeleníti a kért információkat.

Ebben a fejezetben megismerkedünk azokkal a lépésekkel, amelyek segítségével dinamikus HTML dokumentumokat hozhatunk létre a felhasználó által megadott adatok alapján. Azt is megtanuljuk, hogy miként konfigurálhatjuk úgy a számítógépünket, hogy az Web-kiszolgálóként működjön. A létrehozásra kerülő programok saját számítógépünkön belül helyezkednek el és a felhasználók (magunkat is beleértve) képesek lesznek arra, hogy kapcsolatba lépjenek ezzel a kiszolgálóval (a kiszolgáló a *saját gépünkön* fut!), és lekérjék onnan az általunk készített Web-dokumentumokat.

Azon túlmenően, hogy a számítógépünket olyan Web-kiszolgálóvá alakítjuk, mint amilyen a világon meglévő bármelyik kiszolgáló, azt is megtanuljuk, hogyan készítsünk olyan, „méretre szabott" programokat, amelyek együttműködnek a Web-helyünket felkereső felhasználókkal. Egy böngésző segítségével a Weben keresztül kéréseket küldhetünk a kiszolgálónkra és hozzáférhetünk azokhoz a dokumentumokhoz, amelyeket magunk készítettünk, vagyis azokhoz, amelyek a saját gépünkön vannak. A programjaink válaszként viszont a kiszolgáló segítségével, ugyancsak a Weben keresztül, elküldhetik a böngészőnek a kért dokumentumokat. Mindez úgy fogható fel, mintha saját magunkat hívnánk fel telefonon, és „egymással" beszélgetnénk.

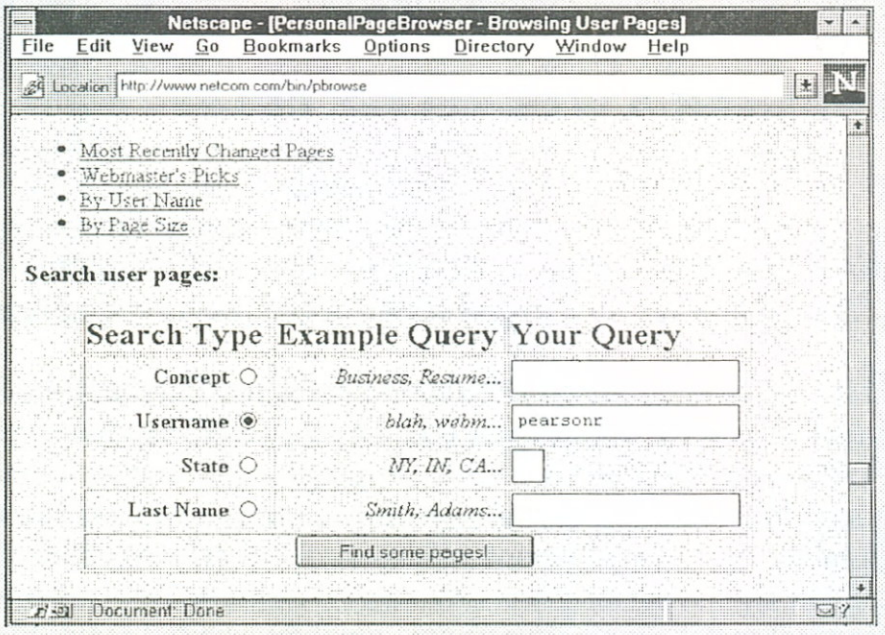
Arra is képesek leszünk, hogy egyidejűleg több felhasználó részére tegyük lehetővé a kiszolgálónkra való kapcsolódást és egyidejűleg többen hozzáférhessenek az általunk készített ugyanazon vagy különböző programokhoz és Web-oldalakhoz. Az ilyen műveleteket lehetővé tevő programozási eljárást CGI (Common Gateway Interface) programozásnak nevezik. A könyvből megtanulhatjuk, hogyan programozhatunk a CGI segítségével, és miképpen lehet belőlünk professzionális

CGI programozó. Ennek során nagyon sokat fogunk hasznosítani az eddigi fejezetekben tanultakból. A fejezet végére érve érteni fogjuk az alábbiakat:

- A CGI parancsállományok (script fájlok) célja, hogy a fejlesztők olyan, interaktív Web-helyeket hozhassanak létre, amelyek feldolgozzák a felhasználók által bevitt adatokat és ezeknek megfelelően válaszolnak.
- A Weben található szoftverek segítségével a számítógépes rendszerünket könnyen átalakíthatjuk Web-kiszolgálóvá.
- Miután telepítettünk a rendszerünkbe kiszolgálói szoftvert, a világ bármely részén lévő felhasználó hozzáférhet a HTML fájljainkhoz.
- Amikor egy felhasználó elküld egy kitöltött űrlapot, a kiszolgáló rendszerünkben lévő program veszi és feldolgozza az űrlap adatait.
- Az űrlap adatait fogadó és feldolgozó program dinamikusan tud létrehozni egy HTML dokumentumot, amelyet aztán a kiszolgáló válaszként megjelenít a felhasználónak.
- Azzal, hogy egy felhasználó adatokat küldhet, és hogy egy program ezen adatok alapján válaszként elküldhet egy HTML dokumentumot, a CGI parancsállományok (script fájlok) interaktív Web-helyek létrehozását teszik lehetővé.
- Olyan programozási nyelvek segítségével, mint a C/C++, olyan programokat írhatunk, amelyek válaszolni tudnak a felhasználó által kitöltött CGI űrlapokra és fel tudják dolgozni a felhasználótól kapott adatokat.

A CGI SCRIPT FÁJLOK – NAGY VONALAKBAN

Amikor szörfözünk a Weben, elkerülhetetlen, hogy ne forduljunk meg olyan helyeken, amelyek a 11.1. ábrán láthatóhoz hasonló űrlapokat tesznek elénk.



11.1. ábra. Kitöltésre váró űrlapot megjelenítő Web-hely

A Web-helyek ehhez hasonló űrlapokat használnak arra, hogy kölcsönös (interaktív) kapcsolatba lépjenek a felhasználóval. Amikor a felhasználó kitölti és elküldi az űrlapot, akkor a böngésző visszaküldi a felhasználó válaszát a kiszolgálónak, mely utóbbi elindít egy olyan programot (a kiszolgálón lévő programot), amely feldolgozza a felhasználó válaszát. A prog-

ram rendeltetésétől függően a program egy adatbázisban tárolhatja a felhasználó válaszait (ami például egy levelezői lista lehet), vagy valamilyen viszontválaszt generálhat.

Ahhoz, hogy válaszolni lehessen a felhasználónak, a programnak létre kell hoznia egy HTML dokumentumot, amit átad a kiszolgálónak, hogy az visszaküldhesse a böngészőnek megjelenítés céljából. Másként megközelítve ez azt jelenti, hogy a programnak olyan utasításokat kell tartalmaznia, amelyek segítségével dinamikusan (röptében) képes HTML bejegyzések létrehozására.

A CGI (Common Gateway Interface) az a szabvány, amely meghatározza azokat az adatformátumokat, amelyeket a böngészők, kiszolgálók és programok az ilyen információcserékhez használnak. Egy CGI parancsállomány (script fájl) olyan – majdnem bármilyen programozási nyelven (C/C++, Perl, TCL és mások) megírt – program, amely feldolgozza a felhasználó által bevitt adatokat, és – szükség esetén – ezek alapján elkészíti a választ (HTML dokumentum formájában).

A CGI SCRIPT FÁJLOK

Ahhoz, hogy a Web-helyek ténylegesen interaktívak legyenek, nem elegendő, ha a helyek csak azt teszik lehetővé, hogy a felhasználó különböző hiperhivatkozásokra kattintva különböző dokumentumokon haladjon végig. A Web-helyeknek azt is lehetővé kell tenniük, hogy valóságos információcsere jöjjön létre kiszolgáló és ügyfél között. CGI programok (más néven CGI parancsállományok vagy *script* fájlok) segítségével tartalomtól függő Web-oldalakat készíthetünk. Mint majd látni fogjuk, CGI script fájlok segítségével a Web-helyünk fogadhatja a felhasználó válaszait, és válaszolhat is azokra.

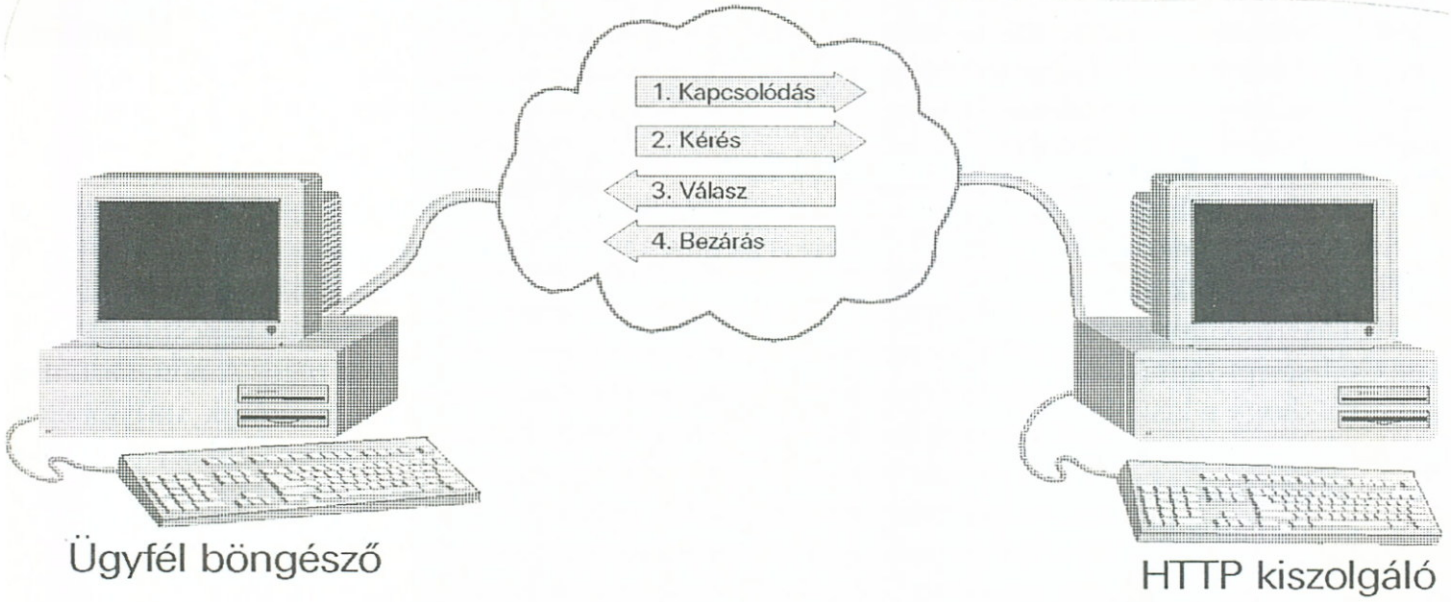
MIÉRT HASZNÁLNAK A WEB-HELYEK CGI-T?

A 6. fejezetben egy egyszerű Web-kiszolgálót készítettünk el. Ha ki akartuk volna bővíteni ezt a kiszolgálót, akkor olyan kódot is belevehettünk volna, amely lehetővé tette volna, hogy a kiszolgáló dinamikusan hozzon létre HTML dokumentumokat. Mivel a kiszolgálónak megvan az a képessége, hogy HTML fájlokat hozzon létre, joggal kérdezhetjük, miért van szükség CGI script fájlok írására.

Az igazsághoz tartozik, hogy nem kell feltétlenül CGI script fájlokat írnunk ahhoz, hogy dinamikus HTML fájlokat hozzunk létre. Egy ilyen script-nyelv nélkül azonban a programozónak minden olyan esetben módosítania kellene a kiszolgáló programját, amikor a helynek új, interaktív, dinamikusan létrehozott Web-oldalra van szüksége. Egy idő után emiatt a Web-kiszolgáló programja rendkívül nagyra nőne. Az ilyen folytonos módosítások elkerülése céljából a fejlesztők a CGI programozást használják. A CGI segítségével a kiszolgáló „albérletbe kiadhatja" a dinamikus Web-dokumentumok készítését annak az alkalmazásnak, amelyet az adott igény kielégítéséhez hozunk létre. Ezt az alkalmazást C, C++, Perl, JavaScript, VBScript vagy más programozási nyelven készíthetjük el.

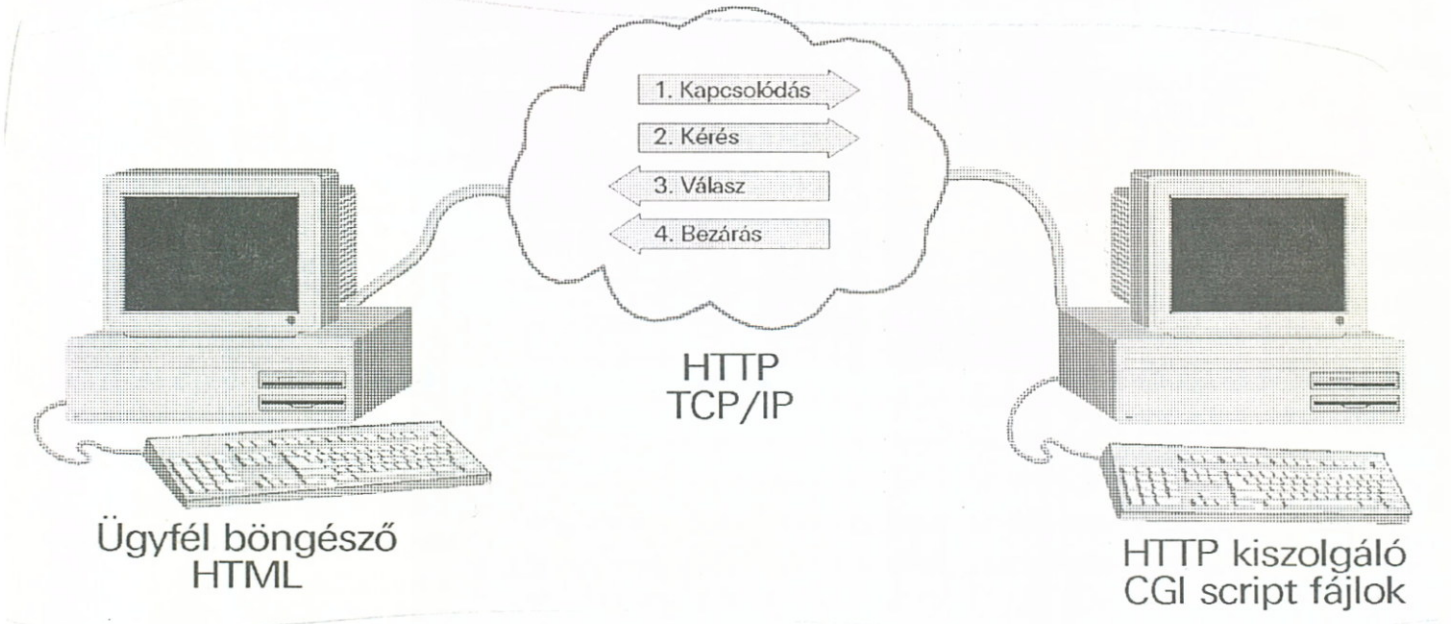
A CGI SZEREPE A HÁLÓZATBAN

Amint tudjuk, amikor egy böngésző egy kiszolgálóval kommunikál, akkor a programok négylépéses HTTP tranzakciót hajtanak végre a 11.2. ábrán látható módon.



11.2. ábra. A négylépéses tranzakció lebonyolításához szükséges szereplők

A 11.3. ábra azt szemlélteti, miként illeszkednek a CGI script fájlok ebbe a négylépéses tranzakcióba.



11.3. ábra. A CGI script fájlok helye a négylépéses HTTP tranzakcióban

Amint a 11.3. ábrán látható, a CGI script fájlok a kiszolgáló számítógépén találhatók, így a kiszolgáló és a programok közvetlenül kommunikálhatnak egymással. Ez a kommunikáció teszi lehetővé, hogy egy CGI script fájl dinamikusan kapjon adatokat, és továbbíthassa azokat a kiszolgáló felé.

A KISZOLGÁLÓ PROGRAMNAK MEG KELL HÍVNIA EGY CGI SCRIPT FÁJLT

CGI script fájlt közvetlenül a böngészőből nem hajthatunk végre. Ahhoz, hogy használhassunk egy CGI script fájlt, olyan számítógépen kell elhelyezni a fájlt, amelyen HTTP kiszolgáló program is van. Ahhoz, hogy egy böngészővel meg tudjuk nézni egy script fájl kimenetét, a kiszolgálónak

futtatnia kell parancsállományunkat. Ebből a könyvből megtudhatjuk, hogyan telepíthetünk és futtathatunk kiszolgáló programot a számítópépünk Windows 95 vagy Windows NT környezetében, hogy a script fájlokat közvetlenül a saját gépünkön hajthassuk végre.

A CGI PROGRAMOZÁS ÁTTEKINTÉSE

Amint tudjuk, ahhoz, hogy egy böngésző megjeleníthessen egy Web-dokumentumot, a böngésző első lépésként kapcsolatba lép a kiszolgálóval és kéri az illető dokumentumot (rendszerint a HTTP GET metódusának elküldésével). Ha a dokumentum létezik, akkor következő lépésként a kiszolgáló a HTML dokumentum elküldésével válaszol a böngésző kérésére, majd zárja a kapcsolatot. Amikor CGI script fájlt írunk, ez a folyamat csak a kiszolgáló oldalán változik meg. A böngésző nem tud arról, hogy a kiszolgáló meghív egy CGI parancsállományt, és azzal sem törődik, milyen útonmódon kapja meg a kért adatokat a kiszolgálótól. Ha a Web programozása során CGI script fájlokat írunk, akkor csak a kiszolgáló I/O műveleteivel kell foglalkoznunk. A böngészőnek csak az a dolga, hogy kapcsolatba lépjen azzal a kiszolgálóval, amelyik majd futtatja a script fájlt. A kapcsolat felvételét követően a script fájlunk elvégzi azokat a műveleteket, amelyek a kívánt kimenet létrehozásához szükségesek. Normál esetben a kiszolgáló bocsátja a script fájlunk kimenetét a böngésző rendelkezésére (egy HTML fájlon keresztül). Ehhez a kiszolgáló program hozzáadja a script fájlunk kimenetéhez (amennyiben van ilyen) a szükséges fejléc-információkat, és ezekkel együtt küldi vissza a script fájl adatait annak a böngészőnek, amelyik eredetileg hívta a kiszolgálót. Ezt követően a kiszolgáló bezárja a böngészővel fennálló kapcsolatát, és újabb kérésre vár.

Talán tudjuk, hogy a 32 bites operációs rendszeren (Windows 95 vagy Windows NT) futó kiszolgálók általában egyidejűleg több felhasználó kérésének kezelésére képesek (legalábbis ez a látszat). Ez azt is jelenti, hogy több felhasználó egyidejűleg használhatja a script fájlunkat anélkül, hogy ehhez speciális kódot kellene írnunk. A fejezet későbbi részében meg is írunk egy script fájlt, amely egyidejűleg több kérés kezelésére képes, és mindegyik felhasználó számára egyedi, dinamikus kimenetet tud létrehozni. Mindeközben a böngészőnek sejtelme sincs arról, hogy egy CGI script fájl fut, ami a kimenetet előállítja. A felhasználók ugyanakkor abból vehetik észre a script fájl közreműködését, hogy dinamikus kimenet jelenik meg a képernyőjükön.

A KISZOLGÁLÓ ÉS A CGI SCRIPT FÁJLOK KAPCSOLATA

Amikor a kiszolgáló program meghívja a CGI script fájlunkat, a kiszolgálónak néhány kulcsműveletet kell elvégeznie: meg kell hívnia a script fájlt és rendelkezésére kell bocsátania a böngészőtől kapott megfelelő adatokat, értékeket kell rendelnie azokhoz a *környezeti változókhoz*, amelyeket a script fájl el tud érni, kezelnie kell a script fájl kimenetét, továbbá ehhez hozzá kell fűznie azokat a fejléc-információkat, amelyekre a böngészőnek van szüksége a script fájl adatainak helyes értelmezéséhez.

Amint tudjuk, a HTTP az a protokoll, amelyet a Web ügyfelei és kiszolgálói az egymással való kommunikációhoz használnak. A HTTP fejléc-információi (ezeket a 3. fejezet mutatta be részletesen) segítik a programokat a hatékony kommunikációban, éppen ezért közelebbről is meg kell vizsgálunk azokat a fejléc-információkat, amelyeket egy kiszolgáló bocsát egy böngésző rendelkezésére. Amikor például egy kiszolgáló program kész arra, hogy adatokat küldjön egy böngészőnek, a kiszolgáló program egy sor fejléc-információt küld, amelyek leírják az adatok állapotát, típusát (ami a fájl tartalmát jelenti) stb. Ezen információk birtokában azután a böngésző a *Content-Type* (tartalom típusa) fejléc adatai alapján már fel tud készülni arra, hogy megjelenítse az ezután következő adatokat. Ezeknek a metainformációknak az elküldéséért a kiszolgáló felelős minden olyan alkalommal, amikor adatokat küld a böngészőnek.

A CGI ÉS AZ ADATBÁZISOK

A felhasználók gyakran használnak CGI script fájlokat adatbázisokban való kereséshez. Mivel az adatbázisok elképesztő mennyiségű adatot tartalmazhatnak, a programozók olyan script fájlokat írnak, amelyek nemcsak a felhasználóval, hanem az adatbázissal is kapcsolatot tartanak. Mint majd látni fogjuk, az ügyfél és a kiszolgáló közötti tranzakció során gyakran fordul elő, hogy a felek dinamikus módon adatokat cserélnek egymással. Ezért tudnunk kell, hogy készíthetünk olyan CGI script fájlokat, amelyek adatbázisokhoz továbbítanak, és onnan lekérnek információkat. Mielőtt azonban CGI adatbázis-programokkal foglalkoznánk (a 13. fejezetben), meg kell ismerkednünk azokkal az alapvető programozási eljárásokkal, amelyek professzionális CGI programok írásához szükségesek.

A SCRIPT FÁJLOK HELYE

A 6. fejezetben elkészítettünk egy kiszolgáló programot. Ezt a programot tovább bővítve megadhatjuk azokat a könyvtárakat, amelyekben a kiszolgáló elérheti a script fájlokat. A CGI szabvány egyébként nem írja elő, hogy hol kell tárolnunk a script fájlokat (a meghajtót és a könyvtárat). Normál esetben a Web-kiszolgálók azt várják el, hogy a CGI script fájlok egy */cgi-bin* nevű alkönyvtárban legyenek, közvetlenül az alatt a könyvtár alatt, amelyben maga a kiszolgáló található. Ha az általunk készített script fájlokat olyan HTTP kiszolgálóra telepítjük, amelyet valaki más üzemeltet, akkor nekünk kell megmondanunk, hogy melyik könyvtárba kerüljenek a fájlok.

A SCRIPT FÁJLOK NEVÉNEK KITERJESZTÉSE

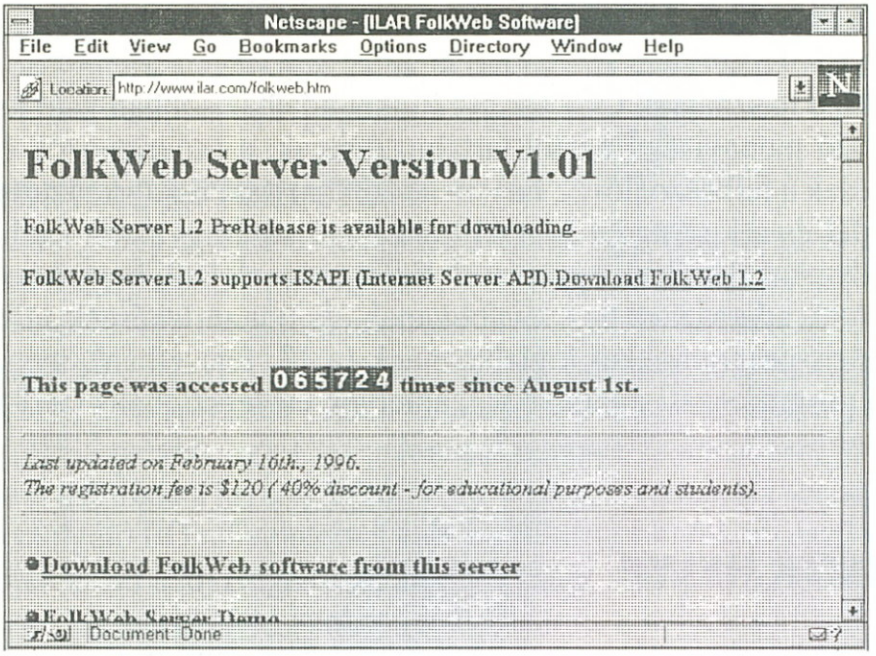
A Windows alapú HTTP kiszolgálók általában az EXE vagy a PL kiterjesztést használják a CGI fájlok nevében. Ha a C programozási nyelv segítségével készítjük el a script fájlt, akkor annak feltehetően EXE lesz a kiterjesztése, míg a Perl nyelvet használva PL lesz a script fájl nevének kiterjesztése. A Perl CGI script fájljaival a 12. fejezetben foglalkozunk részletesen.

Egyes kiszolgálók viszont a CGI kiterjesztést várják a script fájloktól. Számos rendszer a CGI kiterjesztést építi bele a kiszolgáló konfigurációs fájljába. Ha nem vagyunk biztosak abban, hogy egy adott kiszolgáló milyen CGI script fájlnév-kiterjesztéseket támogat, akkor kérdezzük meg a kiszolgáló üzemeltetőjét (az úgynevezett Webmestert). Ha nem igazodunk a beállításokhoz, akkor előfordulhat, hogy a kiszolgáló nem tudja meghívni a script fájljainkat.

A SZÁMÍTÓGÉP KONFIGURÁLÁSA WEB-KISZOLGÁLÓKÉNT

A CGI script fájlok teszteléséhez szükséges, hogy egy HTTP kiszolgáló program fusson ugyanazon a számítógépen, mint amelyiken a script fájljaink is találhatók. Mivel a legtöbb felhasználó nem fér hozzá egy kiszolgálóhoz (hacsak nem ez a tényleges munkája), meglehetősen nehézkes lehet a CGI script fájljaink tesztelése. Ha viszont a Windows 95 vagy a Windows NT operációs rendszer fut a számítógépünkön, akkor letölthetjük és telepíthetjük a *FolkWeb* Web-kiszolgáló programot a következő címről: *http://www.ilar.com/folkweb.htm* (lásd a 11.4. ábrát).

Miután telepítettük a *FolkWeb* kiszolgálót, létrehozhatjuk Web-oldalainkat (annyit, amennyi csak elfér a merevlemezünkön), és a Weben keresztül a világ bármely pontjára „szolgáltathatjuk" ezeket. Sőt még CGI script fájlokat is készíthetünk, amelyek együtt futnak ezzel a kiszolgálóval. Saját kiszolgáló futtatásában mindazonáltal az a legjobb, hogy a dologhoz semmiféle különleges szolgáltatást sem kell igénybe vennünk Internet-szolgáltatónktól (nincsenek járulékos költségek). Még állandó IP címre sincs szükségünk.



11.4. ábra. A *FolkWeb* kiszolgáló letöltése a *http:/www.ilar.com/folkweb.htm* címről

Megjegyzés: A **FolkWeb** *nem ingyenes. Ha megtartjuk a FolkWeb WWW Server programot, akkor fizetnünk kell érte. A licencmegállapodás lehetővé teszi, hogy kipróbáljuk a programot, mielőtt a megvásárlása mellett döntenénk. A licenccel kapcsolatos információk a FolkWeb kiszolgáló docs alkönyvtárában lévő LICENSE.DOC fájlban olvashatók el.*

SAJÁT IP CÍM MEGÁLLAPÍTÁSA

Ahhoz, hogy kapcsolatot teremthessünk a kiszolgáló programunkkal (a *FolkWeb* kiszolgálóval), ismernünk kell azt az IP címet, amelyet az Internet-szolgáltató által az aktuális kapcsolatban a számítógépünkhöz rendelt. Ha nincs állandó IP címünk, akkor valamilyen módon meg kell állapítanunk az adott kapcsolatban érvényes IP címünket (a tizedespontos írásmódú címet). A különböző Internet-szolgáltatóktól függően ehhez más és más lépéseket kell tennünk. Lépjünk kapcsolatba Internet-szolgáltatónkkal és kérdezzük meg, hogyan állapíthatjuk meg az egyes kapcsolatok során a számítógépünknek kiosztott IP címet.

KAPCSOLATFELVÉTEL A KISZOLGÁLÓVAL

Ha fel szeretnénk venni a kapcsolatot a *FolkWeb* kiszolgálóval, akkor a böngésző *Location* (Hely) mezőjébe írjuk be a *http://xxx.xxx.xxx.xxx* címet (ahol az *xxx.xxx.xxx.xxx* a számítógépünk decimális számjegyekből és pontokból álló IP címe), és nyomjuk le az Enter billentyűt. Miután a számítógépünkön fut a kiszolgáló, tudnunk kell, hogyan írhatunk olyan programokat, amelyek kommunikálhatnak ezzel a kiszolgálóval. Azt is tudnunk kell, hogyan írhatunk CGI script fájlokat. Mielőtt azonban rátérnénk ennek ismertetésére, célszerű, ha néhány egyszerű HTML fájl segítségével teszteljük a kiszolgáló telepítését. A teszteléshez készítsünk egy vagy több HTML dokumentumot (a 4. és az 5. fejezetben leírtak szerint), és a böngésző segítségével próbáljuk meg lekérni ezeket a kiszolgálótól. Miután sikerült hozzáférnünk és megjelenítenünk a HTML fájlokat, folytathatjuk a CGI-vel való ismerkedést a következő oldalakon.

AZ ÁLLANDÓ IP CÍMRŐL

Normál esetben, amikor a modemünkön és egy Internet-szolgáltatón keresztül összekapcsolódunk az Internettel, akkor egy véletlenszerű, 32 bites IP (Internet Protocol) címet kapunk a szolgáltatótól xxx.xxx.xxx.xxx alakban, ahol az *x* egy számjegyet jelent, és mindegyik xxx csoport egyetlen bájtnak (8 bitnek) felel meg. Az IP címeknek ezt az alakját az angol szaknyelv *dotted-decimal jelölésnek* nevezi, ami magyarul decimális számjegyekből és pontokból álló jelölésmódot jelent. Így a négy xxx csoport mindegyikének a maximális értéke 255, a minimális értéke pedig 0 lehet. Ebből következően az IP címek tartománya a 000.000.000.000 alsó határtól (amit általában 0.0.0.0 alakban írnak) a 255.255.255.255 felső határig terjed. Ez a 32 bites IP cím azonban nem teljesen véletlen szám, mert megadja az Internetre kapcsolódó adott hálózat méretét, a hálózati illesztőkártyák maximális számát.

A legtöbb esetben nincs befolyásunk arra, hogy milyen IP címet rendel Internet-szolgáltatónk a számítógépünkhöz. A szolgáltató az IP címet általában egy közös *készletből* jelöli ki. Másként fogalmazva, valahányszor kapcsolatba lépünk az Internet-szolgáltatónkkal, gyakorlatilag mindig új IP címet kapunk. Ez egyúttal azt is jelenti, hogy amikor elindítjuk a kiszolgáló programot, akkor a kiszolgálónk mindig más és más címen lesz megtalálható. Ha rendszeresen (a nap mind a 24 órájában) tervezzük a kiszolgáló működtetését, akkor minden bizonnyal állandó IP címre lesz szükségünk, hogy a felhasználók minden alkalommal ugyanazon a címen találják meg a kiszolgálónkat, amikor csak kapcsolatba akarnak lépni vele. Ha viszont a Web-kiszolgáló programot csak tesztelési célokra akarjuk használni, vagy kísérletezgetni szeretnénk vele, hogy eldöntsük, tényleg részt akarunk-e venni kiszolgálóként a Web közösségében, akkor fogadjuk el azt a körülményt, hogy valahányszor kapcsolatba lépünk az Internet-szolgáltatóval, különböző IP címeket fogunk kapni.

A WEB-KISZOLGÁLÓK ÉS A CGI SCRIPT FÁJLOK KÖZÖTTI EGYÜTTMŰKÖDÉS

A Common Gateway Interface specifikációja meghatározza azokat az információkat, amelyeket egy kiszolgálónak egy CGI script fájl rendelkezésére kell bocsátania, valamint azokat az információkat is, amelyeket válaszul a kiszolgáló igényel a script fájltól. Ahhoz, hogy a CGI script fájljaink megfeleljenek a CGI specifikációnak (és hibátlanul együtt tudjanak működni HTTP kiszolgálókkal), tartanunk kell magunkat a CGI specifikációban meghatározott követelményekhez.

Amikor a Web valamely használója olyan hivatkozásra kattint, amelynek következtében a kiszolgáló meghív egy CGI script fájlt, a kiszolgálónak egy sor környezeti változót kell beállítania. Ezeket a környezeti változókat a script fájl a bemeneteként használja. A környezeti változók információkat tartalmaznak a kérést küldő böngészőről, a kérést kezelő kiszolgálóról, valamint azokról az adatokról (ha vannak ilyenek), amelyeket megkap a script fájl. A CGI környezeti változóinak neve „érzékeny" a kis- és a nagybetűs írásmódra. A fejezet következő részében az ezekre a környezeti változókra vonatkozó CGI specifikációt ismertetjük.

A CGI KÖRNYEZETI VÁLTOZÓI

A HTTP kiszolgáló parancssori argumentumokon és környezeti változókon keresztül adja át az információkat egy CGI script fájlnak. A kiszolgáló a script fájl végrehajtásakor rendel értékeket a környezeti változókhoz. A környezeti változók jobb megértéséhez kísérletezzünk a következőkben leírásra kerülő CGI környezeti változókkal. A fejezet későbbi részében majd megírunk egy CGI

script fájlt, amely megjeleníti a kiszolgálónk környezeti változóihoz tartozó értékeket. Egy script fájlon belül a környezeti változók értékeit karakterláncokként kezelhetjük.

AZ AUTH_TYPE VÁLTOZÓ

A CGI script fájlok az AUTH_TYPE környezeti változót használják arra, hogy hozzáférési joggal ruházzák fel a script fájlt elérni kívánó felhasználót. Ha a kiszolgáló úgy van beállítva, hogy támogatja a felhasználók hozzáférési jogosultságát (a *FolkWeb* kiszolgáló beállítható úgy, hogy támogassa az AUTH_TYPE változót), akkor a script fájlt elérni kívánó felhasználónak meg kell adnia felhasználói nevét és jelszavát. A környezeti változó alábbi beállítása például azt írja elő, hogy a felhasználónak alapszinten kell azonosítania magát:

```
AUTH_TYPE = Basic
```

A CONTENT_LENGTH VÁLTOZÓ

A CGI script fájlok a CONTENT_LENGTH környezeti változót használják arra, hogy meghatározzák a csatolt adatokban lévő bájtok pontos számát. Ha például a kérés 1024 bájt hosszúságú dokumentumot tartalmaz, akkor a környezeti változónak az alábbi lesz a tartalma:

```
CONTENT_LENGTH = 1024
```

A CONTENT_TYPE VÁLTOZÓ

A CGI script fájlok a CONTENT_TYPE környezeti változót használják olyan kéréseknél, amelyek csatolt információkat tartalmaznak (Ilyen kérést valósít meg például a HTTP POST metódusa). A CONTENT_TYPE változó tartalma határozza meg a csatolt dokumentum médiatípusát (MIME típust/altípust). Ha például HTML dokumentum van a kéréshez csatolva, akkor a környezeti változónak az alábbi lesz az értéke:

```
CONTENT_TYPE = text/html
```

A GATEWAY_INTERFACE VÁLTOZÓ

A CGI script fájlok a GATEWAY_INTERFACE környezeti változót használják arra, hogy meghatározzák a CGI specifikációnak azt a verziószámát, amelynek a Web-kiszolgáló eleget tesz. A CGI specifikáció verziószámának formátuma: CGI/verziószám. Ha adott esetben a CGI verziószáma 1.1, akkor a környezeti változónak az alábbi lesz az értéke:

```
GATEWAY_INTERFACE = CGI/1.1
```

Megjegyzés: A SERVER_SOFTWARE, SERVER_NAME és a GATEWAY_INTERFACE környezeti változók nem kérésfüggőek, és minden kérésnél értéket kapnak. A többi környezeti változó a kéréstől függ, és az átjáró (gateway) rendel értékeket hozzájuk.

A PATH_INFO VÁLTOZÓ

A CGI script fájlok a PATH_INFO környezeti változót használják arra, hogy megállapítsák az ügyfél által megadott, és az elérési útra vonatkozó kiegészítő információt. Másként fogalmazva egy kiszolgáló úgy férhet hozzá egy script fájlhoz, hogy annak virtuális elérési útjához hozzáfűzi az ebben a változóban megadott kiegészítő információt. Ha ilyen információ érkezik az ügyféltől,

akkor a kiszolgálónak dekódolnia kell az információt, mielőtt továbbküldené a script fájlhoz. Az elérési útra vonatkozó ilyen kiegészítő információk általában egy olyan erőforrást jelölnek meg, amelyet a script fájlnak a munkája végeztével vissza kell küldenie.

Az elérési úttal kapcsolatos információk általában relatív utat adnak meg, amely arra az útra vonatkozik, amelyik a kiszolgáló gyökérkönyvtárához vezet. Vagyis a kiszolgáló gyökérkönyvtára jelenti azt a kiindulást, amihez a PATH_INFO környezeti változóban megkapott relatív elérési utat hozzá kell fűzni. Ha például a teljes útvonal a *c:/cgi-bin/pelda1.exe/sport.html,* akkor a környezeti változó az alábbit tartalmazza:

```
PATH_INFO = /sport.html
```

A PATH_TRANSLATED VÁLTOZÓ

A CGI script fájlok a PATH_TRANSLATED környezeti változót arra használják, hogy a végső, használható alakjában kapják meg a PATH_INFO változóban lévő információt. A kiszolgáló a szükséges virtuális/fizikai leképezések segítségével lefordítja a PATH_INFO változóban lévő információt. Ha például a PATH_INFO változó a */sport/html* értéket tartalmazza, és a kiszolgáló gyökérkönyvtára a *c:*, akkor a környezeti változó tartalma az alábbi lesz:

```
PATH_TRANSLATED = c:/sport.html
```

A QUERY_STRING VÁLTOZÓ

A CGI script fájlok a QUERY_STRING környezeti változót arra használják, hogy (argumentumokból álló) szövegalapú információt fogadjanak, amely a felhasználó által a script fájl futtatásához megadott URL-ben egy kérdőjel (?) után szerepel. A karakterlánc a script fájl bemenetét képezi. A kiszolgáló a karakterláncban lévő szóköz karaktereket plusz (+) jelre cseréli, a nem nyomtatható karaktereket pedig *%dd*-re, ahol a *d* egy hexadecimális számjegyet jelent.

A script fájlnak tartalmaznia kell egy olyan kódot, amellyel dekódolja ezt a karakterláncot. A kiszolgálónak semmilyen módon sem szabad dekódolnia ezt az információt, amikor átadja a script fájlnak. A kiszolgálónak kell beállítania a QUERY_STRING változót, ha a felhasználó ilyen kérésű információt küldött a számára. Ha például adva van a *http://www.jamsa.com./cgi-bin/grandma.exe?name=margaret+alarcon* URL, akkor a környezeti változó az alábbi értéket fogja tartalmazni:

```
QUERY_STRING = name=margaret+alarcon
```

A REMOTE_ADDR VÁLTOZÓ

A CGI script fájlok a REMOTE_ADDR környezeti változót használják arra, hogy megkapják a kérést küldő távoli gazdagép (a böngésző) IP címét. A környezeti változónak például ez lehet az értéke:

```
REMOTE_ADDR = 204.212.52.209
```

A REMOTE_HOST VÁLTOZÓ

A CGI script fájlok a REMOTE_HOST környezeti változót használják arra, hogy megkapják a kérést küldő gazdagép nevét. Ha a kiszolgáló nem ismeri a kérő gazdagép nevét, akkor a kiszolgálónak be kell állítania a REMOTE_IDENT környezeti változót (lásd lejjebb), és nem rendelhet

értéket a REMOTE_HOST változóhoz. Ha például a távoli gazdagép neve *jamsa.com*, akkor a környezeti változó az alábbi értéket tartalmazza:

```
REMOTE_HOST = jamsa.com
```

A REMOTE_IDENT VÁLTOZÓ

A CGI script fájlok a REMOTE_HOST környezeti változót használják arra, hogy megkapják a kérést küldő távoli gazdagép nevét. A Web-kiszolgáló az a program, amelyik meghívja a script fájlokat. Ha a HTTP Web-kiszolgáló támogatja az RFC 931-es azonosító eljárást (Authentication Server Protocol), akkor a kiszolgáló a felhasználótól kapott nevet rendeli ehhez a változóhoz. A script fájlok ezt a változót csak bejelentkezési, naplózási célokra használhatják. Ha például a távoli gazdagép neve *jamsa.com*, és az itt lévő egyik felhasználó neve *pschmauder*, akkor a REMOTE_IDENT környezeti változónak az alábbi lesz a tartalma:

```
REMOTE_IDENT = pschmauder.www.jamsa.com
```

A REMOTE_USER VÁLTOZÓ

A CGI script fájlok a REMOTE_USER környezeti változót használják arra, hogy megkapják a távoli felhasználó nevét. Ha a kiszolgáló támogatja a felhasználók azonosítását, és a script fájl védett, akkor a kiszolgáló felhatalmazza a távoli felhasználó nevét, és hozzárendeli ehhez a változóhoz. Ha például a távoli felhasználó neve *pschamuder*, akkor a környezeti változó az alábbi értéket fogja tartalmazni:

```
REMOTE_USER = pschmauder
```

A REQUEST_METHOD VÁLTOZÓ

A CGI script fájlok a REQUEST_METHOD környezeti változót használják arra, hogy meghatározzák annak a HTTP kérésnek a típusát, amelyet a böngésző a script fájl meghívása céljából a kiszolgálóhoz küldött (a kérés például GET, HEAD vagy POST lehet). Ha például a böngésző GET metódust küldött, akkor a környezeti változó az alábbi értéket tartalmazza:

```
REQUEST_METHOD=GET
```

A SCRIPT_NAME VÁLTOZÓ

A CGI script fájlok a SCRIPT_NAME környezeti változót használják arra, hogy meghatározzák azt a virtuális elérési utat, amely a kiszolgáló által futtatandó script fájlhoz vezet. Ha például adott a *http://www.jamsa.com/cgi-bin/vmiprog.exe* URL, akkor a SCRIPT_NAME környezeti változó az alábbi értéket fogja tartalmazni:

```
SCRIPT_NAME = cgi-bin/vmiprog.exe
```

A SERVER_NAME VÁLTOZÓ

A CGI script fájlok a SERVER_NAME környezeti változót használják arra, hogy megállapítsák a Web-kiszolgáló gazdanevét, tartománynevét vagy IP címét. A változó értékét a kiszolgáló állítja be. Ha például a kiszolgáló egy tizedes jegyekből és pontokból álló IP címet küld vissza, akkor a környezeti változó értéke például az alábbi lehet:

```
SERVER_NAME = 204.212.52.209
```

A SERVER_PORT VÁLTOZÓ

A CGI script fájlok a SERVER_PORT környezeti változót használják arra, hogy megállapítsák azt a portszámot, amelyet az ügyfél (a böngésző) használt a Web-kiszolgálóval való kapcsolatfelvételkor. Ha az alapértelmezés szerinti HTTP portot használta, akkor a változó értéke 80. Ha ezen „jól ismert" HTTP portcímtől eltérő számot használt, például a *http://www.jamsa.com:3000* megadásával, akkor a környezeti változó az alábbi értéket fogja tartalmazni:

```
SERVER_PORT = 3000
```

A SERVER_PROTOCOL VÁLTOZÓ

A CGI script fájlok a SERVER_PROTOCOL környezeti változót használják arra, hogy megadják a Web-kiszolgálóhoz kérést küldő ügyfél (böngésző) által használt protokoll nevét és verziószámát. A SERVER_PROTOCOL környezeti változó tartalmát megvizsgálva a script fájl azonosítani tudja annak a protokollnak a nevét és verziószámát, amelyet használnia kell, amikor adatokat küld a kiszolgálóra. A protokoll nevének és verziószámának formátuma: protokoll/verziószám. Ha például a használandó protokoll a HTTP 1.1, akkor a környezeti változó az alábbi értéket fogja tartalmazni:

```
SERVER_PROTOCOL = HTTP/1.1
```

A SERVER_SOFTWARE VÁLTOZÓ

Amint tudjuk, a kiszolgáló futtatja a CGI script fájlt. Mivel egy script fájl különbözőképpen futhat különböző kiszolgálókon, a CGI script fájlok a SERVER_SOFTWARE környezeti változót használják arra, hogy megállapítsák egy Web-kiszolgálóprogram nevét és verziószámát. A Web-kiszolgáló a következő formátumban adja át a CGI script fájlnak a nevét és verziószámát: név/verzió. Ha például a *FolkWeb* kiszolgáló 1.01-es verzióját használjuk, akkor a környezeti változó az alábbi tartalmú lesz:

```
SERVER_SOFTWARE = FolkWeb/1.01 (Windows-32 bit)
```

EGYÉB KÖRNYEZETI VÁLTOZÓK

Az eddig tárgyalt környezeti változókon túlmenően a kiszolgáló további környezeti változókba helyezi el azokat az adatokat, amelyeket az ügyféltől a kérési fejléc soraiban kapott meg. A kiszolgáló az értékeket olyan változókhoz rendeli, amelyek neve a HTTP_ előtaggal kezdődik, és utána a fejléc neve áll. A kiszolgáló a fejlécnevekben előforduló valamennyi kötőjelet (-) az aláhúzás karakterre (_) cseréli. A kiszolgáló továbbá elhagyhat bármely olyan fejlécet, amelyet már feldolgozott az olyan környezeti változókat használva, mint például az AUTH_TYPE, a CONTENT_TYPE és a CONTENT_LENGTH.

A HTTP_ACCEPT VÁLTOZÓ

A CGI script fájlok a HTTP_ACCEPT környezeti változót használják arra, hogy a böngésző által a kiszolgálónak küldött HTTP fejlécek alapján megállapítsák, milyen MIME típusok fogadására képes a böngésző. Amint tudjuk, egy MIME típus egy típust és egy altípust határoz meg. Többféle MIME típus esetén vesszővel kell elválasztani egymástól a típusokat. A környezeti változó például az alábbi értéket tartalmazhatja:

```
HTTP_ACCEPT = audio/aif, text/html, text/plain
```

A HTTP_USER_AGENT VÁLTOZÓ

A CGI script fájlok a HTTP_USER_AGENT környezeti változót használják arra, hogy megállapítsák a kiszolgálónak kérést küldő böngésző típusát. A környezeti változó például az alábbiakat tartalmazhatja:

```
HTTP_USER_AGENT = Mozilla/2.01 Gold(Win95;I)
```

Megjegyzés: A script fájlt meghívó kiszolgálóprogram más környezeti változókat is definiálhat. A FolkWeb kiszolgáló más környezeti változókat is támogat, mint például a HTTP_REFERER. Ez a változó azt az URL-t tartalmazza, amelyik a script fájlunkra „hivatkozik". Ha például a FolkWeb kiszolgáló fut a **204.212.52.209** *IP című számítógépünkön, és a script fájl által meghívott HTML fájl neve* **example.htm,** *akkor a HTTP_REFERER környezeti változó értéke a* **http://204.212.52.209/example.htm** *lehet. A kiszolgálóhoz tartozó dokumentációban nézhetjük meg, hogy az adott kiszolgáló milyen környezeti változókat támogat.*

A CGI PARANCSSORI BEÁLLÍTÁSAI

A CGI script fájlok általában *parancssori bemenetet* használnak arra, hogy végrehajtsanak egy ISINDEX lekérdezést, mely utóbbi lehetővé teszi, hogy interaktív kulcsszó-keresési képességekkel ruházzuk fel a HTML dokumentumokat. Az ISINDEX lekérdezést azonban nem mindegyik kiszolgáló támogatja. A böngésző elküldi a parancssori bemenetet a kiszolgálóra. A kiszolgáló úgy tudja azonosítani a parancssori bemenetet, hogy megnézi, HTTP GET metódust küldött-e a böngésző, és hogy az URI kereső karakterlánc tartalmaz-e *kódolatlan egyenlőség (=) kataktert.*

Ha a böngésző kérése HTTP GET metódus volt, és a böngésző által elküldött URI kereső karakterlánc nem tartalmaz *kódolatlan egyenlőség (=) kataktert*, akkor a kérés parancssori bemenetet használ. Mielőtt a kiszolgáló meghívná a megfelelő CGI script fájlt, a kiszolgálónak a plusz (+) jelet használva paraméterekre kell bontania a parancssori bemenetet. A kiszolgáló ezt követően még dekódolja (amennyiben szükséges) az URI kereső karakterláncban megkapott paramétereket, és a paraméter-karakterláncokat egy *argv* nevű tömbben tárolja.

A kiszolgáló a dekódolás során az egyes karakterláncokat szétválasztja az ampersand (&) karakternél. Következő lépésként a kiszolgáló tovább bontja ezeket a karakterláncokat úgy, hogy az egyenlőségjelnél elválasztja egymástól a változó nevét (ami az egyenlőségjel bal oldalán áll) és a változó értékét (ami az egyenlőségjel jobb oldalán áll). A kiszolgáló az *argv* tömbben lévő bejegyzések számát az *argc* nevű, egész számú változóban tárolja.

Ha a kiszolgáló egyenlőségjelet talál a QUERY_STRING környezeti változóban, akkor nem küld parancssori bemenetet a CGI script fájlnak. Továbbá, ha valamilyen oknál fogva a kiszolgáló nem tudja elküldeni az *argv* tömböt a CGI script fájlnak, akkor a QUERY_STRING környezeti változóban egy dekódolatlan lekérdező információt bocsát a rendelkezésére.

A STANDARD INPUT (STDIN)

Amikor egy böngésző POST kérést küld egy kiszolgálóra , akkor azok az információk, amelyeket a CGI script fájl megkap, a script fájl *stdin* (standard input) fájlleírójából érkeznek. A kiszolgáló beállítja a CGI script fájlnak a CONTENT_LENGTH környezeti változót, ami azoknak a bájtok-

nak a számát tartalmazza, amelyeket a kiszolgáló ezen a fájlleírón keresztül elküld. A CGI script fájl a CONTENT_LENGTH változó értéke alapján tudja megállapítani, hogy mennyi adatot kell feldolgoznia a *stdin* fájlleíróból. A kiszolgáló a CONTENT_TYPE környezeti változót is a CGI script fájl rendelkezésére bocsátja, aminek alapján a CGI script fájl eldöntheti, hogy miként kezelje az érkező adatokat. Ezen adatfolyam végén a kiszolgáló elküldhet (de nem kötelező) egy fájlvége jelet. Mindenesetre a script fájl feladata, hogy a CONTENT_LENGTH környezeti változó tartalma alapján meghatározza, mennyi adatot kell beolvasnia.

Ha például egy űrlap a HTTP POST metódusát használja (<FORM METHOD="POST">), és a kiszolgálóra küldött adat *name=alberta&husband=art* alakban van kódolva, akkor a kiszolgáló a CONTENT_LENGTH és a CONTENT_TYP környezeti változóhoz az alábbi értékeket rendeli:

```
CONTENT_LENGYTH = 24
CONTENT_TYPE = application/x-www-form-urlencoded
```

A STANDARD OUTPUT (STDOUT)

Miután a CGI script fájl végzett a kiszolgálóról a bemenetére került adatok feldolgozásával, vissza kell küldenie a kimenetét a kiszolgálóra. A CGI script fájlok ezt egyszerűen úgy végzik el, hogy a kimenő adatokat elküldik a *stdout* (standard output) fájlleíróra. A CGI script fájl által a kiszolgálóra visszaküldött adatok formátuma rendszerint HTTP válasz, amely fejlécből, az ezt követő üres sorból, majd a script fájl többi adatából áll. A script fájl kimenete tipikusan egy HTML dokumentum, amelyet maga a script fájl generál.

CGI KIMENET KÜLDÉSE KÖZVETLENÜL A BÖNGÉSZŐRE

Normál esetben a CGI script fájlok olyan kimenetet állítanak elő, amelyeket a kiszolgáló értelmez, és küld vissza a böngészőnek. Annak, hogy a CGI script fájl a kimenetét a kiszolgálón keresztül küldi vissza a böngészőnek, az az előnye, hogy a script fájlnak nem kell minden egyes kéréskor a teljes HTTP fejlécet elküldenie. Vannak ugyanakkor olyan script fájlok, amelyek kikerülik azt az utat, amelyen a kiszolgáló elemezné a fájl kimenetét, és közvetlenül a böngészőnek küldik vissza a kimenő adataikat. Azért, hogy azokat a script fájlokat, amelyek a kimenő adataikat közvetlenül a böngészőre küldik, meg lehessen különböztetni azoktól a fájloktól, amelyek a kimenő adataikat a kiszolgálóra küldik, a CGI protokoll előírja, hogy azon script fájlok neve, amelyek közvetlenül a böngészőre küldik az adataikat, az *nph-* betűkkel kezdődjön (Not to Parse the Header, magyarul: a fejlécet nem kell elemezni). Ha egy CGI script fájl neve az *nph-* betűkkel kezdődik, akkor a kiszolgáló nem elemzi ennek a fájlnak a fejlécét. Ilyen esetekben a script fájl a felelős azért, hogy érvényes HTTP választ küldjön vissza a böngészőre.

A CGI FEJLÉCEI

Mint már volt róla szó, egy CGI script fájl kimenete fejléccel kezdődik. Ez a fejléc szövegsorokból áll ugyanolyan formátumban, mint egy HTTP fejléc, és egy üres sor zárja le (olyan sor, ami csak a CRLF karakterpárt tartalmazza). Ha egy CGI script fájl olyan fejlécet küld ki a kimenetéről, amely nem *kiszolgálói direktíva*, akkor a kiszolgáló ezeket a fejléceket közvetlenül visszaküldi

a kérést kezdeményező böngészőre. A CGI specifikáció jelenleg három kiszolgálói direktívát definiál:

- Content-type
- Location
- Status

Egy CGI fejlécben a *Content-type* mező a script fájl által a böngészőre visszaküldendő adatok MIME típusát/altípusát adja meg. A CGI script fájl általában HTML dokumentumot küld vissza. Ebben az esetben egy CGI fejlécen belül a *Content-type* mező az alábbi értéket tartalmazza:

```
Content-type: text/html
```

Egy CGI fejlécben a *Location* mező egy dokumentumot ad meg. A script fájlok a *Location* mező segítségével hozzák az őket meghívó kiszolgáló tudomására, hogy nem magát a dokumentumot, hanem a dokumentumra való hivatkozást küldik. Ha a *Location* mező egy távoli URL-t tartalmaz (vagyis ha a dokumentum nem ugyanazon a számítógépen található, mint amelyiken a kiszolgáló), akkor a kiszolgáló átirányítja a böngészőt a megfelelő helyre. Ha a *Location* mezőben megadott érték virtuális elérési út (vagyis a fájl ugyanazon a számítógépen található, mint amelyiken a kiszolgáló), akkor a kiszolgáló visszaállítja a megadott dokumentumot, mintha a böngésző eredetileg is ezt a dokumentumot kérte volna. Távoli dokumentum megadásához a CGI fejlécen belüli *Location* mezőnek például ez lehet a tartalma:

```
Location: http://www.jamsa.com/
```

Egy CGI fejlécben a *Status* mező egy HTTP állapotértéket tartalmaz, amelyet a kiszolgáló a script fájltól a böngészőhöz továbbít. A 3. fejezetben részletesen olvashattunk a HTTP állapotkódjairól. A CGI script fájlt meghívó kiszolgáló nagyon sok HTTP állapotkódot használ. Gyakran – főként hibák előfordulásakor – felmerülhet azonban az az igény, hogy a script fájl közvetlenül a böngészőnek küldje el a HTTP állapotkódjait.

Az alábbi példa egy tipikus kimenetet szemléltet, amelyet egy CGI script fájlnak generálnia kell, és el kell küldenie a kiszolgálóra. Miután az adatok megérkeztek a kiszolgálóra, a kiszolgálónak a böngészőhöz kell továbbítania az adatokat.

```
Content-type: text/html <! Üres sor következik >

<HTML>
<HEAD>
<TITLE>Ez a cím</TITLE>
</HEAD>
<BODY>
Ez a CGI script fájlunk által generált törzs.
</BODY>
</HTML>
```

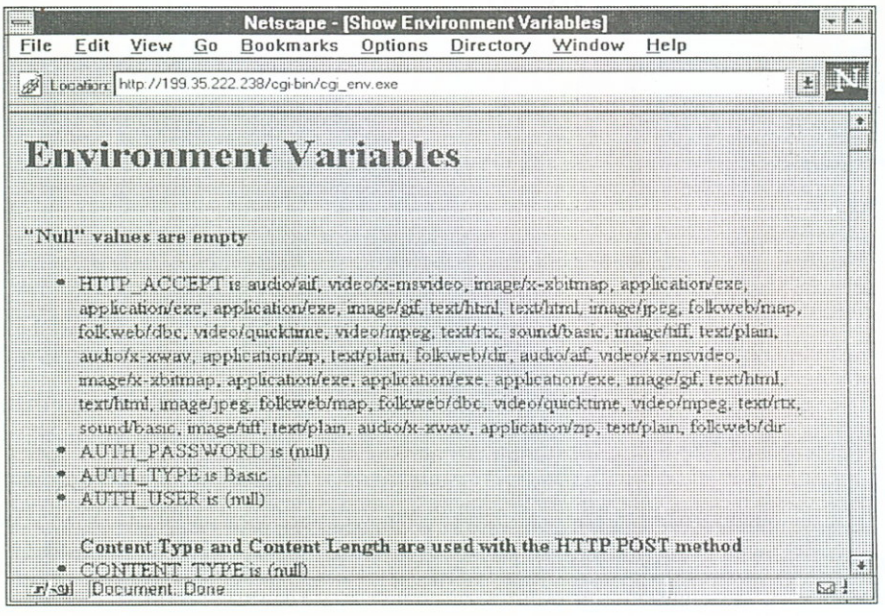
Vegyük észre a *Content-type* mező és az első <HTML> címke közötti üres sort! Erre az üres sorra feltétlenül szükség van. Az alábbi, C nyelvű programrészlet a *printf* függvény segítségével generálja az előző HTML dokumentumot:

```
// További kód az elõzõekben
printf("Content-type: text/html\n");
printf("\n"); // Ne feledjük beírni ide az üres sort
printf("<HTML>\n");
printf("<HEAD>\n");
printf("<TITLE>Ez a cím</TITLE>\n");
printf("</HEAD>\n");
printf("<BODY>\n");
printf("Ez a CGI script fájlunk által generált törzs.\n");
printf("</BODY>\n");
printf("</HTML>\n");
// További kód ezt követõen
```

AZ ELSŐ CGI SCRIPT FÁJL MEGÍRÁSA C NYELVEN

CGI parancsállományokat különböző programozási nyelveken írhatunk meg, így például C, C++, Perl és más nyelven. Mivel a C nyelvet igen széles körben használják, az első script fájlunk megírásához ezt választottuk.

Ennek a CGI_ENV.C script fájlnak az a feladata, hogy megjelenítse azoknak a környezeti változóknak az értékét, amelyeket a kiszolgáló a CGI script fájl rendelkezésére bocsát. A 11.5. ábra néhány olyan értéket sorol fel, amelyet a kiszolgáló rendel néhány környezeti változóhoz. Ha futtatjuk ezt a script fájlt, akkor a környezeti változók megjelenített értékei eltérhetnek az ábrán láthatóktól, mert más IP címet, böngészőt stb. használhatunk, mint amit a példa.



11.5. ábra. CGI környezeti változók megjelenítése C nyelvű script fájl futtatásával

Az alábbi, C nyelvű kód (ami megtalálható az első kötetben levő CD-ROM-on) a CGI_ENV.C script fájlt valósítja meg:

```
//
// A Web programozása, 11. fejezet: CGI_ENV.C
```

```
//
// CGI példa valamennyi környezeti változó megjelenítéséhez
//
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

struct Environment {
char *Auth_Password;
char *Auth_Type;
char *Auth_User;
char *Content_Length;
char *Content_Type;
char *Gateway_Interface;
char *HTTP_Accept;
char *HTTP_From;
char *HTTP_Referer;
char *HTTP_User_Agent;
char *Path_Info;
char *Path_Translated;
char *Query_String;
char *Remote_Addr;
char *Remote_Host;
char *Remote_Ident;
char *Remote_User;
char *Request_Method;
char *Server_Name;
char *Server_Port;
char *Server_Protocol;
char *Server_Software;
char *Script_Name;
char *Server_Admin;
};

void main(void)
  {
    struct Environment EV;

    EV.HTTP_Accept = getenv("HTTP_ACCEPT");
    EV.Auth_Password = getenv("AUTH_PASSWORD");
    EV.Auth_Type = getenv("AUTH_TYPE");
    EV.Auth_User = getenv("AUTH_USER");
    EV.Content_Length = getenv("CONTENT_LENGTH");
    EV.Content_Type = getenv("CONTENT_TYPE");
    EV.HTTP_From = getenv("HTTP_FROM");
    EV.Gateway_Interface = getenv("GATEWAY_INTERFACE");
    EV.Path_Info = getenv("PATH_INFO");
    EV.Path_Translated = getenv("PATH_TRANSLATED");
    EV.Query_String = getenv("QUERY_STRING");
```

```
    EV.HTTP_Referer = getenv("HTTP_REFERER");
    EV.HTTP_User_Agent = getenv("HTTP_USER_AGENT");
    EV.Remote_Addr = getenv("REMOTE_ADDR");
    EV.Remote_Host = getenv("REMOTE_HOST");
    EV.Remote_Ident = getenv("REMOTE_IDENT");
    EV.Remote_User = getenv("REMOTE_USER");
    EV.Request_Method = getenv("REQUEST_METHOD");
    EV.Script_Name = getenv("SCRIPT_NAME");
    EV.Server_Admin = getenv("SERVER_ADMIN");
    EV.Server_Name = getenv("SERVER_NAME");
    EV.Server_Port = getenv("SERVER_PORT");
    EV.Server_Protocol = getenv("SERVER_PROTOCOL");
    EV.Server_Software = getenv("SERVER_SOFTWARE");

// Környezeti változókat tartalmazó, dinamikus HTML
// fájl készítése

    printf("Content-type: text/html\n");
    printf("\n");
    printf ("<!DOCTYPE HTML PUBLIC \"-//W3O//DTD W3 HTML 3.2//EN\">\n");
    printf("<HTML>\n");
    printf("<HEAD><TITLE>Show Environment Variables</TITLE></HEAD>\n");
    printf("<BODY>\n");
    printf("<H1>Environment Variables</H1>\n");
    printf("<HR>\n");
    printf("<B>\"Null\" values are empty</B>\n");
    printf("<UL>\n");
    printf("<LI>HTTP_ACCEPT is %s\n", EV.HTTP_Accept);
    printf("<LI>AUTH_PASSWORD is %s\n", EV.Auth_Password);
    printf("<LI>AUTH_TYPE is %s\n", EV.Auth_Type);
    printf("<LI>AUTH_USER is %s\n", EV.Auth_User);
    printf("<P><B>Content Type and Content Length are used with the HTTP POST
        method</B>\n");
    printf("<LI>CONTENT_TYPE is %s\n", EV.Content_Type);
    printf("<LI>CONTENT_LENGTH is %s\n", EV.Content_Length);
    printf("<LI>HTTP_FROM is %s\n", EV.HTTP_From);
    printf("<LI>GATEWAY_INTERFACE is %s\n", EV.Gateway_Interface);
    printf("<LI>PATH_INFO is %s\n", EV.Path_Info);
    printf("<LI>PATH_TRANSLATED is %s\n", EV.Path_Translated);
    printf("<LI>QUERY_STRING is %s\n", EV.Query_String);
    printf("<LI>HTTP_REFERER is %s\n", EV.HTTP_Referer);
    printf("<LI>REMOTE_ADDR is %s\n", EV.Remote_Addr);
    printf("<LI>REMOTE_HOST is %s\n", EV.Remote_Host);
    printf("<LI>REMOTE_IDENT is %s\n", EV.Remote_Ident);
    printf("<LI>REQUEST_METHOD is %s\n", EV.Request_Method);
    printf("<LI>REMOTE_USER is %s\n", EV.Remote_User);
    printf("<LI>SCRIPT_NAME is %s\n", EV.Script_Name);
    printf("<LI>SERVER_NAME is %s\n",EV.Server_Name);
    printf("<LI>SERVER_PORT is %s\n", EV.Server_Port);
```

```
    printf("<LI>SERVER_PROTOCOL is %s\n", EV.Server_Protocol);
    printf("<LI>SERVER_SOFTWARE is %s\n", EV.Server_Software);
    printf("<LI>SERVER_ADMIN is %s\n", EV.Server_Admin);
    printf("<LI>HTTP_USER_AGENT is %s\n", EV.HTTP_User_Agent);
    printf("</UL>\n");
    printf("</BODY>\n");
    printf("</HTML>\n");
}
```

Figyeljük meg, hogy a program egyszerűen csak átveszi a script fájl környezeti változóit, és a *printf* függvény segítségével a *stdout* kimenetre küldve megjeleníti az értékeiket.

A CGI_ENV SCRIPT FÁJL

Amint látható, a program egy *Environment* nevű C szerkezetet használ arra, hogy tárolja a környezeti paraméterekre mutató mutatókat. A program a futásidejű függvénytár *getenv* függvényét használja arra, hogy egyenként beolvassa a környezeti változók értékeit, és egy EV nevű tömbben tárolja azokat.

Ezt követően a program a *Content-type:text/html* karakterláncot küldi az *stdout* fájlleíróra, ami után egy kocsi vissza–újsor karakter következik, ami a *Content-type* sor után kötelezően megadandó üres sort állítja elő. A CGI script fájlok nem működnének megfelelően, ha a *Content-Type* soruk után nem állna egy üres sor.

Következő lépésként a program a dokumentum típusdeklarációját küldi a *stdout* fájlleíróra, amivel arról tájékoztatja a kiszolgálót (valamint a böngészőt), hogy a HTML 3.2 verziónak megfelelő HTML dokumentumot készít. A továbbiakban a program szabványos HTML címkéket és elemeket küld a *stdout* fájlleíróra, melyek mindegyikét az illető környezeti változó értéke követ.

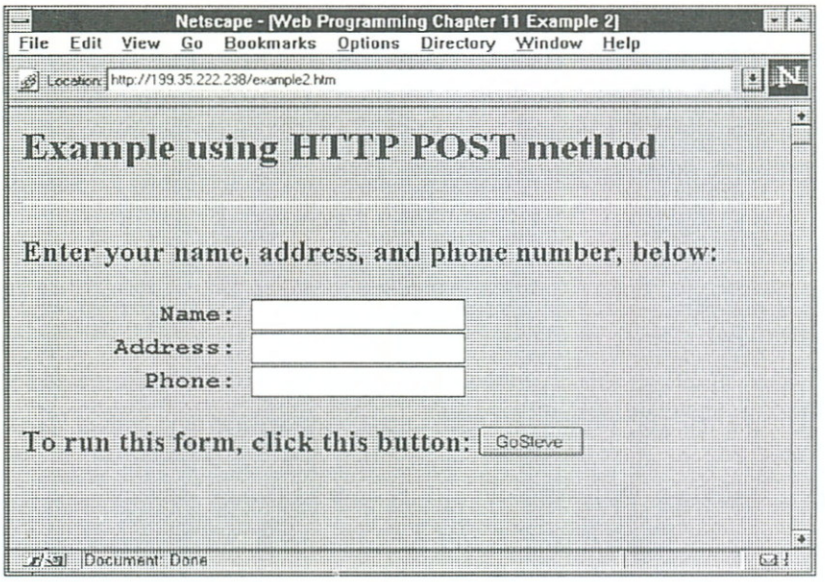
Amint látható, meglehetősen egyszerű egy C nyelvű CGI script fájl megírása. A következő két CGI script példafájl már kicsit bonyolultabb lesz. Ezeket a példákat sablonként használhatjuk valós alkalmazások megtervezéséhez.

Megjegyzés: A fenti példában a PATH_INFO, PATH_TRANSLATED és a QUERY_STRING környezeti változókhoz nem tartoznak értékek. Ez így helyes, mert nincs olyan kiegészítő elérési út, sem pedig lekérési adat, amit át kellene adni a CGI script fájl részére.

CGI SCRIPT FÁJL ÍRÁSA C++ NYELVEN

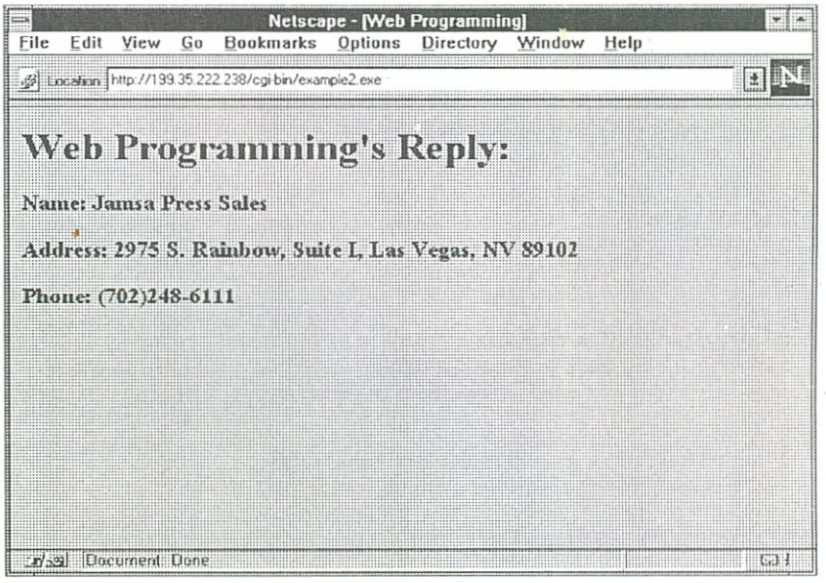
A C++ sokoldalú programozási nyelv, amelynek segítségével „újrahasznosítható" programokat készíthetünk. Mint látni fogjuk, egy C++ nyelvű CGI script fájl megírása némileg bonyolultabb, mint egy hagyományos C nyelvű programé. Miután azonban megtanulunk néhány programozási trükköt, képesek leszünk olyan kódok megírására, amelyeket valóban újra és újra használhatunk.

A létrehozásra kerülő, *CExample2.CPP* nevű, C++ nyelvű script fájl a HTTP POST metódus segítségével továbbítja a kiszolgálóra azokat az információkat, amelyeket a felhasználó egy űrlapba beír. A script fájl futtatásához a *CExample2.HTM* fájlban lévő HTML bejegyzéseket használjuk (a fájl az első kötetben levő CD-ROM-on található). Ekkor a képernyőn a 11.6. ábrán látható űrlap jelenik meg:



11.6. ábra. Űrlap, amely egy CGI script fájlt futtat

Amint látható, az űrlap három beviteli mezője üres. Töltsük ki adatokkal ezeket a mezőket, és kattintsunk a *GoSteve* gombra. Ezzel futtatjuk a C++ nyelvű CGI script fájlt, s a böngészőnk megjeleníti az általunk beírt adatokat például a 11.7. ábra szerint.



11.7. ábra. CGI script fájl segítségével a kiszolgálóhoz elküldött adatok megjelenítése

Mivel az űrlap a HTTP POST metódus segítségével küldi az adatokat a kiszolgálóra, ez a C++ nyelvű CGI script fájl a *stdin* fájlleírót használja a bemenet vételére. A következőkben bemutatjuk a program forráskódjának legfontosabb részeit. A program teljes forráskódja megtalálható az első kötetben levő CD-ROM-on.

Megjegyzés: Ennek a C++ nyelvű CGI script fájlnak a kifejlesztéséhez a Microsoft Visual C++ nyelv 4.0-s verzióját használtuk. A kód vázát az AppWizard segítségével készítettük el. A forrásprogramban úgy különböztettük meg a könyv által az AppWizard alapkódban végrehajtott kiegészítéseket vagy változtatásokat, hogy minden ilyen módosítás elé a „START CUSTOM CODE: Web Programming" (magyarul: EGYÉNI KÓD KEZDETE: A Web programozása), a végéhez pedig az „END MODIFICATIONS: Web Programming" (magyarul: MÓDOSÍTÁSOK VÉGE: A Web programozása) megjegyzést írtuk be. A forráskódban ezeket a megjegyzéseket keresve gyorsan megtaláljuk a módosításokat.

```
// example2.h : main header file
// . . . App Wizard által generált kód

// EGYÉNI KÓD KEZDETE: A Web programozása

class CExample2Doc;
class CMainFrame;
class CExample2View;

// MÓDOSÍTÁSOK VÉGE: A Web programozása

class CExample2App : public CWinApp
  {
    public:
        CExample2App();

        // EGYÉNI KÓD KEZDETE: A Web programozása

        CExample2Doc *m_pDoc;
        CMainFrame *m_pFrame;
        CExample2View *m_pView;
        CStringArray m_saParams;

        // MÓDOSÍTÁSOK VÉGE: A Web programozása

        // Felüldefiniálások
        // ClassWizard virtuális függvény-felüldefiniálások

        //{{AFX_VIRTUAL(CExample2App)

    public:
        // EGYÉNI KÓD KEZDETE: A Web programozása

        virtual BOOL InitInstance();
        virtual int ExitInstance();

        // MÓDOSÍTÁSOK VÉGE: A Web programozása

        //}}AFX_VIRTUAL

        // EGYÉNI KÓD KEZDETE: A Web programozása

        void ParseCmdLine(CString& sIniFile, CString& sContentFile, CString& sOutputFile);
        void ParseIniFile(CString sIniFile);
        void CreateResponse(CString sOutputFile, CString sIniFile);

// MÓDOSÍTÁSOK VÉGE: A Web programozása

// AppWizard által generált további kód
```

A fenti kódrészlet a *CExample2.h* header fájl leglényegesebb kódszegmenseit mutatja be. A listában néhány olyan változtatást láthatunk, amelyeket el kell végezni a fájlban. A figyelmünket a *document*, *frame* (keret), *view* (megjelenítés) és a *string* (karakterlánc) tömbmutatók deklarálására kell összpontosítanunk. A program az *m_pFrame* és az *m_pView* mutatókat használja a *CExample2.h* dokumentumobjektum, a megjelenítésablak és a keretablak eléréséhez. Mivel ebben a példában nincs megjelenítés vagy keret, a program egy „trükköt" használ a két ablak eltávolítására, amellyel a program a *CExample2.CPP* fájlon belül mindkét ablak méretét 0-ra állítja be. A program az *m_saParams* mutatót használja a standard I/O fájlleírókban lévő paraméterek eléréséhez.

A program felüldefiniálja a *CExample2.CPP* fájlon belüli virtuális InitInstance és *ExitInstance* függvényeket. A kód az InitInstance függvényen belül memóriát foglal le, ahol tárolni tudja a program ablakának beállításait. Ezen túlmenően a függvény elemzi a parancssort, valamint az INI fájlban lévő bejegyzéseket. A program az *ExitInstance* függvényen belül felszabadítja a lefoglalt memóriaterületet. A program három további függvényt is deklarál, amelyeket karaktermanipulációkhoz és I/O műveletekhez használ a *CExample2.h* header fájlban.

Mivel az űrlap HTTP POST metódust használ az adatoknak a kiszolgálóra való küldéséhez, a programnak a CGI bemeneti adatait a *stdin* fájlleíróról kell beolvasnia. Miután a program beolvasta az adatokat, a válaszát (az adatait) a *stdout* fájlleírón keresztül küldi a kiszolgálóra. Az alábbi kódrészlet a *CExample2.CPP* programfájl idevonatkozó részletét mutatja be:

```
//
// A Web programozása, 11. fejezet, 2. példa
//
// A HTTP POST metódus egyszerû ûrlapfeldolgozása
//

#include "stdafx.h"
#include "example2.h"
#include "mainfrm.h"
#include "example2view.h"

// Itt a Standard AppWizard kódja áll
// CExample2App construction

CExample2App::CExample2App()
{
    // EGYÉNI KÓD KEZDETE: A Web programozása

    m_pView = NULL;
    m_pFrame = NULL;

    // MÓDOSÍTÁSOK VÉGE: A Web programozása
}
```

Amint látható, a program a *CExample2.h* header fájlban deklarált két mutatóváltozóhoz a NULL kezdeti értéket rendeli. A program esetenként ezeket a mutatóváltozókat használja a megjelenítés és a keret ablak eléréséhez.

AZ INITINSTANCE FÜGGVÉNY

A program az *InitInstance* függvényen belül lefoglalja azt a memóriaterületet, amelyen a program ablakait tárolja (amelyek éppen nem jelennek meg). Ezen túlmenően a függvény elemzi a parancssort és az INI fájl bejegyzéseit. Az *InitInstance* függvényt az alábbi kód valósítja meg:

```
// A CExample2App egyetlen objektuma

CExample2App theApp;

////////////// CExample2App inicializálása

BOOL CExample2App::InitInstance()
  {
    LoadStdProfileSettings(); // standard INI fájlbeállítások betöltése

    // EGYÉNI KÓD KEZDETE: A Web programozása

    m_pFrame = new CMainFrame;
    m_pView = new CExample2View;

    CCreateContext newContext;
    newContext.m_pNewViewClass = NULL;
    newContext.m_pNewDocTemplate = NULL;
    newContext.m_pLastView = NULL;
    newContext.m_pCurrentFrame = NULL;
    newContext.m_pCurrentDoc = NULL;

    DWORD dwStyle = WS_OVERLAPPED;

// 0 méretû keret készítése, mivel ez
// háttérbeli folyamat

    if (!m_pFrame->Create(NULL, NULL, dwStyle, CRect(0,0,0,0),
        NULL, NULL, 0L, &newContext))
      {
        return FALSE;
      }

    m_pMainWnd = m_pFrame;

// 0 méretû megjelenítés készítése

    m_pView->Create(NULL,NULL,WS_CHILD,CRect(0,0,0,0),
        m_pFrame, AFX_IDW_PANE_FIRST,&newContext);

    m_pView->OnInitialUpdate();

    if (m_lpCmdLine[0] != '\0')
      {
```

```
        CString sIniFile;
        CString sContentFile;
        CString sOutputFile;

        ParseCmdLine(sIniFile, sContentFile, sOutputFile);

        ParseIniFile(sIniFile.GetBuffer(0));

        CreateResponse(sOutputFile, sIniFile);
      }

    PostQuitMessage(0);
    return TRUE;
  }
```

Amint látható, a program csak egyetlen alkalmazási objektumot hoz létre *(theApp)*. Továbbá a program felüldefiniálja a virtuális *InitInstance* függvényt, amelyen belül memóriát igényel a Heapből a keret és a megjelenítési ablak objektumokhoz. A program a *CCreateContext* struktúra segítségével hozza létre a dokumentum keret és megjelenítés ablakát, mert a program felüldefiniálja az ablakkészítési eljárást. Mint említettük, a program a keret és a megjelenítés ablak méretét 0-ra állítja (így elkerüli az ablakok megjelenését). A program továbbá létrehoz néhány *CString* változót, amelyeket bemeneti és kimeneti műveletekhez fog használni.

AZ EXITINSTANCE FÜGGVÉNY

A program az *ExitInstance* függvényen belül felszabadítja az előzőleg a megjelenítés és a keret ablak számára lefoglalt memóriaterületet. Az *ExitInstance* függvényt az alábbi kód valósítja meg:

```
int CExample2App::ExitInstance()
  {
    if (m_pView) // Clean up
        delete m_pView;

    if (m_pFrame)
        delete m_pFrame;

    return 0;
  }
```

Amint látható, a függvény egyszerűen felszabadítja azt a memóriaterületet, amelyet a program korábban lefoglalt a keret és a megjelenítés ablakhoz.

A PARSECMDLINE FÜGGVÉNY

A *ParseCmdLine* függvény az *nCmdPos* és az *nWordPos* nevű, integer típusú változót használja a parancssor elemzéséhez. Következő lépésben a függvény az *nCmdPos* változó segítségével lépked végig a parancssori karakterláncon, hogy megtalálja az egyes parancsok elejét és végét. A függvény az *nWordPos* változó segítségével tárolja a parancs kezdetét, majd az *OutputFile* változóban tárolja az egyes parancsokat. A függvény teljes kódja megtalálható az első kötetben levő CD-ROM-on.

A PARSEINIFILE FÜGGVÉNY

A program a *ParseIniFile* függvényt arra használja, hogy elemezze a *ParseCmdLine* függvény által létrehozott *sIniFile*-t. A függvény az *sIniFile*-ban talált minden egyes karakterláncot beírja az *m_saParams* változóba az *m_saParams.add(sLine)* függvény segítségével:

```
void CExample2App::ParseIniFile(CString sIniFile)
  {
    int nIniPos = 0;
    int nBufLen;
    CString sLine;
    char lpszBuf[2048];

    nBufLen = GetPrivateProfileSection("Form Literal",
     (LPSTR)lpszBuf,2048,sIniFile.GetBuffer(0));

    do {
     sLine = "";

     while (nIniPos < nBufLen && lpszBuf[nIniPos] != '\0')
      {
        sLine += lpszBuf[nIniPos];
        nIniPos++;
      }

     nIniPos++;

     if (sLine != "")
      {
        m_saParams.Add(sLine);
      }
  } while (sLine != "");
}
```

A CREATERESPONSE FÜGGVÉNY

A program a *CreateResponse* függvény segítségével generálja azt a dinamikus HTML dokumentumot, amelyet a kiszolgáló a felhasználó adatbeviteleinek megjelenítésére használ. Röviden fogalmazva a függvény visszhangozza a *stdin* fájlleírón keresztül megkapott adatokat. A függvény válasza tartalmazza a HTML 3.2 specifikáció által megkövetelt HTML információkat is:

```
void CExample2App::CreateResponse(CString sOutputFile, CString sIniFile)
  {
    int i;
    int nIndex;
    CString sResponse;
    CFile fOutput;
    CFileStatus fStatus;
    CString sCustomer;
    CString sKey;
```

```
    CString sVal;

// HTML formátum küldése
    sResponse = "Content-type: text/html\r\n";
    sResponse += "\r\n";
    sResponse += "<!DOCTYPE HTML PUBLIC \"-//W3O//DTD W3 HTML 3.0//EN\">\r\n";
    sResponse += "<HTML>\r\n";
    sResponse += "<HEAD>\r\n";
    sResponse += "<TITLE>Web Programming</TITLE>\r\n";
    sResponse += "</HEAD>\r\n";

    for (i = 0; i < m_saParams.GetSize(); i++)
      {
        sKey.Empty();
        sVal.Empty();

          if ((nIndex = m_saParams[i].Find('=')) != -1)
            {
              sKey = m_saParams[i].Left(nIndex);
              sVal = m_saParams[i].Right(m_saParams[i].GetLength() - nIndex - 1);
            }

          if (sVal == "")
            {
              sResponse += "<H1>Please enter all data</H1>\r\n";
                break;
            }

          sCustomer += "<H3>";
          sCustomer += sKey;
          sCustomer += ": ";
          sCustomer += sVal;
          sCustomer += "</H3>\r\n";
      }

    if (i == m_saParams.GetSize())
      {
        sResponse += "<H1>Web Programming's Reply:</H1>\r\n";
        sResponse += sCustomer;
      }

    sResponse += "</BODY>\r\n";
    sResponse += "</HTML>\r\n";

    if (fOutput.Open(sOutputFile.GetBuffer(0),
        CFile::modeCreate | CFile::modeWrite | CFile::typeBinary))
      {
        fOutput.Write(sResponse.GetBuffer(0),sResponse.GetLength());
          fOutput.Close();
```

```
        }
    }

// MÓDOSÍTÁSOK VÉGE: A Web programozása
```

Amint látható, a függvény az *sResponse* karakterláncban egyszerűen egymáshoz fűzi (konkatenálja) a válaszait, amit a program később visszaír a kiszolgálóra.

HASZNOS NYELVEK SCRIPT FÁJLOK ÍRÁSÁHOZ

CGI script fájlok írásához sokféle programozási nyelv közül választhatunk. Tulajdonképpen az is mondható, hogy szinte bármely olyan programozási nyelvet használhatjuk, amely képes olvasni a standard bemeneti eszközről és képes írni a standard kimeneti eszközre. Az itt használt C és C++ programozási nyelv mellett a programozók nagyon gyakran használják a Perl (Practical Extraction Report Language) programozási nyelvet is, amiről a 12. és a 13. fejezetben lesz szó, valamint a Visual Basic, a TCL, a FORTRAN, a JavaScript nyelveket vagy bármilyen UNIX héjat (shellt).

ALTERNATÍVÁK A SCRIPT FÁJLOKHOZ

Keresztül-kasul a Weben igen sok helyen használják a C/C++, Perl vagy egyéb programozási nyelven megírt script fájlokat. Azonban nem csak a script fájlok teszik lehetővé, hogy a Web-helyek interaktív kapcsolatokat tartsanak a felhasználókkal. A CGI script fájlok mellett újabban különböző helyek az úgynevezett *kiszolgáló oldali segítőket* (server-side assists, SSA) használják, amelyekhez maga a kiszolgáló dolgozza fel a felhasználó által bevitt adatokat, és készíti el a HTML válaszdokumentumot – kiküszöbölve ezzel azt a fázist, amelyben a kiszolgálónak egy másik programot kellett futtatnia, mint ahogyan ez a CGI esetén történik. Már arra is van lehetőség, hogy a kiszolgáló ne egy programot futtasson, hanem olyan API-kat (Application Programming Interface, alkalmazásprogramozói felület) használjon, amelyek már a bemenet feldolgozására és a HTML válasz generálására alkalmas függvényeket is tartalmazzák. Függetlenül azonban attól, hogy egy hely milyen eljárást használ, a folyamat ugyanaz. A felhasználó elküldi a kitöltött űrlapot, a kiszolgáló valamilyen szoftver segítségével feldolgozza az űrlap adatait és generál egy HTML választ.

Mindhárom említett eljárásnak megvannak az előnyei és hátrányai. Amint láttuk, a kiszolgáló nagyon összetett program. Ezért meglehetősen nehéz feladat lenne, ha egy kiszolgáló forráskódját minden olyan esetben módosítani kellene, amikor egy űrlap új feldolgozást igényel. Még rosszabb, hogy egy esetleges hiba egy ilyen kiszolgáló kódjában biztonsági problémákat vet fel a rendszerben. A kiszolgáló oldali segítők előnye viszont a sebesség. Az operációs rendszerek többségében az egyik legidőigényesebb feladat egy program futtatása. Ha engedélyeznénk, hogy a kiszolgáló válaszoljon a beküldött űrlapokra, akkor a kiszolgáló oldali segítők szükségtelenné tennék, hogy egy hely folyamatosan különböző programokat futtasson.

A CGI script fájlok előnye, hogy könnyen használhatók. Számos szakcikk egyenesen azt állítja, hogy a CGI script fájlok megírása olyannyira könnyű, hogy a programozásban járatlanok is nyugodtan vállalkozhatnak ilyen feladatokra. Nagyon egyszerű script fájlok esetén bizonyos pontig ez elfogadható, de a valamelyest bonyolultabb script fájlok elkészítéséhez már programozóra van szükség. A script fájlok hátránya, hogy a kiszolgálónak külső programokat kell futtatnia. Ha egy ilyen program hibás vagy szándékosan tartalmaz valamilyen „trójai falovat", akkor a program védtelenné teheti a rendszert a biztonságát fenyegető veszélyekkel szemben.

Végül az API-k, például az Microsoft Internet Server API (ISAPI) előnyéről annyit, hogy itt a szoftver elkülönül a kiszolgálótól, így könnyebben módosítható, ugyanakkor mégis biztonságosabb, mint egy külső, harmadik program. Ha nagyobb jártasságra teszünk szert a Web prog-

ramozásában, akkor részletesebben megismerkedhetünk a kiszolgáló oldali segítőkkel és a kiszolgáló API-kkal.

ÖSSZEFOGLALÁS

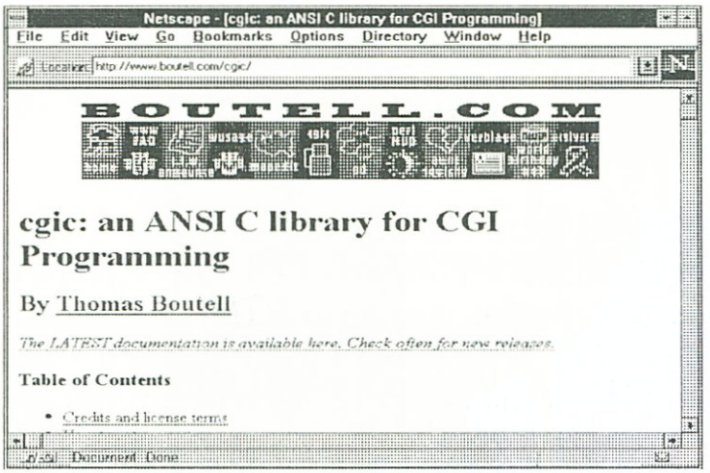
Ebben a fejezetben néhány nagyon fontos ismeretre tettünk szert. Már képesek vagyunk arra, hogy saját HTTP kiszolgálót futtassunk. Azt is megtanultuk, hogyan készíthetünk CGI script fájlokat C és C++ programozási nyelven. Azt is láttuk, hogyan készíthetünk dinamikus HTML dokumentumokat. A következő két fejezetben megismerkedünk a Perl programozási nyelvvel, majd azzal, hogyan írhatunk Perl nyelven script fájlokat. Mielőtt azonban rátérnénk a 12. fejezetre, győződjünk meg arról, hogy értjük az alábbiakat:

- A CGI script fájlok lehetővé teszik, hogy a fejlesztők interaktív Web-helyeket készítsenek.
- A programozók nagyon sokféle programozási nyelven, így például C/C++ és Perl nyelven is elkészíthetik a CGI script fájlokat.
- Amikor egy felhasználó elküld egy CGI űrlapot, a böngésző az űrlapot olyan programnak küldi el, amelyik feldolgozza az űrlap beviteli adatait.
- A CGI űrlap adatait feldolgozó program ugyanott található, ahol a kiszolgáló program.
- Az űrlap adatait feldolgozó program a bemenetét környezeti változókon és parancssori argumentumokon keresztül kapja meg, melyeket a kiszolgáló hoz létre, mielőtt futtatná a programot.
- Válaszul a felhasználó által elküldött űrlapra a program dinamikusan képes egy HTML dokumentum generálására, amit aztán a kiszolgáló elküld a felhasználónak.

CGI SCRIPT FÁJLOKKAL FOGLALKOZÓ FONTOSABB HELYEK

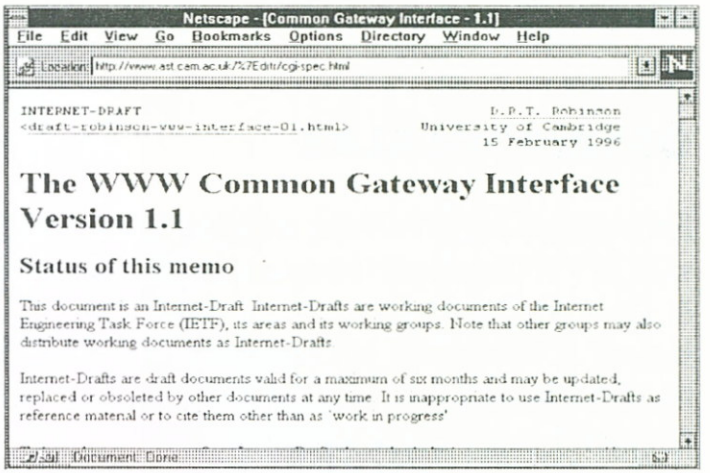
A következő Web-helyek segíthetnek abban, hogy részletesen olvashassunk a CGI programozásról, a környezeti változókkal kapcsolatos speciális tudnivalókról, valamint a biztonságot veszélyeztető kérdésekről. A helyeket kiindulási pontként is használhatjuk a Web felfedezéséhez.

cgic: an ANSI C library for CGI Programming



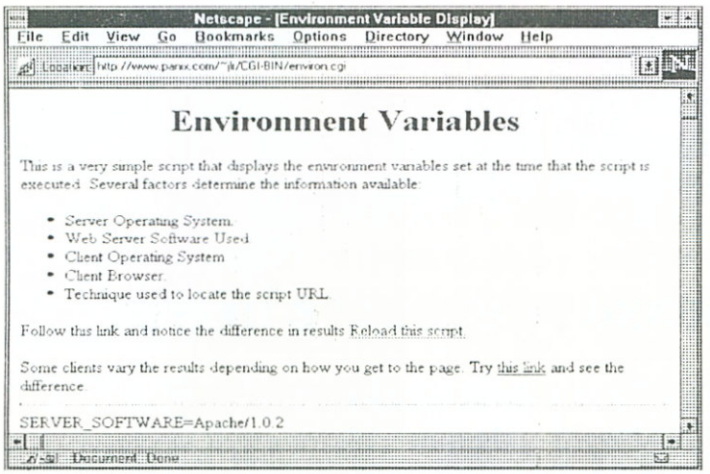
http://www.boutell.com/cgic/

Common Gateway Interface – 1.1



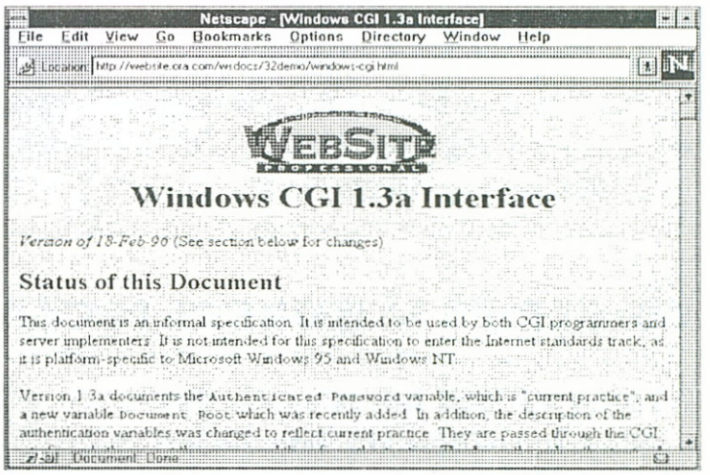
http://www.ast.cam.ac.uk/ %7Edrtr/ cgi-spec.html

Environment Variable Display



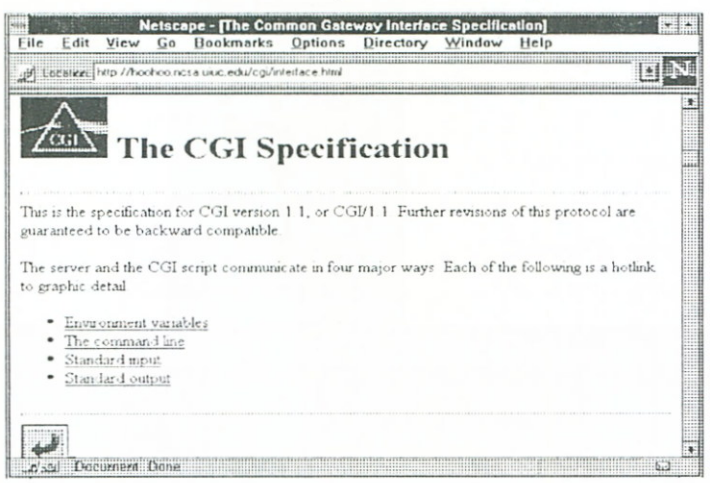
http://www.panix.com/~jlr/CGI-BIN/ environ.cgi

Windows CGI 1.3a Interface



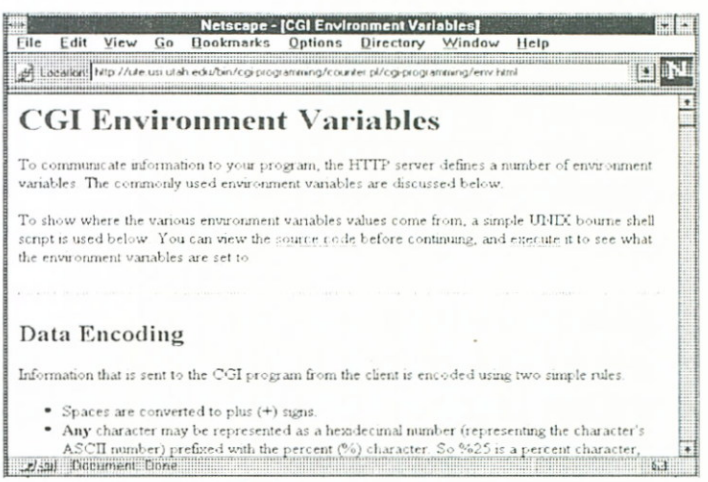
http://website.ora.com/wsdocs/32demo/ windows-cgi.html

The Common Gateway Interface Specification



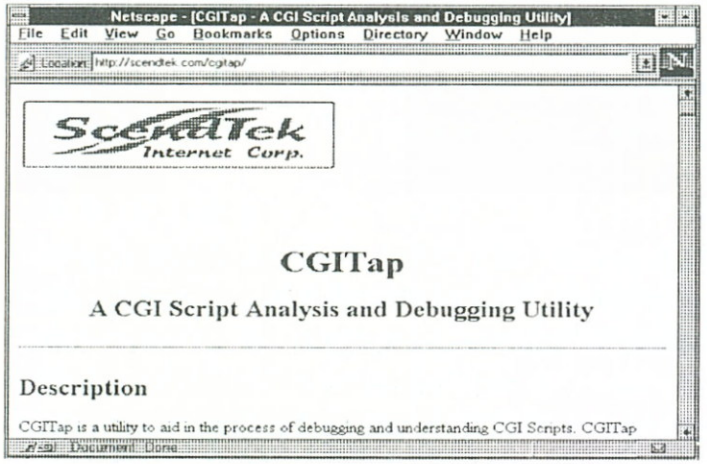
http://hoohoo.ncsa.uiuc.edu/cgi/ interface.html

CGI Environment Variables



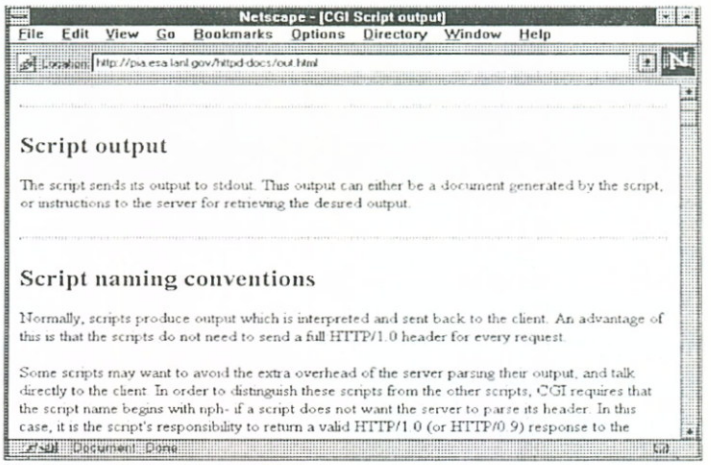
http://ute.usi.utah.edu/bin/cgi-programming/ counter.pl/cgi-programming/env.html

CGITap – CGI Script Analysis and Debugging Utility



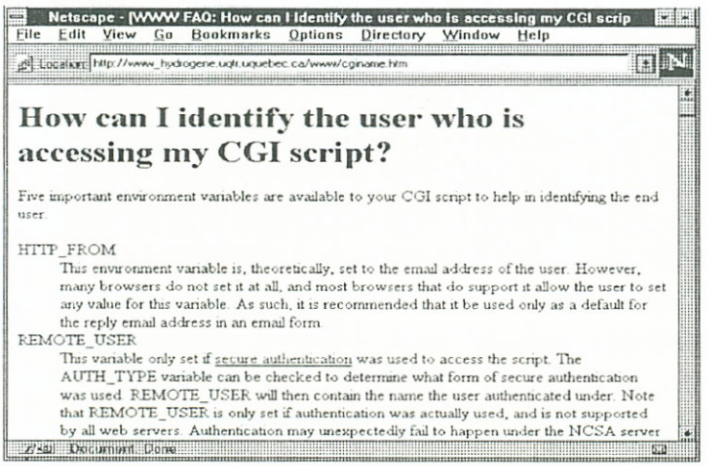
http://scendtek.com/cgitap/

CGI Output



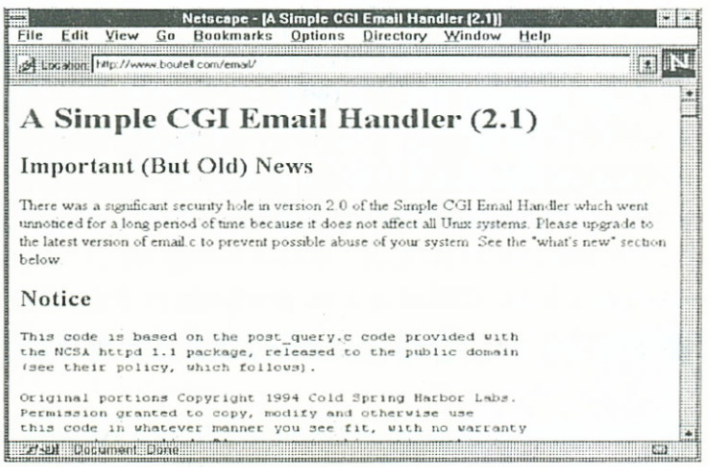
http://pia.esa.lanl.gov/httpd-docs/out.html

Identifying Remote User



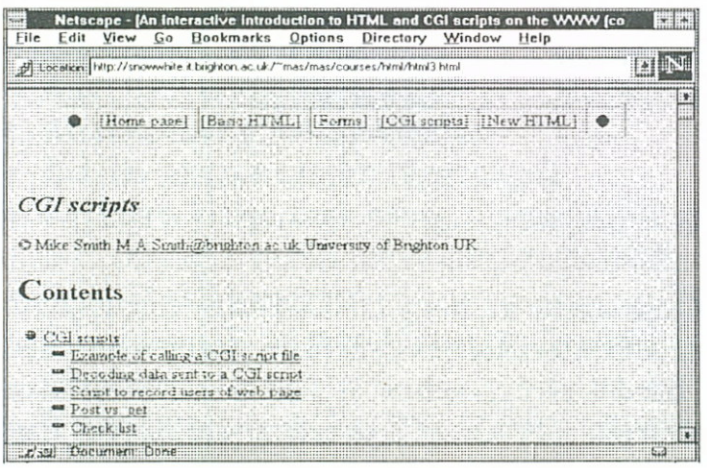
http://www_hydrogene.uqtr.uquebec.ca/www/cginame.htm

CGI E-mail Handler v2.1



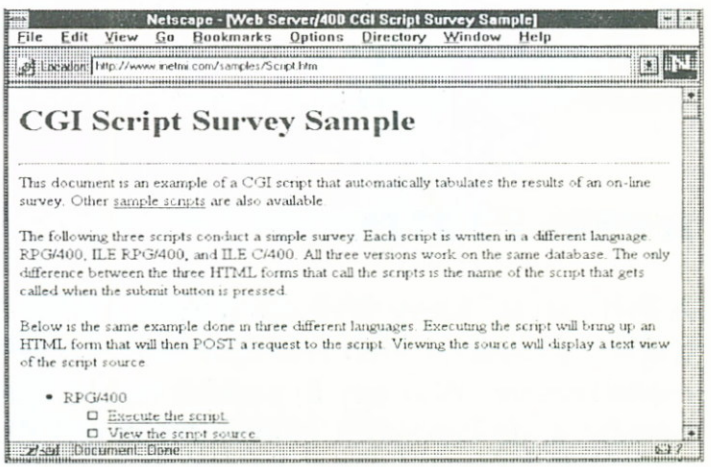
http://www.boutell.com/email/

Introduction to CGI



http://snowwhite.it.brighton.ac.uk/~mas/mas/courses/html/html3.html

CGI Script Survey



http://www.inetmi.com/samples/Script.htm

12. FEJEZET

A PERL PROGRAMOZÁSI NYELV

A 11. fejezetben láttuk, hogy miként írhatunk CGI script fájlokat C és C++ programozási nyelven. Script fájlok írásának legelterjedtebben használatos programozási nyelve azonban a Perl. Ebben a fejezetben ezzel a programozási nyelvvel foglalkozunk. Miután megismerkedtünk a Perl használatával, olyan script fájlokat írhatunk, amelyek révén számos, az Internettel és a Webbel kapcsolatos programozási feladatot oldhatunk meg. Ebben a fejezetben a nyelvnek azon jellemzőit hangsúlyozzuk ki, amelyeket minden bizonnyal használni fogunk a CGI script fájlok írása során. Aki korábban még sohasem programozott Perl nyelven, kiinduló pontként használhatja ezt a fejezetet. A fejezetből megtanulhatjuk, hogyan írjunk Perl nyelven professzionális script fájlokat. A fejezet a Perl nyelven történő programozásnak nemcsak az alapjait, hanem néhány haladó szintű felhasználási lehetőségét is bemutatja. A fejezet végére érve érteni fogjuk az alábbiakat:

- A Perl interpreter típusú programozási nyelv, amely különösen szövegek kezelésére alkalmas.
- A programozók a Perl nyelven megírt programokat gyakran Perl *scripteknek* nevezik.
- A programozók CGI űrlapok készítéséhez használják a Perl nyelvet.
- A Perl támogatja a fájl- és adatbázis-műveleteket, így alkalmas arra, hogy a Webet használók igényeinek megfelelően tárolja és állítsa vissza az információkat.
- A Perl tartalmaz egy hibakereső programot, amelynek segítségével tesztelhetjük Perl kódjainkat.

A PERL PROGRAMOZÁSI NYELV

A Perl (ami a Practical Extraction and Report Language elnevezés kezdőbetűiből álló mozaikszó) hordozható, interpreter típusú nyelv, amely ideálisan használható számos szövegfeldolgozó alkalmazáshoz. A Perl támogatja a magas szintű nyelvek többségében megtalálható strukturált programszerkezeteket, és egy sor olyan beépített képességgel rendelkezik, amelyek az elmúlt években a UNIX környezetben végbement fejlődése során alakultak ki.

A Perl továbbá ingyenes. Fizetség nélkül hozzáférhető, és ez a könyv is ingyen terjesztheti. A könyv első kötetében levő CD-ROM is tartalmazza a Perl 5-ös verzióját. Ebben a fejezetben azonban elsősorban a Perl 4-es verzióját vesszük szemügyre, amit a legszélesebb körben használnak a Weben. A 4-es verzió egyébként teljes mértékben kompatíbilis az 5-ös verzióval.

A PERL TÖRTÉNETE

A Perl „atyja" Larry Wall, aki a nyelvet 1986-ban fejlesztette ki, hogy UNIX környezetben jelentéseket generáljon több szöveges fájlból. Mivel az ehhez szükséges eszközök akkor nem álltak rendelkezésre, Wall egy új eszközt hozott létre a feladat elvégzéséhez. A Perl név eredete kissé homályos, de kezdetben minden bizonnyal Pearl lehetett a Practical Extraction and Report Language kezdőbetűi alapján.

Wall további képességekkel ruházta fel a Perl nyelvet, és biztosította, hogy a legszélesebb körben hozzáférhető legyen. A Perl népszerűsége azóta is folyamatosan nő, és számos programozó szerszámkészletének egyik favorit szerszáma lett. (Amerikában a Perl-programozásban való jártasság a szakmai önéletrajzokban is egyre fontosabb szempontként jön számításba.)

A PERL INTERPRETER TÍPUSÚ PROGRAMOZÁSI NYELV

A Perl interpreter típusú programozási nyelv, ami azt jelenti, hogy egy Perl nyelvű programot úgy futtathatunk, hogy meghívjuk a Perl *interpretert* (értelmező programot), és átadunk neki egy Perl parancsokból álló listát (a lista jelenti a Perl programot). Mivel az interpreter ilyen módon olvassa be és hajtja végre a Perl parancsokat, a Perl programokat *script* fájloknak (parancsfájloknak) is nevezik.

A UNIX világából jövőknek bizonyára ismerősek az olyan script fájlok, mint a *shell, sed* stb. Feltehetően értékelni is tudják egy nagy teljesítményű script-nyelv hasznosságát. A DOS és Windows környezetet használók a BAT (batch- vagy parancsfájlok) és a BASIC programok kapcsán találkozhattak interpreter típusú nyelvvel. Ha a DOS parancsfájljainak fényében tekintünk a script fájlokra, akkor szkeptikusak lehetünk a tekintetben, vajon megoldhatók-e a csak picit is bonyolultabb feladatok egy interpreter típusú nyelven. Ha előítéletektől mentesen tanulmányozzuk a fejezetet, akkor rövidesen értékelni tudjuk a Perl tudását.

A PERL ÉS A C/C++ PROGRAMOZÁSI NYELVEK ÖSSZEHASONLÍTÁSA

A Perl nyelvnek a C programozási nyelvhez nagyon hasonló a struktúrája, és egy Perl program első pillantásra úgy is néz ki, mint egy C program. A C valamennyi operátora (műveleti jele) megtalálható benne, és a C programvezérlő szerkezeteinek (mint például az *if* és a *for* utasítások) többsége is, bár egy kicsit módosított formában. A C nyelvvel szemben viszont a Perl nyelv nem tartalmaz mutatókat, struktúrákat és definiált típusokat. A C programozási nyelvnek természetesen megvan a maga létjogosultsága, de nem kell feltételeznünk, hogy a C program mindig jobb a vele azonos Perl programnál. Mint minden eszköznek, a C és a Perl nyelvnek is megvan az a területe, ahol a legjobban használható. Mindkét nyelvet alaposan meg kell ismernünk ahhoz, hogy eldönthessük, mikor használjuk az egyiket és mikor a másikat.

A PERL GAZDAG VÁLASZTÉKA

A programozók gyakran úgy emlegetik a Perl nyelvet, mint egy mindentudó „svájci zsebkést", és okkal. Az alábbiakban felsorolunk néhányat a Perl legfontosabb képességei közül. A fejezet későbbi részében mindegyik használatára kitérünk.

- Asszociatív tömbök.
- Automatikus típuskonverzió egész számok, lebegőpontos számok és karakterláncok között.
- Automatikus tömbátméretezés.
- Bináris adatkonverziós függvények.
- Reguláris kifejezések bőséges támogatása, amelyeket egy program keresési, helyettesítési vagy más szövegelemzési műveletekben használ.
- Fájl I/O műveletek.
- C-hez hasonló formázott kiviteli függvények, sablon alapú jelentésgeneráló képességekkel kiegészítve.
- C operátorok teljes készlete szövegösszehasonlító operátorokkal kibővítve.
- Listamanipuláló függvények, amelyek támogatják a vermeket, sorokat és más lista-adattípusokat.
- Rendszerszolgáltatási függvények.
- Számos utasítástípus és vezérlőszerkezet, a szubrutinokat is beleértve.

A PERL HASZNÁLATÁRÓL

A Perl használatának megértéséhez Perl kódokat kell tanulmányoznunk. A fejezet következő részében rövid példákon keresztül vizsgáljuk meg a Perl használatát. Miután elsajátítottuk a Perl programozási nyelv alapjait, nekifoghatunk Perl nyelvű CGI script fájlok megírásának.

A PERL MINT ADATSZŰRŐ

A UNIX segédeszközei sűrűn használják az adatszűrőket. Ezek olyan programok, amelyek megvizsgálják a beérkező adatáramot (például egy fájl tartalmát) és kiszűrik belőle a nemkívánatosakat. Az MS-DOS operációs rendszer szintén használ ilyenfajta adatszűrést. A UNIX *grep* segédeszköze klasszikus példa egy adatszűrőre. A *grep* segédeszköz egy adott szövegminta alapján keresi meg a beérkező szövegsorokban azokat a sorokat, amelyek azonosak a mintával. A program a mintával egyező sorokat átengedi a szűrőn a kimenő áramba. A nem egyező sorokat viszont kiszűri, mintegy „lenyeli".

A Perl ideális eszköz adatszűrők készítéséhez. A *grep* program leegyszerűsített változatát már ezzel a rövid Perl kóddal is megvalósíthatjuk:

```
$minta = shift(@ARGV); # parancssori minta beolvasása

while (<>)
  { # sorok beolvasása az érkező adatáramból
    print if (/$minta/); # egyező sor átengedése a kimenetre
  }
```

Ebben az esetben a script fájl egy ciklusban beolvassa a program (átirányított) bemenetét, és sorról sorra megvizsgálja, hogy az megegyezik-e a program által az első parancssori argumentumként megadott szöveggel. A script fájl tartalmával egyelőre ne törődjünk. A fejezet későbbi részében mindegyik utasításra részletesen kitérünk.

A PERL MINT BIZTONSÁGOS ÁTJÁRÓ

Más hálózati programozási feladatokhoz hasonlóan a CGI programozásban is kiemelt fontossága van a biztonságnak. Sokszor lehet szükség arra, hogy a fájljainkat és a rendszer más erőforrásait megvédjük a hálózat felelőtlen vagy rosszindulatú használóival szemben. Ez különösen fontos a Web-kiszolgálók (és más, például FTP kiszolgálók) esetében, amelyek az Internethez kapcsolódnak, ahol biztosra vehető a rosszindulatú felhasználók jelenléte. A rendszer ilyen támadásokkal szembeni megvédésének egyik módja, hogy az összes adatot átfolyatjuk egy biztonsági átjárón. Ilyen módon csak azok az adatok juthatnak be a rendszerünkbe, amelyeket az átjáró program „biztonságosnak" talált.

Hagyományosan nagyon sok Internet-kiszolgálót C nyelven írtak meg és UNIX környezetben futtattak (és futtatnak ma is). Míg egyrészről a C programozási nyelv rendkívül hatékony, addig másrészről ha egy programozó hibásan használja a C mutatóit, akkor könnyen írhat meghibásodásra hajlamos programot, ami komolyan veszélyeztetheti a biztonságot.

Biztonságos átjáró programok írásánál a Perl nyelv egyik előnye, hogy a karakterlánc típusú változók korlátozás nélkül, automatikusan a kívánt méretre nőnek, hogy tárolni tudják a script fájl által hozzájuk rendelt karaktereket. Perl nyelv használatakor a programnak nincs lehetősége úgy írni értéket egy változóba, hogy azzal tönkretegye egy másik változó értékét.

A Perl nyelvnek van egy különleges, *taintperl* nevű változata, amely megvizsgálja az adatok hovatartozását és megakadályozza, hogy bármely rendszerparancs olyan adatokat engedjen át a kiszolgálóra, amelyek nem megbízható forrásból származnak. A *taintperl* program minden

parancssori értéket, környezeti változót és bemenő adatot „fertőzöttként" jelöl meg, és végzetes hibajelzéssel leállítja az adatátvitelt, ha fertőzött érték kerül egy rendszerparancs hatókörébe.

A PERL HASZNÁLATA ADATBÁZIS-ELŐTÉTKÉNT

Egy adatbázis *előtétje* (frontend) olyan program, amely egyszerűbbé teszi más programok hozzáférését az adatbázis-kiszolgálóhoz. Az előtétprogram feldolgozza a felhasználó adatbázisra vonatkozó kérését és generálja azokat az adatbázis-lekérdezéseket, amelyek a kiszolgálón lévő adatok eléréséhez szükségesek. Az előtétprogram fel is dolgozhatja a lekérdezés eredményét, és jelentést készítve vissza is küldheti a felhasználónak.

Mint látni fogjuk, a programozók teljes mértékben Perl nyelven is készítenek egyszerű adatbázis-alkalmazásokat anélkül, hogy szükség lenne ehhez egy különálló adatbázis-kiszolgálóra. A Perl beépítve tartalmazza azt a képességet, amellyel egy *asszociatív tömb* (lásd később) egy adatbázis-fájlra leképezhető. Ennek köszönhetően egy Perl script fájlon belül egy adatbázis fájl elérése nem jelent nagyobb feladatot, mint a tömbelemek elérése, mert a fájl I/O műveletek láthatók a script fájl számára.

Fejlett adatbázis-alkalmazásoknál a Perl kapcsolódhat egy kereskedelmi adatbázis-kiszolgálóhoz és az adatbázis előtétjeként működhet. A Perl különböző speciális változatát készítették el, és látták el olyan bővítésekkel, amelyek adott adatbázis-kiszolgálókkal tudnak együttműködni. Az *oraperl* segítségével például Oracle adatbázis-kiszolgálók érhetők el.

A PERL MINT CGI SCRIPT FÁJLOK NYELVE

Amint az előző fejezetben láttuk, a CGI olyan Web-helyeket létesít, amelyek lehetővé teszik az ügyfélprogramok (tipikusan a böngészők) számára az interaktivitást. A Web-helyek számos esetben CGI script fájlokat használnak adatbázisok eléréséhez, amikor egy ügyfélnek és kiszolgálónak adatokat kell cserélnie egymással. CGI script fájl beiktatásával egy felhasználó hétköznapi Web-böngészővel is hozzáférhet egy adatbázishoz a Weben keresztül. A CGI script fájl beolvassa és feldolgozza a HTML űrlap tartalmát, hozzákapcsolódik az adatbázishoz, továbbítja a lekérdezést, a lekérdezés eredményét új HTML dokumentumként megformázza és visszaküldi ezt a HTML dokumentumot a felhasználónak. E lépések megtételekor természetesen messzemenően figyelembe kell venni a biztonsági követelményeket.

BEVEZETŐ A PERL NYELVBE

Az eddig leírtak remélhetőleg mindenkit meggyőztek arról, hogy CGI script fájlok írásakor legalább vegyék fontolóra a Perl nyelvet. A fejezet következő részében megtanuljuk, hogyan futtathatjuk a Perlt és miként készíthetünk Perl nyelven hasznos script fájlokat. A Perl és a C programozási nyelv hasonlóságát kihasználva a fejezet összehasonlítja és egymással szembeállítja a Perl és a C nyelv számos tulajdonságát. Az összehasonlítás során a hasonlóságokkal szemben nagyobb figyelmet kapnak a különbségek.

A HALLÓ VILÁG! PERL NYELVEN

A programozási nyelveket oktató tankönyvek többsége az egyszerű „Halló világ!" programmal kezdődik. Tartva magunkat ehhez a szokáshoz, ebben a könyvben is bemutatunk néhány „Halló világ!" példát. Az alábbi kód programutasításai három különböző eljárással, háromszor írják ki a képernyőre a Halló világ! szöveget. (A Perl nem képes a magyar ékezetek helyes megjelenítésére. Ékezeteket csak a könnyebb olvashatóság céljából használunk. – A fordító megjegyzése.)

```
# Három eljárás a Halló világ! megjelenítésére Perl nyelven

printf("Halló világ!\n");
printf("%s\n", 'Halló világ!');
print "Halló világ!", "\n";
```

A kód első sora egy megjegyzés. A Perl nyelvben a *hashmark* (létra, #) karakter jelzi a megjegyzést. Amikor a Perl ezzel a karakterrel találkozik, akkor a jel után a sor végéig tartó szöveget figyelmen kívül hagyja. A Perlben ezzel az egyetlen karakterrel lehet megjegyzéseket beszúrni. A C nyelvtől eltérően a Perlben nincsenek többsoros megjegyzés-szerkezetek.

Első ránézésre a *printf* utasítások ugyanúgy néznek ki, mint a C nyelvben. Rövidesen azonban tapasztalni fogjuk, hogy a program nem tartalmaz *main* függvényt. Bár a Perl script fájlok támogatják a szubrutinokat (amelyek a C függvényekhez hasonlítanak), a script fájlok nem definiálnak olyan szakaszt, amelyik a kód fő törzsét tartalmazná. A Perl interpreter a script fájl végrehajtását a fájlban lévő első utasításnál kezdi.

A fenti kód második *printf* utasítása is nagyon hasonlít egy C programutasításhoz azzal a különbséggel, hogy a második karakterlánc kétszeres idézőjelek ("") helyett egyszeres idézőjelek ('') között áll. A Perl nyelvben a kétszeres idézőjelek közötti szöveg még bizonyos fokú átalakításon megy keresztül. Így például a Perl interpreter a kétszeres idézőjelek közötti „\n" újsor karakterláncot újsor karakterre fordítja le. A Perl nyelvű script fájlok egyszeres idézőjeleket (aposztrófokat) használnak egy karakterlánc literális (betű szerinti) megjelölésére. A Perl például az egyszeres idézőjelek közötti '\n' karaktereket nem újsor karakternek tekinti, amely új sorba állítaná a kurzort, hanem megjelenítené a "\" (backslash) és az „n" karaktert.

Végül a harmadik utasítás a *print* függvényt használja, ami a C nyelvben nem található meg. Ebben a sorban a függvény legszembetűnőbb jellemzője, hogy hiányoznak a szokásos zárójelek. Mint látni fogjuk, a Perl függvényekben mindenütt használhatjuk a zárójeleket, de a legtöbb esetben ez nem kötelező. A Perl csak abban az esetben igényli a zárójelek használatát, ha ezek nélkül egy kifejezés két- vagy többértelmű lehet. Ezzel együtt jó szokás a zárójelhasználat, és tartsuk is magunkat ehhez a szokáshoz.

A PERL MEGHÍVÁSA

Itt az ideje, hogy magunk is kísérletezzünk a Perl nyelvvel. Ha a Perl nyelvi környezet nincs a rendszerünkben, akkor telepítenünk kell. A telepítésre vonatkozó információkat az első kötetben levő CD-ROM-on található *Perl_Installation.txt* nevű fájl tartalmazza (valamint a Perl nyelvi környezetet is). Miután sikeresen telepítettük a Perl programjait, az alábbihoz hasonlók beírásával futtathatunk egy Perl script fájlt:

```
C:\PERL> perl script_fájl_neve <Enter>
```

Készítsünk el például egy *hallo.pl* nevű fájlt, amely csak az alábbi *print* függvényt tartalmazza:

```
print "Halló világ!\n";
```

A parancssorba most írjuk be az alábbi parancsot:

```
C:\PERL> perl hallo.pl <Enter>
```

A képernyőn az alábbi szöveg jelenik meg:

```
Halló világ!
```

Mint említettük, ha UNIX környezetben dolgozunk, akkor egy Perl script fájlt olyan látszatot keltve is meghívhatunk, mintha a program önállóan futna. Szerkesszük az előző fájlt és írjuk be az első sorába az alábbiakat:

```
#!/usr/bin/perl
print "Halló világ!\n";
```

Az első sor arra utasítja a rendszert, hogy a Perl környezetet használja ennek a script fájlnak a futtatásához. Ez a sor történetesen egy Perl megjegyzés (nem véletlenül), így a Perl ezt figyelmen kívül hagyja. A legtöbb UNIX-os parancshéj (shell) azonban az első két karaktert figyeli egy végrehajtható script fájlban. Ha az első két karakter *#!*, akkor a parancshéj a sor további részét annak a parancsnak tekinti, amellyel a script fájlt végre kell hajtania. A fenti példában ez a parancs a */usr/bin/perl*. A héj a script fájl nevét automatikusan átadja a Perlnek. A script fájl futtatásához a *chmod* paranccsal be kell állítanunk a script fájl végrehajtási bitjét (mint például *chmod +x hallo.pl*). A héjtól függően esetleg még a *rehash* beírásával azt is közölnünk kell a héjjal, hogy új parancsot vettünk fel. A Perl elérési útjának módosítására is szükség lehet, ha az a rendszernek nem a */usr/bin* könyvtárába lett telepítve.

Megjegyzés: A Perl nyelvű script fájlok többsége a .pl fájlnév-kiterjesztést használja, de nem kötelező ennek a használata, sőt, akár el is hagyható. Ha Perl nyelvű script fájlokat önálló programként futtatunk, akkor talán célszerű a kiterjesztés elhagyása.

A PERL HIBAKERESŐJE

Talán kicsit korainak tűnhet, hogy máris szóba hozzuk a Perl hibakeresőjét (debugger). Látni fogjuk azonban, hogy nagyon jó hasznát vehetjük a Perl tanulásához. A hibakereső segítségével könnyen kipróbálhatjuk a fejezetben előforduló példákat. Bármilyen Perl utasítást beírhatunk a hibakeresőbe, amely ilyen módon egyfajta „interaktív Perlként" használható. Számos képességet tesztelhetünk így, ami sokkal könnyebb, mintha minden példához egy komplett script fájlt készítenénk és futtatnánk.

A Perl hibakeresője magába a Perl programba van beépítve. A hibakeresőt úgy hívhatjuk meg, hogy a parancssorba a *perl* programnév után beírjuk a *-d* kapcsolót:

```
C:\PERL> perl -d hallo.pl <Enter>
```

Ekkor a Perl betölti a *hallo.pl* script fájlt, és elindítja a hibakeresőt. Ha UNIX környezetben dolgozunk, akkor a script fájl első sorába beírhatjuk a *#!/usr/bin/perl -d* megjegyzést.

Ha csak magát a hibakeresőt szeretnénk futtatni anélkül, hogy betöltenénk egy script fájlt, akkor az alábbi parancsot adjuk ki:

```
C:\PERL> perl -de 0 <Enter>
```

A fenti példában a *-d* parancssori argumentum arra utasítja a Perlt, hogy indítsa el a hibakeresőt, a *-e 0* argumentumok pedig arra, hogy „hajtsa végre az egysoros *0* script fájlt". Mivel a *0* script fájl nem létezik, a Perl csak a hibakeresőt indítja el. Ha a Perl helyesen van telepítve a rendszerünkbe,

akkor a képernyőn az alábbiakhoz hasonló üzenetek jelennek meg, arról tájékoztatva, hogy a hibakeresőben vagyunk:

```
Loading DB routines from $RCSfile: perldb.pl,v $$Revision: 4.0.1.3
$$Date: 92/06/08 13:43:57 $

Emacs support available.

Enter h for help.

main'(pl000159:1):      0

 DB<1>
```

Ha a Perl válaszol, de például a *Can't locate perldb.pl in @INC* hibaüzenetet jeleníti meg, akkor a Perl nincs megfelelően telepítve a rendszerbe. Ha ilyen hibák fordulnak elő, akkor olvassuk el a Perl tartozékát képező „readme" fájlokat. A hibakeresőbe bármilyen Perl kifejezést beírhatunk, amit a hibakereső azonnal végrehajt. A hibakeresőben még az alábbi parancsokat használhatjuk:

- h – súgó, megjeleníti a hibakeresőben használható parancsokat
- n – végrehajtás a következő utasításig (átlép a szubrutinokon)
- <CR> – megismétli az utolsó n vagy s parancsot
- p kif – a *print kifejezés* rövidítése
- q – kilépés (quit)
- r – végrehajtás egy szubrutinból való visszatérésig
- s – a script fájl lépésenkénti végrehajtása (belépés a szubrutinokba)

Az alábbi hibakereső parancs például a *print* függvény segítségével jeleníti meg a *Halló világ!* szöveget:

```
 DB<1> print "Halló világ!\n"; <Enter>
Halló világ!
 DB<2>
```

A Perl hibakeresőjének használatakor egy kifejezés értékének megjelenítéséhez jó szolgálatot tesz a print parancs *p* alakú rövidítése. Az is a kényelmünket szolgálja, hogy egy utasítás végére nem kell beírnunk a pontosvesszőt, mert a hibakereső ezt mindig megteszi helyettünk. Továbbá a hibakereső mindig megjelenít egy új sort, ahová beírhatjuk a következő parancsunkat. Az alábbi példa a *p* rövidítés használatát szemlélteti:

```
 DB<2> p "Halló világ!" <Enter>
Halló világ!
 DB<3>
```

Amint látható, ha kiadunk egy parancsot, a hibakereső megnöveli a DB prompt jobb oldalán látható számlálóját. A fejezet tanulmányozása során bizonyára futtatni fogjuk a Perl hibakeresőjét, hogy példákat írjunk be és együttműködjünk a Perllel.

Megjegyzés: Ha többsoros utasítást írunk be a hibakeresőbe, akkor minden egyes sor végére be kell írnunk a "\" sorfolytató karaktert. Ellenkező esetben a hibakereső az első sor után hibaüzenetet küld.

Az alábbi parancs a sorfolytató karakter használatát mutatja be a Perl hibakeresőben:

```
 DB<3> for ($i = 0; $i < 10; $i++) { \ <Enter>
cont:    print $i; \ <Enter>
cont: } <Enter>
0123456789
 DB<4>
```

A fenti példa a Perl *for* ciklusát használja arra, hogy 0-tól 9-ig megjelenítse a számokat. A *for* ciklussal a fejezet későbbi részében foglalkozunk.

A PERL ADATTÍPUSAI

A programokban különböző adattípusokat használunk változók létrehozásához. Egy adattípus egy értékkészletet és egy művelethalmazt definiál. Az értékkészlet azokat az értékeket tartalmazza, amelyek egy változóban tárolhatók, a művelethalmaz pedig azokat a műveleteket, amelyeket a program a változón végre tud hajtani. A Perl nyelvben egy adat szám vagy karakterlánc lehet. Egyetlen adatot *skaláris értéknek*, vagy egyszerűen csak *skalárnak* neveznek. Az alábbiakban néhány *skaláris értéket* sorolunk fel, amelyeket a Perl literálisan (a karakterei szerint) jelenít meg:

- decimális: 127 vagy 127.0 vagy 1.27E2
- hexadecimális 0x7F vagy 0x7f
- oktális: 0177 (a vezető 0 jelenti az oktális számrendszert)
- karakterlánc: „Halló világ!\n" vagy 'Halló világ!'

Az alábbi parancs például arra használja a Perl hibakeresőt, hogy megjelenítse a 0177 oktális értéket, ami decimális számrendszerben a 127-nek felel meg:

```
 DB<4> p 0177 <Enter>
127
```

A felszín alatt a Perl a literális értékeket a saját, belső formátumára alakítja át. Amikor a Perlnek oktális vagy hexadecimális értékeket kell megjelenítenie, akkor ezeket előbb decimális számrendszerbe alakítja át, mint a fenti példában láttuk.

Megjegyzés: Mint majd látni fogjuk, a script fájljaink a printf függvény segítségével különleges formátumban, például oktális vagy hexadecimális formátumban is megjeleníthetnek értékeket.

A Perl saját magán belül minden számot dupla pontosságú, lebegőpontos számként tárol. Ez egyszersmind azt is jelenti, hogy a Perlben nincs egész számú (integer) adattípus. Ezzel azonban az esetek többségében nem kell törődnünk, mert a Perl „tudja a dolgát". Ha például egy számot olyan környezetben kell használnunk, ahol csak egész számnak van értelme (mint például egy bitenkénti művelet valamely operandusának), a Perl automatikusan egész számmá alakítja a lebegőpontos számot.

Megjegyzés: Akik C nyelven programoznak, és megszokták, hogy az osztás művelete automatikusan egész számot eredményez, ne feledjék, hogy a Perlben az **int***() függvénnyel kell a hányadosból egész számot előállítani.*

Az alábbi parancsok azt szemléltetik, hogy miként kezeli a Perl az egész és a lebegőpontos értékeket:

```
print 6 & 3;                    # a mûvelet eredménye 2
print 6.9 & 3.1;                # ugyanaz, mint fent
print 7 / 3;                    # a mûvelet eredménye 2.3333, ami nem egész szám
print int(7/3);                 # a mûvelet eredménye 2
```

Ugyanúgy, ahogyan a Perl a lebegőpontos számokat egész számokká alakítja, amikor egy script fájl a számot egész számokkal dolgozó környezetben használja, a Perl – amikor van értelme – a számokat karakterláncokká alakítja, és fordítva. Ha például egy script fájl egy számot olyan környezetben használ, ahol csak karakterláncoknak van értelmük – például karakterláncok egymáshoz fűzésénél – akkor a Perl a számot karakterlánccá alakítja át. Hasonlóképpen, ha egy script fájl egy karakterláncot olyan környezetben használ, ahol csak számoknak van értelmük, a Perl a karakterláncot számmá alakítja át. Amikor Perl nyelvű script fájlokat írunk, általában nem kell törődnünk azzal, hogy a Perl saját magán belül hogyan jeleníti meg a skaláris értékeket.

A Perl ugyan támogatja a logikai értékeket (Boolean true és false), de ezek megjelenítésére nincs külön adattípusa. A C nyelvhez hasonlóan a Perl egy numerikus értéket *igaznak* (true) tekint, ha az érték nem egyenlő nullával. Ugyanígy, egy karakterlánc-értéket igaznak tekint, ha a karakterlánc nem üres, azaz nem "" vagy '0'. Egyes Boole-operátorok, mint például a nagyobb (>), mint műveleti jel igaz eredmény esetén az 1 értéket, hamis eredmény esetén a 0 értéket adja vissza. A Perl nyelvű script fájljainkban azonban egyszerűen tekintsünk minden nem nulla értéket igaznak, legyen szó akár számról, akár karakterláncról.

A Perl nyelvű script fájlok a skaláris értékeket *listába* csoportosíthatják. Ha a script fájl a listát egy változóban tárolja, akkor ezt a változót *tömbváltozónak*, vagy röviden csak *tömbnek* nevezzük.

A PERL VÁLTOZÓI

A Perl a következő három változótípust használja: *skalár, tömb* és *asszociatív tömb*. A változónevek a C nyelvhez hasonlóan a Perl nyelvben is érzékenyek a kis- és nagybetűs írásmódra, vagyis a VALT, *Valt* és a *valt* nevek mindegyike más változót jelent. Továbbá a Perl mindegyik változótípushoz saját névterületet rendel, ami azt jelenti, hogy ugyanazon script fájlon belül lehet például egy *valt* nevű skalár változó és egy ugyancsak *valt* nevű tömbváltozó. A Perl az ilyen nevű változókat a környezetük alapján különbözteti meg egymástól (aszerint, ahogyan a script fájl használja őket).

Megjegyzés: A változók típusokba sorolása nem úgy történik, mint a C nyelvben. Egy skalár változó például bármilyen skaláris értéket tárolhat, és az adattípusok között automatikus a konverzió. Azt is észrevehettük, hogy a Perl nyelvben egy script fájlnak nem kell deklarálnia a változókat. Ha egy változó nincs deklarálva, akkor azt a Perl globális változónak tekinti. A fejezet későbbi részében megvizsgáljuk a változók deklarálását és hatókörét.

SKALÁR VÁLTOZÓK

Mint említettük, egy skalár változó egyetlen értéket tárol. A Perl nyelvben a skalár változók nevei mindig a dollár jelével ($) kezdődnek. Az alábbi Perl utasítások a 35 értéket rendelik a *$kor* nevű skalár változóhoz, és a „Pista" karakterláncot a *$nev* változóhoz. A harmadik utasítás a *print* függvény segítségével megjeleníti a változók értékét:

```
$kor = 35;
$nev = 'Pista';
print ($nev, ' éveinek száma ', $kor);
```

Ha a fenti utasításokat egy SKALAR.PL nevű fájlba mentjük, akkor futtathatjuk a programot:

```
C:\PERL> perl SKALAR.PL <Enter>
Pista éveinek száma 35
```

TÖMBVÁLTOZÓK

Mint tudjuk, egy tömb olyan változó, amely egy sor skalár értéket tárol. Az alábbi Perl utasítások tömbváltozókra mutatnak példát:

```
@napok = ('Va', 'Hé', 'Ke', 'Sze', 'Csü', 'Pé', 'Szo');

print(@napok);                          # az eredménye 'VaHéKeSzeCsüPéSzo' lesz
print($napok[4]);                       # az eredménye 'Csü' lesz
@munkanapok = @napok[1..5]              # @munkanapok tömbbe a ('Hé', 'Ke',
                                        # 'Sze', 'Csü', 'Pé') tömbelemek kerülnek
@ureslista = ()
```

A tömbváltozókra való hivatkozások általában a @ karakterrel kezdődnek, és a név után általában szögletes zárójelek között ([]) egy index áll. A C nyelvhez hasonlóan a Perlben is csak egész szám lehet az index, ami normál esetben 0-tól kezdődik. E két általános szabály alóli kivételekről a fejezet későbbi részében még lesz szó.

A harmadik utasítás (amelyik a *$napok[4]* tömbelemre hivatkozik) skalár hivatkozású tömbváltozóra mutat példát. Mivel az utasítás csak egyetlen tömbelemre vonatkozik, a kapott érték skalár érték. A @ jel helyett a $ jelet használva a script fájl skalár értékre hivatkozik. Ez apró, de nagyon fontos megkülönböztetés. A szögletes zárójelek jelzik, hogy a script fájl tömbváltozóra hivatkozik. A dollárjel azt jelzi, hogy a script fájl skalár értékre hivatkozik.

A negyedik utasítás úgy inicializálja a *@munkanapok* tömböt, hogy a tömbhöz a *@napok* tömb egy *szeletét* (slice) rendeli. A szelet egy lista része, és a Perl ilyen módon hivatkozik egy tömb részére.

Az előző példa első két sorában a *@napok* tömb nem tartalmaz indexet. Ha egy script fájl a tömb neve mellől elhagyja az indexet, akkor a Perl úgy tekinti ezt, mint a teljes tömbre való hivatkozást. Ugyancsak az előző példa első sora karakterláncok listájával inicializálja a *@napok* tömbváltozót. Karakterláncok mellett azonban változók értéke, sőt más tömb is hozzárendelhető tömbökhöz, az alábbiak szerint:

```
@valami = ($kor, $nev);
@AzEnBaratom = ('Feri', 'Mari', @ATeBaratod);
```

Néhány további példa tömbszeletekre:

```
@hetvege = (@napok[0,6];                # a @hetvege tömbbe a 'Va' és a 'Szo'
                                        # érték kerül
print(@napok[1..5,0,6]);                # a print függvény kimenete a
                                        # 'HéKeSzeCsüPéVaSzo' lesz
```

Ha egy script fájl egy tömbváltozót *skalár környezetben* használ, akkor a változó értéke a tömbben tárolt elemek száma lesz. Skalár környezet minden olyan környezet, ahol csak skalár érték használatának van értelme. Az alábbi utasítás például skalár környezetben használja a *valami* tömböt, hogy megállapítsa a tömbben lévő elemek számát. Ha az elemek száma nagyobb vagy egyenlő 2-vel, akkor a script fájl hibaüzenetet jelenít meg és befejeződik:

```
(@valami >= 2) || die "Túl sok elem!\n";
```

A *die* függvény arra utasítja a Perlt, hogy fejezze be a script fájl végrehajtását, és jelenítse meg a megadott üzenetet. Ha a script fájl nem tartalmaz üzenetet, akkor a *die* függvény egyszerűen befejezi a script fájl végrehajtását.

A Perl egy különleges, *$#valt* alakú szerkezetet is használ, ami egy tömb utolsó elemét adja vissza. Az alábbi *for* utasítás például a *$[* szerkezetet használja a tömb első indexének, a *$#* szerkezetet pedig a tömb utolsó indexének megállapítására. A *for* utasítás a tömb valamennyi elemét megjeleníti:

```
for ($i = $[; $i <= $#valami; $i++)
  {
    print $valami[$i];
  }
```

Könnyen belátható, hogy az előző *for* utasítás más módon, de ugyanazt hajtja végre, mint az alábbi *print* utasítás:

```
print @valami;
```

SKALÁR VAGY TÖMB KÖRNYEZET

Figyeljük meg, hogy a listakészítő operátor (,) ugyanaz, mint a sorrendi kiértékelés operátor (,). Hogy a Perl mikor melyik operátort használja, az attól a környezettől függ, amelyikben előfordulnak – pontosabban attól, hogy egy script fájl tömböt vagy skalár értéket használ-e. A Perl tömb környezetben a listakészítő operátort használja, skalár környezetben pedig a sorrendi operátort. Tekintsük az alábbi kifejezéseket:

```
@egy_tomb = (1,2,3,4,5);
$egy_skalar = (1,2,3,4,5);
```

Az első példa egy tömböt inicializál, míg a második példa a *$egy_skalar* változóhoz az 5-ös értéket rendeli, és az első négy értéket eldobja.

Most tekintsük az alábbi utasításokat:

```
print $assoc{1,2};
print @assoc{1,2};
```

Az első utasítás egyetlen értéket jelenít meg egy kétkulcsos asszociatív tömbből, míg a második utasítás két értéket jelenít meg egy egykulcsos asszociatív tömbből.

Végül tekintsük az alábbi utasításokat, melyek közül az első átmásol egy listát egy másik tömbbe, míg a második a lista méretét rendeli egy skalár változóhoz:

```
@x = @lista;
$x = @lista;
```

ASSZOCIATÍV TÖMBVÁLTOZÓK

Egy asszociatív tömb típusú változó annyiban hasonlít egy tömbváltozóra, hogy az előbbi is egy sor skalár változót tárol. A különbség abban van, hogy míg egy tömbnek egész indexeket kell használnia a tömb valamely elemének kijelöléséhez, addig az asszociatív tömb bármilyen értéket használhat ehhez. Egy asszociatív tömb indexértékeit kulcsoknak (keys) nevezik. Tekintsük az alábbi példákat:

```
$korok{'Feri'} = 35;
$korok{'Mari'} = 25;
$, = ' '; # megjelenítéskor a listaelemeket elválasztó vesszők
# helyén szóközt jelenít meg

print @korok{'Feri','Mari'};          # az eredmény '35 25' lesz
print keys(%korok);                   # az eredmény 'Mari Feri' lesz

for $nev (keys(%korok))
  {
    print "$nev eveinek szama $korok{$nev}\n";
  }
```

Amint látható, a program értéket rendel egy '$,' változóhoz (vesszőnek nevezett skalár változóhoz). A script fájl azért tartalmazza ezt az utasítást, hogy ne „szaladjon össze" a kimenete, amikor a *print* operátor segítségével a későbbiekben megjelenít egy listát. A '$,'-hez hasonló különleges változókkal a fejezet későbbi részében foglalkozunk.

A Perl nyelvű script fájlok úgy azonosítanak egy asszociatív tömböt, hogy annak indexét (a kulcsot) kapcsos zárójelek ({}) közé teszik. A Perl script fájlok – a közönséges tömbökhöz hasonlóan – az asszociatív tömböknél sem adnak meg indexértékeket, ha a teljes tömbre akarnak hivatkozni.

Az *@korok{'Feri','Mari'}* hivatkozás kapcsos zárójelek közötti indexet használ, ami jelzi, hogy asszociatív tömb. A tömb előtt a @ jel egy tömbértéket jelez – hasonlóan ahhoz, ahogyan egy dollárjel egy tömbhivatkozás előtt egy skalár értéket jelez.

Megjegyzés: Két kulcs van megadva. Ezek a @ jellel kombinálva azt jelzik, hogy a script fájl az asszociatív tár egy szeletét igényli, és hogy az eredménynek egy listában kell lennie. Az utasítás egyenértékű a ($korok{'Feri'},$korok{'Mari'}) alakú utasítással, aminek (35,25) az eredménye.

A *print keys(%korok)* példa a Perl *key* operátorát mutatja be, ami egy asszociatív tömb teljes kulcslistáját adja vissza. A *%korok* hivatkozásban a % előtag jelzi, hogy a hivatkozás a teljes asszociatív tömbre vonatkozik.

Az utolsó példa is tartalmazza a *keys* operátort és bemutatja, hogyan jeleníthető meg a *for* ciklussal az asszociatív tömbben lévő valamennyi asszociáció. Figyeljük meg, hogy a *for* ciklus a kettős idézőjelek között lévő karakterláncon belüli változókra hivatkozik. A Perl a karakterlánc kiértékelésekor a hivatkozásokat a változók értékére cseréli le. A Perl programozók ezt a behelyettesítési eljárást a *változók helyettesítésének* vagy *interpolációnak* nevezik.

Megjegyzés: A Perl az egyszeres idézőjelek (aposztrófok) közötti karakterláncokban lévő változókat nem interpolálja.

A PERL OPERÁTORAI

A Perl adattípusai és változói eltérnek a C programozási nyelvben megszokottaktól, azonban operátorai és utasításai már jóval ismerősebbek lesznek. A Perl nyelv a C valamennyi operátorát tartalmazza, kivéve a *típuskonverziós* operátort (type cast), a **ptr* indirekt mutatóhivatkozást és a *valt.tag* vagy *valt->tag* struktúratag-választó operátort. A Perl továbbá számos olyan új operátort és utasítást tartalmaz, amelyeket különböző műveletek végrehajtására, például karakterláncok összehasonlítására és manipulálására használhatunk.

ARITMETIKAI OPERÁTOROK

Az aritmetikai operátorok (műveleti jelek) numerikus értékeken végeznek műveleteket, és az eredményük is numerikus érték. Ha egy kifejezés karakterlánc operandust tartalmaz, akkor a Perl a kifejezés kiértékelése előtt számmá alakítja át a karakterlánc értékét. A Perl a karakterláncot olyan módon alakítja át számmá, ami megfelel a C nyelv *atof()* futásidejű függvénytári függvényének. A Perl jelenleg az alábbi aritmetikai operátorokat használja:

- + összeadás
- - kivonás
- * szorzás
- / osztás (csak lebegőpontos számok)
- % modulo (csak egész számok)

Tekintsük meg a Perl alábbi aritmetikai műveleteit:

```
$x = 2.5;
$y = 3;
print ($x + 2 * $y);                    # az eredmény 8.5
print (7 / $y);                         # az eredmény 2.333333
print int(7 / $y);                      # az eredmény 2
print (7 % $y);                         # az eredmény 1
print (7.5 % $y);                       # az eredmény ugyancsak 1
```

Megjegyzés: A Perl nyelvben a "/" operátor mindig lebegőpontos eredményt ad, míg a % modulo (maradék) operátor mindig egész számot eredményez. Ténylegesen a modulo operátor mindkét operandust egész számmá alakítja át, mielőtt kiszámítaná a maradékot.

Tekintsük az alábbi maradékszámítási műveletet:

```
print (7.9 % 3.6);                      # az eredmény 1 lesz, mert (7 % 3) = 1
```

A Perl az inkrementáló és dekrementáló műveleteket is használja:

- ++ előzetes vagy utólagos inkrementálás
- -- előzetes vagy utólagos dekrementálás

Tekintsük a Perl alábbi inkrementáló és dekrementáló műveleteit:

```
$x = 4;
++$x;
print $x;                   # az eredmény 5 lesz
$y = $x--;                  # x értéke elõször y-ba kerül, majd x dekrementálódik
print "$y $x";              # az eredmény 5 4 lesz
```

Végül lássuk a Perl új, hatványozáshoz használható operátorát (**). Az alábbi példák a hatványozó operátor használatát szemléltetik:

```
$x = 2 ** 3;                    # 2 a köbön egyenlõ 8
$x = 2 ** 0.5                   # 2 négyzetgyöke megközelítõleg 1.414
$x = -2 ** -3                   # 1/(-2 a köbön) egyenlõ -1/8 (-0.125)
```

BITMŰVELETEK OPERÁTORAI

A bitműveletek operátorai egész számok bináris alakjain végeznek műveleteket és egész számokat adnak eredményül. Ha valamelyik operandus karakterlánc vagy tört szám, akkor a Perl az operandust először egész számmá alakítja át úgy, hogy ehhez az operandus long word méretét használja (ami általában 32 bites). A C programozási nyelv valamennyi bitműveleti jele rendelkezésre áll:

- | bitenként VAGY művelet
- & bitenként ÉS művelet
- ^ bitenkénti KIZÁRÓ-VAGY művelet
- ~ bitenkénti invertálás
- << bitenkénti léptetés balra
- >> bitenkénti léptetés jobbra

Tekintsük a Perl alábbi bitműveleteit:

```
$x = 5;                         # binárisan 101
$y = 3;                         # binárisan 011

print $x | $y;                  # az eredmény 7 (binárisan 111)
print $x & $y;                  # az eredmény 1 (binárisan 001)
print $x ^ $y;                  # az eredmény 6 (binárisan 110)
print $x & -1;                  # az eredmény 5 (binárisan 101)
print $x << 2;                  # az eredmény 20 (binárisan 10100)
print $x >> 1;                  # az eredmény 2 (binárisan 10)
```

A C nyelvhez hasonlóan a Perl is különleges módon végzi el a jobbra léptetés műveletét (>>), ha az operandus negatív (vagy ha a legnagyobb helyiértékű bit értéke 1). A művelet egyes implementációi mindig 0-t léptetnek be, míg mások megismétlik a legnagyobb helyiértékű bitet, hogy megtartsák az operandus előjelét.

ÖSSZEHASONLÍTÓ OPERÁTOROK

Az összehasonlító operátorok két operandus értékét hasonlítják össze. A Perl az aritmetikai operátorokhoz hasonlóan a karakterlánc operandusokat számokká alakítja át, mielőtt elvégezné az összehasonlítást. Annak érdekében, hogy a script fájlok olyan karakterláncokat is összehasonlíthassanak egymással, amelyek nem számok, a Perl további karakterlánc-összehasonlító operátorokat bocsát a rendelkezésünkre. Ezek az operátorok a karakterláncokat az ASCII értékeik alapján hasonlítják össze. Ha egy karakterlánc-összehasonlító operátor valamelyik operandusaként szám van megadva, akkor a Perl a számot előbb karakterlánccá alakítja át. A 12.1. táblázat sorolja fel a Perl összehasonlító operátorait:

Numerikus	Karakterlánc	Jelentése
==	eq	egyenlő
!=	ne	nem egyenlő
>	gt	nagyobb mint
<	lt	kisebb mint
>=	ge	nagyobb vagy egyenlő
<=	le	kisebb vagy egyenlő
<=>	cmp	nem egyenlő (előjeles eredmény)

12.1. táblázat. A Perl összehasonlító operátorai

Az összehasonlítás eredményének 1 lesz az értéke, ha az összehasonlítás igaz, és 0 lesz az eredménye, ha az összehasonlítás hamis. Az utolsó operátor (<=> vagy cmp) azonban a -1, 0 vagy 1 értéket adja vissza attól függően, hogy az első operandus kisebb, egyenlő vagy nagyobb mint a második operandus.

Megjegyzés: A Perl **cmp** *operátora hasonlóan viselkedik, mint a C nyelv* **strcmp**() *futásidejű függvénytári függvénye.*

Tekintsük az alábbi Perl nyelvű összehasonlítást:

```
$x = 5;                          # 5 hozzárendelése a $x változóhoz
print ($x < 4);                  # a kifejezés eredménye hamis, a kimenõ érték a 0
```

LOGIKAI OPERÁTOROK

A logikai operátorok Boole-kifejezéseket vizsgálnak, és eredményül az igaz vagy a hamis értéket szolgáltatják. A Perl egy logikai operátor operandusait Bool-értékekként kezeli – vagyis igazként vagy hamisként. A Perl logikai operátorai:

- || logikai VAGY
- && logikai ÉS

A Perl a logikai kifejezéseket mindig balról jobbra haladva értékeli ki. A Perl mindig befejezi egy logikai kifejezés kiértékelését, ha egy *logikai VAGY* művelet operandusa igaz, vagy ha egy *logikai ÉS* művelet operandusa hamis. A Perl ezt a „rövidzár" kiértékelést használja, hogy gyorsan ki tudjon értékelni egy kifejezést.

Az általánosan használt logikai operátorokon túlmenően a Perl még az alábbi logikai operátorokat is használja:

- ! logikai tagadás (NEM)
- ?: feltételes
- , felsorolás

A logikai tagadás operátora (!) negálja az operandus Boole-értékét úgy, hogy az operanduson *logikai NEM* műveletet végez. A logikai tagadás eredménye mindig igaz vagy hamis.

A C programozási nyelvhez hasonlóan a Perl feltételes operátorának (?:) is három operandusa van. Egy feltételes operátort tartalmazó kifejezés az alábbi alakban írható:

```
feltétel ? igaz eredmény : hamis eredmény
```

Ehhez hasonlóan az alábbi utasítás a feltételes operátort használja ahhoz, hogy Ferinek teljes hozzáférést, mindenki másnak pedig csak korlátozott hozzáférést biztosítson:

```
# Ferinek teljes hozzáférés, mindenki másnak csak korlátozott
$hozzaferes = ($felhasznalo eq 'Feri' ? 'Teljes' : 'Korlátozott');
```

A felsorolás (,) operátor szigorúan véve nem logikai operátor, mert nem vizsgálja az operandusok igaz voltát. Ugyanakkor viszont semmiféle más kategóriába sem fér bele. A Perl a felsorolás operátor operandusait balról jobbra haladva értékeli ki, és a jobb szélső operandus értékét adja vissza. A felsorolás operátort a C nyelvből vették át, ahol tipikusan arra használják, hogy egy szerkezeten belül több műveletet lehessen végrehajtani. Az alábbi példában egy *for* ciklus látható, ami a felsorolás operátort használja:

```
for ($i=0, $j=10; $i<10; $i++, $j-)
  {
    print $i,' ', $j
  }
```

KARAKTERLÁNC-KEZELŐ OPERÁTOROK

Mivel a Perl nyelvet elsősorban karakterláncok feldolgozására fejlesztették ki, nem meglepő, hogy a Perlben van néhány új karakterlánc-kezelő operátor. Ezek a következők:

- . konkatenálás (összefűzés)
- x ismétlés
- =~ változóhoz kötés mintaegyezőség vizsgálatához
- !~ változóhoz kötés mintaegyezőség vizsgálatához, majd az eredmény negálása

Az első kettő könnyedén szemléltethető az alábbi példán keresztül:

```
print 'f' . 'el' x 2 . 'et';          # eredménye: felelet
```

Amint látható, a fenti utasítás a konkatenálás és az ismétlés operátor segítségével állítja össze a „felelet" karakterláncot.

A Perl nyelvű script fájlok az utolsó két operátort mintaegyezőségek vizsgálatára használják. A mintaegyezőségek vizsgálatát ebben a fejezetben „Reguláris kifejezések" címmel ismertetjük. Az alábbi utasítások arra mutatnak példát, hogy miként vizsgálja a Perl a minta egyezőségét:

```
$valt = 'narancs';
print ($valt =~ /ara/) ? TRUE : FALSE;
```

Ebben a példában az utasítások a Perl mintaegyezőséget vizsgáló operátorát (=~) használják arra, hogy megállapítsák, tartalmazza-e a *$valt* változó az *ara* mintát. A fenti példában az utasítás a TRUE értéket jeleníti meg.

ÉRTÉKADÓ OPERÁTOROK

Az eddigiekben a Perlnek csak az egyenlőségjel (=) értékadó operátorát használtuk, amihez nem kell sok magyarázat. Akik járatosak a C programozási nyelvben, azoknak a Perl többi értékadó operátora is ismerős lesz. A C nyelvhez hasonlóan ezek az operátorok arra utasítják a Perlt, hogy végezze el a megadott műveletet a jobb értéken (azon a kifejezésen, ami az operátor jobb oldalán áll), majd hajtsa végre az értékadást:

```
=     +=    -=    *=    /=    %=    |=    &=
^=    ~=    <<=   >>=   **=   .=    x=
```

A BAL ÉRTÉKEK (LVALUES)

A C programozási nyelvhez hasonlóan a Perlben is *bal érték* (lvalue) a neve mindannak, ami egy értékadó operátor bal oldalán áll. Másként fogalmazva a *bal érték* azt az egységet, például változót jelenti, aminek értéket adhatunk. Egy Perl nyelvű script fájl egy literálnak például nem adhat értéket úgy, hogy "Feri" = 32, mert a "Feri" nem bal érték. A fájl a *$Feri* nevű változónak viszont már adhat értéket úgy, hogy *$Feri= 32*, mert a *$Feri* nevű változó már bal érték.

A Perlben minden olyan egység, valószínűleg bal érték, ami bal értékként értelmes. Az alábbi, *becsomagoló* és *kicsomagoló* utasítások közül az első egy listaváltozót (tömböt), a második pedig három skalár változót használ bal értékként:

```
@szinek = ($r, $g, $b);                    # színeket csomagol egy listába
($r, $g, $b) = @szinek;                    # kicsomagolja a listát
```

Amikor listákkal dolgozunk a Perlben, egy értékadó művelet nem kívánja meg, hogy a teljes listával dolgozzunk. A script fájl a lista adott elemeihez is rendelhet értékeket az alábbiak szerint:

```
@listaelemek[2,4,7] = (100, 200, 300);
```

A fenti példában az értékadó operátor három listaelemhez rendel értéket. Ehhez hasonlóan az alábbi értékadó utasítás úgy csomagolja ki egy lista elemeit, hogy az első két elemet egy-egy skalár változóhoz, az összes többit pedig egy listaváltozóhoz rendeli:

```
($arg1, $arg2, @tobbi) = @ARGV;            # vegyesen használható skalár és tömb
```

Megjegyzés: Nincs értelme annak, hogy a bal érték listájában egy listaváltozó után további változók szerepeljenek, mert a listaváltozó (ebben az esetben a @tobbi) fogja tárolni az összes többi értéket, amikor azok kicsomagolásra kerülnek, és az utána következő változók definiálatlanok maradnak.

LISTAOPERÁTOROK

A C programozási nyelvtől eltérően a Perl tartalmaz néhány listaoperátort:

- , listakészítő operátor
- .. tartományoperátor
- x ismétlő operátor

A listakészítő operátort már használtuk tömbök inicializálásához és bal értékként változók listájának elkészítéséhez.

A Perl *tartomány* (range) operátora egymásra következő egész számok tartományát adja vissza, amely a bal operandussal kezdődik és a jobb operandussal végződik, ezeket is beleértve. A script fájlok gyakran használják a *tartomány*operátort a *listakészítő* operátorral együtt listák készítéséhez. Az alábbi utasítás például a tartományoperátort használja arra, hogy létrehozzon egy *@szamjegyek* nevű listát, amely a 0-tól 9-ig terjedő tartományba eső számjegyeket tartalmazza:

```
@szamjegyek = 0..9;                        # az elkészített lista (0,1,2,3,4,5,6,7,8,9)
```

Ehhez hasonlóan az alábbi utasítás arra használja a tartományoperátort, hogy tömbindexek tartományát hozza létre. Feltéve, hogy a *@napok* lista a hét napjait tartalmazza (vasárnaptól kezdődően), az utasítás a 'Hétfő' és a 'Péntek' közötti napokat rendeli a *@munkanapok* listához:

```
@munkanapok = @napok[1..5];
```

Végül az alábbi utasítás két tartományoperátort használ arra, hogy elkészítse az érvényes hexadecimális számjegyek listáját:

```
@hexa_jegyek = (0..9,a..f);
```

A Perl ismétlő operátora, az *x*, adott számban megismétli (másolja) az adott operandusokat. Az alábbi utasítás például arra használja az ismétlő operátort, hogy háromszor megismételje az 1, 2, 3 listaértékeket:

```
@lista = (1,2,3) x 3;                    # az eredő lista (1,2,3,1,2,3,1,2,3)
```

FÁJLTESZTELŐ OPERÁTOROK

A Perl bőségesen tartalmaz fájlkezelő operátorokat. Nem kevesebb, mint 27 olyan operátora van, amelyek segítségével különböző információkhoz juthatunk egy fájlról anélkül, hogy megnyitnánk azt. Ezen operátorok többsége csak UNIX környezetben használható, de a következő operátorok minden rendszerben használhatók:

- -d megállapítja, hogy a fájl könyvtár-e
- -e megállapítja, hogy létezik-e a fájl
- -s megállapítja a fájl méretét
- -w megállapítja, hogy írható-e a fájl

Az első két operátor Boole-értéket ad vissza (igaz vagy hamis). A harmadik operátor a fájl bájtokban mért méretét adja vissza. Az alábbi utasítások ennek a három fájloperátornak a használatát mutatják be:

```
if (-e 'Perl.exe')
  {
    print 'A fájl mérete: ' -s 'perl.exe';
  }
else
  {
    print 'Nem található a perl.exe fájl\n';
  }
(-w 'Vmilyen_fájl') || die "Nem írható a Vmilyen_fájl\n";
```

AZ OPERÁTOROK PRECEDENCIÁJA

A többi programozási nyelvekhez hasonlóan a Perl is precedenciát rendel az operátoraihoz, amelylyel meghatározza a műveletek elvégzésének sorrendiségét. A 12.2. táblázat felsorolja a Perl operátorainak precedenciáit a legmagasabbtól a legalacsonyabbig:

++ –
! ~ unáris vagy egy operandusú mínusz
**
=~ !~
* / % x
+ - .
<< >>
-d -e -s -w (és a többi fájloperátor)
< > <= >= lt gt le ge
== != <=> eq ne cmp
&
| ^
&&
||
..
?:
= += -= *=
,

12.2. táblázat. A Perl operátorainak precedenciája a legmagasabbtól a legalacsonyabbig

Script fájlokban a kifejezések zárójelbe tételével változtathatjuk meg azt a sorrendet, amely szerint a Perl a műveleteket végrehajtja.

A PERL UTASÍTÁSAI

A Perl nyelv a C nyelv valamennyi utasítását használja, csaknem változatlan alakban. Az *if, while, do, for* és *goto* utasítások például a C nyelvben használatos alakban írhatók. Amint látni fogjuk, a Perlben a *continue* utasításnak valamelyest módosult a jelentése (a régi funkcióját a *next* utasítás vette át), a *break* utasításnak pedig *last* lett a neve. A Perl nem használja a *switch* utasítást. Néhány utasításnak megváltozott az írásmódja, és a Perl számos új utasítással bővítette az utasításkészletét.

EGYSZERŰ ÉS ÖSSZETETT UTASÍTÁSOK

Egy egyszerű *kifejezés* operátorok és operandusok bármilyen érvényes kombinációjából áll. A Perlben egy *utasítás* egyszerű kifejezés, amit pontosvessző zár le. A Perlben a C nyelvhez hasonlóan mindegyik utasítást pontosvesszővel kell befejeznünk. Talán észrevettük, hogy a hibakeresőben a pontosvessző beírása nélkül is dolgozhatunk, de csak azért, mert a hibakereső beírja helyettünk a pontosvesszőt. Az alábbi értékadó utasítás egy egyszerű kifejezést szemléltet Perl nyelven:

```
$Cim = 'A Web programozása';
```

Ugyancsak a C nyelvhez hasonlóan a Perl script fájlok *blokk-* (vagy *összetett*) *utasítást* is létrehozhatnak úgy, hogy kapcsos zárójelek ({ }) közé teszik az utasításokat:

```
{
  # egyszerû utasítások
  # blokkutasítások
}
```

A Perl bonyolultabb utasításaihoz sűrűn fogunk blokkutasításokat használni. A blokkutasításokat arra is használhatjuk, hogy *hatókört* adjon a helyi változóknak. Egy blokkon belül azonban a helyi hatókör automatikus – a script fájlnak a *local* kulcsszóval deklarálnia kell a helyi változókat. A fejezet későbbi részében még visszatérünk a változók hatókörére.

BEHÚZÁSOK ÉS ÜRES SOROK A KÖNNYEBB OLVASHATÓSÁG ÉS MEGÉRTÉS CÉLJÁBÓL

Amikor blokkutasításokat írunk script fájlokba, üres sorok és behúzások beiktatásával tegyük olvashatóbbá és érthetőbbé fájljainkat. A C programokhoz hasonlóan Perl script fájlokban is két szóközzel húzzuk be a blokkutasításokat az alábbiak szerint:

```
if (vmilyen_feltetel)
  {
    ket_szokozzel_beljebb_mint_a zarojel;

    if (masik_feltetel)
      {
        ket_tovabbi_szokoz_itt;
      }
  }
```

Amint látható, a kapcsos zárójeleket követően a kódot beljebb húzva könnyen megállapíthatjuk, hogy mely utasítások tartoznak egy adott kódblokkhoz. A behúzásokon túlmenően még az is célszerű, ha üres sorok beszúrásával csoportosítjuk a logikailag összetartozó utasításokat.

FELTÉTELES UTASÍTÁSOK

Az előző példákban már többször használtuk az *if* utasítást. A Perlben majdnem pontosan úgy használható az *if* utasítás, mint a C nyelvben. Az eltérés az, hogy amíg a C nyelvben egy *if* utasítás egyszerű utasítást is használhatott, addig a Perlben az *if* utasításnak mindig blokkot kell alkotnia:

```
if (kifjezes)
  utasitas;                          // C-ben érvényes, a Perlben nem

if (kifjezes)
  {
    utasitas;                        # a Perl megköveteli a blokkutasítást
  }
```

Ehhez hasonlóan az *else* utasítás is némileg másképp használható a Perlben, mint a C nyelvben. A Perl az *else* utasításnál is megköveteli a blokkot. Az alábbi utasítások egy érvényes *if-else* szerkezetet mutatnak be C nyelven:

```
// C nyelvû példa (Perlben ez sem érvényes)

if (kif_1)
  utasitas_1;
else if (kif_2)
  utasitas_2;
else
  utasitas_3;
```

A fenti szerkezet a Perl *elsif* szerkezetét használva:

```
# az ekvivalens Perl szerkezet

if (kif_1)
  {
    utasitas_1;
  }
elsif (kif_2)
  {
    utasitas_2;
  }
else
  {
    utasitas_3;
  }
```

A PERL UNLESS OPERÁTORA

A C programozási nyelvben a *logikai NEM (!)* operátort használhatjuk egy Boole-változó értékének ellenkezőjére változtatásához (igazról hamisra és fordítva):

```
if (!(kif))                          // A C nyelv logikai NEM operátora
  {
    utasitas;
  }
```

Perl nyelvű script fájlokban a *logikai NEM* operátor mellett gyakran használják az *unless* (=hacsak nem) operátort is. Az alábbi Perl utasítások például az *unless* operátort használva ugyanazt valósítják meg, mint az előző C nyelvű kód:

```
unless (kif)
  {
    utasitas;
  }
```

Megjegyzés: A Perl a C nyelvtől eltérően nem használja a **switch** *utasítást.*

A DO UTASÍTÁS

A különleges blokkutasítások egyike a *do* utasítás, amely lehetővé teszi, hogy az utasítás egy értéket adjon vissza. A *do* utasítás a blokkon belül utoljára kiértékelt kifejezés eredményét adja vissza. Az alábbi példában a *do* utasítás a *$Honap* karakterlánc-változó tartalmát összehasonlítja az év hónapjaival és a hónap napjainak számát hozzárendeli a *$NapokSzama* változóhoz:

```
$NapokSzama = do
  {
    if ($Honap eq 'Szeptember' || $Honap eq 'Április' ||
     $Honap eq 'Június' || $Honap eq 'November')
      {
        30;
      }
    elsif ($Honap eq 'Február')
      {
        $Ev & 3 ? 28 : 29;              # szökõév vizsgálata
      }
    else
      {
        31;
      }
  };
```

Jegyezzük meg, hogy a Perl elvárja a pontosvesszőt a *do* blokk végén. Ne keverjük össze a *do* blokkot a *do while* utasítással, mely utóbbira rövidesen kitérünk.

CIKLUSSZERVEZŐ ÉS ELÁGAZTATÓ UTASÍTÁSOK

A Perl a C programozási nyelvtől csak némileg eltérő módon használja a *for, while* és *do* ciklusokat. A Perl és a C ciklusszerkezetei között az a leglényegesebb eltérés, hogy a Perl szerkezetekben a blokkutasításokat kapcsos zárójelek ({ }) közé kell tenni. Továbbá látni fogjuk, hogy a Perl néhány újabb alakkal bővíti ki a *ciklusszerkezeteket.* Az alábbi *for, while* és *do* ciklusok C és Perl nyelven egyaránt működnek:

```
for ($i = 0; $i < 100; $i++)
  {
    printf("%d\n", $i);
  }

while ($i > 0)
  {
```

```
    printf("%d\n", $i-);
  }

do
  {
    printf("%d\n", $i++);
  } while ($i < 0);
```

A Perl ciklusszerkezetei abban is különböznek a C hasonló szerkezeteitől, hogy a Perlben nincs *break* utasítás, és a *continue* utasítás teljesen más műveletet végez. Szerencsére azonban a Perlnek van néhány új, rugalmasabban használható szerkezete:

- last kilépés a ciklusból (mint a C nyelvben a break)
- next indítja az újabb közelítést (mint a C nyelvben a continue)
- redo megismétli az aktuális közelítést

A Perl ciklusszerkezeteinek megismeréséhez azt is tudnunk kell, hogyan használja a Perl a *continue* blokkot. Tekintsük az alábbi *while* ciklust, amely a *continue* blokkot használja:

```
$i = 100;

while ($i > 0)
  {
    print $i;
  } continue {$i-}
```

Tekintsünk úgy a Perl *continue* blokkjára, mint egy *for* ciklus harmadik kifejezésére, amelyet a ciklus minden egyes közelítéskor végrehajt. Ehhez hasonlóan hajtja végre a Perl a *continue* blokkot minden egyes közelítés végén. Amint azonban látni fogjuk, a *continue* blokk nagyobb lehetőséget ad a script fájlnak a feldolgozás vezérlésére, mint a *for* ciklus kifejezése. Amikor egy Perl ciklus a *next* szerkezetet használja, a ciklus végrehajtja a *continue* blokkot (amennyiben van ilyen). Ha viszont a ciklus a *redo* szerkezetet használja, akkor a ciklus nem hajtja végre a *continue* blokkot.

CÍMKÉK HASZNÁLATA VEZÉRLŐSZERKEZETEKBEN

Perl nyelvű script fájlon belül egy címke egyszerűen egy név, amely a fájlon belül egy helyet azonosít. A címkék neve után kettőspont áll (mint például *kulsociklus*:). A *goto* utasítás segítségével a script fájlok ráugorhatnak ilyen címkékre, és így elágazásokat hozhatnak létre. Sőt, a script fájlok a *last, next* és *redo* szerkezetet is használhatják arra, hogy a program végrehajtása egy címkétől folytatódjon. Az alábbi példa azt szemlélteti, miként használható a *last* szerkezet arra, hogy a program végrehajtása egy címkétől folytatódjon:

```
kulsociklus: while ($i > 0)
  {
    while ($j > 0)
      {
        # Egyéb feldolgozás

        if ($beKellFejezniACiklust)
```

```
            {
              last kulsociklus;
            }
        }
    }
```

Ebben a példában a *last* szerkezet segítségével ágazik el a program a *kulsociklus* címkére, és fejezi be a ciklus végrehajtását (arra utasítva a Perlt, hogy tegyen úgy, mintha a ciklus végrehajtotta volna az utolsó közelítését). Amikor egy script fájl a ciklusszerkezetek valamelyikét, például a *last* szerkezetet egy címkével együtt használja, akkor a program az aktuális szerkezetből ráugorhat a megadott ciklusra. A C programozók viszont csak a *continue* és a *break* utasítást használhatják a legbelső ciklusban. A Perl címkével használható ciklusszerkezete feloldja ezt a korlátozást.

AZ UNTIL CIKLUS

A Perl nyelvben az *until* ciklusszerkezet is használható, ami a *while* ciklus ellentettje. Amint láttuk, a *while* ciklus addig folytatja az utasítások végrehajtását, amíg egy adott feltétel igaz. Ezzel szemben az *until* ciklus addig folytatja az utasítások végrehajtását, amíg a feltétel igazzá nem válik. Tekintsük például az alábbi *while* ciklust, ami addig ismétli a végrehajtást, amíg egy kifejezés értéke hamis:

```
while (!(kif))
  {
    utasitas;
  }
```

Az *until* utasítást használva egy ezzel azonos ciklust készíthetünk:

```
until (kif)
  {
    utasitas;
  }
```

Ehhez hasonlóan az alábbi *do while* ciklus a *logikai NEM* operátort használja arra, hogy egy ciklus mindaddig fusson, amíg egy kifejezés hamis:

```
do
  {
    utasitas;
  } while (!(kif));
```

A *do until* ciklust használva ugyanezt a ciklust valósíthatjuk meg anélkül, hogy használnunk kellene a *logikai NEM* operátort:

```
do
  {
    utasitas;
  } until (kif);
```

A FOR ÉS A FOREACH CIKLUS

A Perl a C programozási nyelvvel azonos módon használja a *for* ciklust:

```
for (utasitas_1; kifejezes; utasitas_2)
  {
    utasitas_3;
  }
```

Az alábbi utasítások a *for* ciklust használják arra, hogy 0-tól 100-ig megjelenítsék a számokat:

```
for ($szam = 0; $szam <= 100; $szam++)
  {
    prin $szam, '';
  }
```

A Perlben egy *foreach* nevű ciklusszerkezet is használható, amely lehetővé teszi, hogy a script fájllistákban és tömbökben végezzenek közelítéseket. Néhány példa ennek használatára:

```
@lista = ('a', 'b', 'c');

foreach $arg (@lista)
  {
    print "Listaelem: $arg\n";
  }

foreach $i (1..10)
  {
    print "kozelites $i\n";
  }
```

Az első példában a *foreach* ciklus a *@lista* listaváltozóban lévő értékeken keresztül halad át, míg a másodikban az 1 és 10 közötti számtartományban.

Mint láttuk, egy *foreach* cikluson belül a lista literálokból vagy tömbváltozóból állhat. Amikor a ciklus végrehajtja az egyes közelítéseket, egy adott skalár változót rendel (ez az első példában a *$arg*, a másodikban a *$i*) a következő listaelem értékéhez. A *foreach* ciklus a skalár változó hatókörét (a forráskód azon részeit, ahol a script fájlnak tudomása van a változó létezéséről) a ciklus törzsére korlátozza. Ennélfogva a *foreach* ciklus skalár változója nem kerül összeütközésbe más, olyan, azonos nevű változóval, amelyik a cikluson kívül létezik. Az alábbi utasítások a *$i* nevű változót egy *foreach* cikluson belül és azon kívül is használják:

```
$i = 1001;

foreach $i (1..9)
  {
    print "$i";                          # a ciklus kimenete 123456789
  }
print "$i\n";                            # az utasítás kimenete 1001
```

Amint látjuk, az a mód, ahogyan a *foreach* ciklus a (helyi) *$i* skalár változóját kezeli, nem befolyásolja a cikluson kívül lévő *$i* változó értékét.

A *foreach* ciklus egyik különleges, tömbváltozókkal kapcsolatban használható tulajdonsága az, hogy a ciklus képes a tömb elemeinek megváltoztatására. *(Legyünk óvatosak ennek a képességnek a használatakor!)* Tekintsük az alábbi *foreach* ciklust, amely 10-et ad hozzá egy tömb valamennyi eleméhez:

```
@lista = 1..5;

foreach $i (@lista)
  {
    $i += 10;
  }

$, = ' ';
print @lista;                          #a kimenet 11 12 13 14 15 lesz
```

Még egy végső megjegyzés a *foreach* ciklussal kapcsolatban. A Perl a *foreach* és a *for* neveket egymás szinonimájának tekinti. Ebből következően szabadon felcserélhetjük ezeket a ciklusneveket a script fájlokban. A Perl a ciklus környezete alapján meg tudja állapítani, hogy adott esetben melyik típusú ciklust kell használnia. Azonban csak azért, mert a Perl képes erre a különbségtételre, ne hagyatkozzunk rá. Használjuk mindig az adott célnak megfelelő ciklusnevet, hogy érthetőbbé tegyük a kódjainkat.

A GOTO UTASÍTÁS

A Perl a C programozási nyelvvel azonos módon használja a *goto* utasítást. Az alábbi példa a *goto* utasítást használja arra, hogy megjelenítse az 1 és 10 közötti számokat:

```
$i = 1;
ciklus:
  print $i++, ' ';
  if ($i <= 10)
    {
      goto ciklus;
    }
```

Ami a fenti példát illeti, ugyanezt az eredményt érhetjük el a Perl más ciklusszerkezeteivel is, amelyek kódjai jobban olvashatók. Általános szabályként az javasolható, hogy kerüljük a *goto* utasítás használatát.

EGYSZERŰ UTASÍTÁSOK MÓDOSÍTÁSA

A Perlben az *if, unless, while és until* szerkezetek különböző alakban használhatók. Ezeket az alakokat egy egyszerű utasításhoz hozzáfűzve módosíthatjuk az utasítás végrehajtását. Bizonyos helyzetekben ezekkel a speciális alakokkal olvashatóbbá és érthetőbbé tehetjük kódjainkat. A script fájlokban úgy válasszuk meg az utasítások formátumát, hogy azok célja a lehető legnyilvánvalóbb legyen. Tekintsük az alábbi példát, amely a *die* függvényt használja arra, hogy befejezze egy script fájl végrehajtását, ha a *$szamlalo* változó értéke 10-nél kisebb lesz:

```
if ($szamlalo < 10)
  {
    die;
  }
```

Ha az *if* szerkezet elé az alábbi módon beszúrjuk a *die* függvényt, akkor a fenti négy programsort egy sorra csökkenthetjük:

```
die if ($szamlalo < 10);
```

Ehhez hasonlóan az alábbi példa a Perl „rövidzár" kiértékelését használja arra, hogy egy feltétel leírásához szükséges programsorok száma egy sorra csökkenjen:

```
($szamlalo >= 10) || die;
```

Ha ebben a példában a *$szamlalo* értéke nagyobb vagy egyenlő 10-zel, akkor a Perl befejezi a feltétel kiértékelését (mert annak az eredménye igaz, bármi álljon is a *logikai VAGY* operátor jobb oldalán), és nem hívja meg a *die* függvényt.

Végül az alábbi példa az *unless* szerkezetet használja arra, hogy a feltételhez szükséges programsorok számát egy sorra csökkentse:

```
die unless ($szamlalo >= 10);
```

Hasonlóképpen, az alábbi két – egyszerű és módosított szerkezetű – *while* ciklus egymással egyenértékű:

```
$i = 0;
while ($i < 10)
  {
    $i++;
  }

$i = 0;
$i++ while ($i < 10);
```

Amint látható, a szerkezet módosításával a négysoros ciklus egyetlen sorosra csökkenthető. Ugyanígy, az alábbi két *until* ciklus is egyenértékű:

```
$i = 10;
until ($i >= 10)
  {
    $i++;
  }
```

```
$i = 10;
$i++ until ($i >= 10);
```

Megjegyzés: Mind a négy módosított szerkezetre vonatkozik, hogy még akkor is, amikor a szintaxisban a kifejezés az utasítás után áll, a Perl először kiértékeli a feltételt és csak ezt követően hajtja végre az utasítást.

DINAMIKUS KIFEJEZÉSEK GENERÁLÁSA AZ EVAL FÜGGVÉNNYEL

Mivel a Perl interpreter típusú nyelv, a script fájljaink a Perl segítségével „röptében" is tudnak kódot generálni. Tulajdonképpen így működik a Perl hibakeresője is. A Perl hibakeresője valójában egy Perl nyelvű program *(Perldb.pl)*. Amikor script fájlokat tervezünk, ilyen dinamikus kódot használhatunk dinamikus változónevek vagy speciális szubrutinok készítéséhez.

A Perl a dinamikus kifejezéseket az *eval* függvény segítségével értékeli ki. Az alábbi utasítások például egy dinamikus utasítást hoznak létre úgy, hogy a kívánt Perl utasítást tartalmazó szöveget hozzárendelik egy változóhoz. Ezt követően az utasítások az *eval* függvény segítségével kiértékelik (végrehajtják) az utasítást:

```
$perl_utasitas = 'print "Halló világ!\n";';

eval $perl_utasitas;                    # a Halló világ! szöveget jeleníti meg

$i = 1001;
$valt_nev ='$i';
print eval $valt_nev;                   # a $i értékét jeleníti meg
```

Megjegyzés: Egy script fájlban az **eval** *függvény használata veszélyes – különösen akkor, ha a fájl felhasználó által bevitt adatokat ad át a függvénynek. Az eval függvény segítségével egy script fájl bármilyen Perl parancsot végre tud hajtani (beleértve a rendszerparancsokat is). Az a körülmény, hogy egy felhasználó ilyen módon átveheti a vezérlést a program fölött, különösen hálózatban, az Interneten és Web programokban nagyon kockázatos.*

SZUBRUTINOK

Mint minden strukturált nyelv, a Perl is használja a szubrutinokat. Perl nyelvű script fájlon belül a *sub* kulcsszóval definiálunk szubrutint:

```
sub demo_sub
  {
    print "demo_sub hívása\n";
  }
&demo_sub;                              # a demo szubrutin hívása
```

A fenti példa egy *demo_sub* nevű szubrutint hoz létre. A szubrutin meghívásához egy ampersand (&) karaktert kell beírni a szubrutin neve elé. A Perl nyelvben szubrutin hívásakor nem kötelező a zárójelek használata.

Script fájl forráskódjának bármelyik részén deklarálható szubrutin, mert a Perl a fájl végrehajtása előtt a teljes forráskódot átvizsgálja. Szubrutinokat a forrásfájlnak azon a részén is deklarálhatunk, amely a szubrutin első használata után következik – ez az *előre hivatkozás*.

A szubrutinoknak argumentumaik és visszatérési értékeik lehetnek. Az alábbi példa egy *ertek_megjelenito* nevű szubrutint hoz létre, amely az argumentumában megkapott értéket jeleníti meg:

```
sub ertek_megjelenito
  {
    print 'Az érték ', $_[0];
  }

&ertek_megjelenito(1001);
```

A Perl szubrutinjai formálisan nem deklarálnak változókat az argumentumok tárolásához. Ehelyett a Perl egy @_ nevű tömbváltozót ad át a szubrutinnak, amely tömb az argumentumok értékeit tartalmazza. Ezt követően a szubrutin a *$_[0], $_[1]* stb. alakú tömbindexekkel tud hozzáférni az argumentumokhoz. Mivel azonban az argumentumok tömbindexeken keresztüli elérése nehezen olvasható és érthető kódot eredményezhet, a szubrutinok többsége helyi változókba másolja az argumentumokat.

Az alábbi, *ket_ertek_megjelenitese* nevű szubrutin két argumentum értékét jeleníti meg:

```
sub ket_ertek_megjelenitese
  {
    print 'Az elsõ érték ', $_[0], "\n";
    print 'A második érték ', $_[1], "\n";
  }

&ket_ertek_megjelenitese(1001, 2002);
```

Végül az alábbi, *minden_ertek_megjelenitese* nevű szubrutin valamennyi megkapott argumentum értékét megjeleníti. A szubrutin a @_ tömbváltozót használja az argumentumok számának meghatározásához:

```
sub minden_ertek_megjelenitese
  {
    for ($i = 0; $i < @_; $i++)
      {
        print 'A(z) ', $i,'. argumentum értéke ',$_[$i], "\n";
      }
  }

&minden_ertek_megjelenitese(1001, 2002, 3003, 4004);
```

Mint említettük, a Perl szubrutinjai értéket adhatnak vissza. A szubrutinok a *return* utasítást használják egy adott érték visszaadásához. A C programozási nyelvtől eltérően a Perl nem igényli a *return* utasítást. Ha egy szubrutin nem tartalmazza a *return* utasítást, akkor a szubrutin a *do* blokk-

hoz hasonlóan az utoljára kiértékelt kifejezés értékét adja vissza. Tekintsük például az alábbi utasításokat, amelyek két argumentumot adnak össze és az összeget adják vissza:

```
sub ertekek_osszege
  {
    return $_[0] + $_[1];
  }

print 'Az eredmény: ', &ertekek_osszege(1001, 2002);
```

SZUBRUTIN-KÖNYVTÁR ELÉRÉSE

A C programozási nyelvtől eltérően a Perl nem igazán támogatja a könyvtárak használatát. Van azonban egy olyan mechanizmus a Perlben, amelynek segítségével a script fájlok képesek más fájlok beépítésére. Tegyük fel például, hogy az *ertekek_osszege* szubrutint egy *sumertek.pl* nevű fájlban tároljuk. A *require* (=igényel) utasítás segítségével ekkor más Perl script fájlok is hozzáférhetnek a szubrutinhoz az alábbi módon:

```
require "sumertek.pl";
print &ertekek_osszege(10, 11);
```

A Perl require utasítását úgy tekinthetjük, mint a C nyelv *#include* előfordító direktívát. A forrásfájl megtalálásához a Perl először megnézi a függvénytár alapértelmezés szerinti könyvtárát (ellenőrizzük a Perl telepítését), majd az aktuális könyvtárat. A fájl neve előtt abszolút és relatív elérési utat is megadhatunk.

A Perl megjegyzi magának, hogy mely fájlokat „igényeltük", és mindegyiket csak egyszer tölti be, még ha többször is használjuk őket. A Perlnek számos szabványos könyvtára (függvénytára) van, amelyek kibővítik a képességeit. Szánjunk időt arra, hogy megnézzük a Perl meglévő függvénytárait, és képet alkothassunk arról, hogy milyen további képességek állnak rendelkezésünkre.

SZUBRUTINOK ELKÜLÖNÍTÉSE CSOMAGOKBA

Ha sok szubrutinunk van és főleg olyanok, amelyeket különböző fájlokban tárolunk, akkor a Perl package utasítását használhatjuk arra, hogy csökkentsük a változónevek ütközéséből adódó problémák előfordulásának valószínűségét (azonos változónevek különböző célokra történő használatát). Amint láttuk, szubrutinokon belüli helyi változók deklarálásával elkerülhetők ezek az ütközések. Ha viszont különböző szubrutinoknak megosztva kell használniuk meghatározott adatokat, akkor ezek az adatok *globális* hatókört igényelhetnek, ami viszont már névütközésekhez vezethet. A *package* utasítás segítségével globális adatokat privát *névterületekre* csoportosíthatunk (amely területen kívül a globális változó nevei nem ismertek). Tekintsük az alábbi egyszerű példát, amelyben két, különböző forrásfájlban található szubrutin saját, privát *névterületet* használ:

```
# Más kódokkal együtt az egy.pl nevû script fájlban található
sub sub_egy
  {
    package demo_egy;
    $vmilyen_adat = 10;
  }
```

```
#*********************************************************************
# Más kódokkal együtt a ketto.pl nevû script fájlban található
sub sub_ketto
  {
    package demo_ketto;
    $vmilyen_adat = 20;
  }
```

Amint látható, az első szubrutin a *demo_egy* nevű csomagot (package), a második szubrutin pedig a *demo_ketto* nevű csomagot tartalmazza. Mindkét szubrutin beállíthatja és használhatja a *$vmilyen_adat* változót anélkül, hogy összeütközésbe kerülnének egymással a globális változók.

Ha egy script fájl ismeri annak a csomagnak a nevét, amelyben egy változó található, akkor a fájl úgy férhet hozzá a változóhoz, hogy a változó neve elé beírja a csomag nevét. Az alábbi utasítások például a *csomag_egy* és a *csomag_ketto* csomagneveket használják a *vmilyen_adat* változónév előtt:

```
&sub_egy
&sub_ketto

print "A csomag_egy csomagban lévõ érték $csomag_egy'vmilyen_adat\n";
print "A csomag_ketto csomagban lévõ érték $csomag_ketto'vmilyen_adat\n";
```

Ha Perl csomagokkal dolgozunk, akkor az aktuális forrásfájl számára létrehozhatunk egy egyedi névterületet, ha a fájl elejére beírjuk a *package* utasítást:

```
package vmilyen_csomag_neve;

$vmilyen_adat = 1;

sub vmilyen_szub
  {
    return $vmilyen_adat;
  }
```

Ebben a példában a *$vmilyen_adat* változó csak a csomagban létezik, így védve van a véletlen hozzáférésekkel szemben. Ilyen csomag használatával a C nyelvhez hasonlóan lehet szabályozni az adatelérési hatókört – a C nyelvben ugyanis lehetnek olyan globális adataink, amelyekhez csak azon a forrásfájlon belül lehet hozzáférni, amelyben azok definiálva vannak. Másik script fájlban lévő szubrutin hívásához a híváskor a csomag nevét kell használni:

```
require 'vmilyen_csomag.pl';
print &vmilyen_csomag_neve'vmilyen_szub;
```

KARAKTERLÁNCOK KEZELÉSE

Az előző példákban láttuk, hogyan készíthetünk változók behelyettesítésével literálokat. Azt is láttuk, hogyan konkatenálhatunk literálokat és karakterlánc típusú változókat. A fejezetnek ebben a részében egy sor további függvénnyel ismerkedünk meg, amelyek segítségével különböző módon kezelhetünk karakterláncokat.

A CHOP FÜGGVÉNY

A Perl *chop* függvénye törli a karakterlánc utolsó karakterét. A chop függvény szintaxisa:

```
$karakter = chop(kar_lánc);
```

A *chop* függvény a levágott karaktert adja vissza. A Perl nyelvű script fájlok gyakran használják a *chop* függvényt arra, hogy egy fájlban olvasott sor végéről eltávolítsák az újsor karaktert.

AZ INDEX FÜGGVÉNY

A Perl *index* függvénye egy karakterláncban egy rész-karakterláncot keres. Az *index* függvény szintaxisa:

```
$hely = index(kar_lánc, rész_lánc[, eltolás]);
```

Az *index* függvény a *rész_lánc* karakterlánc-részletnek a *kar_lánc* teljes karakterláncon belüli első előfordulásának helyét adja vissza. A függvényben nem kötelező jelleggel egy *eltolási érték* is megadható, amely azt a helyet jelöli, ahonnan a karakterlánc-részletnek a teljes láncon belüli keresését kezdeni kell. Ha a függvény nem találja a karakterlánc-részletet, akkor a -1 értéket adja vissza. Az alábbi utasítás az *index* függvény segítségével állapítja meg, hogy a *'le'* részlánc a harmadik karakterhelyet követően hányadik karakterhelyen kezdődik a teljes karakterláncban (a karakterhelyek számlálása 0-tól kezdődik):

```
print index('felelet', 'le', 3);          # az eredmény 4 lesz
```

A RINDEX FÜGGVÉNY

A Perl a *rindex* függvény segítségével karakterlánc-részletet keres egy teljes karakterláncban, és a részlánc jobb szélső előfordulásának kezdő karakterhelyét adja vissza. A függvény szintaxisa:

```
$hely = rindex(kar_lánc, rész_lánc);
```

A függvény hasonlít az *index* függvényre azzal az eltéréssel, hogy a *rindex* a részláncnak nem az első, hanem az utolsó előfordulását adja vissza. Az alábbi utasítás arra használja a *rindex* függvényt, hogy a *'felelet'* karakterláncban megkeresse a *'le'* részlánc jobb szélső előfordulásának a helyét:

```
print rindex('felelet', 'le');          # az eredmény 4 lesz
```

A LENGTH FÜGGVÉNY

A Perl *length* függvénye egy karakterlánc karaktereinek számát adja vissza. A *length* függvény szintaxisa:

```
$hossz = length(kar_lánc);
```

Az alábbi utasítás a *length* függvény segítségével jeleníti meg a karakterlánc karaktereinek számát:

```
print length('felelet');                    # az eredmény 7 lesz
```

A SUBSTR FÜGGVÉNY

A Perl *substr* függvénye kiemel vagy lecserél egy karakterlánc-részletet egy karakterláncban. A *substr* függvény szintaxisa:

```
$reszlanc = substr(kar_lánc, eltolás[, hossz]);
```

A függvény egy karakterlánc-részletet ad vissza, amelynek a hossza nem haladja meg a nem kötelező *hossz* paraméter által megadott karakterek számát. A karakterlánc-részlet a teljes karakterláncon belül az *eltolás* paraméterrel megadott karakterhelyen kezdődik. Ha a *hossz* paraméter nincs megadva, akkor a karakterlánc-részlet a teljes karakterlánc végéig tart. Ha a *hossz* paraméter értéke negatív, akkor a függvény az *eltolás* paramétert a karakterlánc végétől visszafelé értelmezi. A script fájlok a *substr* függvényt bal értékként használhatják egy karakterlánc-részlet lecserélésére. Az alábbi utasítások szemléltetik a *substr* függvény használatát:

```
print substr('narancs', 3);                 # az eredmény 'ancs'
print substr('narancs', -2);                # az eredmény 'cs'
print substr('narancs', 2, 2);              # az eredmény 'ra'
$k_lanc = 'szilva';
substr($k_lanc, -4) = 'eder';
print $k_lanc;                              # az eredmény 'szeder'
```

Megjegyzés: A Perl reguláris kifejezéseket használó műveletei sok esetben hatékonyabbak, mint a **substr** *függvénnyel történő keresés és csere. A reguláris kifejezéseket használó műveletekről a fejezet későbbi részében, a „Reguláris kifejezések" címmel olvashatunk.*

A JOIN FÜGGVÉNY

A Perl *join* függvénye listaelemeket kapcsol össze egyetlen karakterlánccá úgy, hogy adott karakterrel választja el egymástól az elemeket. A join függvény szintaxisa:

```
$uj_lanc = join(elválasztó, lista);
```

A join függvény karakterláncokká alakítja át a lista elemeit, és konkatenálja azokat egy karakterláncba. Az alábbi utasítások a *join* függvény használatát mutatják:

```
$k_lanc = join(',', 0..4, 10, 20);      # a létrehozott karakterlánc a
                                        # '0,1,2,3,4,10,20'
$k_lanc = join("\t", $a, $b, $c);       # tab karakterrel elválasztott lista
```

A SPLIT FÜGGVÉNY

A Perl *split* függvénye egy karakterlánc tartalmát listaelemekre bontja. A *split* függvény szintaxisa:

```
split(elválasztó, kar_lánc[, limit];
```

Az *elválasztó* paraméter a lista elemeit egymástól elválasztó karaktert adja meg – mint például vessző, szóköz, tabulátor stb. A nem kötelező *limit* paraméter a függvény által kibontható listabejegyzések maximális számát adja meg. Az alábbi utasítások szemléltetik a *split* függvény használatát:

```
($a, $b, $c) = split("\t", $k_lanc);    # a join függvény ellentettje
                                        # részekre bont egy listát
$k_lanc = 'egy tesztelõ szöveg';
@szavak = split(' ', $k_lanc);          # eredménye 'egy', 'tesztelõ', 'szöveg'
```

LISTAKEZELŐ FÜGGVÉNYEK

A fejezet eddigi példáiban már láttuk, hogyan készíthetünk listát, hogyan tárolhatunk listákat változókban, közelítésekkel hogyan lépkedhetünk végig listaelemeken, és hogyan férhetünk hozzá adott listaelemhez. A fejezet ezen részében néhány olyan függvénnyel ismerkedünk meg, amelyeket ugyancsak listakezeléshez használhatunk.

A REVERSE FÜGGVÉNY

A Perl *reverse* függvénye megfordítja egy listában a listaelemek sorrendjét. A reverse függvény szintaxisa:

```
@uj_lista = reverse(@lista);
```

A *reverse* függvény új listát ad vissza, amelyben a listaelemek fordított sorrendben helyezkednek el. Az alábbi utasítások a *reverse* függvény használatát szemléltetik:

```
@lista = reverse(1..5);                 # a @lista tartalma 5,4,3,2,1
@lista = reverse(@lista);               # a @lista új tartalma 1,2,3,4,5
```

A SORT FÜGGVÉNY

A Perl *sort* függvénye sorba rendezi egy lista elemeit. A *sort* függvény szintaxisa:

```
@uj_lista = sort(@lista);
```

vagy

```
@uj_lista = sort(szubrutin @lista);
```

vagy

```
@uj_lista = sort(blokkutasítás @lista);
```

A *sort* függvény az ASCII karaktertábla szerinti sorrendbe rendezi egy lista elemeit. A *reverse* függvényhez hasonlóan a *sort* függvény is új listát ad vissza és nem módosítja az eredetit. Az alábbi utasítások a *sort* függvény egyszerű használatát mutatják be:

```
@lista = sort(1,5,2,3,4);                # a @lista tartalma: 1,2,3,4,5
@lista = sort(1,2,10);                   # a @lista tartalma 1,10,2 (ASCII sorrend)
```

Egy script fájlon belül szubrutinnal vagy blokkal megváltoztathatjuk a rendezési sorrendet. A blokknak össze kell hasonlítania a *sort* által a rendelkezésére bocsátott *$a* és *$b* változót, és a -1, 0 vagy az 1 értéket kell visszaadnia. Az alábbi utasítások a *sort* szubrutin használatát mutatják be:

```
@lista = sort({$a <=> $b} (2,1,10));    # a @lista tartalma 1,2,10
@lista = sort({$b <=> $a} (2,1,10));    # a @lista tartalma 10,2,1

sub hasonlit
  {
    $b <=> $a
  }
@lista = sort(hasonlit (2,1,10));       # a @lista tartalma 10,2,1
```

TÖMBFÜGGVÉNYEK

Amint tudjuk, egy tömb olyan adatszerkezet, amely egy vagy több, azonos típusú értéket tárol – például 100 érdemjegyet vagy 100 tanuló nevét. A Perl számos beépített függvénnyel támogatja a tömbben tárolt értékek kezelését. A fejezet következő részében néhány tömbkezelő függvénnyel ismerkedünk meg.

A PUSH ÉS A POP FÜGGVÉNY

A Perl nyelvű script fájlok arra használják a *push* és a *pop* függvényt, hogy tételeket adjanak egy tömb végéhez, illetve töröljenek onnan. Azt is mondhatjuk, hogy a *push* és a *pop* függvény segítségével a script fájlok LIFO (last in, first out = az utoljára a verembe helyezett érték vehető ki elsőként) típusú veremműveleteket végezhetnek. A *push* függvény egy tételt helyez a tömb végére. A *push* függvény szintaxisa:

```
push (@TOMB, LISTA);
```

Az alábbi utasítások szemléltetik a *push* függvény használatát:

```
@lista = ();
push(@lista, 10, 20);                    # a @lista tartalma most (10, 20)
push(@lista, 1..3);                      # a @lista tartalma most (10, 20, 1, 2, 3)
```

Ezzel ellentétben a *pop* függvény eltávolítja a tömb végéről az oda helyezett elemet, és az eltávolított értéket adja vissza. A *pop* függvény szintaxisa:

```
$ertek = pop(@TOMB);
```

Az alábbi utasítások szemléltetik a *pop* függvény használatát:

```
# feltesszük, hogy megmaradt az előző példából a @lista (10, 20, 1, 2, 3) tartalma

print pop(@lista);                       # a kimenet 3 lesz
print pop(@lista);                       # a kimenet 2 lesz
print pop(@lista);                       # a kimenet 1 lesz

# a @lista tartalma most (10, 20)
```

A SHIFT FÜGGVÉNY

A Perl *shift* függvénye eltávolítja és visszaadja egy tömb első elemét. A függvény hasonlít a *pop* függvényhez, azzal a különbséggel, hogy ez a függvény elején dolgozik (és így egy FIFO listát készít). A *shift* függvény szintaxisa:

```
$ertek = shift(@TOMB);
```

A következő utasítások a *shift* függvény használatát szemléltetik:

```
# feltételezve, hogy a @lista tömb tartalma (10, 20) maradt

print shift(@lista);                     # a kimenet 10 lesz
print shift(@lista);                     # a kimenet 20 lesz

# a @lista tartalma most ()
```

AZ UNSHIFT FÜGGVÉNY

A Perl *unshift* függvénye egy vagy több elemet szúr be egy tömb elejére. Az *unshift* függvény szintaxisa:

```
unshift(@TOMB, lista);
```

Az alábbi utasítások az *unshift* függvény használatát szemléltetik:

```
# feltételezve, hogy a @lista tartalma ()
unshift (@lista, 5, 10, 20);          # a @lista tartalma most (5, 10, 20)
unshift (@lista, 1..3);               # a @lista tartalma most (1, 2, 3, 5, 10, 20)
```

A SPLICE FÜGGVÉNY

A Perl nyelvű script fájlok arra használják a *splice* függvényt, hogy kiemeljenek tételeket egy listából és a helyükre másik lista elemeit szúrják be. A *splice* függvény szintaxisa:

```
slice(@Tömb, eltolás[, számláló[, lista]]);
```

A *splice* függvény a *@Tömb* tömbből az *eltolás* által megadott pozíciótól kezdődően kiemeli a *számláló* paraméterben megadott számú elemet és ezek helyére beszúrja a lista paraméterrel megadott másik *lista* elemeit. Ha a függvény hívásakor nem adjuk meg a *számláló* paramétert, akkor a függvény a tömb végéig terjedően emeli ki az elemeket. Ha a függvény nem definiál helyettesítő listát, akkor nem szúr be elemeket a tömbbe. Az alábbi utasítások szemléltetik a *splice* függvény használatát:

```
@lista = 1..10;
splice(@lista, 1, 8, 5, 6);           # a @lista tartalma (1, 5, 6, 10)
```

A SCALAR FÜGGVÉNY

A Perl *scalar* függvénye egy lista elemeit számolja össze. A *scalar* függvény szintaxisa:

```
eredmény = scalar(lista);
```

A Perl script fájloknak a tömbökkel kapcsolatban általában nincs szükségük a *scalar* függvény használatára, mert amikor egy fájl *skalár* környezetben egy tömbre hivatkozik, a tömb visszaadja elemeinek számát. A fájloknak azokban az esetekben lehet szükségük erre a függvényre, ha az esetek környezete nem egyértelmű, vagy ha a lista nem tömb. A következő utasítások a *scalar* függvény használatát szemléltetik:

```
@lista = 1..10;
print scalar(@lista);                 # a @lista méretét adja eredményül
```

A GREP FÜGGVÉNY

A Perl *grep* függvénye kiszűri egy listából azokat az elemeket, amelyekre egy kifejezés hamis eredményt ad. A *grep* függvény szintaxisa:

```
@lista = grep(kifejezés, lista);
```

A *grep* függvény végiglépked egy lista valamennyi elemén és mindegyik elemre vonatkozóan kiértékel egy adott kifejezést. A *grep* függvény a *$_* változóhoz hozzárendeli a lista aktuális elemét

és kiértékeli a kifejezést. Ha a kiértékelés igaz eredményt ad, akkor a *grep* az illető elemet felveszi az eredő listába. Az alábbi utasítások a *grep* függvény használatát szemléltetik:

```
@lista = grep($_ & 1, 1..10);          # a @lista tartalma (1,3,5,7,9)

@lista = ('a', '', 'b');               # a @lista tartalma ('a', ' ', 'b')
@lista = grep($_ ne '', @lista);       # a @lista tartalma (a, b)
```

Megjegyzés: Ha a kifejezés módosítja a $_ változót, akkor az eredeti lista is módosul.

ASSZOCIATÍV TÖMBÖKET KEZELŐ FÜGGVÉNYEK

Amint láttuk, egy asszociatív tömb olyan tömb, amelynek indexelésére nem numerikus érték, például név is használható. Az asszociatív tömbök kezelésének egyszerűsítése céljából a Perl különböző beépített függvényeket tartalmaz.

A KEYS FÜGGVÉNY

A Perl *keys* függvénye az asszociatív tömbnek megfelelő kulcsokat adja vissza. A *keys* függvény szintaxisa:

```
@kulcs_lista = keys(%tömb);
```

A *keys* függvény reguláris listaként adja vissza a tömb kulcsait. Az alábbi utasítások a *keys* függvény használatát mutatják be:

```
$korok{'Feri'} = 25;
$korok{'Mari'} = 30;
$korok{'Kati'} = 15;
@lista = keys(%korok);                 # a kulcsok 'Kati', 'Mari', 'Feri'
@lista = sort keys %korok;             # a kulcsok 'Feri', 'Kati', 'Mari'

for $kulcs (sort keys %korok)
  {
    # az életkorok megjelenítése
    print "$kulcs eveinek szama $korok{$kulcs}\n";
  }
```

A VALUES FÜGGVÉNY

A *values* függvény normál tömböt ad vissza, amely egy asszociatív tömb értékeit tartalmazza. A *values* függvény szintaxisa:

```
@érték_lista = values(%tömb);
```

A *values* függvény reguláris listaként adja vissza egy asszociatív tömb értékeit. Az alábbi utasítások a *values* függvény használatát mutatják be:

```
# az elõzõ példa értékeit használva
$korok{'Feri'} = 25;
$korok{'Mari'} = 30;
$korok{'Kati'} = 15;

@lista = sort values %korok;          # a @lista tartalma (15, 25, 30)

@lista = %korok;                      # a @lista tartalma ('Kati',15,
                                      # 'Mari',30, 'Feri',25)
```

AZ EACH FÜGGVÉNY

A Perl *each* függvénye sorra előveszi egy asszociatív tömb tételeit. Az *each* függvény szintaxisa:

```
@kulcs_érték = each(%tömb);
```

Amikor egy script fájl meghívja az *each* függvényt, a függvény egy kételemes listát ad vissza, amely az egymáshoz tartozó kulcs/érték párokat tartalmazza. Amikor a függvény a lista végére ér, üres listát ad vissza. A függvény következő hívásakor elölről kezdődik az elemek elővétele. Az alábbi utasítások az *each* függvény használatát mutatják be:

```
# az elõzõ példa értékeit használva
$korok{'Feri'} = 25;
$korok{'Mari'} = 30;
$korok{'Kati'} = 15;

while (($nev, $kor) = each %korok)
  {
    # a nevek és életkorok megjelenítése
    print "$nev eveinek szama $korok{$nev}\n";
  }
```

A DELETE FÜGGVÉNY

A Perl *delete* függvénye eltávolít egy elemet egy asszociatív tömbből. A *delete* függvény szintaxisa:

```
delete $Tömb{kulcs};
```

Az alábbi utasítás a *delete* függvény segítségével a *$Alkalmazottak* asszociatív tömbből eltávolítja a *Feri* kulcsnak megfelelő értéket:

```
delete $Alkalmazottak{Feri};
```

PARANCSSORI ARGUMENTUMOK ELÉRÉSE

A Perlben nagyon egyszerűen érhetik el a script fájlok a parancssori argumentumokat. A Perl egy script fájl minden egyes futtatásakor egy *@ARGV* nevű listaváltozóba helyezi a parancssori argumentumokat. Az alábbi ciklus például sorra veszi a parancssori argumentumokat, és megjeleníti azokat a képernyőn:

```
while ($arg = shift @ARGV)
  {
    print "$arg\n";
  }
```

A KÖRNYEZETI VÁLTOZÓK ELÉRÉSE

A Perlben a környezeti változókhoz is könnyen hozzáférhetnek a script fájlok. A Perl egy script fájl minden egyes futtatásakor egy *%ENV* nevű asszociatív tömbbe másolja a környezeti változókat. Az alábbi utasítás a *%ENV* tömböt használja arra, hogy megjelenítse az aktuális elérési utat:

```
print "$ENV{PATH}\n";                    # megjeleníti az aktuális elérési utat
```

Azon túlmenően, hogy a script fájlok elolvashatják a *%ENV* tömbben lévő környezeti változókat, módosíthatják is a tömb elemeit. A *%ENV* tömbben ilyen módon végzett minden változtatás meg fog jelenni a script fájl által létrehozandó minden leszármazott (gyermek) folyamat környezeti beállításaiban is. Az alábbi utasítás például arra használja a *%ENV* tömböt, hogy megváltoztassa az aktuális elérési utat:

```
$ENV{PATH} = 'c:\\masik_ut;'.$ENV{PATH};
```

Megjegyzés: Egy script fájl által a **%ENV** *tömbben végzett változtatások nem változtatják meg a környezeti változók eredeti beállításait. Amikor a script fájl futása befejeződik, a rendszer környezeti változóinak beállítása változatlan marad.*

FÁJL I/O MŰVELETEK

A Perl programozási nyelvet úgy tervezték meg, hogy könnyedén írhassa és olvashassa a szöveges fájlokat. Ugyanakkor látni fogjuk, hogy fájlok véletlenszerű elérését is lehetővé teszi és bináris fájlokon is végezhetők vele I/O műveletek. A Perl fájlműveletei *fájlazonosítót (file handle)* igényelnek, ami nem más, mint egy, az illető fájlnak megfelelő változó. Alapbeállítás szerint minden Perl nyelvű script fájlhoz három standard fájlazonosító tartozik, amelyeket a Perl a script fájl indításakor automatikusan megnyit. Ezek neve *STDIN, STDOUT* és *STDERR*. Ez a három fájlazonosító megfelel a C programozási nyelvben meglévő *stdin, stdout* és *stderr* fájlazonosítónak. Egy Perl script fájl ezeken túlmenően még további, más fájlokra vonatkozó fájlazonosítókat is meg tud nyitni.

FÁJLOK ÉS MÁS ADATFOLYAMOK MEGNYITÁSA

Ahhoz, hogy egy Perl nyelvű script fájl használni tudjon egy fájlt, az *open* függvénnyel meg kell hívnia azt. Az *open* függvény szintaxisa:

```
open(fájlazonosító[, fájlnév]);
```

A C futásidejű könyvtári *open* függvényétől eltérően a Perl *open* függvényhívásában nincs *mode* paraméter. A Perl a megnyitás módját a fájlnév formátuma alapján határozza meg. A 12.3. táblázat sorolja fel a megnyitás módjára vonatkozó megállapodásokat:

Fájlnév	Művelet
"FÁJL"	Megnyitja olvasásra a FÁJL-t (mint a fopen "r")
"<FÁJL"	Megnyitja olvasásra a FÁJL-t (ua. mint a FÁJL)
">FÁJL"	Létrehozza írásra a FÁJL-t (mint a fopen "w")
">>FÁJL"	Megnyitja hozzáfűzésre a FÁJLT-t (mint a fopen "r+")
"+>FÁJL"	Létrehozza írásra/olvasásra a FÁJL-t (mint a fopen "rw")
"+<FÁJL"	Megnyitja írásra/olvasásra a FÁJL-t (mint a fopen "rw+")
"| CMD"	Megnyit egy csővezetéket egy "CMD" parancsot futtató folyamat felé
"CMD |"	Megnyit egy csővezetéket egy "CMD" parancsot futtató folyamattól

12.3. táblázat. A Perl fájlmegnyitási módjának megállapodásai

Megjegyzés: A csővezetékes adatfolyamot nem minden rendszer támogatja.

Ha egy *open* függvény nem tartalmaz fájlnevet, akkor a Perl abból indul ki, hogy egy *$FileHandle* nevű skalár karakterlánc-változó tartalmazza a kívánt fájl nevét. Amikor a script fájl befejezte a munkáját a fájlon, a *close* függvény meghívásával zárja be azt:

```
close(fájlazonosító);
```

Az alábbi utasítások az *open* és a *close* függvények használatát szemléltetik:

```
open(FajlBe, "teszt.dat") || die;       # olvasásra megnyitja a teszt.dat fájlt
                                        # (input)

open(FajlKi, ">teszt.dat") || die;      # létrehozza a teszt.dat fájlt

$MasikFajl = ">>teszt.dat";
open(Masik, $MasikFajl) || die;         # hozzáfûzésre megnyitja a teszt.dat fájlt

close(FajlBe);
close(FajlKi);
close(Masik);
```

Figyeljük meg, hogy a fájlazonosítók nevét nem előzi meg a szokásos egykarakteres előtag. Mint majd látni fogjuk, a Perl script fájlok a fájlazonosítók nevét karakterláncként egy skalár változóban tárolják, és fájlazonosítóként ezt a skalárt adják át azoknak a függvényeknek, amelyek igénylik ezt. Ha a Perl olyan környezetben találkozik ezzel a karakterlánccal, amely fájlazonosítót igényel, akkor konvertálja az értéket.

Csak MS-DOS környezetben a Perl egy további, *binmode* nevű függvényt is használhat, amely a fájl I/O hozzáférési módját szövegesről binárisra kapcsolja át. A legtöbb rendszer nem tesz különbséget a szöveges és a bináris mód között. Az MS-DOS rendszerben azonban az újsor karakter két karakterből áll (CR+LF, kocsi vissza+soremelés). Mivel a legtöbb program nem vár két karaktert a sorok végén, az I/O rendszernek kell elvégeznie az átalakítást. A *binmode* függvény használatához a megfelelő fájlazonosítónak már nyitva kell lennie. A *binmode* függvény szintaxisa:

```
binmode(Fájlazonosító)
```

ADATOK SORONKÉNTI ÍRÁSA ÉS OLVASÁSA

Egy Perl script fájl legegyszerűbben a *<FÁJLKEZELŐ>* operátor segítségével olvashat sorokat a fájlkezelőből. A Perlben a csúcsos zárójelekkel közrefogott fájlkezelőből *beviteli szimbólum (input-symbol)* lesz. Az alábbi utasítások azt szemléltetik, hogy miként használható egy beviteli szimbólum a *Teszt.dat* nevű fájl tartalmának olvasására és megjelenítésére:

```
open(FajlBe, "Teszt.dat") || die;

while ($sor = <FajlBe>)
  {
    print $sor;                          # a sor megjelenítése
  }
close(FajlBe);
```

Amikor a beviteli szimbólum eléri a fájl végét, a szimbólum a hamis értéket adja vissza, aminek következtében befejeződik a *while* ciklus.

Van egy különleges, <> alakú (üres) beviteli szimbólum, ami meglehetősen furcsán viselkedik, de jól használható. Amikor egy script fájl első alkalommal használja a <> beviteli szimbólumot, akkor a Perl megvizsgálja a script fájl parancssori argumentumait. Ha az *@ARGV* üres, akkor a <> beviteli szimbólum a *STDIN*-ről olvas. Ha viszont az *@ARGV* nem üres, akkor a Perl megnyitja az *@ARGV* tömbben lévő első fájlt, és beolvassa a tartalmát. Miután a script fájl végzett az első beolvasásával, a Perl megnyitja a következőt. A <> beviteli szimbólum csak azt követően ad vissza hamis értéket, miután valamennyi fájl beolvasása megtörtént.

A Perl script fájlok arra is használják a beviteli szimbólumot, hogy egy fájl teljes tartalmát beolvassák egy tömbbe úgy, hogy egy sor egy elemnek felel meg. Az alábbi utasítás például a STDIN fájlazonosítóról olvas be adatokat a *@sorok* tömbbe:

```
@sorok = <STDIN>;
```

Írni ugyancsak könnyen lehet egy fájlazonosítóra. Egyébként minden alkalommal ezt tesszük, amikor a *print* függvényt használjuk. A *print* függvény teljes szintaxisa:

```
print [Fájlazonosító] lista;
```

Ha a *print* függvény nem kap fájlazonosítót, akkor a kimenetét a *STDOUT* nevű standard kimenetre (vagyis a képernyőre) küldi. Az alábbi utasítások azt szemléltetik, miként lehet használni a *print* függvényt arra, hogy adatokat fűzzünk egy kiviteli fájlhoz:

```
open(NaploFajl, '>>naplo.dat') || die;

# dátum bekérése (hónap számának növelése, mivel a 0 értékről indul)
($hh, $nn, $ee) = (localtime(time)) [4,3,5];
$hh = ++$hh;
print NaploFajl "A főnök naplója, Stardate $hh/$nn/$ee\n";
close(NaploFajl);
```

Megjegyzés: A fájlazonosító és a kimeneti lista között nincs vessző.

ADATBLOKKOK OLVASÁSA ÉS ÍRÁSA

A programozók a szöveges fájlokat gyakran szövegfolyamnak is nevezik, mert ezek folyamatos karaktersorozatot alkotnak egészen a fájl-vége jelig. Ha a script fájlunknak olyan fájlon kell dolgoznia, amelyik nem ilyen folyam jellegű, hanem blokkokból áll, akkor a script fájlunk a *sysread* és a *syswrite* függvényt használhatja a rögzített méretű adatblokkok feldolgozásához. A *sysread* és a *syswrite* függvény szintaxisa:

```
$eredmény = sysread(FájlAzonosító, $Vált, Hossz[, Eltolás]);
$eredmény = syswrite(FájlAzonosító, $Vált, Hossz[, Eltolás]);
```

A *sysread* és a *syswrite* függvény egy skalár karakterlánc-változónak adja át az adatait. Mivel a függvények rögzített méretű adatblokkokat dolgoznak fel, az adatok bináris értékeket, továbbá nullákat és fájl-vége jeleket is tartalmazhatnak. Ha a függvényhívás megad (bájtokban) egy fájlon belüli *eltolást,* akkor a függvény előbb az eltolásban megadott abszolút pozícióra áll, mielőtt elvégezné az I/O műveletet.

Adatblokkokon dolgozva jól használhatjuk az alábbi I/O függvényeket:

```
$eredmény = seek(FájlAzonosító, Pozíció, Bázis);
$eredmény = tell(FájlAzonosító);
$eredmény = eof(FájlAzonosító);
```

A Perl *seek* függvénye ugyanarra használható, mint a C nyelv futásidejű függvénykönyvtárának *fseek* függvénye. A *Pozíció* paraméter a *Bázis* paraméterrel megadott helyhez viszonyított, bájtokban megadott hely. A *Bázis* paraméter értéke a következők egyike lehet:

- 0 Keresés a fájl elejétől
- 1 Keresés az aktuális pozíciótól
- 2 Keresés a fájl végétől

A Perl *tell* függvénye ugyanúgy működik, mint a C nyelv futásidejű függvénytárának *ftell* függvénye. A függvény a fájlon belüli aktuális pozíciót adja vissza. Végül az *eof* függvény a C nyelv

futásidejű függvénytárának *feof* függvényéhez hasonlóan igaz vagy hamis értéket ad vissza attól függően, hogy az aktuális fájlpozíció elérte-e a fájl végét.

BINÁRIS ADATOK KEZELÉSE

Bár a Perl elsősorban szövegfeldolgozásra van „kihegyezve", bináris adatok is kezelhetők vele. A script fájlok a bináris adatokat karakterlánc-változók segítségével blokkokban mozgathatják, és bájtokra vonatkozó I/O műveleteket hajthatnak végre rajtuk (a *sysread* és a *syswrite* függvény révén). Ahhoz azonban, hogy bármilyen „hasznos" műveletet végezzünk, a script fájlnak az adatokat először át kell alakítania a Perl eredendő, skalár formátumára, majd ugyanezt visszafelé is meg kell tennie.

BINÁRIS ADATOK TÁROLÁSA A PERL NYELVBEN

Amikor egy script fájl a *sysread* függvény segítségével beolvas egy bináris adatblokkot, akkor a függvény az adatblokkot egy skalár karakterlánc-változóba helyezi el. A Perl nem törődik azzal, hogy az adatblokk tartalmaz-e nem ASCII értékeket vagy akár nullákat is. A Perl nem tesz különbséget az egyes bájtok között. A C nyelvtől eltérően a Perl nem egy lezáró nulla karakter alapján határozza meg a karakterlánc hosszát. Ha az adatok történetesen ASCII karakterek, akkor a script fájl úgy tekintheti ezeket, mintha normál karakterláncot alkotnának. Ha viszont az adatok bináris értékeket tartalmaznak, akkor a script fájlnak először ki kell bontania őket, mielőtt a Perl kezelhetné az adatokat.

BINÁRIS ADATOK KIBONTÁSA PERL VÁLTOZÓKBA

Ahhoz, hogy egy script fájl hozzáférhessen bináris adatokhoz, a fájlnak először *ki kell bontania* ezeket a Perl eredendő skalár formátumába. A script fájlok az adatokat az *unpack* függvény segítségével bonthatják ki. Az *unpack* függvény szintaxisa:

```
$eredmény = unpack(Sablon, Kifejezés);
```

A *Kifejezés* általában olyan karakterlánc-változó, amely a *sysread* függvénnyel beolvasott bináris adatokat tartalmazza, de bármilyen más kifejezés is lehet, ami karakterláncként értelmezhető. A *Sablon* az a karakterlánc, amely megadja, hogy miként kell értelmezni a kifejezésben lévő értékeket. Az alábbi utasítások az *unpack* függvény használatát szemléltetik:

```
($r, $g, $b) = unpack("C3", $szinek);   # három előjel nélküli karaktert bont ki
@longwords = unpack("L*", $adatok);     # longword típusú adatok listáját
                                        # bontja ki
@valami = unpack("S2L", $bin);          # két short és egy long típusú adatot
                                        # bont ki
```

Az *unpack* függvényben minden egyes sablonkarakter után beírható egy számláló. Ha számlálóként a csillag (*) karaktert írjuk be, akkor a művelet a karakterláncban még meglévő valamennyi adatra kiterjed. Ha nincs megadva számláló, akkor a művelet csak egy adatra vonatkozik. A script fájl tetszés szerinti számú sablonkaraktert tehet egymás mellé a *Sablon* karakterláncban. A 12.4. táblázat felsorolja a *Sablon* karaktereit és röviden leírja a szerepüket.

Sablonkarakter	Leírás
a	Kitöltés nélküli, nullokat esetleg tartalmazó ASCII karakterlánc
A	Kitöltés nélküli, nullokat kiszűrt ASCII karakterlánc
b	Bit karakterlánc (először a legkisebb helyiértékű)
B	Bit karakterlánc (először a legnagyobb helyiértékű)
c	Előjeles karakter (bájt)
C	Előjel nélküli karakter (bájt)
d	Dupla pontosságú lebegőpontos érték
f	Egyszeres pontosságú lebegőpontos érték
h	Hexadecimális karakterlánc (először a legkisebb helyiértékű számjegy)
H	Hexadecimális karakterlánc (először a legnagyobb helyiértékű számjegy)
i	Előjeles integer (egész szám)
I	Előjel nélküli integer (egész szám)
l	Előjeles long word
L	Előjel nélküli long word
n	Short integer hálózati sorrendben (először a nagyobb helyiértékű)
N	Long integer hálózati sorrendben (először a nagyobb helyiértékű)
P	Mutató karakterláncra
s	Előjeles short integer
S	Előjel nélküli short integer
u	Undecode-olt karakterlánc
v	Short integer VAX sorrendben (először a kisebb helyiértékű)
V	Long integer VAX sorrendben (először a nagyobb helyiértékű)
x	Ugrás előre egy bájttal
X	Ugrás hátrafelé egy bájttal
@	Ugrás a karakterláncon belül megadott helyre

12.4. táblázat. A Perl sablonkarakterei

ADATOK BECSOMAGOLÁSA BINÁRIS ADATLÁNCOKBA

Bináris adatok kiviteléhez a script fájlnak a skalár értékeket bináris karakterláncba kell becsomagolnia. A fájl ezt a *pack* függvény segítségével teheti meg, amelynek szintaxisa:

```
$eredmény = pack(Sablon, Lista);
```

Az alábbi utasítások a *pack* függvény használatát szemléltetik:

```
$szinek = pack("C3", $r, $g, $b);
$adatok = pack("L*", @longwords);
$bin = pack("S2L", @valami);
```

A *pack* függvényben ugyanazok a sablonkarakterek használhatók, mint az *unpack* függvényben az alábbiak kivételével:

- a Kitöltés nélküli, nullokat esetleg tartalmazó ASCII karakterlánc
- A Kitöltés nélküli, nullokat kiszűrt ASCII karakterlánc
- u Undecode-olt karakterlánc
- x Ugrás előre egy bájttal
- X Ugrás hátrafelé egy bájttal
- @ Ugrás a karakterláncon belül megadott helyre

KÖNYVTÁR-INFORMÁCIÓK BEOLVASÁSA PERLBEN

A fájlkezelő műveletek mellett a Perl számos jól használható könyvtárkezelő függvényt is a rendelkezésünkre bocsát. A fejezet következő részében néhány ilyen függvényt vizsgálunk meg részletesebben.

AZ OPENDIR, READDIR ÉS CLOSEDIR FÜGGVÉNY

Fájlok megnyitásához és olvasásához hasonlóan a Perl azt is lehetővé teszi, hogy a script fájlok könyvtárakat nyissanak meg és beolvassák a bennük tárolt fájlok nevét. Valamely könyvtár az *opendir* függvény segítségével nyitható meg, amelyben meg kell adni egy könyvtár-azonosítót és a könyvtár elérési útját. A könyvtárlistából a *readdir* függvény segítségével olvasható be egy fájl. Végül a script fájl a *closedir* függvény meghívásával zárhatja be a könyvtárlistát. Az alábbi példa az aktuális könyvtárban lévő fájlok listájának megjelenítésére használja a *readdir* függvényt:

```
opendir(Dir, $INC[2]) || die;

while ($fajl = readdir(Dir))
  {
    print "$fajl\n";
  }

closedir(Dir);
```

A fenti példa a *$INC[2]* változót használja az aktuális könyvtár eléréséhez. Ha ezt a változót a *$ARGV[0]* változóra cseréljük, akkor a script fájl a parancssorban megadott könyvtár fájljait listázza ki.

A fent bemutatott könyvtárkezelő függvények mellett a Perl még olyan függvények használatát is lehetővé teszi, amelyek segítségével az aktuális mutatónak a könyvtárlistán belüli helyét határozhatjuk meg:

```
$eredmény = rewinddir(KönyvtárAzonosító);
$eredmény = telldir(KönyvtárAzonosító);
$eredmény = seekdir(KönyvtárAzonosító, Pozíció);
```

SCRIPT FÁJLOK KIVITELÉNEK FORMÁZÁSA

A fejezet eddigi részében különböző lehetőségeket láttunk arra, hogy miként tudja megformázni a kivitelét egy script fájl a *print* függvény segítségével. A C programozási nyelvhez hasonlóan a Perl is lehetővé teszi, hogy a *printf* és a *sprintf* függvényeket használva formázott kivitelt állítsunk elő. Ezen túlmenően a Perl nyelvnek jelentéskészítő képességei is vannak, amelyek segítségével a script fájlok űrlapsablonok alapján oszlopokba rendezett jelentéseket készíthetnek.

EGYSZERŰ KIVITEL KÉSZÍTÉSE A PRINT FÜGGVÉNNYEL

A fejezet eddigi részeiben sűrűn használtuk a *print* fügvényt. A Perlben van azonban néhány olyan különleges változó, amelyek révén befolyásolhatjuk a *print* függvény működését. A 12.5. táblázat röviden felsorolja ezeket a különleges változókat:

Változó	Rendeltetése
$,	Listaelemek elválasztása
$"	Listaelemek elválasztása karakterlánc-behelyettesítésben
$\	Rekordok elválasztása

12.5. táblázat. A print függvény kiviteli formátumát módosító különleges változók

Ezekhez a különleges változókhoz hozzárendelhetjük azt az értéket, amelyet egy adott feladat elvégzéséhez a kivitelben meg akarunk jeleníteni. Az alábbi példa a *$,* változót használja a listaelemeket egymástól elválasztó karakter meghatározására:

```
$, = '*';
@lista = 1..10;
print @lista;              # a képernyõn az 1*2*3*4*5*6*7*8*9*10 lista jelenik meg
```

A *$,* változó a valóságban minden karakterláncra, s nemcsak a *print* függvénynek átadott karakterláncra vonatkozik. Ezt a változót azonban legtöbbször a *print* függvény által használt érték megváltoztatására használjuk.

KÜLSŐ PROGRAMOK MEGHÍVÁSA PERL SCRIPT FÁJLBÓL

A Perl lehetővé teszi, hogy egy script fájl külső programokat hívjon meg. Arról azonban ne feledkezzünk meg, hogy ha engedélyezzük egy script fájlnak rendszerparancsok kiadását, akkor ezzel veszélyeztetjük rendszerünk biztonságát. Általános szabályként tartsuk ahhoz magunkat, hogy script fájlból ne adjunk ki külső parancsokat. Ha mégis végre *kell* hajtania egy fájlnak valamilyen külső parancsot, akkor ehhez a *system, exec* vagy a *fork* beépített függvényeket kell meghívni.

FORMÁZOTT KIVITEL KÉSZÍTÉSE A PRINTF FÜGGVÉNNYEL

A Perlben használhatjuk a *printf* és a *sprintf* függvényeket, amelyek nagyon hasonlítanak a C nyelv azonos nevű és futásidejű függvénytári függvényeihez. A függvények szintaxisa:

```
$eredmény = printf([FájlAzonosító] Formátum, Lista);
$eredmény = sprintf(Formátum, Lista);
```

A *printf* függvény formázott kivitelt küld az alapértelmezés szerinti STDOUT fájlazonosítóra, a *sprintf* függvény pedig formázott karakterláncot ad vissza. A *Formátum* karakterlánc mindkét függvénynél majdnem azonos a C nyelv hasonló karakterláncával, azzal a különbséggel, hogy a Perl függvények nem használják a hossz jelzőjét (*). Az alábbi utasítások a *printf* és a *sprintf* függvény használatára mutatnak példát

```
$pontossag = 2;
$pi = 3.1415;
printf("%.2f\n", $pi);                    # a kivitel 3.14 lesz
printf(".${pontossag}f", $pi;             # a kivitel 3.14 lesz
```

REGULÁRIS KIFEJEZÉSEK

A fejezet eddigi részeiben már láttunk néhány példát kereső függvényekre. E függvények jó néhánya úgynevezett *reguláris kifejezések* alapján végzi a keresést. Amint látni fogjuk, a Perl nyelvű script fájlok sűrűn használnak reguláris kifejezéseket szövegek kezeléséhez. A továbbiakban néhány ilyen kifejezéssel ismerkedünk meg.

A REGULÁRIS KIFEJEZÉSEK ÁTTEKINTÉSE

A *reguláris kifejezés* a számítástechnikai szakzsargonban használt fogalom, amely valamilyen karaktermintát jelent. A Perl script fájljai ilyen karaktermintákat használnak arra, hogy az adatok meghatározott szakaszokra bontásával leegyszerűsítsék a beérkező adatok elemzését. Egy script fájl a beérkező adatokat gyakran az úgynevezett kitöltő karakterek (szóköz, tabulátor, vessző vagy valamilyen más, ismert elválasztó karakter) alapján is tudja elemezni. Ha azonban a beérkező adatok formátuma kötetlen, vagy a szakaszok definíciója túl bonyolult, akkor a reguláris kifejezések jelentik a megfelelő megoldást.

A REGULÁRIS KIFEJEZÉSEK SZINTAXISA

A Perl az általában használatos reguláris kifejezések többségét támogatja. A Perlben úgy határozták meg a különleges karaktereket, hogy a lehető legkevesebb slash (\) karaktert kelljen használni. A 12.6. táblázat néhány olyan karaktert sorol fel, amelyeket a script fájlok reguláris kifejezésekben használhatnak.

Karakter	Leírás
.	Bármelyik karakterrel megegyezik (az újsor karaktert kivéve)
(...)	Csoportosít egy elemsorozatot
+	Az előtte álló karakterrel egyezik meg, egyszer vagy többször
?	A mintával nullszor vagy egyszer egyezik meg
*	A mintával nullszor vagy többször egyezik meg
[...]	Az adott készlet valamelyik karakterével egyezik meg

Karakter	Leírás
[^...]	A negált készlet valamelyik karakterével egyezik meg
(...|...|...)	Az adott választási lehetőségek valamelyikével egyezik meg
^	A karakterlánc elejétől kezdődik az egyezőség vizsgálata
$	Egy karakterlánc végén lévő mintával egyezik meg
[n,m]	A mintával n-től m-szer egyezik meg
[n]	A mintával pontosan n-szer egyezik meg
[n,]	A mintával legkevesebb n-szer egyezik meg
\n\t stb.	Újsorral, tabulátorral stb. egyezik meg
\b	Szóhatárral egyezik meg
\B	Szóhatáron belül egyezik meg
\d	Egy számjeggyel egyezik meg
\D	Nem számjeggyel egyezik meg
\s	Fehér térközzel (white space) egyezik meg
\S	Nem fehér térközzel egyezik meg
\w	Alfanumerikus karakterrel egyezik meg
\W	Nem alfanumerikus karakterrel egyezik meg

12.6. Reguláris kifejezésekben használható karakterek

A Perl a reguláris kifejezéseket (mintákat) ferde vonalakkal fogja közre, mint például /kifejezés/. Az alábbi bejegyzések a Perl reguláris kifejezéseire mutatnak példát:

```
# az alábbi reguláris kifejezések igazak, ha:
/ap/                          # a karakterlánc tartalmazza az 'ap' (rész)láncot

/(k|l|n)ap/                   # a karakterlánc tartalmazza a 'kap', 'lap' vagy 'nap
                              # (rész)láncot

/[0-9]+/                      # a karakterlánc tartalmaz számjegyet

/[A-Za-z] [A-Za-z0-9_]*/      # a karakterlánc egy azonosítót tartalmaz
```

Ne aggódjunk, ha ezek a kifejezések még nem tűnnek világosnak. Rövidesen részletesebben is kitérünk néhányukra. Most elegendő annyit megjegyeznünk, hogy a Perlben a reguláris kifejezéseket ferde vonalak közé kell tennünk.

KULCSSZAVAK KERESÉSE REGULÁRIS KIFEJEZÉSEKKEL

A Perl karakterláncok összehasonlításának leegyszerűsítéséhez használja a reguláris kifejezéseket. Ha meg szeretnénk állapítani, hogy egy karakterlánc tartalmaz-e egy mintát, akkor ehhez az alábbi módon használhatunk egy reguláris kifejezést:

```
if ($str =~ /minta/);
```

A fenti példában a kifejezés kiértékelése igaz eredményt ad, ha a minta megtalálható a karakterláncban *($str)*. Ha a karakterlánc nem tartalmazza a mintát, akkor a kifejezés kiértékelése hamis eredményt ad. Az alábbi utasítás például azt vizsgálja, hogy a karakterlánc tartalmazza-e *A Web programozása* szöveget:

```
if ($str =~ /A Web programozása/;
```

A teljes egyezőség vizsgálatához a kifejezés az összehasonlítást a karakterlánc elejéhez és a végéhez rögzítheti. Az alábbi kifejezés kiértékelése például akkor és csak akkor ad igaz eredményt, ha a *$str* a „banana", „bananana" vagy a „banananana" szövegek valamelyikét tartalmazza:

```
($str =~ /^ba(na){2,4}$/);
```

Hasonlóképpen az alábbi kifejezés akkor és csak akkor lesz igaz, ha a *$str* változó a „vas" szót tartalmazza, és nem része más olyan szónak, mint például „havas".

```
($str =~ /\bvas\b/);
```

BEVITELI ADATOK ELEMZÉSE REGULÁRIS KIFEJEZÉSEKKEL

Bonyolultabb Perl script fájlokban gyakran többet szeretnénk tudni annál, hogy egy karakterlánc megegyezik-e egy adott mintával. Előfordulhat például, hogy egy karakterláncból bizonyos értékeket ki akarunk emelni. Egy reguláris kifejezésben a () csoportosító karakterek segítségével a script fájl kiemelhet bizonyos értékeket a karakterláncból, és egy listához rendelheti azokat. Az alábbi utasítások például reguláris kifejezés segítségével emelik ki a hónapot, napot és évet egy karakterláncból:

```
$str = " January 1, 1997, ";
($hh,$nn,$ee) = $str =~ /\s*(S*)\s+(\d+)\D+(>d{4})/;
```

A fenti példában lévő reguláris kifejezést a következőképpen értelmezhetjük:

- Ugorja át az összes vezető fehér térközt
- Helyezze az összes nem fehér térköz-karaktert a $hh változóba (hónap)
- Ugorja át a fehér térköz karaktert
- Helyezze az összes számjegyet a $nn változóba (napok száma)
- Ugorjon át minden nem számjegy karaktert
- Helyezzen 4 számjegyet a $ee változóba (évek száma)

A fenti eredményt más összeállítású reguláris kifejezéssel is megkaphatjuk, melyek közül egyesek lehetnek általánosabbak, míg mások pontosabbak. Összeállíthatnánk például olyan kifejezést, amelynek többféle dátumformátum feleltethető meg, de olyant is, amelyik csak meghatározott hónapneveket enged meg. A Perl nyelvben a mintaillesztő operátornak (=~) az az alakja is használható, amelyik negálja az eredményt (!~). Ez az operátor ekvivalens a !($str =~ /minta/) kifejezéssel.

KARAKTERLÁNCOK KERESÉSE ÉS CSERÉJE REGULÁRIS KIFEJEZÉSEKKEL

Az eddigiekben script fájljaink csak a mintaillesztő operátort használták. Ezen túlmenően a Perl két másik reguláris kifejezést is használ, amelyek a karakterláncot a mintára cserélik. Az első esetben a Perl a karakterláncnak a mintával megegyező részét a csere karakterláncra cseréli:

```
$str =~ s/minta/csere/;
```

Az alábbi utasítás például a „colour" szó egyetlen előfordulását a „color" alakra cseréli:

```
$str =~ s/\bcolour\b/color/;
```

Némileg módosítva az előző kifejezést az utasítás a „colour" szó valamennyi előfordulását a „color" alakra cseréli:

```
$str =~ s/\bcolour\b/color/g;
```

A fenti példában a kifejezés végén álló *g* arra utasítja a Perlt, hogy globális cserét hajtson végre.

Az *i* toldalék használatával arra utasíthatjuk a kifejezést, hogy a kis- és a nagybetűs írásmódtól függetlenül hajtsa végre a keresést (nem megkülönböztetve a kis- és a nagybetűket).

A mintaillesztő operátor második módosító alakja a sima egyeztetés helyett a cserét is elvégzi:

```
$str =~ tr/KeresõLista/LecserélõLista/;
```

Az alábbi utasítás például egy kisbetű összes előfordulását a nagybetűs megfelelőjére cseréli:

```
$str =~ tr/a-z/A-Z/;                    # nagybetûsre változtatja a $str karakterláncot
```

Tekintsük az alábbi példákat:

```
$betuk = "abcde";
print "$betuk\n";                       # az abcde jelenik meg

$betuk =~ tr/a-z/A-Z/;
print "$betuk\n";                       # az ABCDE jelenik meg
```

ÖSSZEFOGLALÁS

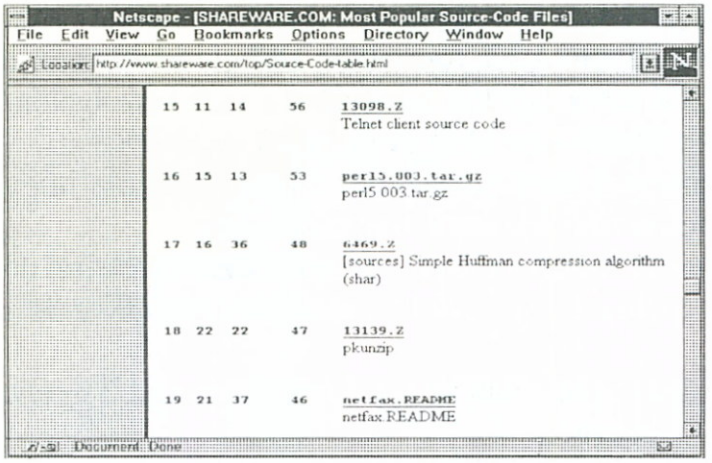
Ebben a fejezetben megismerkedtünk a Perl nyelvű programozás alapjaival. Az itt tanultakat hasznosítva sokat tudó CGI script fájlokat írhatunk Perl nyelven. A 13. fejezetben eddigi ismereteinkre építve megtudjuk, hogyan készíthetünk Perl nyelvű CGI script fájlokat, amelyeket a saját kiszolgálónkon futtathatunk. Mielőtt azonban továbblépnénk, győződjünk meg arról, hogy értjük az alábbiakat:

- A Perl interpreter típusú programozási nyelv, amelyet arra használnak a programozók, hogy script fájlokat írjanak a Weben és az Interneten felmerülő feladatok megoldásához.
- A Perl programszerkezetei sok esetben nagyon hasonlítanak a C nyelv programszerkezeteire, de a Perl számos olyan képességgel is rendelkezik (például egyszerű karakterlánc- és fájlkezelési képességek), amelyek hiányoznak a C nyelvből.
- A Perl nyelv rugalmassága, tömörsége és biztonságos volta miatt az egyik legalkalmasabb nyelv, amelyen CGI programokat írhatunk a Webre és az Internetre.

PERL NYELVVEL FOGLALKOZÓ FONTOSABB HELYEK

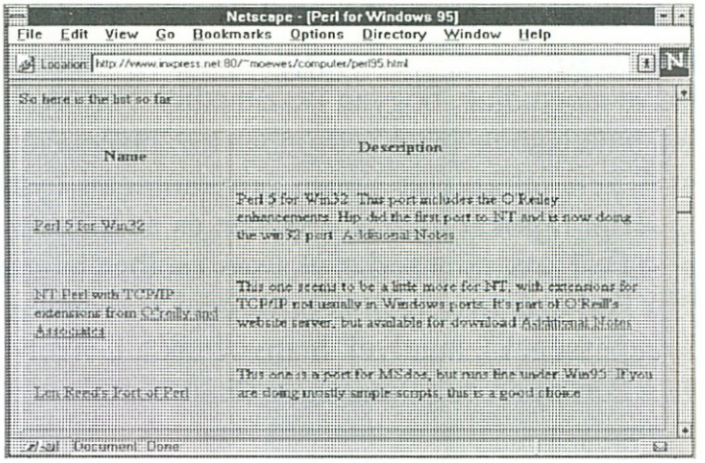
A következő Web-helyek segíthetnek abban, hogy részletesen olvashassunk Perl script fájlokról, valamint a Perl erőforrásairól és függvénytárairól. A helyeket kiindulási pontként is használhatjuk a Web felfedezéséhez.

SHAREWARE:COM Most Popular Source-Code Files



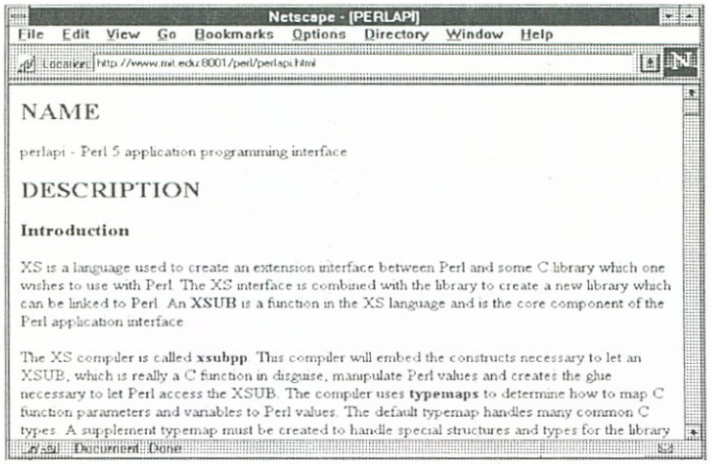
http://www.shareware.com/top/Source-Code-table.html

Perl for Windows 95



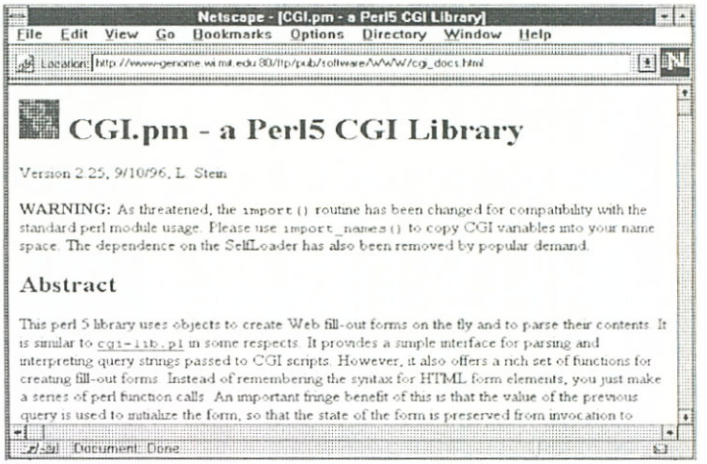
http://www.inxpress.net:80/~moewes/computer/perl95.html

PERLAPI



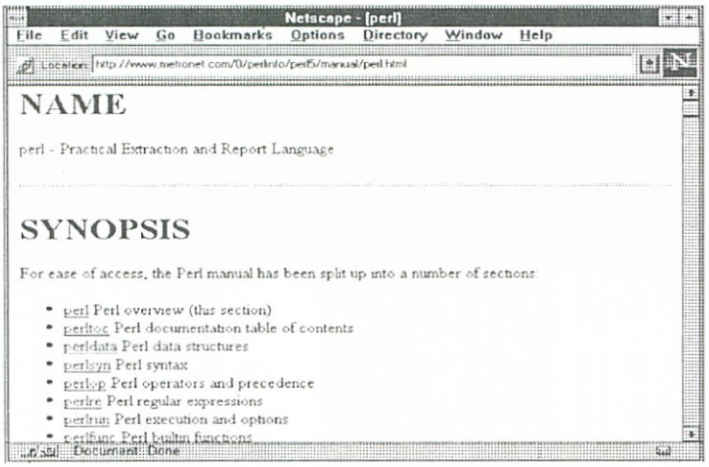
http://www.mit.edu:80001/perl/perlapi.html

CGI.pm – a Perl5 CGI Library



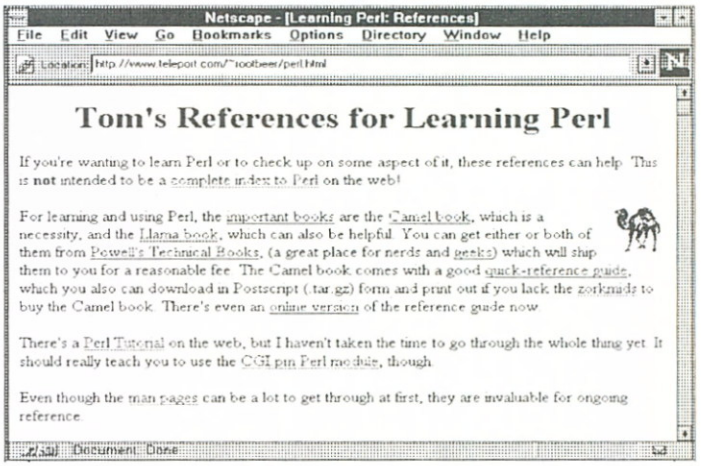
http://www.genome.wi.mit.edu:80/ftp/pub/software/WWW/cgi_docs.html

Perl



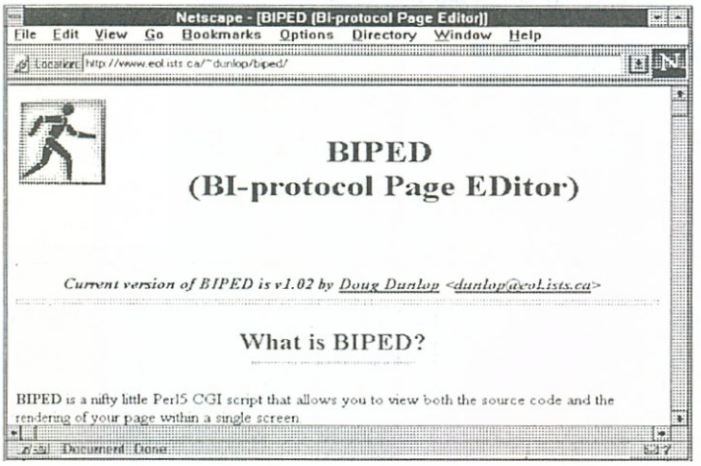
http://www.metronet.com/0/perlinfo/perl5/manual/perl.html

Learning Perl References



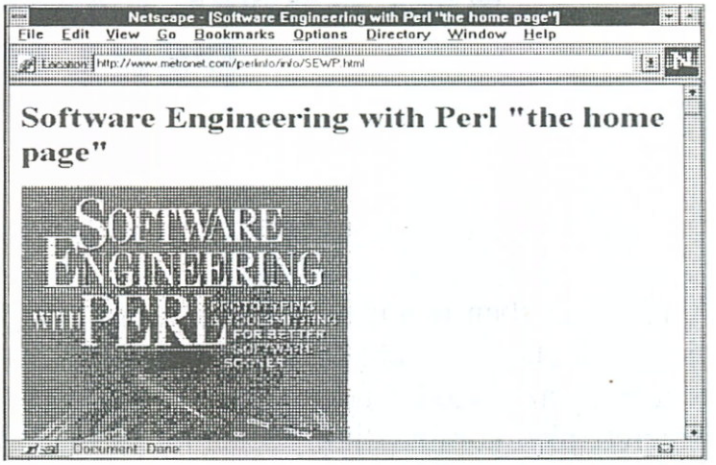
http://www.teleport.com/~rootbeer/perl.html

BIPED (Bi-protocol Page Editor)



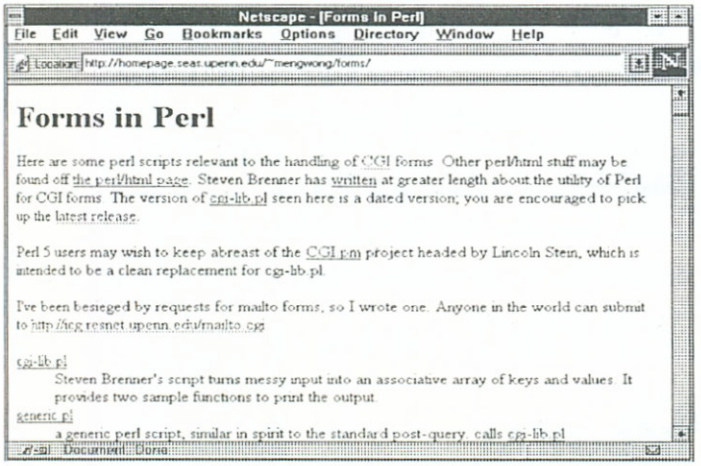
http://wwweol.ists.ca/~dunlop/biped/

Software Engineering with Perl



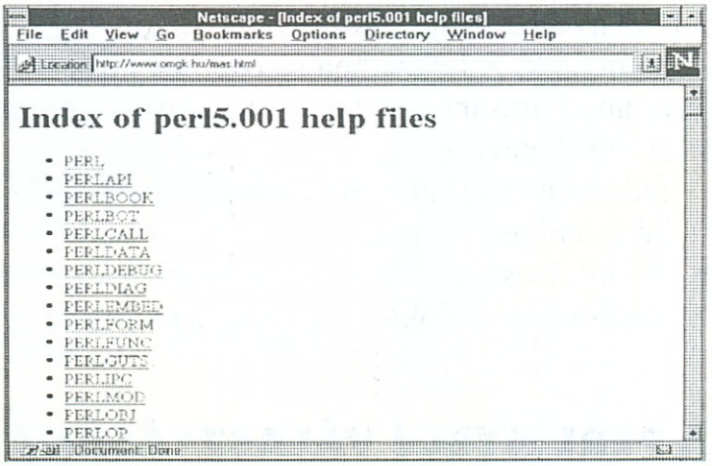
http://www.metronet.com/perlinfo/info/SEWP.html

Forms in Perl



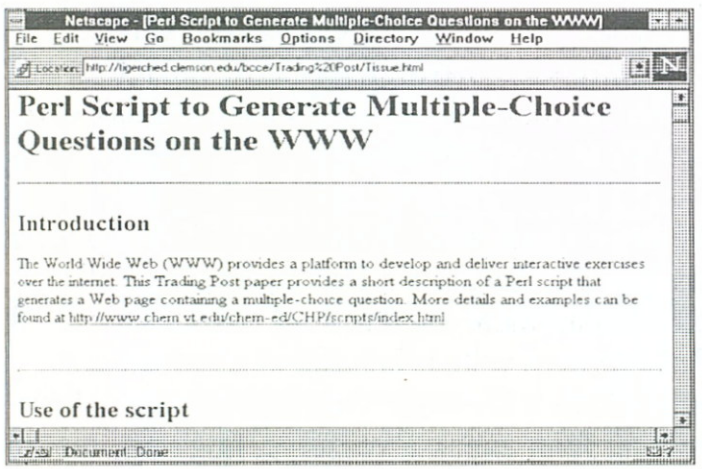
http://homepage.seas.upenn.edu/~mengwong/forms/

Index of perl5.001 help files



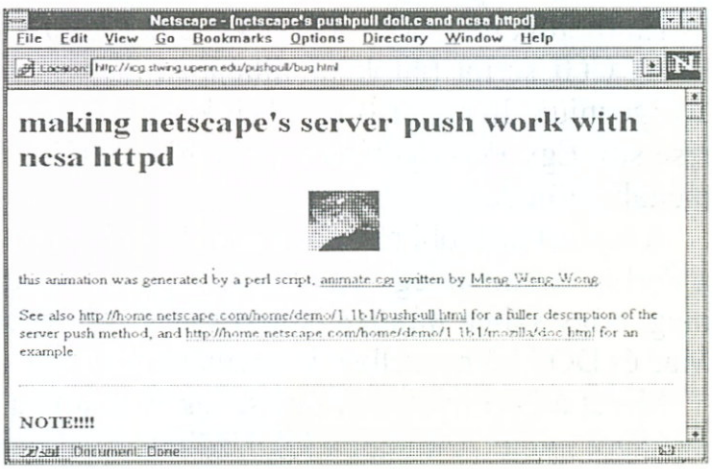
http://www.omgk.hu/mas.html

Perl Script to Generate Multiple-Choice Questions



http://tigerched.clemson.edu/bcce/Trading%20Post/Tissue.html

Perl Animation Script



http://icg.stwing.upenn.edu/pushpull/bug.html

13. FEJEZET

CGI SCRIPT FÁJLOK KÉSZÍTÉSE PERL NYELVEN

A 12. fejezetben megismerkedtünk a Perl programozási nyelv alapjaival. Ebben a fejezetben azt vizsgáljuk, hogy miként használhatjuk a Perl nyelvet CGI script fájlok írására. Lehetőségünk nyílik arra, hogy az előző két fejezetben a Perl nyelvvel és a CGI programokkal kapcsolatban megszerzett ismereteinket kombináljuk. Menet közben néhány Perl programozási trükköt is megtanulunk.

Amint tudjuk, CGI script fájlokat szinte bármilyen programozási nyelven írhatunk. Ahhoz, hogy sikeres Web-programozók legyünk, nem csak azt kell tudnunk, hogy miként írjunk Web-programokat, hanem azt is, hogy egy adott feladat megoldásához milyen eszközöket válasszunk. Ha a feladathoz a legalkalmasabb nyelvet (eszközt) választjuk, akkor jelentősen megnövekedhet programjaink hatékonysága, továbbá időt és fáradságot takaríthatunk meg. Ha a CGI script fájlok írásához a Perl nyelvet választjuk, akkor viszonylag rövid idő alatt robusztus és professzionális fájlokat készíthetünk. A fejezetben elsősorban arról tanulunk, hogy miként használjuk a Perl nyelvet CGI programok írásához. A fejezet végére érve érteni fogjuk az alábbiakat:

- A Perl hatékony eljárást kínál robusztus script fájlok készítéséhez anélkül, hogy – más programozási nyelvekhez, mint például a C és a C++ nyelvhez képest – nagy terjedelmű kódot kellene készítenünk.
- A Perl 5-ös verziója számos olyan új képességgel bővült, amelyek támogatják az objektumorientált script fájlokat és javítják a változók hatókörének kezelését.

A PERL NYELV VÁLASZTÁSÁNAK INDOKAI

Az előző két fejezet áttanulmányozása után már eléggé világosan megfogalmazhatjuk magunknak a CGI script fájlok mibenlétét, és számba vehetjük azokat a nagy teljesítményű szövegfeldolgozási képességeket is, amelyek a Perl nyelvet alkalmassá teszik CGI script fájlok kifejlesztésére. A Perl nyelv használatát emellett még olyan körülmények is indokolják, mint az adatbázisok támogatása, a hordozhatóság és a hálózati biztonság.

Említettük már, hogy a Perl a jelenleg létező legnagyobb tudású szövegfeldolgozó nyelvek egyike. A CGI script fájloknak általában nagyon nagy mennyiségű szövegfeldolgozó műveletet kell elvégezniük. Ilyen művelet a beérkező adatok elemzése, adatbázisok elérése, HTML oldalak készítése stb. Egy Perl nyelven megírt átlagos CGI script fájl mérete egy hasonló, C++ nyelvű programénak a töredéke.

A fejezet későbbi részében látni fogjuk, hogy a Perl támogatja az adatbázisok elérését. Sőt maga a Perl is tartalmaz egy egyszerű, beépített adatbázis-kezelőt. Továbbá a Perl nyelvet úgy alkották meg, hogy szinte minden ismert számítógépes platformra átvihető legyen, s így Windows, UNIX, Mac és DOS környezetben is használni lehessen. A Perl ingyenes volta is hozzájárul a vonzerejéhez.

Mivel a Perl nyelvben nincsenek mutatók, a nyelv a C és a C++ nyelvhez képest nagyobb biztonságot nyújt. A mutatók kiküszöbölésével már nem áll fenn annak a veszélye, hogy egy mutató „eltéved" és memóriaelérési problémákat okozva sebezhetővé teszi a rendszert. A Perl hatékony reguláris kifejezéseit használva könnyen megvizsgálhatjuk a beérkező adatok épségét, és kiszűrhetjük az olyan, beágyazott escape-sorozatokat, amelyek károsíthatnák a rendszerünket. Végül a biztonság további fokozása céljából a Perl egy speciális *taintperl* nevű verziója megakadályozza, hogy bármelyik beérkező adat (ami „szennyezettnek" tekintendő) eljuthasson egy rendszer-

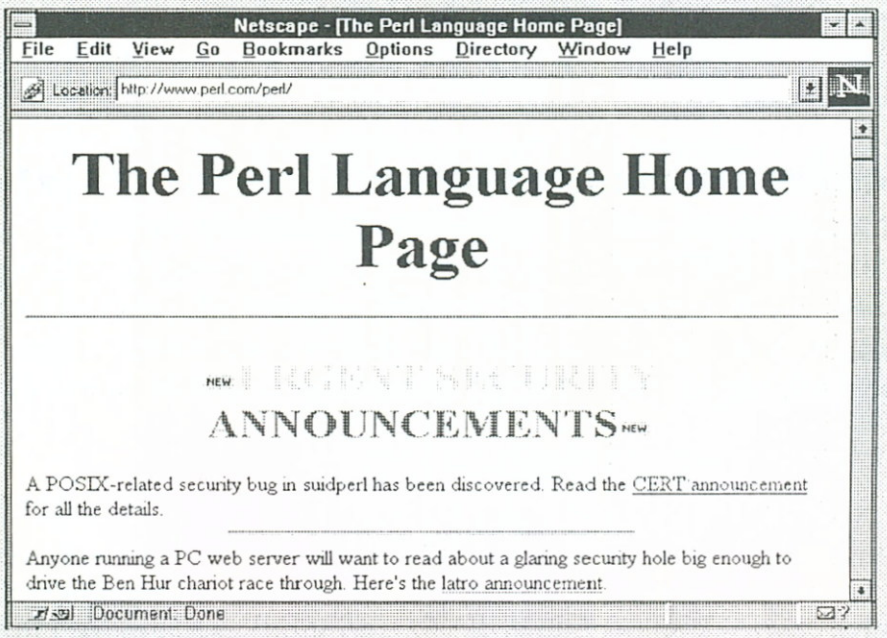
parancshoz. A *taintperl* verzióról a fejezet végén felsorolt Web-helyeken juthatunk további információkhoz.

A PERL 4 ÉS A PERL 5 VERZIÓ

A 12. fejezetben részletesen megvizsgáltuk a Perl programozási nyelvet. A fejezetben bemutatott képességek a Perl 4-es verziójára vonatkoztak. Azért választottuk ezt a verziót, mert ennek az elemei azonnal és változtatás nélkül használhatók az 5-ösben is, továbbá mert a Weben jelenleg található Perl script fájlok többsége alapvetően a 4-es verzió szerkezeteiből építkezik. A Perl 5-ös verziója számos új, kulcsfontosságú képességgel gazdagodott. Néhány ezek közül:

- Beépített nyelvi támogatás, ami lehetővé teszi Perl rutinok C/C++ programokba történő beágyazását és fordítva.
- Szélesebb körű reguláris kifejezések használata a szövegfeldolgozásban.
- Változók hatókörének javított kezelése.
- Egymásba ágyazott adatszerkezetek.
- Objektumorientált programozás és az öröklés támogatása csomagok segítségével.

További információkhoz juthatunk a Perl 5-ös verziójáról, ha letöltjük a Perl 5 dokumentációját a *http://www.perl.com/perl* helyről (lásd a 13.1. ábrát).



13.1. ábra. A Perl 5-ös verziójával kapcsolatos információk letöltése

PERL NYELVŰ CGI SCRIPT FÁJL MEGHÍVÁSA

UNIX alapú rendszerekben egy Perl nyelvű script fájl végrehajtható fájlként futtatható. Másként fogalmazva ez azt jelenti, hogy egy script fájl meghívásához semmiféle különleges dolgot nem kell tenni. Ezzel szemben DOS és Windows alapú rendszerekben egyes kiszolgálók nem hajtják végre automatikusan a script fájlokat. Ilyen esetben egy parancsfájlt (batch) kell írnunk, amely meghívja a script fájl futtatásához szükséges Perl parancsot. Más rendszerekben esetleg tanulmányoznunk kell a HTTP kiszolgáló dokumentációját és az ott leírtak szerint kell meghívnunk a Perlt.

A legtöbb HTTP kiszolgáló azt várja, hogy a CGI script fájlok egy *cgi-bin* nevű könyvtárban legyenek. Egy script fájlt ekkor az alábbi URL megadásával hívhatunk meg:

```
http://saját-tartomány/cgi-bin/saját-script
```

Megjegyzés: Az URL a **cgi-bin** *könyvtárat adja meg, de az aktuális hely bárhol lehet a rendszerben. A helyet a HTTP kiszolgáló telepítésekor kell megadni.*

PERL NYELVŰ CGI SCRIPT FÁJL MEGHÍVÁSA UNIX ALATT

Ha a script fájlunkat a *cgi-bin* könyvtárba helyezzük, és a fájlt végrehajthatóvá tesszük, akkor a felhasználók a fájlt közvetlenül az URL megadásával hívhatják meg a fenti példa szerint – de ehhez még néhány lépést kell tennünk. Első lépésként a script fájl elejére be kell szúrnunk azt a bejegyzést, amely a fájl tartalmát Perl programként azonosítja:

```
#!/usr/bin/perl
```

Megjegyzés: Attól függően, hogy a Perlt a rendszerünk mely részére telepítettük, az elérési út ettől eltérő lehet.

Második lépésként a fájlt végrehajthatóvá kell tennünk úgy, hogy a *chmod* parancs segítségével magasra állítjuk a végrehajtási bitet a fájlengedélyekben:

```
chmod +x script-fájlunk
```

PERL NYELVŰ CGI SCRIPT FÁJLOK MEGHÍVÁSA DOS ÉS WINDOWS ALATT

DOS és Windows rendszerekben a script fájlok önmagukban nem hajthatók végre. A végrehajtásukhoz futtatni kell a PERL.EXE nevű interpretert úgy, hogy a parancssorba beírjuk a végrehajtandó script fájl nevét. Az eljárás egyszerűsítése céljából minden egyes script fájlhoz elkészíthetünk egy parancsfájlt (batch), amely a PERL.EXE interpretert a megfelelő script fájllal együtt hajtja végre. Tegyük fel például, hogy a HALLO.BAT parancsfájl az alábbi parancsokat tartalmazza:

```
@echo off
perl Hallo.PL %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Tegyük fel továbbá, hogy a Perl nyelvű *Hallo.PL* script fájl az alábbiakat tartalmazza:

```
print "Halló, az argumentumok '@ARGV'\n";
```

Amint látható, a parancsfájl egyszerűen futtatja a PERL.EXE interpretert úgy, hogy a *Hallo.PL* script fájlt parancssori bejegyzésként adja meg.

Ha viszont a Perl nyelvű script fájlokat Web-kiszolgálóról futtatjuk, akkor tapasztalni fogjuk, hogy a HTTP kiszolgálók többsége (mint például a *FolkWeb* kiszolgáló is) észleli a script fájlokat, így nem kell külön parancsfájlt írnunk. Másként fogalmazva ez azt jelenti, hogy ha olyan HTTP kiszolgálóval dolgozunk, mint amilyen például a *FolkWeb*, akkor semmilyen különleges dolgot nem kell tennünk egy Perl nyelvű script fájl meghívásához. Más esetekben szükség lehet parancsfájlok készítésére.

Ha a kiszolgáló a script fájl futtatásához igényli a parancsfájl meglétét, akkor a parancsfájlt (mint amilyen a HALLO.BAT) a *cgi-bin* könyvtárba kell helyeznünk, és feltehetően a script fájlt *(Hallo.PL)* is ott kell tárolnunk.

SZÖVEG ÉS HTML DOKUMENTUMOK GENERÁLÁSA PERL NYELVEN

A Perl nyelv segítségével rendkívül egyszerűen generálhatunk egy szöveges dokumentumot. Mindössze arról kell meggyőződnünk, hogy a dokumentum szövegének kezdete elé beszúrtunk egy érvényes HTML fejlécet. Az alábbi script fájl például egy olyan, hétköznapi (plain) dokumentumot hoz létre, amely a *Halló világ!* üzenetet jeleníti meg:

```
print "Content-type: text/plain\n\n";
print "Halló világ!\n";
```

Ugyanilyen egyszerű egy HTML dokumentum elkészítése is. Az alábbi utasítások például egy egyszerű HTML dokumentumot hoznak létre:

```
print <<HTML;
Content-type: text/html

<HTML>
<HEAD><TITLE>HTML dokumentum tesztelése</TITLE></HEAD>
<BODY>
<H1><CENTER>
Halló világ!
</CENTER></H1>
</BODY></HTML>
HTML
```

A fenti példa inkább hasonlít egy HTML forrásfájlra, mint egy Perl nyelvű programra. Valóban, ha törölnénk belőle néhány sort, akkor egy HTML forrásfájlt kapnánk. A példa a Perl egyik olyan képességét használja, amelyről a korábbiakban nem volt szó, de igen könnyen megérthető.

A <<HTML és a HTML utasítások egy *itt dokumentum*nak (here document) nevezett szerkezetet alkotnak, amely a UNIX héjprogramozásból átvett szerkezet. Egy *itt* dokumentum valójában többsoros literál (mint egy hosszú karakterlánc). A Perl az *itt* dokumentumot úgy kezeli, mint egy kettős idézőjelek közé tett karakterláncot.

Mivel az *itt* dokumentum könnyen olvashatóvá teszi a Perl forráskódot, ideális eszköz HTML dokumentumok generálására. *Itt* dokumentumot használva a Perl script fájloknak nincs szükségük idézőjelekre és újsor karakterekre, továbbá nincs szükség a *print* utasítások ismételt kiadására. Ha megvizsgáljuk a fejezet további részében szereplő példákat, akkor többször is találkozhatunk ilyen *itt* dokumentumokkal.

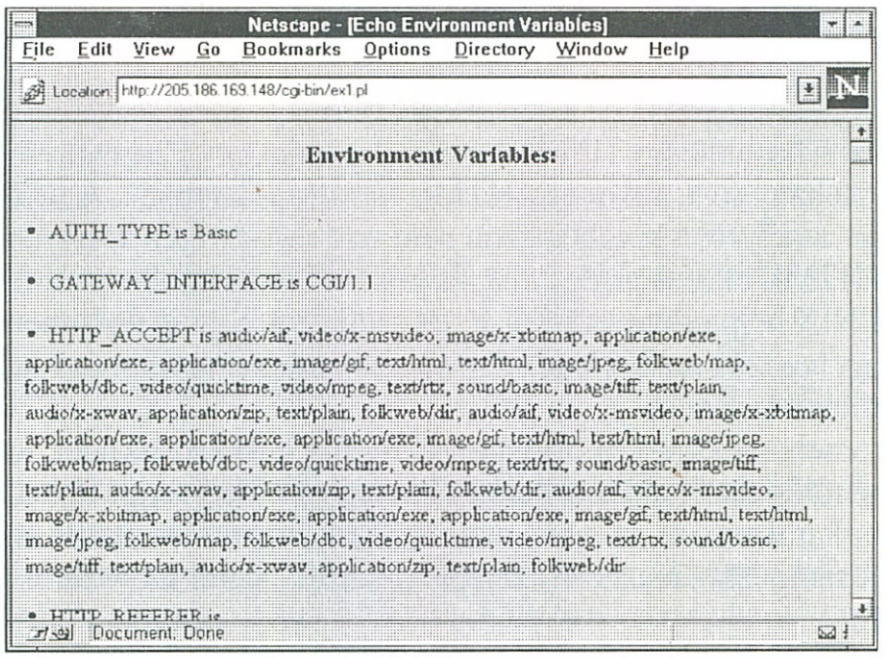
DOKUMENTUM DINAMIKUSSÁ TÉTELE

Ha egy CGI script fájltól mindössze annyira „telne", hogy elkészítsen egy statikus űrlapot, akkor a Perl (vagy akár maga a CGI is) elveszíthetné a létjogosultságát. A CGI script fájlok igazi ereje abban van, hogy dinamikusan képesek Web-lapok létrehozására. A 11. fejezetben egy C++ nyelvű

programot vizsgáltunk. Ez a program olyan script fájlt hozott létre, amely *visszhangozta* (echo) a script fájl környezeti változóit. Az alábbi, Perl nyelvű script fájl ugyanezt a feladatot végzi el. Figyeljük meg, hogy mennyivel egyszerűbb ez a Perl kód, mint a 11. fejezetben látható C++ nyelvű változata:

```
print <<HTML;
Content-type: text/html

<HTML>
<HEAD><TITLE>Környezeti változók echója</TITLE></HEAD>
<BODY>
<H3><CENTER>
Környezeti változók:<HR>
</CENTER></H3>
HTML

for $env (sort keys %ENV)
  {
    print "<LI>$env tartalma $ENV{$env}<BR>";
  }

print "</BODY></HTML>\n";
```

A fenti példa egy statikus fejlécet generál, majd formázott HTML utasítások segítségével megjeleníti a script fájl környezeti változóinak tartalmát. A script fájl arra is példát mutat, hogy miként használhatunk egy *itt* dokumentumot egy hagyományos *print* kivitellel kombinálva. Ha a HTTP kiszolgálóról végrehajtjuk ezt a script fájlt, akkor a böngésző a 13.2. ábrán láthatóhoz hasonlót jelenít meg a képernyőn:



13.2. ábra. Egy script fájl környezeti változóinak megjelenítése

HOZZÁFÉRÉS A LEKÉRDEZŐ KARAKTERLÁNCHOZ

A 11. fejezetben láttuk, hogy a *lekérdező karakterláncot* (query string) használva a CGI script fájl könnyen hozzájuthat a bemeneti adataihoz. A böngésző az adatokat az URL részeként küldi el a HTTP kiszolgálóra. A kiszolgáló mindazt, ami az URL-ben a kérdőjel (?) után áll, a lekérdező karakterlánc részének tekinti.

Egy CGI script fájl kétféle módon férhet hozzá a lekérdező karakterlánchoz. Egyik módja, hogy a kiszolgáló parancssori argumentumokként adja át a karakterláncot a script fájlnak, a másik pedig az, hogy a kiszolgáló a QUERY_STRING környezeti változóhoz rendeli a karakterláncot. Visszatérhetünk például az előző, a környezeti változókat megjelenítő script fájlhoz, ha egy lekérdező karakterlánccal meghívjuk azt.

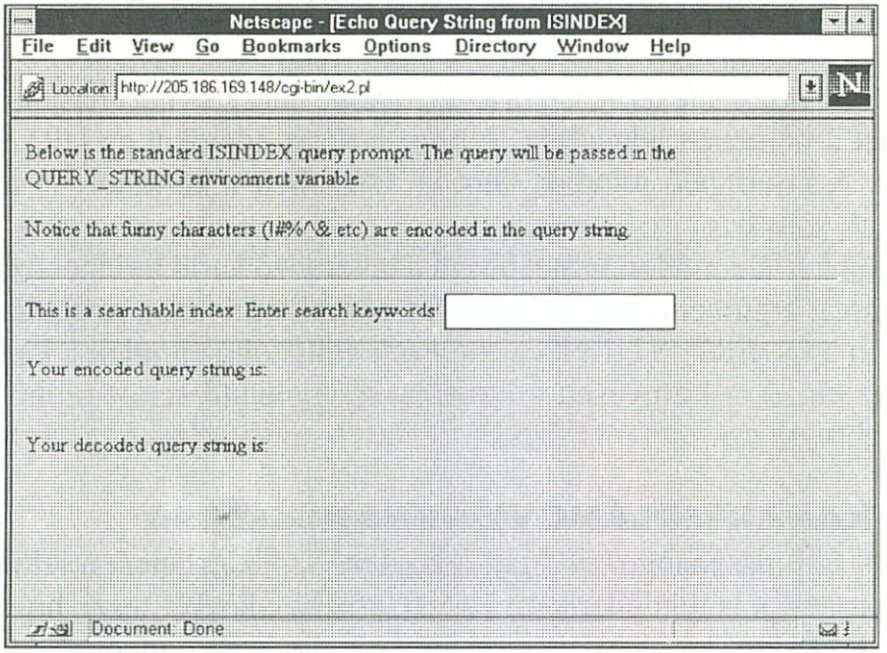
A LEKÉRDEZŐ KARAKTERLÁNC VISSZHANGOZÁSA

Az alábbi Perl script fájl visszhangozza a fájl lekérdező karakterláncát a felhasználónak. A script fájl egy ISINDEX promptot is rendelkezésre bocsát, amely lehetővé teszi, hogy a felhasználó egyszerűen beírhassa a lekérdező karakterláncot. A felhasználó Web-böngészője feldolgozza az ISINDEX promptot, és a bevitt adatokat az URL-hez fűzi:

```
print <<HTML;
Content-type: text/html

<TITLE>Lekérdezõ karakterlánc visszhangozása az ISINDEX promptból</TITLE>

Alant a szabványos ISINDEX lekérdezõ prompt látható.
A lekérdezés bekerül a QUERY_STRING
környezeti változóba.<P>

Figyeljük meg a fura karaktereket (!#%^& etc)
a kódolt lekérdezõ karakterláncban.<P>

<ISINDEX>
HTML

print "A kódolt lekérdezõ karakterlánc: <BR>\n";
$query = $ENV{QUERY_STRING};
print "$query<P>\n";

print "A dekódolt lekérdezõ karakterlánc: <BR>\n";
$query =~ s/\+/ /g;
$query =~ s/%([0-9A-H]{2})/pack('C',hex($1))/eg;
print "$query<P>\n";
```

Ha a Netscape Navigator böngészőből hívjuk meg ezt a script fájlt, akkor a böngésző a fájl kimenetét a 13.3. ábrán látható módon jeleníti meg.

Amikor egy script fájl vagy egy böngésző lekérdező karakterláncot illeszt az URL végéhez, akkor a karakterlánc nem tartalmazhat olyan karaktert, amely érvénytelenné tenné az URL-t. Az érvényben lévő megállapodás szerint a szóközöket plusz (+) jelre kell cserélni, a további, érvénytelen karaktereket pedig az ASCII kódjuk hexa alakjára úgy, hogy mindegyik karakter elé be kell szúrni a százalék (%) jelet. A böngésző egy űrlap minden adatát e szabályok szerint kódolja.



13.3. ábra. A lekérdező karakterlánc eredményének megjelenítése

A *QUERY_STRING* környezeti változóban megkapott lekérdező karakterláncot a script fájlnak kell dekódolnia. Számos kiszolgálónál előfordul, hogy mielőtt a parancssorba helyezné a lekérdező karakterláncot, dekódolja azt, hogy megkönnyítse a script fájl részére a feldolgozását. Mivel azonban a parancssor hossza korlátozott, a *QUERY_STRING* környezeti változón keresztüli átadás biztonságosabb eljárást kínál.

A lekérdező karakterlánc kódolásához a script fájlok két reguláris kifejezést használnak. Az első reguláris kifejezés a plusz (+) jeleket szóközökre cseréli:

```
$query =~ s/\+/ /g;
```

Amint látható, a reguláris kifejezés az *s/MINTA/CSERE/* alakú keresés és cserélés formátumot használja. A reguláris kifejezés úgy módosítja a *$query* változó tartalmát, hogy a plusz jel minden egyes előfordulását szóközre cseréli. A reguláris kifejezés végén álló *g* arra utasítja a Perlt, hogy globális műveletet végezzen, azaz a cserét ne csak az első előforduláskor, hanem a teljes karakterláncban végezze el. Figyeljük meg, hogy a kifejezés (a backslash karakter eléírásával) megakadályozza, hogy a plusz jelet a reguláris kifejezésben a módosított jelentésben értelmezze.

A második reguláris kifejezés a százalékjel (%) és az azt követő két hexadecimális számjegy minden előfordulását a megfelelő ASCII karakterre cseréli:

```
$query =~ s/%([0-9A-H]{2})/pack('C',hex($1))/eg;
```

Ebben az esetben – amint a reguláris kifejezéshez fűzött *e* betű jelzi – a lecserélendő érték nem karakterlánc, hanem egy Perl kifejezés. A *pack('C',hex($1))* kifejezés a két hexa számjegynek megfelelő ASCII karaktert adja vissza. A *$1* változó a karakterláncnak arra a részére hivatkozik, amely a mintának a zárójelek közötti *([0-9A-H]{2})* részét a két hexa számjegynek felelteti meg. Ha egynél több ilyen zárójelpár lenne, akkor ezeknek az értékeit a script fájl a *$2, $3* stb. speciális változókon keresztül kapná meg.

HTML ŰRLAPOK DEKÓDOLÁSA A GET METÓDUSSAL

Amint láttuk, az *ISINDEX* segítségével könnyedén létrehozhatunk egy lekérdezést. Ha viszont egynél több adatot szeretnénk fogadni a felhasználótól, akkor ehhez űrlapot kell készíteni. Az itt következő Perl script fájl egy űrlapot generál. A *GET* metódus segítségével a script fájl arra utasítja a böngészőt, hogy a lekérdezett adatokat az URL részeként küldje el, ugyanúgy, mintha az *ISINDEX* promptot használná. A *GET* és az *ISINDEX* eljárás között az a különbség, hogy a *GET* használatakor a böngésző több mező értékét fűzheti egybe egyetlen lekérdező karakterlánccá úgy, hogy az egyes mezőket az ampersand (&) karakter választja el egymástól.

Azért, hogy a script fájl azonosíthassa a mezők értékeit, a böngésző beszúrja a mezőneveket a lekérdező karakterláncba. Ha például egy adatbázis három mezőt tartalmaz (név, kor, születésnap) a (Bob, 27, 11-1-68) értékekkel, akkor a lekérdező karakterlánc a következő formátumban fogja tartalmazni a mezőértékeket: "nev=Bob&kor=27&szuletesnap=11-1-68". Az alábbi script fájl dekódolja a mezőket, és egy HTML űrlapot készítve megjeleníti azokat:

```
($cgi_bin, $cgi_script) = ($0 =~ m:(.*)[/\\](.*):);

$query = $ENV{QUERY_STRING};

if ($query eq '') {
    # ûrlap generálása
    print <<FORM;
Content-type: text/html

<HTML>
<HEAD><TITLE>Egyszerû GET ûrlap</TITLE></HEAD>
<BODY>
Mi a választása? <P>

<FORM METHOD="GET" ACTION="$cgi_script">

Jelölõnégyzet<BR>
<INPUT TYPE="checkbox" NAME="check" VALUE="on"><P>

Választógombok csoportja<BR>
<INPUT TYPE="radio" NAME="button" VALUE="1"> 1<BR>
<INPUT TYPE="radio" NAME="button" VALUE="2"> 2<BR>
<INPUT TYPE="radio" NAME="button" VALUE="3"> 3<P>

Adatbeviteli mezõ<BR>
<INPUT NAME="field"><P>

Az adatok elküldése<BR>
<INPUT TYPE="submit">

</FORM>
</HTML>
FORM
```

```
} else {
    # a lekérdezés eredményének megjelenítése
    print "Content-type: text/html\n\n";
    print "<HTML>\n";
    print "<HEAD><TITLE>GET ûrlap eredményei</TITLE></HEAD>\n";
    print "<BODY>\n";
    print "A választásai:<P>\n";
    @fields = split('&', $query);
    foreach (@fields) {
        Switch: {
          /^check=(.*)/ && do {
            $check = $1;
            last Switch;
          };
          /^button=(.*)/ && do {
            $button = $1;
            last Switch;
          };
          /^field=(.*)/ && do {
            $field = &decode($1);
            last Switch;
          };
        }
    }

    print "Jelölõnégyzet: $check<BR>\n";
    print "Választógomb: $button<BR>\n";
    print "Adatbeviteli mezõ: ", &html($field), "<BR>\n";
    print "</HTML>\n";
  }

sub decode {
    local ($value) = @_;
    $value =~ s/\+/ /g;
    $value =~ s/%([0-9A-H]{2})/pack('C',hex($1))/eg;
    return $value;
  }

sub html {
    local ($value) = @_;
    $value =~ s/</&lt;/g;
    $value =~ s/>/&gt;/g;
    return $value;
  }
```

Szánjunk időt a script fájl első sorának tanulmányozására:

```
($cgi_bin, $cgi_script) = ($0 =~ m:(.*)[/\\](.*):);
```

Ez a kifejezés hasonlít az előző példában vizsgált kifejezéshez. Ebben az esetben azonban a script fájl a teljes elérési utat könyvtárra és fájlnévre bontja. A script fájl a fájlnév segítségével határozza meg az űrlap ACTION attribútumával megadandó URL-t. Ebben a példában ugyanaz a script fájl generálja az űrlapot és dolgozza fel annak a kimenetét, ami megkönnyíti a munkát, és mivel mindent ugyanoda helyez, ez az űrlapok kezelésének javasolt módja.

A script fájl megvizsgálja, hogy küldött-e válaszokat a felhasználó, és ettől függően vagy űrlapot generál, vagy feldolgozza a beérkezett adatokat. A lekérdező karakterlánc feldolgozásához a script fájl a *split* függvény segítségével mezőkre darabolja a karakterláncot. Ezt követően a lekérdező mezőket összehasonlítja a várt mezőnevekkel.

Alaposan tanulmányozzuk az alábbi *foreach* ciklust is:

```
foreach (@fields) {
    Switch: {
      /^check=(.*)/ && do {
        $check = $1;
        last Switch;
      };
```

A *foreach* ciklus normál, *for $VÁLT(LISTA)* alakja egy iterációs változót tartalmaz ($VÁLT). Ha a *foreach* ciklusból hiányzik ez a változó, akkor a Perl az alapértelmezés szerinti *$_ változót* használja helyette. Hasonlóképpen, a reguláris kifejezés operátora általában *$VÁLT =~ /MINTA/* alakú. Ha a kifejezés nem tartalmazza a változót, akkor a Perl az alapértelmezés szerint *$_ változót* használja helyette. Ilyen módon a két utasítás megfelel egymásnak. Ha viszont túl sokszor hagyatkozunk a Perl alapértelmezés szerinti változóira, akkor meglehetősen zavaros lehet a kódunk. A mi esetünkben az alapértelmezés szerinti változók használata világosabb és könnyebben olvasható kódot eredményez.

A következőkben tekintsük a */^mező=(.*)/* alakú reguláris kifejezést. A „kalap" (^) karakter biztosítja hogy az egyezőség keresése a karakterlánc elejétől, és ne esetleg valamilyen más mező közepétől kezdődjön. Más szavakkal ez azt jelenti, hogy a mező nevének és az egyenlőségjelnek (=) egyezniük kell egymással. A reguláris kifejezés további része a mező értékét egyezteti és kiemeli a karakterláncból a *$1* változóba. Mivel a *$1* csak ideiglenes változó, a script fájl átmásolja a tartalmát az egyes mezők nevével megadott változókba.

A script fájl a *decode* szubrutinban dekódolja a mezőértékekben lévő, a böngésző által oda bekódolt karaktereket. A dekódoló szubrutinban használt reguláris kifejezéseket az előző példában ismertettük.

Végül a script fájl a *html* szubrutinban úgy kódolja az adatokat, hogy azokat HTML szövegként lehessen megjeleníteni. A script fájlok a szövegek többségét bármilyen kódolás nélkül elküldhetik a böngészőnek megjelenítés céljából. Mivel azonban a HTML nyelv a csúcsos zárójeleket a HTML elemek megjelölésére használja, a script fájlnak a zárójeleket az *<* és az *>* HTML karaktersorozatokkal kell körülvennie.

HTML ŰRLAPOK DEKÓDOLÁSA A POST METÓDUSSAL

Az alábbi script fájl csak annyiban különbözik az előzőtől, hogy ez a POST metódust használja az űrlapok adatainak elküldéséhez. A POST metódus arra utasítja a böngészőt, hogy az űrlap adatait ne a lekérdező karakterlánccal, hanem a script fájl standard bemenetét használva küldje el. A POST metódus azért hasznos, mert nagy mennyiségű adat kezelését teszi lehetővé, míg a GET metódus-

nál korlátot jelent a kiszolgálón a környezeti változók számára rendelkezésre álló memóriaterület, valamint a böngésző URL-jének hossza.

A beérkező adatok standard bemenetről történő olvasásához a script fájl a kívánt bájtok számával meghívja a *sysread* függvényt. A lekérdező karakterlánc bájtokban számított mérete a CONTENT_LENGTH nevű környezeti változóban olvasható el.

Miután a script fájl beolvasta a lekérdező karakterláncot, az adatokat az előző példában látottakhoz hasonlóan dolgozza fel. Azért, hogy a script fájl ne legyen túlságosan terjengős, a fájl a karakterláncot egy kicsit furfangosabb módon elemzi:

```
($cgi_bin, $cgi_script) = ($0 =~ m:(.*)[/\\](.*):);

$content_length = $ENV{CONTENT_LENGTH};
if ($content_length > 0) {
    sysread(STDIN, $query, $content_length);
  }

if (!defined($query) || $query eq '') {
    # ûrlap generálása
    print <<FORM;

Content-type: text/html
<HTML>
<HEAD><TITLE>Egyszerû POST ûrlap</TITLE></HEAD>
<BODY>
Mi a választása? <P>

<FORM METHOD="POST" ACTION="$cgi_script">

Jelölõnégyzet<BR>
<INPUT TYPE="checkbox" NAME="check" VALUE="on"><P>

Választógombok csoportja<BR>
<INPUT TYPE="radio" NAME="button" VALUE="1"> 1<BR>
<INPUT TYPE="radio" NAME="button" VALUE="2"> 2<BR>
<INPUT TYPE="radio" NAME="button" VALUE="3"> 3<P>

Adatbeviteli mezõ<BR>
<INPUT NAME="field"><P>

Az adatok elküldése<BR>
<INPUT TYPE="submit">

</FORM>
</HTML>
FORM
  }
else {
    # a lekérdezés eredményének megjelenítése
```

```
    print "Content-type: text/html\n\n";
    print "<HTML>\n";
    print "<HEAD><TITLE>POST ûrlap eredményei</TITLE></HEAD>\n";
    print "<BODY>\n";
    print "A választásai: <P>\n";
    @fields = split('&', $query);

    foreach (@fields)
      {
        /([^=]+)=(.*)/ && do {
          local ($field, $value) = ($1, $2);
          $query{$field} = &decode($value);
        }
      }

    print "Jelölõnégyzet: $query{check}<BR>\n";
    print "Választógomb: $query{button}<BR>\n";
    print "Adatbeviteli mezõ: ", &html($query{field}), "<BR>\n";
    print "</HTML>\n";
  }

sub decode {
    local ($value) = @_;
    $value =~ s/\+/ /g;
    $value =~ s/%([0-9A-H]{2})/pack('C',hex($1))/eg;
    return $value;
  }

sub html {
    local ($value) = @_;
    $value =~ s/</&lt;/g;
    $value =~ s/>/&gt;/g;
    return $value;
  }
```

A script fájl egyetlen reguláris kifejezést használ a lekérdező karakterlánc valamennyi mezőjének elemzésére:

```
/([^=]+)=(.*)/ && do {
    local ($field, $value) = ($1, $2);
    $query{$field} = &decode($value);
  }
```

Ez a script fájl a lekérdezett értékeket nem külön-külön változókhoz rendeli, hanem az összes értéket egy asszociatív tömbben tárolja. A fájl a tömb elemeinek indexeléséhez a mezőneveket használja.

Figyeljük meg a reguláris kifejezés szögletes zárójelek közötti *[^=]* részét! Ebben a környezetben a „kalap" (^) karakter – az előző példától eltérően – nem a karakterlánc kezdetéhez köti a

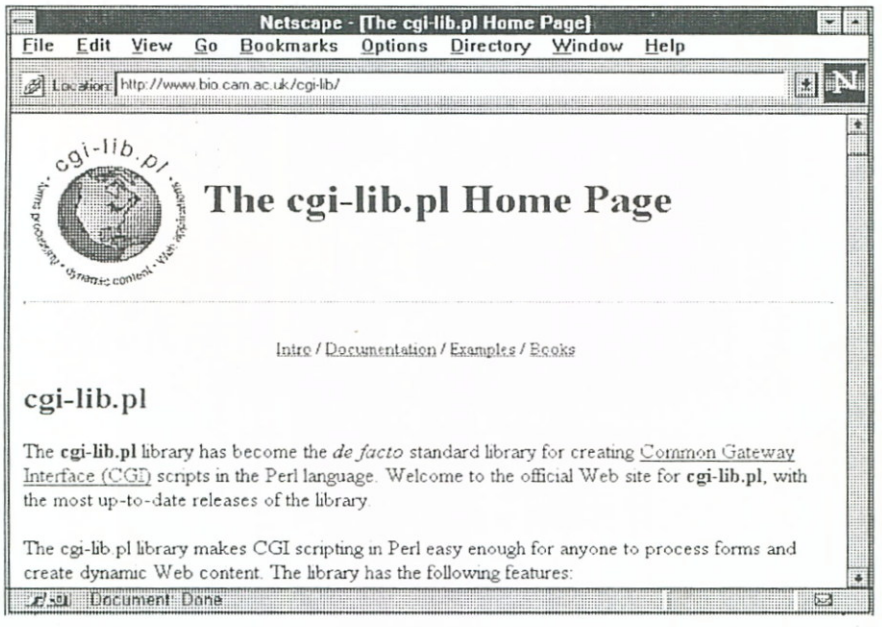
keresést. A kalap karakter ebben az esetben negálja a *[=]* karaktert. A reguláris kifejezés a következőképpen olvasható: „keressen egy vagy több olyan karaktert, amelyek között nincs egyenlőségjel, azután egy karakter egyezzen meg az egyenlőségjellel, majd jöjjön az összes többi karakter". A kalap karakterre mint a keresést a karakterlánc elejéről indító karakterre nincs szükség, mert a *([^=]+)* kifejezés tartalmaz egy globális helyettesítő (+) karaktert, amely a karakterlánc elejétől kezdve mindegyik karakterrel megegyezik. Ehhez hasonlóan a (.*) kifejezés a karakterlánc végéig minden karakterrel megegyezik.

A CGI-LIB FÜGGVÉNYTÁR HASZNÁLATA ŰRLAPOK DEKÓDOLÁSÁRA

A Perl szabadon hozzáférhető *cgi-lib.pl* nevű függvénytára leegyszerűsíti a CGI űrlapok feldolgozását. A függvénytár számos hasznos szubrutint tartalmaz, de elsősorban a *ReadParse* tarthat számot az érdeklődésre, ami beolvassa és elemzi az űrlap adatait. A *cgi-lib* egyik legnagyobb előnye, hogy transzparens módon kezel mindenféle típusú űrlapot (ISINDEX, GET, POST), sőt többrészes űrlapok és igen nagy mennyiségű adat feldolgozására is képes.

Néhány rutin, mint például a *PrintHeader, HtmlTop* és *HtmlBot* szabványos HTML szekvenciákat generál, de ezek meglehetősen egyszerűek és nem túlságosan fontosak. A *cgi-lib* függvénytárat ugyanúgy használhatjuk, mint a Perl bármelyik másik függvénytárát: a függvénytárat a *require* utasítással kell beemelnünk a forrásprogramba. A *cgi-lib.pl* függvénytárat célszerű a Perl függvénytárak standard könyvtárába telepíteni (DOS vagy Windows alapú rendszerben ez általában a C:\PERL\LIB) vagy a teljes elérési úttal is hivatkozhatunk rá.

A *cgi-lib.pl* függvénytárról a *http://www.bio.cam.ac.uk/cgi-lib* címen kaphatunk további információkat, ahonnan a függvénytár le is tölthető (lásd a 13.4. ábrát).



13.4. ábra. A szabadon hozzáférhető *cgi-lib.pl* függvénytár letöltése

ÖSSZEFOGLALÁS

Ebben a fejezetben megtanultuk, hogyan használható a Perl nyelv bonyolult CGI script fájlok írására. A fejezetben megismert eljárások segítségével professzionális minőségű, az üzleti életben is használható Web-szolgáltatásokat nyújthatunk. A 14. fejezetben egy másik érdekes programozási

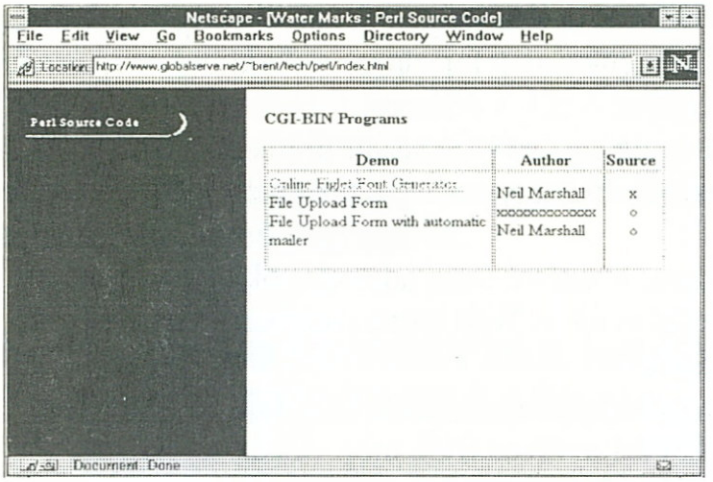
nyelvvel, a Javával ismerkedünk meg. Mint majd látni fogjuk, a Java nyelv jól használható animációs és multimédiás Web-helyek létrehozására. A 14. fejezetben még egy másik script nyelvet, a JavaScript nyelvet is áttekintjük. Mielőtt azonban rátérnénk erre a fejezetre, győződjünk meg arról, hogy értjük az alábbiakat:

- A Perl sokoldalú programozási nyelv, amelynek segítségével CGI script fájlokat fejleszthetünk a Web programozásához, továbbá hétköznapi használatú, hagyományos programokat is írhatunk vele.
- A Perl nagyon jól használható szövegfeldolgozásra, lehetővé teszi adatbázisok egyszerű elérését, hordozható, és megfelelő hálózati biztonságot is nyújt. A Web környezetében történő fejlesztések során ezek a jellemzők mind nagyon fontosak.
- A Perl a szöveg elemzésekor sűrűn használja a reguláris kifejezéseket.
- A Weben jelenleg található script fájlok többségét Perl nyelven írták meg. A Perl nyelvnek ezt a monopóliumát azonban feltehetően hamarosan megtörik az olyan programozási nyelvek, mint a JavaScript és a VBScript.

PERL NYELVŰ CGI SCRIPT FÁJLOK ÍRÁSÁVAL FOGLALKOZÓ FONTOSABB HELYEK

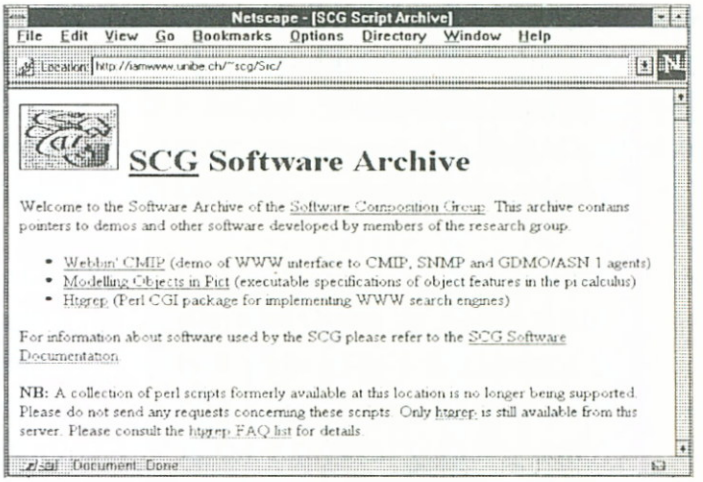
A következő Web-helyek segíthetnek abban, hogy részletesen olvashassunk Perl nyelvű script fájlok írásáról, valamint az objektumorientált script fájlokról. A helyeket kiindulási pontként is használhatjuk a Web felfedezéséhez.

Watermarks: Perl Source Code



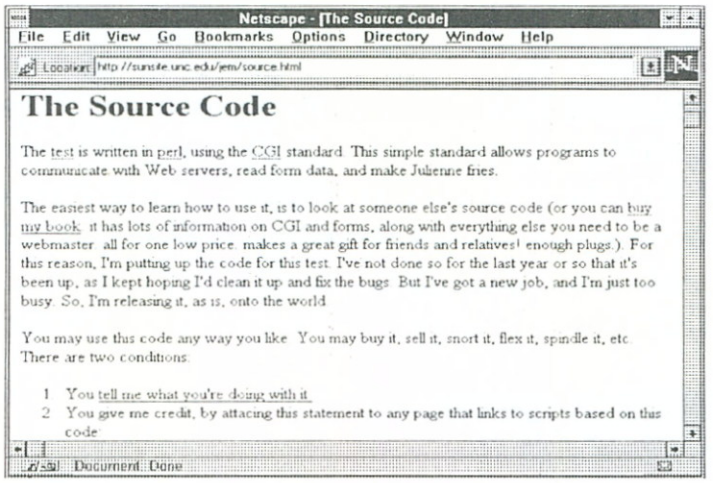
http://www.globalserve.net/~brent/tech/perl/index.html

SCG Script Archive



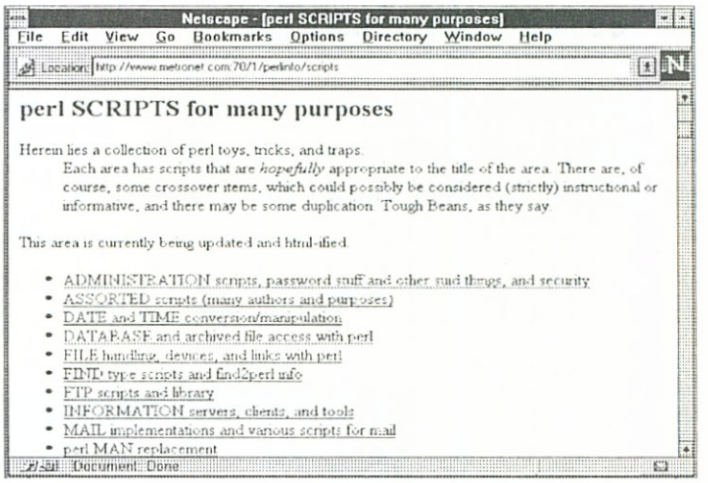
http://iamwww.unibe.ch/~scg/Src/

The Source Code



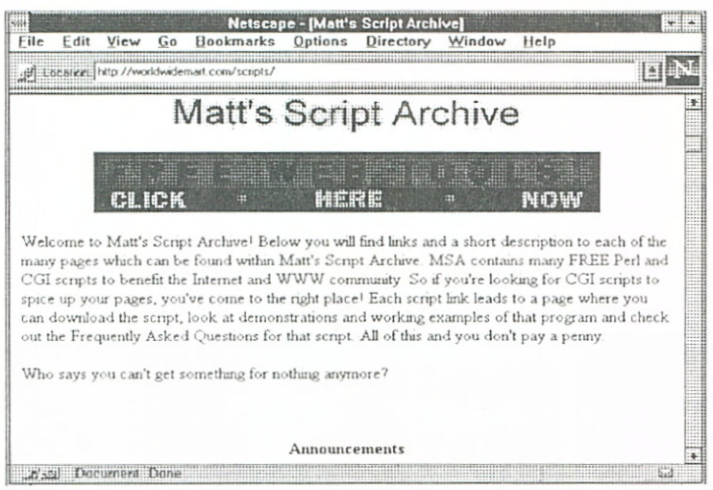
http://sunsite.unc.edu/jem/source.html

perl SCRIPTS for many purposes



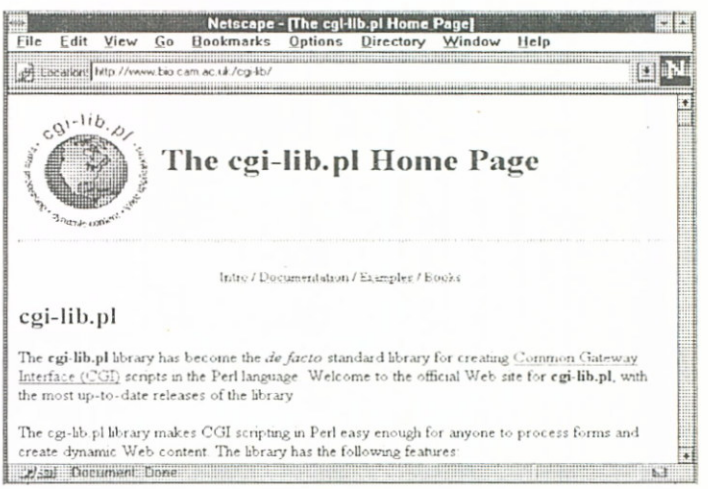
http://www.metronet.com:70/1/perlinfo/scripts

Matt's Script Archive



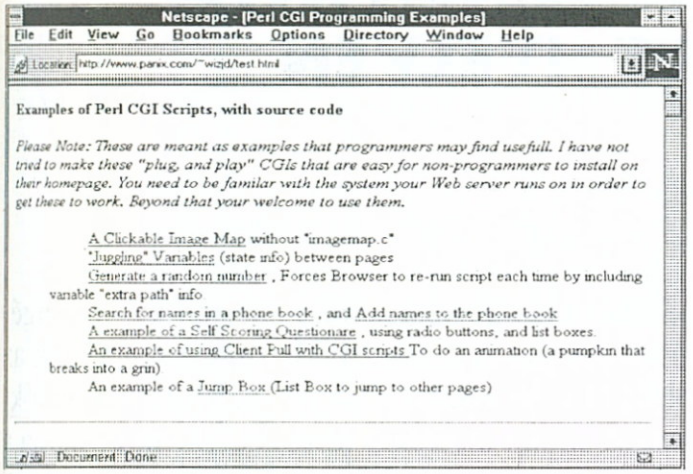
http://worldwidemart.com/scripts/

The cgi-lib.pl Home Page



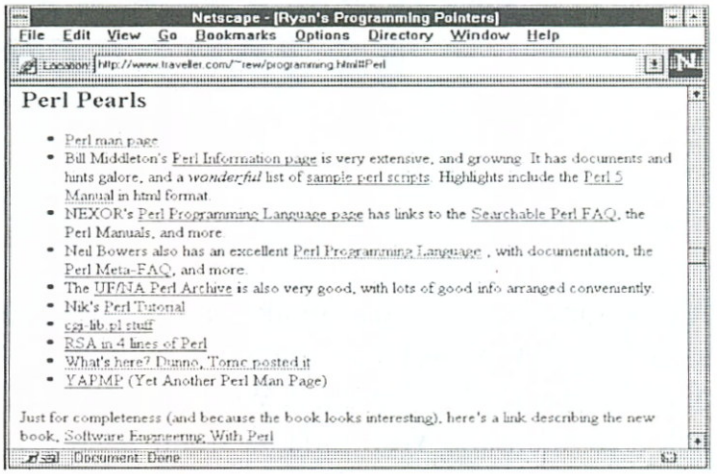
http://www.bio.cam.ac.uk/cgi-lib/

Perl CGI Programming Examples



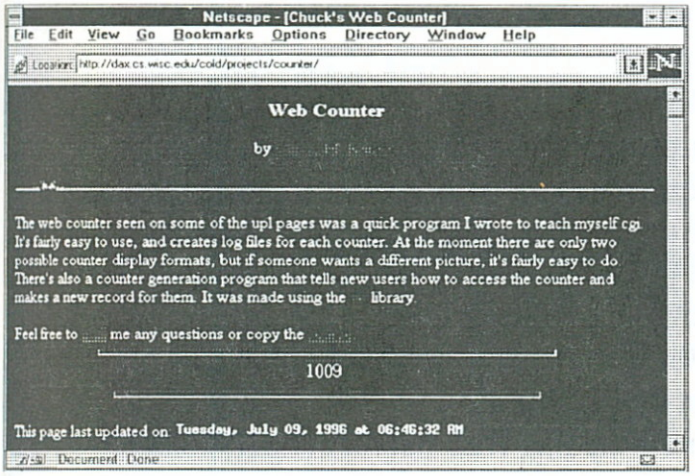
http://www.panix.com/~wizjd/test.html

Ryan's Programming Pointers



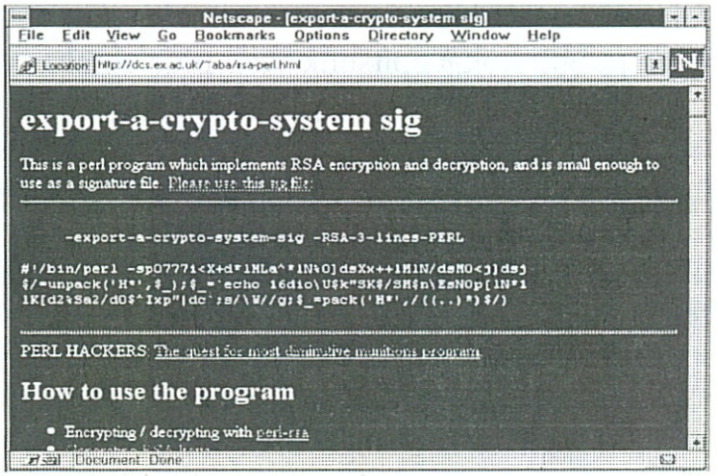
http://www.traveller.com/~rew/programming.html#Perl

Chuck's Web Counter



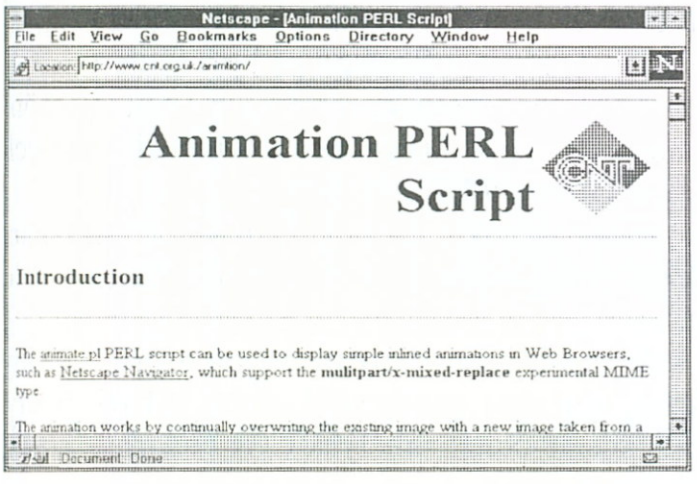
http://dax.cs.wisc.edu/cold/projects/counter/

export-a-crypto-system sig



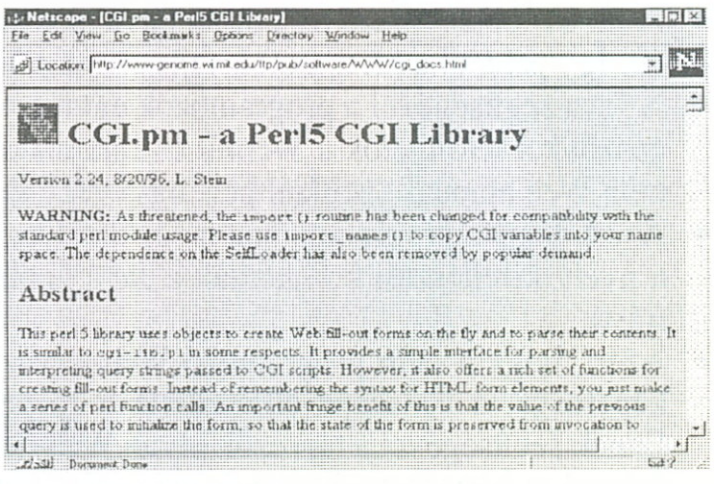
http://dcs.ex.ac.uk/~aba/rsa-perl.html

Animation Perl Script



http://www.cnt.org.uk/animtion/

CGI.pm – a Perl5 CGI Library



http://www-genome.wi.mit.edu/ftp/pub/software/WWW/cgi_docs.html

14. FEJEZET

A WEB PROGRAMOZÁSA JAVA ÉS JAVASCRIPT NYELVEN

A programozás erőforrásainak – mint amilyenek például a multimédia-könyvtárak vagy a böngészők beépített nyelvi támogatásai – bővülésével párhuzamosan a felhasználók igényei is nőttek az alkalmazásokkal szemben. A World Wide Web berobbanása óta eltelt néhány évben a felhasználóknak a mindentudó Web-oldalak iránti elvárásai legalább olyan ütemben növekedtek, mint maga a Web. A közeli jövőben a Web-oldalak fejlődésének egyik elsődleges hajtómotorja a Java programozási nyelv lesz. A Java segítségével a programozók (ma már) olyan multimédia alapú Web-helyeket készíthetnek, amelyeken animációk, hangok (beszéd és zene), sőt akár videoklipek integrálásával szolgáltathatnak információkat a felhasználóknak. Ebben a fejezetben a Java programozási nyelv alapjaival és Java programok készítésével ismerkedünk meg. Miután áttanulmányoztuk a fejezetben található Java programokat (az úgynevezett appleteket), képesek leszünk arra, hogy ilyen programokat építsünk Web-oldalainkba.

A Java programozási nyelv tanulmányozásán túlmenően a fejezet betekintést nyújt a JavaScript script (parancsfájl alapú) nyelvbe, amely a „nem programozók" számára is lehetőségeket kínál, hasonlóan a C/C++ és a Perl nyelvű script fájlokhoz.

A fejezet célja, hogy olyan információkat nyújtson az olvasóknak, amelyek azonnal használhatóvá teszik a Java és a JavaScript nyelvet. A fejezet tartalmazza továbbá néhány fontos információforrás Web-címét is. A fejezet végére érve érteni fogjuk az alábbiakat:

- A Java a Sun Microsystem Inc. által kifejlesztett programozási nyelv, amelynek segítségével Web-alapú programokat, úgynevezett appleteket készíthetünk.
- Java appletek segítségével olyan animációs és multimédiás Web-helyeket hozhatunk létre, amelyek zenét, hangokat és mozgó képeket integrálnak magukba.
- A Java programozási nyelv nagyon hasonlít a C++ nyelvre. Aki ismeri a C++ nyelvet, akár egy héten belül is képes megérteni a Javát, szinte minden vonatkozásban.
- Jelenleg a Java fejlesztőkészlet (Java Developer's Kit, röviden: JDK) ingyenesen letölthető a Sun Web-helyéről.
- A Java robbanásszerű terjedését látva más szoftvercégek is elkészítették a maguk Java változatát. Így például a Microsoft kifejlesztette a Visual J++ nyelvet, a Borland pedig a Latte-t.
- A Java segítségével a programozók egyrészt appleteket készíthetnek, amelyek a Javát értő böngészőkkel letölthetők a Webről és futtathatók, másrészt önálló programokat írhatnak, amelyek nem Web alapúak (például pénzügyi vagy számlázó programokat).
- Azért, hogy a programozókat megakadályozzák vírusprogramok Java nyelven történő írásában, a nyelv kifejlesztői korlátozták a Java appletek bizonyos képességeit (például a lemezre való írást).
- Azért, hogy a nem programozó Web-tervezők számára is rendelkezésre álljon a Java néhány képessége, a Netscape elkészítette a JavaScriptet, ami egy parancsfájl alapú HTML nyelv. Ennek segítségével a tervezők olyan Web-helyeket készíthetnek, amelyek CGI-t használhatnak, űrlapokat jeleníthetnek meg és még sok mást tartalmazhatnak.
- A JavaScript és más, script fájl alapú nyelvek, mint például a Perl közötti elsődleges különbség az, hogy a Java script fájlok utasításait a böngésző hajtja végre és nem a kiszolgálón található program.

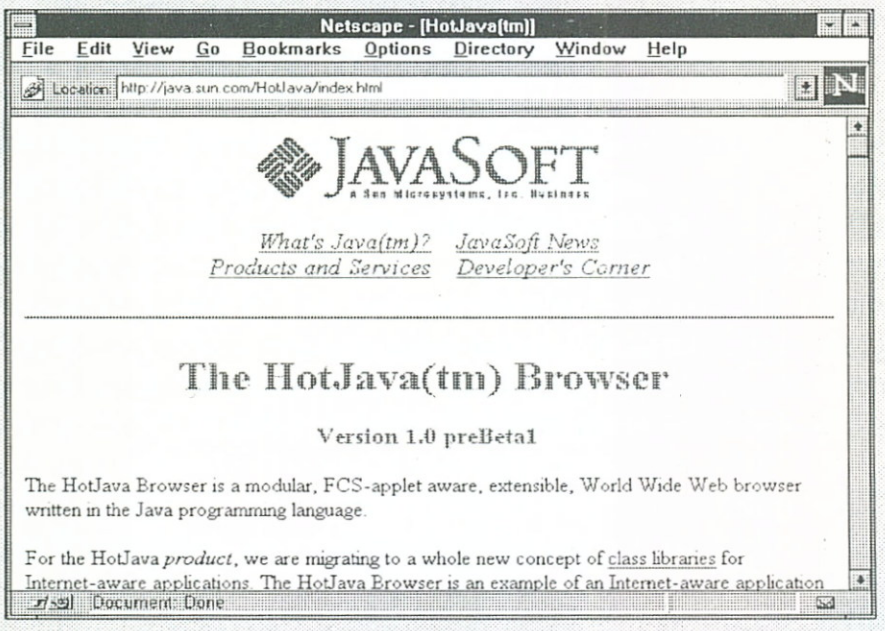
A JAVA HASZNÁLATA

WINDOWS 3.1 ALATT NEM HASZNÁLHATÓ A JAVA

Ahhoz, hogy a Java fordító segítségével Java appleteket készíthessünk, a Windows valamelyik 32 bites változatát – mint amilyen például a Windows 95 vagy a Windows NT – kell futtatnunk. Windows 3.1 alatt nem futtathatók Java programok. Jelenleg olyan böngészők sincsenek, amelyek Windows 3.1 alatt futtathatnák a Javát – eggyel több ok, hogy áttérjünk a Windows 95-re.

A HOTJAVA JAVA NYELVEN ÍRT BÖNGÉSZŐ

A Javát nem ismerő programozók és felhasználók gyakran nincsenek tisztában a Java és a HotJava közötti különbséggel. Amint említettük, a Java programozási nyelv, csakúgy, mint a C vagy a C++. Ezzel szemben a HotJava böngésző program, ami a Netscape Navigator vagy a Microsoft Internet Explorer böngészőjének leegyszerűsített változatához hasonlítható. A HotJava böngészőt az teszi különlegessé, hogy azt Java programozási nyelven írták meg. A HotJava böngészőről további információkat kapunk, ha meglátogatjuk az alábbi helyet: *http://java.sun.com/HotJava/index.html* (lásd a 14.1. ábrát).



14.1.ábra. Információk a *HotJava* böngészőről

A JAVA ÉS A HÁLÓZAT BIZTONSÁGA

Amikor programokat töltünk le a Webről, akkor tisztában kell lennünk azzal a potenciális veszéllyel, hogy egy letöltött program számítógépes vírust is tartalmazhat. Korábban, amikor a felhasználók jórészt csak szörföztek a világhálón (és nem töltöttek le programokat), a vírusoktól nemigen kellett tartani. A felhasználó gépét csak akkor fenyegette veszély, ha fertőzött programot töltött le, és azt futtatta.

Amint említettük, a programozók a Java nyelv segítségével appleteket készítenek, amelyeket böngészők töltenek le és futtatnak. Ezért első látásra úgy tűnhet, hogy a Java appletek kiváló lehetőséget teremtenek arra, hogy rosszindulatú programozók vírusokat készítsenek és terjesszenek. Az ilyen veszély elkerülése céljából a Java nyelv kifejlesztői azt a megoldást választották, hogy korlátozzák az appletek által végrehajtható műveleteket – vagyis az appletek képességeinek csökkentése útján megakadályozzák, hogy azok vírust terjeszthessenek. Így például egy Java applet nem képes fájl I/O műveletek végrehajtására. Ilyen módon egy applet nem tud információkat (például vírusokat) tárolni annak a felhasználónak a merevlemezén, aki letöltötte az appletet. A Java „inváziós" képességeinek korlátozása révén a Java-fejlesztők lényegesen csökkentették a hálózat biztonságával kapcsolatos problémákat.

A JAVA ESZKÖZ- ÉS RENDSZERFÜGGETLEN

A Java egyik legnagyobb előnye más programozási nyelvekhez képest az, hogy olyan appletkódot állít elő, amely eszköz- és rendszerfüggetlen. Más szavakkal ez azt jelenti, hogy miután elkészítettünk egy Java appletet, ugyanazt futtathatjuk Windows, Mac vagy akár UNIX-os rendszerben is. Ha viszont olyan programozási nyelvet használtunk volna, mint amilyen például a C++, akkor három végrehajtható fájlt kellene készítenünk – külön- külön mindegyik rendszerhez.

A Java rendszerfüggetlenségének fontosságát akkor érthetjük meg igazán, ha végiggondoljuk, hogy a felhasználók miként futtatják a Java appleteket. Amikor egy felhasználó szörfözik a hálón, különböző Web-helyeket látogat meg, ahonnan dokumentumokat tölt le (ezek szöveget, grafikát vagy akár Java appleteket is magukban foglalhatnak). Ha a felkeresett helyen vannak Java appletek, akkor a felhasználó böngészője futtatni fogja azokat. Ha egy másik felhasználó látogat el ugyanerre a helyre, akkor a folyamat megismétlődik, és ennek a felhasználónak a böngészője is futtatni fogja az appleteket. Vegyük észre, hogy a vázolt esetben nincs szó arról, hogy az egyik felhasználó Windows rendszert, míg a másik esetleg Macintosh rendszert használ. Mivel a Java rendszerfüggetlen, nem számít, hogy a végfelhasználó milyen rendszerrel dolgozik.

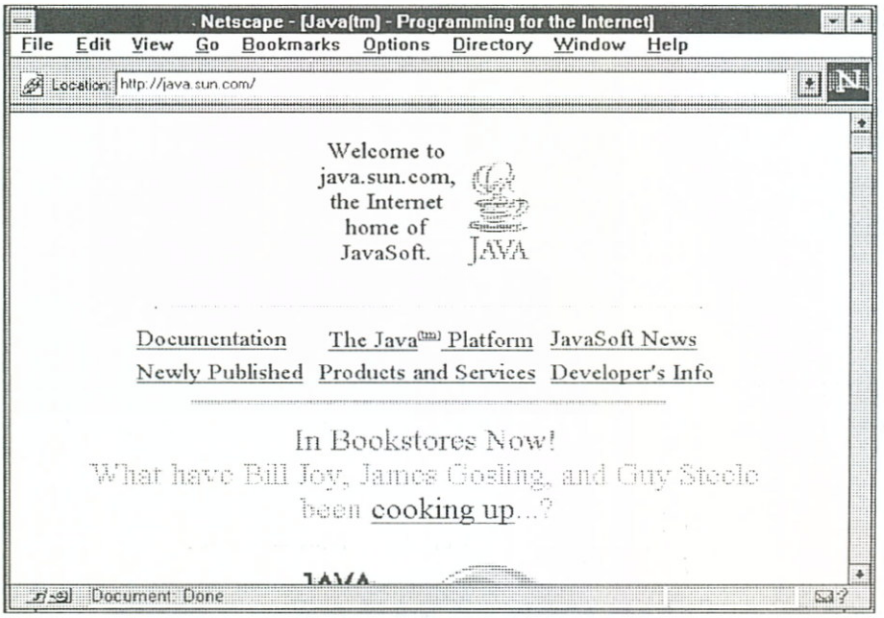
A számítógépek 0-kkal és 1-esekkel dolgoznak. Amikor programot írunk, egy fordítóprogram alakítja át a programozási nyelv számunkra érthető utasításait a számítógép számára értelmezhető 0-k és 1-esek sorozatává. Ha más programozási nyelveken, például C++ nyelven írunk programot, akkor a fordítóprogram által előállított 0-k és 1-esek sorozata processzorfüggő lesz – vagyis vagy csak egy Intel processzor, vagy csak egy Motorola processzor számára lesznek érthetőek. Ezzel szemben a Java fordítóprogram másként működik. Ahelyett, hogy a 0-k és az 1-esek processzorfüggő sorozatát állítaná elő, a Java fordítóprogram egy közbenső, *virtuális gépi kódot* készít, ami processzorfüggetlen. A *Javát értő* böngészők megértik ezt a virtuális kódot, és a bennük lévő 0-k és 1-esek sorozatát arra a formátumra alakítják át (fordítják le), amelyet a felhasználó processzora megért.

Tegyük fel például, hogy egy Mac rendszerben dolgozó felhasználó letölt egy appletet. Ebben az esetben a Mac alapú böngésző a virtuális gépi kódot a 0-k és 1-esek olyan sorozatává alakítja át, amelyet a Motorola processzor megért. Hasonlóképpen, ha egy felhasználó Windows rendszerben böngészik, akkor a böngészője a virtuális gépi kódot a 0-k és 1-esek olyan sorozatává alakítja át, amelyet az Intel processzor ért meg.

A virtuális gépi kódnak az az előnye, hogy a programozónak csak egyetlen appletet kell elkészítenie, amelyet számos különböző rendszer támogat. A Java rendszerfüggetlenségéért a felhasználónak a sebességgel kell fizetnie. Mivel a böngészőnek a virtuális gépi kódot le kell fordítania processzorfüggő kódra, a Java alapú programok nem futnak olyan gyorsan, mint a processzorfüggő kódok. Ha egymás mellé teszünk egy-egy egymással azonos Java és C++ programot, akkor a C++ program (amelyik processzorfüggő kódot használ) 10-20-szor gyorsabban hajtódik végre!

A JAVA LETÖLTÉSE ÉS TELEPÍTÉSE

Más programozási nyelvekhez, mint például a C-hez vagy a C++-hoz hasonlóan egy Java applet készítéséhez *fordítóprogramot* (compiler) kell használnunk, ami a Java program utasításait arra a virtuális gépi kódra fordítja le, amit a böngésző végre tud hajtani. Eltérően azonban a C vagy a C++ nyelvektől, amelyek fordítóprogramjai megvásárolhatók, a Java fordítóprogram jelenleg csak a Sun Web-helyéről tölthető le, ingyenesen. A Sun Web-címe: *http://java.sun.com* (lásd a 14.2. ábrát).



14.2. ábra. Java fordító letöltése a Sun Java Web-helyéről

A Sun Web-oldaláról a Java Windows alapú változata (Windows 95-höz vagy Windows NT-hez – Windows 3.1-hez nincs), Mac alapú változata és UNIX alapú változata tölthető le.

Attól függően, hogy melyik változatot töltjük le, más és más lépéseket kell tennünk a Java fejlesztőkészletének telepítéséhez. A Sun Web-oldalain azonban mindegyik esetre megkapjuk a telepítéshez szükséges információkat.

Megjegyzés: A Javában a kezdeti lépések megtételét segítendő a könyv első kötetében található CD-ROM tartalmazza a Java fejlesztőkészletét (JDK). A CD-ROM-ról telepíthetjük a Java programjait, hogy ne kelljen azokat a Sun Web-helyéről letölteni (a fájlok mérete meghaladja a 4 Mb-ot.)

JAVA FEJLESZTŐKÉSZLET LETÖLTÉSE ÉS TELEPÍTÉSE WINDOWS ALATT

Ha a Windows 95-öt vagy a Windows NT-t használjuk, akkor az alábbi lépések megtételével telepíthetjük a Java fejlesztőkészletet rendszerünkbe:

1. Az Interneten keresztül lépjünk rá a Sun Web-oldalára a *http://java.sun.com* címen.
2. A Web-oldalról töltsük le a Java fejlesztőkészlet Windows változatát. Rendszerünk a Java fejlesztőkészletét egyetlen végrehajtható fájlként fogja letölteni. Amikor végrehajtjuk ezt a programot, a program önmaga kicsomagolja a benne lévő fájlokat, és elhelyezi a merevlemezünkön.
3. Mielőtt még futtatnánk a Java fejlesztőkészletet tartalmazó programot, válasszuk ki azt a merevlemezt, amelyiken a készletet tárolni szeretnénk. Ezt követően válasszuk ki azt a könyvtárt (ez feltehetően a gyökérkönyvtár), amelyikben a programmal létre akarjuk hozatni azt a JAVA könyvtárat és alkönyvtárakat, ahová a programnak el kell helyeznie a fejlesztőkészletet.

4. Futtassuk az előbb letöltött programot, amely kicsomagolja a benne lévő fájlokat a merevlemezünkre, a megfelelő alkönyvtárakba.
5. Szerkesszük a rendszerünk AUTOEXEC.BAT állományát, és a PATH bejegyzés sorába vegyük be a BIN alkönyvtárt, amely a Java könyvtár alatt helyezkedik el. Ha például a Java fejlesztőkészletet a C meghajtó gyökérkönyvtárába telepítettük, akkor a PATH bejegyzés sora ilyen lehet:

```
PATH C:\WINDOWS;C:\WINDOWS\COMMAND;C:\JAVA\BIN
```

6. Az AUTOEXEC.BAT állományban egy SET parancs felvételével hozzunk létre egy HOME nevű bejegyzést, amelyik arra a meghajtóra és könyvtárra mutat, amelyiken belül a program létrehozta a JAVA könyvtárt. Ha például a Java fejlesztőkészletet a C meghajtó gyökérkönyvtárába telepítettük, akkor a SET utasítás ilyen lehet:

```
SET HOME=C:\
```

7. Olvassuk át a Java fejlesztőkészlet részeként megkapott dokumentációt és bizonyosodjunk meg arról, hogy a telepítés itt megadott lépései a könyv írása (és fordítása) óta nem változtak meg.

MIT TARTALMAZ A JAVA FEJLESZTŐKÉSZLETE?

A Java fejlesztőkészlete a Java fordítóprogram mellett még néhány segédprogramot is tartalmaz, többek között az *appletnézőt* (appletviewer), amellyel futtathatjuk és megjeleníthetjük az appleteket, egy *hibakeresőt* (debugger), amellyel megkereshetjük és kijavíthatjuk az appletekben lévő hibákat, és egy *dokumentumgenerátort*, amely leegyszerűsíti a kód dokumentálásával kapcsolatos munkát. A 14.1. táblázatban felsoroljuk a Java fejlesztőkészletben található programokat.

Program neve	Program rendeltetése
java	Java interpreter, amellyel az önálló programokat futtathatjuk
javac	Java fordítóprogram (compiler)
javadoc	Java dokumentumgenerátor
javah	C fájlgenerátor, amely fejlécet és forrásfájlokat hoz létre az osztálydefiníciókhoz
javap	Java osztály-diszasszembler
jdb	Java hibakereső (debugger)

14.1. táblázat. A Java fejlesztőkészletben lévő segédprogramok

JAVA FEJLESZTŐKÉSZLET TELEPÍTÉSE A CD-ROM-RÓL

Ha a Windows 95 vagy a Windows NT operációs rendszert használjuk, akkor az alábbi lépések végrehajtásával telepíthetjük a Java fejlesztőkészletét a könyvhöz megvásárolható CD-ROM-ról:

1. Helyezzük a CD-ROM-ot a meghajtóba.
2. Ha a Windows 95 operációs rendszert használjuk, akkor a Start menüből válasszuk a Futtatás menüpontot. Ha Windows NT rendszert használunk, akkor a Programkezelő Fájl menüjéből válasszuk a Futtatás parancsot.
3. A Futtatás párbeszédpanelen írjuk be a CD-meghajtó betűjelét, majd a JDK102 programnevet. Ha például a CD-meghajtónk betűjele D:, akkor a D:\JDK102 szöveget kell beírnunk. A program kibontja a Java fájlokat a merevlemez megfelelő alkönyvtáraiba.
4. Szerkesszük a rendszerünk AUTOEXEC.BAT állományát és a PATH bejegyzés sorába vegyük be a BIN alkönyvtárt, amely a Java könyvtár alatt helyezkedik el. Ha például a Java fejlesztőkészletet a C meghajtó gyökérkönyvtárába telepítettük, akkor a PATH bejegyzés sora ilyen lehet:

```
PATH C:\WINDOWS;C:\WINDOWS\COMMAND;C:\JAVA\BIN
```

5. Az AUTOEXEC.BAT állományban egy SET parancs felvételével hozzunk létre egy HOME nevű bejegyzést, amelyik arra a meghajtóra és könyvtárra mutat, amelyiken belül a program létrehozta a JAVA könyvtárat. Ha például a Java fejlesztőkészletet a C meghajtó gyökérkönyvtárába telepítettük, akkor a SET utasítás ilyen lehet:

```
SET HOME=C:\
```

6. Olvassuk át a Java fejlesztőkészlet részeként megkapott dokumentációt és bizonyosodjunk meg arról, hogy a telepítés itt megadott lépései a könyv írása (és fordítása) óta nem változtak meg.

ELSŐ JAVA APPLETÜNK ELKÉSZÍTÉSE

Ugyanúgy, mint a C/C++ vagy a Perl nyelvben, egy Java applet elkészítéséhez is szövegszerkesztő segítségével kell beírnunk azokat az utasításokat, amelyek leírják a Java applet által elvégzendő feladatot. Ezeket a programutasításokat egy fájlban, az applet úgynevezett *forrásfájljában* tároljuk. A forrásfájl létrehozásához valamilyen szövegszerkesztőt, például a DOS-ban lévő EDIT szerkesztőt vagy a Windows Jegyzettömbjét (Notepad) kell használnunk. A szövegszerkesztő segítségével hozzuk létre a *TesztPróba.java* fájlt, amely az alábbi utasításokat tartalmazza: (Programokon belül fájlnevekben és változónevekben nem célszerű az ékezetes betűk használata. Programokon kívül a könnyebb olvashatóság céljából használunk ékezetes betűket. – A fordító megjegyzése.)

```
import java.awt.*;
import java.applet.*;

public class TesztProba extends Applet
```

```
    public void paint(Graphics g)
      {
        g.drawString("A Java tesztpróbája!", 5, 25);
      }
}
```

Mint majd látni fogjuk, a fenti utasításokat végrehajtva az applet megjeleníti *A Java tesztpróbája* szöveget az applet ablakában. Nagyon figyelmesen, pontosan az itt látható módon írjuk be ezeket az utasításokat. Ha valahol elírunk valamit, például kifelejtünk egy pontosvesszőt, vagy kisbetűt írunk ott, ahol az applet nagybetűt használ, akkor a Java fordító a fordításkor hibát jelez, amint a következőkben látni fogjuk.

AZ APPLET LEFORDÍTÁSA

Amint láttuk, az appletünk forrásfájlja a *java* kiterjesztést használja, így a fájl teljes neve *TesztProba.java*. Attól függően, hogy miként futtatjuk az appletet, az *appletnézőnk* vagy a böngészőnk végrehajtja az applet osztályfájljában *(TesztProba.class)* lévő 0-k és 1-esek sorozatát (a virtuális gépi kódot).

A Java programutasításainak virtuális gépi kódra való átalakításához a forrásfájlt egy speciális programmal, a *Java fordítóprogrammal* le kell fordítani. A Java fordítóprogram megtalálható a Java fejlesztőkészletben.

Más Windows alapú programoktól eltérően a Java fordítóprogramot a parancssorból futtatjuk. Windows 95-ben úgy érhetjük el a parancssort, hogy a Start menüből a Futtatás parancsot választjuk, a Futtatás mezőbe beírjuk a COMMAND parancsot, majd lenyomjuk az ENTER billentyűt. Windows NT rendszerben úgy érhetjük el a parancssort, hogy a Programkezelő Fájl menüjéből kiadjuk a Futtatás parancsot, a párbeszédablakba beírjuk a COMMAND parancsot, majd lenyomjuk az ENTER-t. Esetünkben a *TesztProba.java* forrásfájl lefordításához az alábbi parancsot írjuk be:

```
C:\Web_prog> javac TesztProba.java <Enter>
```

Ebben az esetben a *javac* parancs indítja a fordítóprogramot, ami viszont lefordítja a parancssorban megadott forrásfájlt. Ne feledkezzünk meg arról, hogy a Java „érzékeny" a kis- és a nagybetűs írásmódra. A forrásfájl nevében ugyanúgy kell használnunk a kis- és a nagybetűket, mint ahogyan azokat a Java programutasításaiban használjuk.

Más szavakkal ez azt jelenti, hogy ha az applet neveként a *TesztProba* nevet használtuk, a parancssorba viszont a *tesztproba* nevet írjuk be, akkor a Java fordítóprogram hibaüzenetet fog küldeni (a parancssorba a *TesztProba.java* nevet kell beírnunk, amelyben a kis- és a nagybetűk pontosan megegyeznek a forrásprogramban használt írásmóddal.)

Ha a Java utasításokat pontosan úgy írtuk be, ahogyan azok a könyvben láthatók, akkor a fordítóprogram nem küld hibaüzenetet. Ha viszont elírtunk valamint, akkor a fordítóprogram egy vagy több szintaktikai hibaüzenetet jelenít meg. Mielőtt a fordítóprogram sikeresen lefordítaná a forrásfájlt, szerkesztenünk kell azt, és ki kell javítanunk benne az összes szintaktikai hibát. A hibák kijavítása után ismét futtatnunk kell a fordítóprogramot.

A CLASS FÁJL

Ha sikerült lefordítanunk a Java appletet és kilistáznunk a könyvtárban lévő fájlokat, akkor a listában észrevehetünk egy *TesztProba.class* nevű fájlt. Az appletnek ez a *class* fájlja tartalmazza a virtuális gépi kódot, amit az *appletnéző* vagy valamilyen böngésző végre tud hajtani:

```
C:\Web_prog>dir

 C meghajtóban lévő kötetnek nincs címkéje.
 A kötet sorozatszáma: 1E65-12E7
 C:\Web_prog könyvtára

.              <DIR>          96.12.02  15.07 .
..             <DIR>          96.12.02  15.07 ..
TESZTP~1 JAV           217  96.12.02  15.09 TesztProba.java
TESZTP~1 CLA           427  96.12.02  15.11 TesztProba.class
        2 fájl            644 bájt
        2 könyvtár   903 086 080 bájt szabad.
```

Az applet futtatásához most létre kell hoznunk egy HTML fájlt, amelyre az *appletnézőnek* vagy valamilyen böngészőnek szüksége van ahhoz, hogy hozzáférhessen az applethez.

HTML FÁJL LÉTREHOZÁSA

A könyv 4. és 5. fejezetében részletesen foglalkoztunk a HTML fájlokkal. Amint láttuk, a Webhelyek fejlesztői sűrűn használják a HTML nyelvet. A Web-tervezők a HTML segítségével adják meg a Web-oldalon elhelyezendő szöveget, grafikát és hangot. Java applet Web-oldalra történő elhelyezéséhez a tervezők az <APPLET> címkével adják meg az applet class fájlját, valamint olyan más beállításokat, mint az applet ablakának mérete.

A *TesztProba* applet futtatásához hozzuk létre *a TesztProba.HTML* fájlt, amely az alábbi HTML bejegyzéseket tartalmazza:

```
<HTML><TITLE>TesztProba Applet</TITLE>
<APPLET CODE="TesztProba.class" WIDTH=200 HEIGHT=100></APPLET></HTML>
```

Ebben az esetben a HTML <APPLET> címkéje adja meg az applet class fájlját és az ablak képpontokban mért méretét. Bár a *TesztProba.HTML* fájlnév megfelel az applet nevének, a HTML fájlnak bármilyen nevet adhatunk, ami például az *ElsőApplet.HTML* is lehet. Amint azonban növekszik létrehozott appletjeink száma, tapasztalni fogjuk, hogy a tartalomra utaló HTML fájlneveket használva könnyebben eligazodunk a fájljaink között.

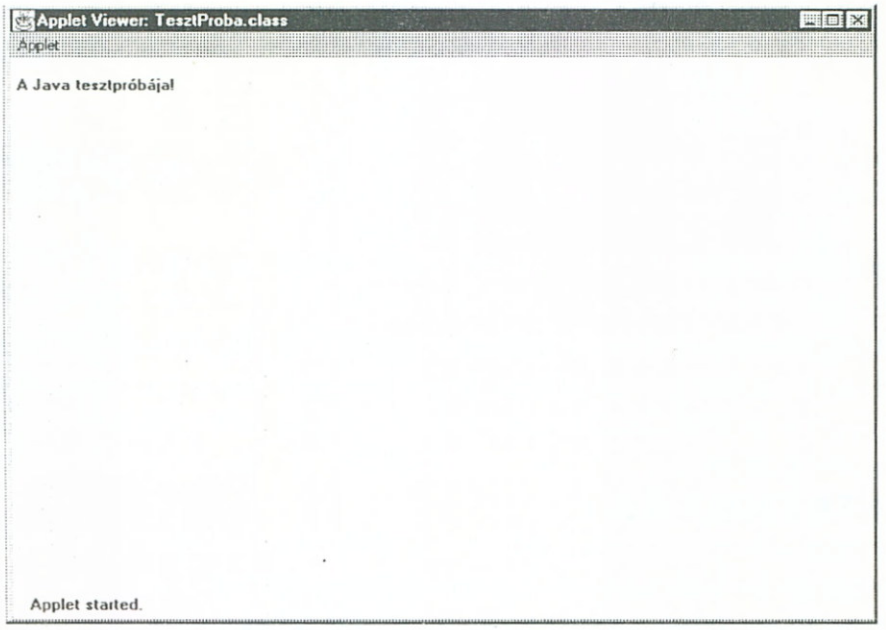
AZ APPLETNÉZŐ HASZNÁLATA

Létrehozott appletjeinket kétféle módon futtathatjuk. Ha időt akarunk megtakarítani, akkor használjuk az *appletnézőt* (appletviewer). Az appletnéző speciális program, amely a Java fejlesztőkészlet része, és amellyel gyorsan elindíthatunk és megnézhetünk egy appletet. Másik lehetőségként böngészőnkkel betölthetjük az applet megfelelő HTML dokumentumát a merevlemezünkön tárolt

fájlból (az URL címet használva, mint például *fájl://valamilyen_út*). A *TesztPróba* applet *appletnézővel* való futtatásához írjuk be a következő parancsot:

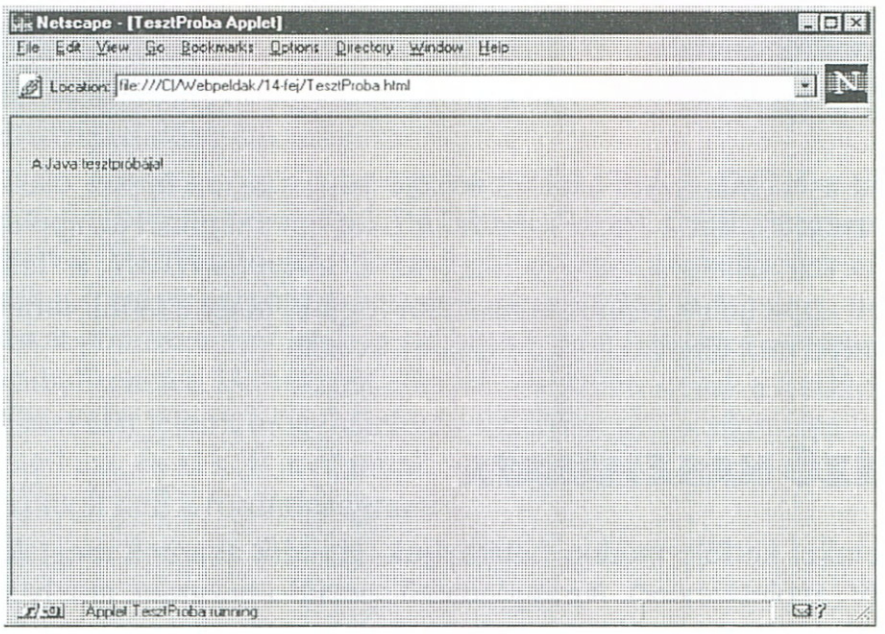
```
C:\Web_prog> appletviewer TesztProba.HTML <Enter>
```

Amint látjuk, ez a parancs futtatja az *appletviewer* nevű appletnézőt, és megadja azt a HTML fájlt, amely a *TesztPróba* appletnek megfelelő <APPLET> címkét tartalmazza. Az appletnéző ekkor megnyitja a 14.3. ábrán láthatóhoz hasonló applet ablakot, amelyben megjelenik az *A Java tesztpróbája!* szöveg. Az applet befejezéséhez zárjuk be az applet ablakát.



14.3. ábra. Az appletnéző futtatja a *TesztPróba* appletet

A *TesztPróba* applet valamely böngészővel való futtatásához írjuk be a böngésző címmezőjébe a HTML fájlnak a lemezen elfoglalt helyét. Ha például a HTML fájl a *C:\Web_prog* könyvtárban van, akkor a 14.4. ábrán látható URL-t (Unified Resource Locator) kell beírnunk.



14.4. ábra. A *TesztProba* applet futtatása a Netscape Navigator böngészővel

MEGJEGYZÉSEK HASZNÁLATA A JAVA NYELVBEN

Amikor Java appleteket készítünk, megjegyzéseket írunk a programba, amelyek elmagyarázzák az applet rendeltetését, és megkönnyítik mások számára az egyes utasítások céljának megértését. A Java – a C++ nyelvhez hasonlóan – lehetővé teszi mind az egy-, mind a többsoros megjegyzések beírását. A következő utasítások a Java egysoros megjegyzéseire mutatnak példát:

```
int szamlalo;                          // Találatok száma a mai napon
                                       // (a hely látogatóinak száma)
String_HelyNev = Wow Dude!";           // Új hely az MTV-n
float belepodij = 4.95;                // Felhasználó költsége óránként
```

Amikor a Java fordító ilyen kettős ferde törtvonallal (//) találkozik, az utána álló sort figyelmen kívül hagyja. Több sorra kiterjedő megjegyzéseket az alábbi parancsformátumban készíthetünk:

```
/* A megjegyzés kezdete
a megjegyzés további sorai
a megjegyzés vége */
```

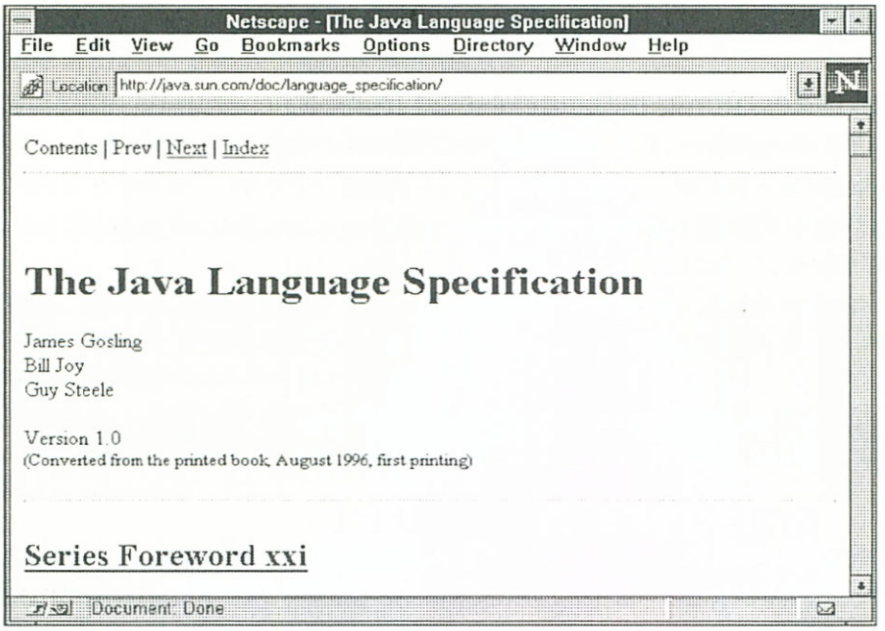
Ebben az esetben a Java fordítóprogram a /* és a */ jelek közötti összes szöveget figyelmen kívül hagyja.

A JAVA NYELV KÜLÖNBSÉGET TESZ A KIS- ÉS A NAGYBETŰK KÖZÖTT

Az egyik legfontosabb dolog, amiről egy Java applet készítése során nem szabad megfeledkeznünk, az az, hogy a Java nyelv „érzékeny" a kis- és nagybetűs írásmódra. Ez azt jelenti, hogy a Java másként kezeli a nagybetűket és másként a kisbetűket. Ezért a Java a *számláló, SZÁMLÁLÓ* és a *Számláló* nevű változókat három különböző változónak tekinti. Rendszeresen érkeznek az olvasóktól olyan elektronikus üzenetek, amelyek különböző problémákról számolnak be. A problémák döntő többsége abból származik, hogy a programozók megfeledkeznek erről a szabályról, és a kisbetű helyett nagybetűt írnak vagy fordítva. Amikor a Java fordítóprogram hibát jelez, akkor elsőként azt ellenőrizzük, hogy nem hibáztuk-e el valahol a megfelelő írásmódot.

A JAVA NYELVI SPECIFIKÁCIÓJÁNAK LETÖLTÉSE

Ennek a fejezetnek az a célja, hogy segítsen a Java programozási nyelv alapjainak elsajátításában és használatában. Ezért a fejezetben azokkal az alapgondolatokkal ismerkedünk meg, amelyek lehetővé teszik egyszerűbb Java appletek készítését. Amint azonban az appleteink egyre összetettebbé válnak, további információkra lehet szükségünk olyan témakörökben, mint például a Java csomagok, hálózati műveletek, adatbázis-feldolgozások stb. Az ilyen témakörökben az egyik legjobb (és ingyenes) információforrás a Java Language Specification (Java nyelvi specifikáció), ami a Web *http://java.sun.som/doc/language_specification/* címén található meg (lásd a 14.5. ábrát).



14.5. ábra. A Java Language Specification letöltése

ÖNÁLLÓ PROGRAMOK ÉS APPLETEK

A Java applet kisméretű alkalmazás (innen származik a neve is: application = nagyobb méretű alkalmazás), amelyet a felhasználó *appletnéző* vagy böngésző segítségével futtathat. A programozók a Weben történő használatra készítenek Java appleteket. Röviden már említettük, hogy a Java tervezői a vírusok terjeszthetőségének csökkentése céljából korlátozták az appletek által végrehajtható műveleteket. Így például egy applet nem képes I/O műveletek végrehajtására (nem tud információkat írni a távoli felhasználó lemezére, és olvasni sem tud onnan).

Ezzel szemben egy *önálló program* olyan program, amely a ,,saját erejéből" fut (vagyis nem egy böngészőben vagy az *appletnézőben*). Amikor a C/C++ programozási nyelv segítségével megírunk egy programot, akkor végrehajtható (EXE kiterjesztésű) fájlt hozunk létre, ami önálló program. Más szavakkal ez azt jelenti, hogy miután elkészült az EXE program, az minden más segédeszköz nélkül futtatható. Ezért az EXE programok önálló programok.

Bár a Java programozók a ,,nem applet" programokat önálló programoknak nevezik, a dolog azért nem egészen igaz. Ahhoz, hogy egy Java programot futtassunk, egy speciális, JAVA.EXE nevű programot is futtatnunk kell. A JAVA.EXE program a Java értelmező (interpreter) programja.

A Java programok könnyebb megértése céljából készítsük el a *HalloVilág.java* forrásprogramot, amely az alábbi utasításokat tartalmazza:

```
class HalloVilag
  {
    public static void main(String args[])
      {
        System.out.println("Halló, Világ!");
      }
  }
```

Ez a program a képernyőnkön jeleníti meg a *Halló, Világ!* üdvözletet és nem egy applet ablakában. A program használatához a fenti forrásfájlt még le kell fordítanunk a Java fordítóprogramjával:

```
C:\Web_prog> javac HalloVilag.java <Enter>
```

A fordítóprogram létrehozza a *HalloVilag.class* nevű class fájlt. Ha kilistázzuk a könyvtárt, akkor abban az alábbi bejegyzéseket láthatjuk:

```
C:\Web_prog>dir

 C meghajtóban lévő kötetnek nincs címkéje.
 A kötet sorozatszáma: 1E65-12E7
 C:\Web_prog könyvtára

.              <DIR>          96.12.02  15.07 .
..             <DIR>          96.12.02  15.07 ..
HALLOV~1 JAV             150  96.12.03  12.08 HalloVilag.java
HALLOV~1 CLA             475  96.12.03  12.10 HalloVilag.class
         2 fájl             625 bájt
         2 könyvtár   892 665 856 bájt szabad.
```

Az önálló Java programunk futtatásához a *java* parancsot kell beírnunk az alábbiak szerint:

```
C:\Web_prog> java HalloVilag <Enter>
Halló, Világ!
```

A Java interpretert így használhatjuk önálló Java programok futtatásához.

HOSSZÚ FÁJLNEVEK

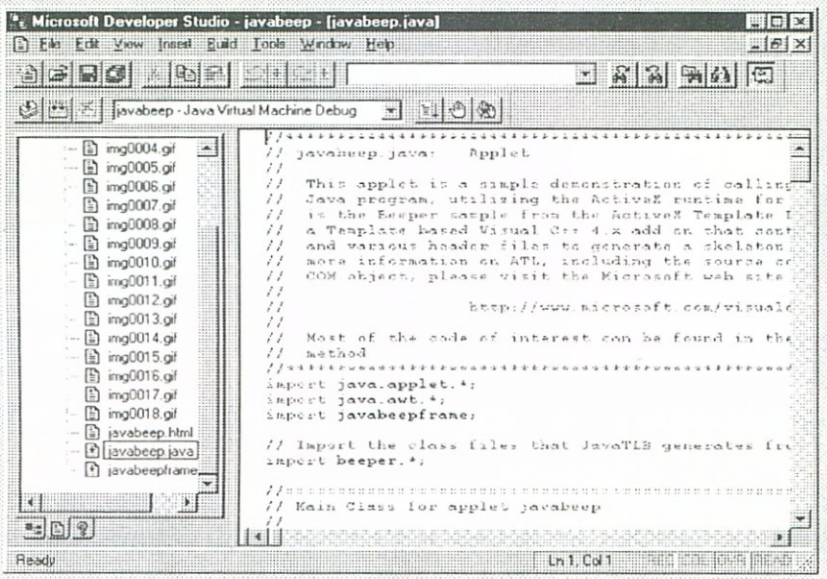
Eleinte sok programozót megzavarhat, ha egy könyvtár kilistázásakor megcsonkított fájlneveket lát, mint például *ForgoV~1.jav*, és nem tudja eldönteni, hogy melyiket kell használnia.

Amint láttuk, ha a Sun Java fejlesztőkészletével dolgozunk, akkor mind a fordítóprogramot, mind az *appletnézőt* a parancssorból kell meghívnunk. A Java appletek legtöbbjének hosszú, jelentést hordozó fájlneve van, mint például *ForgóVilágDemo.java* vagy *UgatóDalmata.java*. Mivel ezek a fájlnevek meghaladják a DOS-ból ismert 8.3-as formátumot (8 karakteres fájlnév és 3 karakteres fájlnév-kiterjesztés), kezdetben sok programozó nem tudja, hogy adott esetben melyik fájlt kell használnia, ha a nevek csonkított formában jelennek meg, mint például *ForgóV~1.jav* vagy *UgatóD~1.jav*. Mivel a Java fordító vagy az *appletnéző* számára a fájlneveket pontosan úgy kell megadni, ahogyan azok az applet osztálynevében szerepelnek, mindig a teljes, csonkítatlan fájlnevet kell használnunk. Ha a Windows 95 rendszerben megvizsgáljuk a könyvtárlistákat, akkor a lista bal oldalán a csonkított, míg a jobb oldalán a teljes fájlneveket láthatjuk.

Ha használjuk a könyv első kötetében található CD-ROM-on tárolt appleteket, és nem látjuk a hosszú fájlneveket, akkor próbáljuk ki a CD-t egy másik felhasználó rendszerében. Ha egy másik CD-meghajtóban megjelennek a hosszú fájlnevek, akkor feltehetően valós módú illesztőprogramot használunk a meghajtóhoz, amit célszerű frissítenünk. (A hosszú fájlnevek nemcsak így válnak olvashatóvá, de ez a jobb megoldás.)

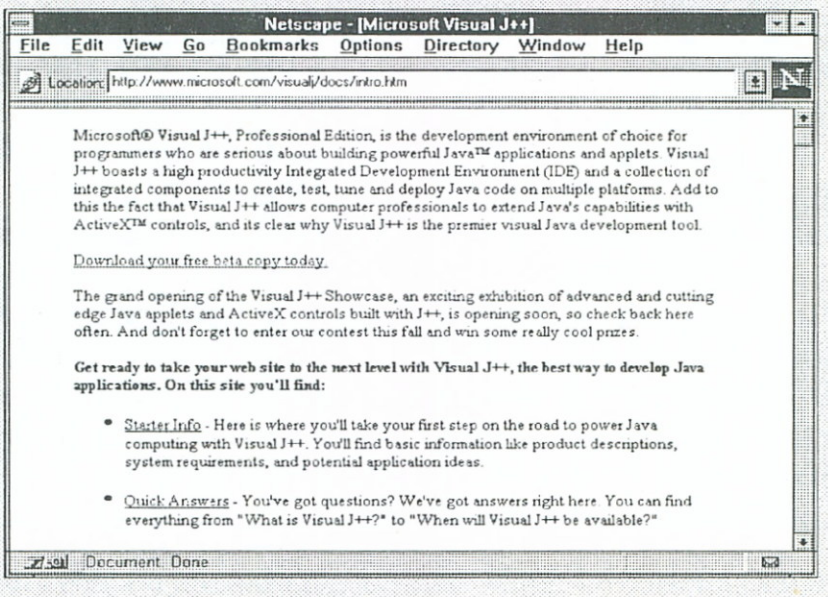
VISUAL J++ (KORÁBBI NEVE JAKARTA)

A Java programozási nyelvet a Sun Microsystems fejlesztette ki. A Java hatalmas sikerét látva nem telt el sok idő, és a Microsoft is megjelent a Java piacán: a Visual C++ eszközkészletére építve piacra dobta a Visual J++ grafikus programozási nyelvet, amelynek segítségével a felhasználók Java appleteket készíthetnek, tesztelhetnek és futtathatnak. Amint a 14.6. ábrán látható, a Visual J++ környezete nagyon hasonlít a Visual C++ környezetéhez



14.6. ábra. A Visual J++ programozási környezet

Általában a Visual J++ programozási környezetben készített Java appleteket ugyanúgy lehet végrehajtani, mint a Java fejlesztőkészletével létrehozottakat. A Visual J++ környezet gyors fordítóprogramot, beépített hibakeresőt és sok mást tartalmaz. Ezen túl a Visual J++ lehetővé teszi az ActiveX (OLE) vezérlők elérését, amivel a 15. fejezet foglalkozik részletesen. További információkat kaphatunk a Visual J++ programozási nyelvről a következő címen: *http://www.microsoft.com/visualj/docs/ intro.htm* (lásd a 14.7. ábrát).



14.7. ábra. Információk a Microsoft *VisualJ++* programozási nyelvéről

AZ ÖNÁLLÓ PROGRAMOK NEM BÖNGÉSZHETŐK

Eltérően egy Java applettől, amelyik valamilyen böngészőben futtatható, az önálló Java programok böngészővel nem használhatók. Amint láttuk, a Java fejlesztői annak érdekében, hogy csökkentsék a vírusok terjesztésének lehetőségét, korlátozták az applet által végrehajtható műveleteket. Ugyanakkor az önálló programokat nem sújtják ezek a korlátozások. Ezért vannak olyan műveletek, amelyeket csak az önálló Java programok tudnak végrehajtani, az appletek viszont nem.

A JAVA OSZTÁLYKÖNYVTÁRAI

Mint minden programozási nyelv, a Java is rendelkezésünkre bocsát egy terjedelmes futásidejű könyvtárat, ami számos előre definiált függvényt tartalmaz. E függvények segítségével rengeteg programozási és tesztelési időt takaríthatunk meg. A Java programozók általában *osztálykönyvtáraknak* nevezik ezeket az előre definiált könyvtárakat.

Java appletek készítése során igen sokszor használunk ilyen osztálykönyvtárakat. A könyvtárban lévő kódok lehetővé teszik, hogy ne kelljen megírni azokat a kódokat, amelyek az egeret vezérlik, alacsonyabb szintű grafikai és betűműveleteket végeznek stb. Ehelyett igénybe vehetjük a könyvtárakban már meglévő kódokat.

Valamely osztálykönyvtár használatához az appletnek tartalmaznia kell az *import* utasítást, ami megadja a Java fordítóprogram részére annak az osztálykönyvtárnak a nevét, amelyik a használni kívánt kódot tartalmazza. Amikor lefordítjuk az appletet, a Java fordító automatikusan „bedolgozza" a könyvtárban lévő kódot az appletbe. Ha megvizsgáljuk a Java appleteket, láthatjuk, hogy mindegyik egy vagy több *import* utasítással kezdődik.

Ha megnézzük az előző appleteinket, látjuk, hogy az első két utasításuk *import* utasítás, amelyek azt mondják meg a Java fordítóprogramnak, hogy az applet használni akarja az *awt* és az *applet* osztálykönyvtárat. Az alábbi utasítások szemléltetik az *awt* és az *applet* osztálykönyvtár használatát:

```
import java.awt.*;
import java.applet.*;
```

Az *awt* osztálykönyvtár azokat a kódokat tartalmazza, amelyek az *applet* ablakműveleteit vezérlik. Ezek a kódok vezérlik a szöveg és a grafika megjelenítését, az ablak méretét stb. Az *applet* osztálykönyvtár azt a kódot tartalmazza, amelyiket egy böngésző használ az applet futtatásához, majd a leállításához, továbbá azt a kódot is, amelyik az applet végrehajtását vezérli. Mivel ez a két osztálykönyvtár alapvetően fontos műveleteket végez, általában minden applet elején megtalálhatók azok az *import* utasítások, amelyek ezeket a könyvtárakat hívják.

AZ APPLET OSZTÁLY KIBŐVÍTÉSE

A Java appletekben általában találkozhatunk az alábbihoz hasonló utasítással, amely *kibővíti* (extends) az applet osztályt:

```
public class ValamilyenNév extends Applet
```

Amint az előbb említettük, az *applet osztálykönyvtár* egy *applet* végrehajtását vezérlő kódot tartalmazza. Pontosabban fogalmazva az osztálykönyvtár annak az *applet osztálynak* a kódját tartalmazza, amellyel a Java fordítóprogram létrehozza az applet objektumot. Amikor létrehozunk egy

Java appletet, akkor lényegében olyan programutasításokat adunk meg, amelyek a létező *applet osztályt* bővítik ki. Így az előző utasítás azt közli a java fordítóprogrammal, hogy egy *ValamilyenNév* nevű appletet készítünk, amelyik kibővíti a létező *applet osztályt*. Az utasítás elején álló *public kulcsszó* teszi lehetővé, hogy a Java böngésző futtathassa az appletünket. Ha kihagynánk a *public kulcsszót*, akkor a böngésző nem tudna hozzáférni az applethez.

A JAVA VÁLTOZÓTÍPUSAI

A legtöbb programhoz hasonlóan a Java appletek is változókban tárolják az értékeket a végrehajtásuk során. Egy Java appleten belül mindegyik változónak meghatározott típusúnak kell lennie, mint például *int* (egész számú értékek) vagy *float* (lebegőpontos értékek) stb. Egy változó típusa általánosságban egyrészt azt az értékkészletet határozza meg, amelyet az illető típusú változó tárolni tud (az *int* típusú változók például egész számokat tárolnak), másrészt azokat a műveleteket, amelyek az illető típusú változón elvégezhetők (egy program például össze tud szorozni két számot, de karakterláncok szorzása nem értelmezhető). A 14.2. táblázat a Java változótípusait sorolja fel.

Típus	Tárolt érték
boolean	True vagy false (igaz vagy hamis) logikai érték
byte	–128 és 127 közötti értékek
char	Alfanumerikus karakterek (betűk, számok és szimbólumok) a 16 bites Unicode karakterek alapján
double	$-1{,}7 \times 10^{-308} - 1{,}7 \times 10^{308}$ tartományban lévő értékek
float	$-3{,}4 \times 10^{-38} - 3{,}4 \times 10^{38}$ tartományban lévő értékek
int	–2 147 483 648 – 2 147 483 647 tartományban lévő értékek
long	–9 223 372 036 854 775 808 – 9 223 372 036 854 775 807 tartományban lévő értékek
short	–32 768 – 32 767 tartományban lévő értékek

14.2. táblázat. A Java változótípusai

A C/C++ programozási nyelvekhez hasonlóan mielőtt egy Java appletben használhatnánk egy változót, deklarálnunk kell azt. Deklaráláskor a változó nevének és típusának megadásával mintegy „bemutatjuk" a változót a Java fordítóprogramnak. Az alábbi utasítások például *int, float* és *long* típusú változókat deklarálnak:

```
int eletkor;
float belepodij = 4.95;
long a_Mars_tavolsaga;
```

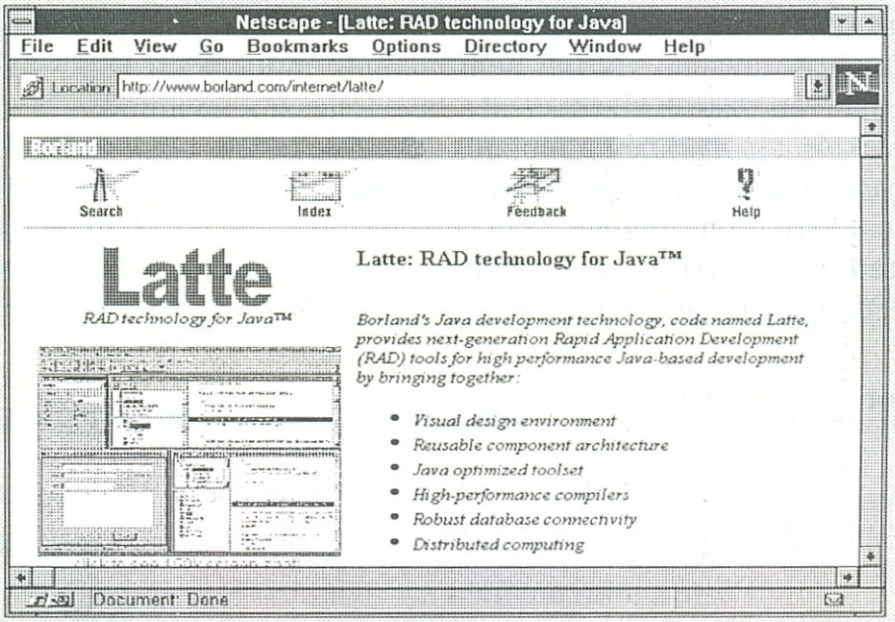
Amint látjuk, az értékadás operátorával (=) már a változó deklarálásakor értéket adhatunk a változónak.

Ha már programoztunk más nyelveken, mint például C/C++ nyelven, akkor ismerősek lehetnek a globális változók, amelyek értékei a teljes programon (vagy osztályon) keresztül ismertek. Mivel a globális változók olyan hibákat okozhatnak, amelyeknek nagyon nehéz kideríteni az okát, a Java nem teszi lehetővé globális változók létrehozását.

A C/C++ programozási nyelvben járatosak feltehetően mutatókat is használtak programjaikban. Mivel egy hibás mutatóművelet „kevés a memória" helyzethez vezethet, aminek következtében egy hibásan elkészített applet a rendszer teljes leállását is okozhatja, a Java nem teszi lehetővé a mutatóműveleteket.

LATTE PROGRAMOZÁSI NYELV

Amint láttuk, a Microsoft a Visual J++ nyelv piacra hozatalával nagyobb teljesítményű programozási környezetet bocsátott a Java programozók rendelkezésére. Hasonlóképpen a Borland is megjelent a piacon a *Latte* nevű vizuális Java programozási környezetével. A Latte alapjául a Borland C/C++ programozási eszközei szolgáltak. A Borland gyors alkalmazásfejlesztői környezetként népszerűsíti a termékét a Java programozók között. A Latte grafikus tervezői környezetet, nagy sebességű fordítóprogramot, adatbázis-támogatást és elosztott működést bocsát a programozók rendelkezésére. Röviden, a Latte segítségével nagyon gyorsan készíthetünk, tesztelhetünk és futtathatunk mindenféle olyan appletet, amelyet a Java fejlesztőkészletével is létrehozhatunk. További információkat kaphatunk a Latte programozási környezetről, ha meglátogatjuk a következő helyet: *http://www.borland.com/internet/latte/* (lásd a 14.8. ábrát).



14.8. ábra. Információk a Borland Latte nevű programozási környezetéről

A JAVA KULCSSZAVAI

Java appletek és programok készítése során az ütközések elkerülése céljából névazonosítóként (változók és függvények neveként) nem használhatjuk a 14.3. táblázatban felsorolt kulcsszavakat.

abstract	boolean	break	byte	case
cast	catch	char	class	const
continue	default	do	double	else
extends	final	finally	float	for
goto	if	implements	import	instanceof
int	interface	long	native	new
null	operator	package	private	protected
public	return	short	static	super
switch	synchronized	this	throw	throws
transient	try	unsigned	virtual	void
volatile	while			

14.3. táblázat. Java kulcsszavak

JAVA MŰVELETEK ÉS OPERÁTOROK

Amikor értékeket rendelünk változókhoz, sűrűn használjuk az olyan műveleteket, mint az összeadás, kivonás, szorzás stb. A Java a C programozási nyelvben található legtöbb műveletet támogatja, és néhány újat is definiál. A Javában viszont nem találunk mutatókkal kapcsolatos műveleteket. A mutatók és a memóriahivatkozások kizárásával a Java fejlesztői javítottak a Java biztonságán. Eltérően a C/C++ nyelvektől, amelyekben a mutatók lehetővé tették, hogy egy véletlenül kiválasztott memóriahelyre írjunk és onnan olvassunk, a Java appletek nem teszik lehetővé ezeket a műveleteket.

A C programozási nyelvhez hasonlóan a Java precedenciát rendel a műveletekhez, aminek révén megadható az a sorrend, ami szerint a Java a kifejezéseket kiértékeli. A 14.4. táblázat a műveleteket és az operátorokat a precedenciájuk sorrendjében sorolja fel.

Operátor	Neve	Példa
.	Tagválasztó	objektum.tag_neve
[]	Index	tömb[elem]
()	Függvényhívás	kifejezés(paraméterek)
++	Postfix inkrementáló	változó++
++	Prefix inkrementáló	++változó
--	Postfix dekrementáló	változó--
--	Prefix dekrementáló	--változó
~	1-es komplemens	~kifejezés
!	NOT művelet	!kifejezés
instanceof	objektum példánya	if(objektum_a instanceof osztálynév)
new	Allokáló művelet	új típus
*	Szorzás	kifejezés * kifejezés
/	Osztás	kifejezés / kifejezés
%	Modulo	kifejezés % kifejezés

Operátor	Neve	Példa
+	Összeadás	kifejezés + kifejezés
–	Kivonás	kifejezés – kifejezés
<<	Bitenkénti léptetés balra	kifejezés << kifejezés
>>	Bitenkénti léptetés jobbra	kifejezés >> kifejezés
>>>	Nullával feltöltő jobbra léptetés	kifejezés >>> kifejezés
<	Kisebb mint	kifejezés < kifejezés
>	Nagyobb mint	kifejezés > kifejezés
<=	Kisebb vagy egyenlő	kifejezés <= kifejezés
>=	Nagyobb vagy egyenlő	kifejezés >= kifejezés
==	Egyezőség	kifejezés == kifejezés
!=	Nem egyezőség	kifejezés != kifejezés
&	Bitenkénti AND	kifejezés & kifejezés
^	Bitenkénti kizáró OR	kifejezés ^ kifejezés
|	Bitenkénti OR	kifejezés | kifejezés
&&	Logikai AND	kifejezés && kifejezés
||	Logikai OR	kifejezés || kifejezés
?:	if-else	(boolean_kifejezés) ? igaz_kifejezés: hamis_kifejezés
= operátor	Értékadás	változó *= kifejezés;

14.4. táblázat. A Java operátorai a precedenciájuk szerinti sorrendben

PROGRAMVEZÉRLŐ SZERKEZETEK

Bonyolultabb Java appletek írásakor szükségünk lesz olyan szerkezetek használatára, amelyek a program végrehajtását vezérlik. Ilyen például az *if* és a *while utasításokból* álló, ciklust alkotó szerkezet. Amint látni fogjuk, a Java programvezérlő szerkezetei nagyon hasonlítanak a C programozási nyelvben használtakhoz. A fejezet példaprogramjai számos ilyen szerkezetet mutatnak be. A 14.5. táblázat mindegyik programszerkezetre bemutat egy-egy példát.

Programszerkezet	Formátum
if	if (feltétel) utasítás;
if else	if (feltétel) utasítás; else utasítás;
do	do utasítás; while (feltétel);

Programszerkezet	Formátum
for	for (kifejezés; feltétel; kifejezés) utasítás;
while	while (feltétel) utasítás;
switch	switch (kifejezés) { case 1: utasítás; break; case 2: utasítás; break; default: utasítás;}

14.5. táblázat. Programvezérlő szerkezetek a Java nyelvben

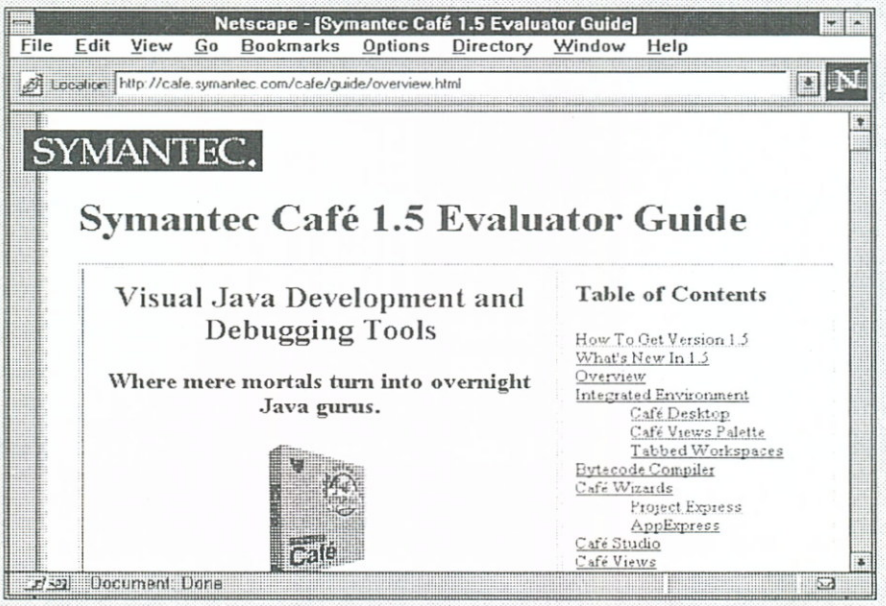
A Java programvezérlő szerkezeteiben az utasítások *egyszeresek* és *összetettek* attól függően, hogy hány utasítást kapcsolnak össze. Az egyszeres szerkezet egyetlen utasítást, míg az összetett több, egymással kapcsolatban lévő utasítást tartalmaz. Az ilyen, egymással kapcsolatos utasításokat kapcsos zárójelek közé kell tenni:

```
if (feltétel)
 egyszeres_utasítás;

if (feltétel)
  {
    elsõ_utasítás;
    második_utasítás
    harmadik_utasítás;
  }
```

Függetlenül attól, hogy egyszeres vagy összetett a programszerkezetünk, behúzásokkal tagoljuk azokat a fent látható módon, hogy megkönnyítsük mások számára az összetartozó utasítások felismerését.

A SYMANTEC CAFÉ

A Microsoft és a Borland szoftverfejlesztő cégekhez hasonlóan a Symantec is kifejlesztette a Symantec Café névre keresztelt szoftverét, hogy integrálhassa a Javát vizuális programozási környezetébe. Általában a Symantec Café integrált fejlesztő és hibakereső programozási környezetet bocsát a Java nyelv rendelkezésére. A programozási környezeten túlmenően a Café bőséges dokumentációt is közzétesz a hálózaton. A Symantec Café programozási környezetről a *http://café.symantec.com/café/guide/overview.html* címen található bővebb információ (lásd a 14.9. ábrát).



14.9. ábra. Információk a Symantec Café programozási környezetről

A JAVA FÜGGVÉNYEI

A programozók a nagyobb C/C++ programozási feladataikat kisebb, jobban kezelhető függvényekre bontják. Ugyanezt tesszük a Javában is. Egy appleten belül egy-egy függvénynek egyetlen, jól meghatározott feladatot kell ellátnia. Minden olyan esetben, amikor az appletnek végre kell hajtania valamilyen feladatot, az applet *meghívja* az ezért „felelős" függvényt és átadja azokat a *paramétereket*, amelyekre a függvénynek a feladat végrehajtásához szüksége van.

A Javában létrehozandó függvényeknek nevet kell adnunk, a név elé be kell írnunk a függvény *visszatérési típusát* (mint pl. *int* vagy *float*), a név után pedig zárójelek között fel kell sorolnunk a paraméterlistát. A függvényhez tartozó utasításokat kapcsos zárójelekkel kell körülvenni:

```
visszateresi_tipus fuggvény_neve(parameter_tipusa parameter_neve)
  {
    // függvényutasítások
  }
```

Egy függvény használatához az appletnek *meg kell hívnia* a függvényt. A következő, *HalloVilag.java* nevű applet egy *MonddHallo* nevű függvényt tartalmaz, amelynek az a feladata, hogy a képernyőn megjelenítse a *Halló, Világ!* szöveget. Ebben az esetben amikor elindul az applet, a böngésző meghívja a *paint* függvényt, hogy frissítse az applet ablakát. A paint függvény viszont meghívja a *MonddHallo* függvényt, ami kiírja a képernyőre az üzenetet:

```
import java.awt.*;
import java.applet.*;

public class HalloVilag extends Applet
  {
```

```
    public void paint(Graphics g)
      {
        MonddHallo(g);
      }

    public void MonddHallo(Graphics g)
      {
        g.drawString("Halló, Világ!", 5, 15);
      }
}
```

Az alábbi HTML bejegyzésekkel hozzuk létre *a HalloVilag.html* fájlt, hogy futtathassuk az appletet:

```
<HTML><TITLE>Hallo Vilag Applet</TITLE>
<APPLET CODE="HalloVilag.class" WIDTH=300 HEIGHT=200></APPLET></HTML>
```

Amint látjuk, a *paint* függvény meghívja a *MonddHallo* függvényt, és átadja neki a *g* változót. Ez a változó tartalmazza azt a *grafikus* környezetet, amit a függvény az üzenet megjelenítéséhez használ. A grafikus *környezet* határozza meg a használandó betűtípust, színt, a szöveg koordinátáit stb.

Amikor az applet a *paint* függvényen belül találkozik a *MonddHallo* függvényhívással, akkor végrehajtja a *MondHallo* függvényben megadott utasításokat. Miután az applet a függvényben definiált valamennyi utasítást végrehajtotta, rátér a közvetlenül a függvényhívást követő első utasítás végrehajtására.

PARAMÉTEREK ÁTADÁSA JAVA FÜGGVÉNYEKNEK

Amint láttuk, a Java appletek értékeket (paramétereket) adhatnak át a függvényeknek, amelyekkel azok végrehajtják a feladatukat. Ha olyan függvényt definiálunk, amelyik paramétereket használ, akkor minden egyes paraméterre meg kell adnunk a paraméter típusát, mint például *int* (egész érték), *float* (lebegőpontos érték), *char* (karakterlánc típusú érték) stb. A következő, *FejezetSzám* nevű függvény például két paramétert használ. Az első paraméter egy *Graphics* nevű objektum, ami a grafikus környezetet tartalmazza arra vonatkozóan, hogy a függvény miként jelenítse meg a fejezetre vonatkozó információt. A második paraméter egy *int* típusú érték, ami a fejezet számának felel meg:

```
void FejezetSzam(Graphics g, int fejezet)
  {
    g.drawString("A fejezet száma ": + 14, 5, 25);
  }
```

Amint látjuk, a függvény az összeadás + műveleti jelét használja arra, hogy a második függvényben (a *drawString* függvényben) egy karakterlánchoz hozzáfűzze (konkatenálja) a fejezet számát. Amikor az appletünk meghívja a *FejezetSzám* függvényt, akkor az appletnek értéket kell átadnia a függvény számára a következők szerint:

```
FejezetSzam(g, 14);
```

A Java ekkor az átadott értéket behelyettesíti a függvényen belül a *g* és az *érték* paraméter helyére.

A JAVA NEM HASZNÁL MUTATÓKAT

A Java és a C/C++ nyelv függvényei között az legjelentősebb eltérés, hogy a Java-függvényekben nem használhatók mutatók. Ebből következően a Java-függvények nem változtathatják meg egy paraméter értékét. A Java azért mondott le a mutatók használatáról, hogy megakadályozza a mutatókkal kapcsolatos hibák fellépését, és elejét vegye annak, hogy rosszindulatú programozók vírust állíthassanak elő vagy információhoz jussanak egy távoli rendszerről.

ÉRTÉK VISSZAADÁSA A FÜGGVÉNY HÍVÓJÁNAK

Szóltunk már arról, hogy egy appleten belül mindegyik függvénynek megvan a maga speciális feladata, amelyet végre kell hajtania. A függvények többsége a feladat végrehajtása után valamilyen eredményt ad vissza a hívójának. Ha egy függvény visszaad valamilyen értéket, akkor a Java tudomására kell hoznunk a visszaadásra kerülő érték típusát. Ezt úgy tehetjük meg, hogy a függvény neve elé beírjuk a visszatérési érték típusát. A következő, *ÁtlagÉrték* nevű függvény például két érték átlagát adja vissza:

```
public float AtlagErtek(int a, int b)
  {
    float eredmeny;
    eredmeny = (a + b) / 2;
    return(eredmeny);
  }
```

Ebben a példában a függvény neve előtt a *float* (lebegőpontos) típusjelző adja meg a függvény által *visszaadandó érték típusát*. A C/C++ nyelvekhez hasonlóan a Java is a *return* (visszatérés) utasítást használja arra, hogy egy függvény értéket adjon vissza a hívójának. Amikor az applet a *return* utasítással találkozik, akkor az applet visszaadja a megadott értéket, befejeződik a függvény végrehajtása, és a vezérlést visszakapja az applet. A függvény hívója viszont használhatja a visszakapott értéket, például így:

```
eredmeny = AtlagErtek(1001, 2002);
```

Ebben a példában az applet a visszatérési értéket az *eredmény* nevű változóhoz rendeli. Az applet a függvény által visszaadott értéket a *drawString* függvény segítségével közvetlenül a képernyőre is küldheti az alábbiak szerint:

```
g.drawString("Az átlagérték: " + AtlagErtek(10, 20), 5, 25);
```

Az *ÁtlagÉrték* függvény előző megvalósításánál három külön utasítást használtunk a könnyebb érthetőség kedvéért. A függvényt ugyanakkor egyetlen *return* utasításra is lerövidíthetjük:

```
public float AtlagErtek(int a, int b)
  {
    return((a+b) / 2);
  }
```

NÉHÁNY ALAPVETŐ FONTOSSÁGÚ FÜGGVÉNY

Folyóiratokban, könyvekben és egyéb helyeken található Java appletek kódjait tanulmányozva láthatjuk, hogy az appletek sűrűn használnak néhány alapvető fontosságú függvényt, mint amilyen az *init*, a *start*, a *stop* és a *paint*. A következő fejezetrészben az ilyen Java függvények rendeltetésével ismerkedünk meg.

AZ INIT FÜGGVÉNY

Akik otthonosak a C/C++ nyelvben, tudják, hogy egy program végrehajtása mindig egy *main* nevű függvénnyel kezdődik. Ezzel szemben egy Perl parancsfájlban elsőként a parancsfájl első utasítása hajtódik végre. Java appleten belül viszont elsőként általában egy *init* nevű függvény végrehajtására kerül sor (feltéve, hogy van benne ilyen függvény).

Bonyolultabb Java appletekben egyre többször lesz szükségünk arra, hogy az appletek különböző kép- és hangfájlokat töltsenek be, vagy más objektumokat, mint például betűtípusokat inicializáljanak. Azért, hogy az appletek az ilyen programrészeknek egyetlen helyen belül adhassanak kezdeti értékeket, a Java ezeket a műveleteket egyetlen, speciális, *init* nevű függvényben fogja össze. Mint már volt róla szó, amikor elindul egy applet, a böngésző automatikusan meghívja az *init* függvényt az alábbiak szerint:

```
public void init()
  {
    // utasítások
  }
```

Az *init* függvényt csak arra használjuk, hogy kezdeti értékeket adjunk az appletek változóinak. Az appleteknek a feladatuk végrehajtását a *start* függvényen belül kell elkezdeniük (lásd később).

A START FÜGGVÉNY

Miután az applet futtatta az *init* függvényt, meghív egy speciális, *start* nevű függvényt, aminek az a rendeltetése, hogy elindítsa az applet végrehajtását. Normál esetben a *start* függvény létrehoz egy vagy több végrehajtási szálat, amelyek ténylegesen végrehajtják az appletet. Az alábbi utasítások egy minta *start* függvényt mutatnak be, amely létrehoz és futtat egy szálat:

```
Thread valamilyenSzal = null;

public void start()
  {
    if (valamilyenSzal == null)
```

```
    {
      valamilyenSzal = new Thread(this);        // létrehozza a thread objektumot
      valamilyenSzal.start();                   // elindítja a thread objektumot
    }
}
```

Ugyanúgy, ahogyan egy applet az indulásakor automatikusan meghívja a *start* függvényt, a befejeződésekor automatikusan meghívja a *stop* függvényt.

A STOP FÜGGVÉNY

Mint majd látni fogjuk, a Java lehetővé teszi több szál egyidejű végrehajtását. Leegyszerűsítve úgy képzeljünk el egy végrehajtási szálat, mint egy olyan programot, amelyik az appletünket futtatja. Azáltal, hogy a Java támogatja a többszálas végrehajtást, lehetővé teszi, hogy „több program" hajtsa végre ugyanannak az appletnek az utasításait. Tegyük fel például, hogy van egy bonyolult animációnk, amelyik különböző objektumokat mozgat a képernyőn. Végrehajtási szálakat használva mindegyik objektumhoz hozzárendelhetünk egy-egy szálat. Így mindegyik szál csak egyetlen objektum mozgását vezérli (ami leegyszerűsíti a programozásukat is). A böngésző ugyanakkor észleli az egyes szálak futását, és a vezérlést gyorsan képes váltogatni a különböző szálak között.

Ha az appletünk több szálat használ, akkor szükség lehet arra, hogy valamilyen módon vezéreljük azok leállítását, amikor a böngésző megállítja az appletet. Ellenkező esetben ugyanis az applet egy háttérzene leállása után is folytathatja egy objektum mozgatását a képernyőn, vagy fordítva – aminek következtében az appletnek „döcögős" lehet a leállása.

Másrészt egy Web-oldal tartalmától függően előfordulhat, hogy a böngésző állítja le az appletet, amikor a felhasználó elhúzza az appletet a látómezőből. Később, amikor az applet ismét láthatóvá válik, a böngésző újraindítja azt. Egy applet ilyen módon történő indításához és leállításához a böngésző az applet *start* és *stop* függvényét használja.

Minden olyan esetben, amikor a böngésző állít le appletet, egy speciális, *stop* nevű függvényt hív meg. A *stop* függvényen belül az applet az általunk kívánt sorrendben állíthatja le a szálakat. Ha viszont az applet nem ilyen manuális módon állítja le a szálakat, akkor a Java a saját választása szerinti sorrendben teheti meg ezt. A következő példa azt mutatja be, hogy miként használhatjuk a *stop* függvényt egy vagy több szál befejezéséhez:

```
public void stop()
  {
    if (zeneSzal != null)
      {
        zeneSzal.stop();
        zeneSzal = null;
      }
  }
```

Ebben a példában az utasítások a *null* értéket rendelik minden egyes olyan szálhoz, amelyiket a függvény leállít. Annak következtében, hogy a *Thread* objektum a *null* értéket kapja, a Java később, a szemétgyűjtés során figyelmen kívül hagyhatja az objektumot. Ha a böngésző később újraindítja a szálat, a *start* függvény megállapíthatja, hogy *null* értéke van az objektumnak, így szükség szerint létrehozhat és indíthat új szálat.

A PAINT FÜGGVÉNY

Akik jártasak a Windows alapú programozásban, bizonyára tudják, hogy minden olyan alkalommal, amikor a programnak frissítenie kell a képernyőt, a program általában meghív egy különleges függvényt, amelyik újrarajzolja azt. Ha például egy programnak meg kell változtatnia a képernyőn való megjelenését, akkor a változtatásokat az ablak frissítő függvényében végzi el. Ugyanígy, ha egy felhasználó átméretez vagy áthelyez a képernyőn egy ablakot, a programnak újra meg kell jelenítenie az ablak tartalmát. Másként fogalmazva ez azt jelenti, hogy sem a böngésző, sem a Windows nem kíséri figyelemmel egy applet ablakának tartalmát. Az ablak tartalmának a frissítése a *paint* függvény dolga. Ha például az applet ablakában valamilyen háttérkép jelenik meg, akkor a *paint* függvénynek kell újrarajzolni a képet minden olyan esetben, amikor frissíti az ablak tartalmát. Ha az ablak szöveget tartalmaz, ugyancsak a *paint* függvénynek kell visszaállítania azt. Röviden: teljes mértékben a paint függvény vezérli az ablak tartalmát.

Az alábbi *paint* függvény például a *Halló, Web!* üdvözletet jeleníti meg minden olyan esetben, amikor az appletnek frissítenie kell az ablakot:

```
public void paint(Graphics g)
  {
    g.drawString("Halló, Web!", 50, 100);
  }
```

Az appletnek a *paint* függvényen belül kell az ablak minden olyan elemét újrarajzolni, amit korábban már megjelenített az ablakban. Tegyük fel például, hogy az applet különböző alakzatokat rajzol meg, és a felhasználó minimális (ikon) méretűre zsugorítja az ablakot. Ha a felhasználó később visszaállítja az ablak eredeti méretét, akkor a *paint* függvénynek kell újrarajzolnia az ablak korábbi tartalmát (ebben az esetben az alakzatokat).

Az applet által végzett művelettől függően lehetnek olyan esetek, amikor az appletnek kell újrarajzolnia az ablakát. Ilyenkor az applet meghívhatja a *repaint* függvényt, ami viszont a maga részéről meghívja a *paint* függvényt:

```
repaint();
```

A JAVA OSZTÁLYAI

Az objektumorientált programozást illetően a Java nyelvben kiemelkedő szerepet játszik az osztály. Az objektum valamilyen „dolgot" jelent, például egy könyvet. A Java osztálya lehetővé teszi az appletek számára, hogy különböző tulajdonságokat definiáljanak egy objektum számára. Ha az objektum történetesen egy könyv, akkor az osztály olyan adattagokat tartalmazhat, mint a könyv címe, a szerző neve vagy a kötet ára. Ezen túlmenően egy objektumnak lehetnek metódusai (függvényei), amelyek meghatározott műveleteket végeznek az objektumon, mint például *cím_hozzárendelés, szerző_megjelenítése* stb.

Az osztály lehetővé teszi, hogy egy Java applet összefogja az adatokat és azokat a függvényeket, amelyek műveleteket végeznek az adatokon. Úgy hozhatunk létre Java osztályt, hogy egyedi nevet adunk az osztálynak, majd kapcsos zárójelek között megadunk egy vagy több tagot:

```
class osztaly_neve
  {
```

```
    int adat_tag;                    // Adattag
    void fuggveny_tag(int);          // Függvénytag
}
```

Miután definiáltuk az osztályt, deklarálhatjuk az osztály típusának megfelelő változókat (az ún. *objektumokat*):

```
osztály_neve elso_objektum, masodik_objektum, harmadik_objektum;
```

Az alábbi definíció egy *könyv* nevű osztályt hoz létre, amely adatváltozókat és metódusdefiníciókat tartalmaz:

```
class konyv
  {
    public String Cim;
    public String Szerzo;
    public double Ar;

    public void CimMegjelenito(Graphics g)
      {
        g.drawString("Cím: " + Cím, 5, 10);
        g.drawString("Ár: " + Ar, 5, 35);
        g.drawString("Szerzõ: " + Szerzo, 5, 60);
      }
  }
```

Ebben a példában az osztály három változótagot és egy függvénytagot tartalmaz. A Java osztály tagjai az elérhetőségüket tekintve *private, protected, public* minősítésű tagokként definiálhatók. A példában valamennyi tag *public* minősítésű, ami azt jelenti, hogy az applet a pont (.) operátor segítségével bármelyik taghoz hozzáférhet (mint például *valamilyen_könyv.Ar = 49.95*). Miután az appleten belül definiáltuk az osztályt, az ilyen osztálytípusú objektumokat (változókat) az alábbi módon deklarálhatjuk az osztály nevével és a *new* operátorral:

```
konyv Webkönyv = new konyv();
konyv Javakonyv = new konyv();
konyv CGIkonyv = new konyv();
```

A Java nyelvben valamennyi objektum dinamikus, ami azt jelenti, hogy az objektum deklarálásához a *new* operátort kell használni. Ha e nélkül deklarálnánk egy objektumot, akkor a Java fordító ugyan tudni fogja az objektum nevét, de addig nem létezik aktuális objektum, amíg a *new* operátorral nem hozunk létre egyet.

OSZTÁLYTAGOK

Az osztály lehetővé teszi, hogy információkat, ún. tagokat csoportosítsunk egyetlen változóba. Valamely taghoz a Java *pont operátora (.)* segítségével rendelhetünk értéket, és ugyanígy férhetünk

hozzá egy tag értékéhez is. Az alábbi utasítások például a *Webkönyv* nevű változó különböző tagjaihoz rendelnek értékeket:

```
Webkonyv.Cim = "A Web programozása";
Webkonyv.Ar = 49.95;
Webkonyv.Szerzo = "Jamsa, Lalani és Weakly";
```

Egy osztálytag eléréséhez meg kell adnunk az objektum nevét, amelyet egy pont követ, majd meg kell adnunk a tag nevét. Hasonlóképpen kell használnunk a pont operátort egy osztály tagfüggvényének eléréséhez is. Az alábbi utasítás például a *Webkönyv* objektum *CimMegjelenito* tagfüggvényét hívja meg:

```
Webkonyv.CimMegjelenito(g);
```

OSZTÁLY HASZNÁLATA APPLETBEN

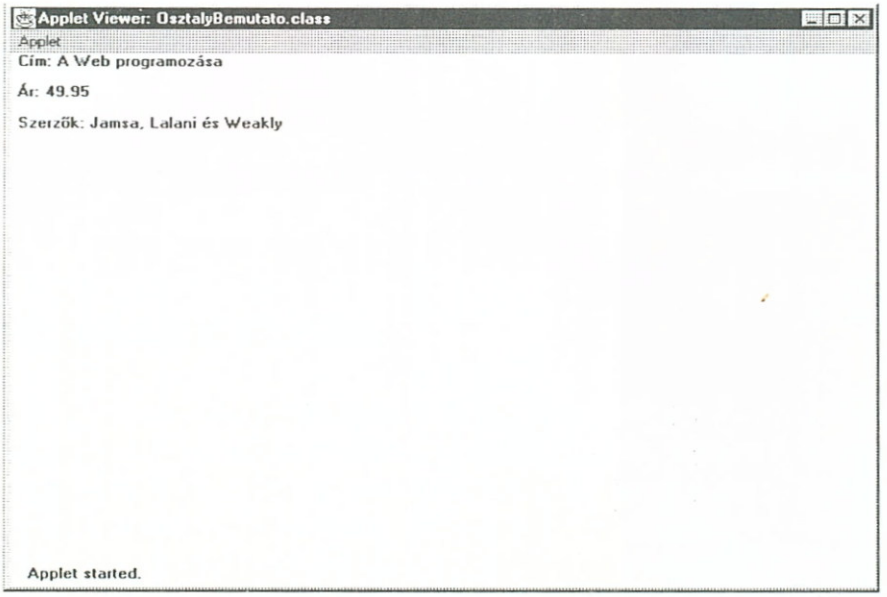
Az itt következő, *OsztályBemutató.java* applet létrehoz egy *Könyv objektumot*. Az applet a *pont operátor* segítségével értékeket rendel az adattagokhoz. Ezt követően az applet a *CimMegjelenito* tagfüggvény segítségével információkat jelenít meg a könyvről:

```
import java.awt.*;
import java.applet.*;

class konyv
  {
    public String Cim;
    public double Ar;
    public String Szerzo;
    public void CimMegjelenito(Graphics g)
      {
        g.drawString("Cím: " + Cim, 5, 10);
        g.drawString("Ár: " + Ar, 5, 35);
        g.drawString("Szerzõk: " + Szerzo, 5, 60);
      }
  }

public class OsztalyBemutato extends Applet
  {
    public void paint(Graphics g)
      {
        konyv Webkonyv = new konyv();

        Webkonyv.Cim = "A Web programozása";
        Webkonyv.Ar = 49.95;
        Webkonyv.Szerzo = "Jamsa, Lalani és Weakly";
        Webkonyv.CimMegjelenito(g);
      }
  }
```

Az alábbi HTML bejegyzések beírásával hozzuk létre az *OsztályBemutató.html* fájlt, hogy futtathassuk az appletet:

```
<HTML><TITLE>OsztalyBemutato Applet</TITLE>
<APPLET CODE="OsztalyBemutato.class" WIDTH=400 HEIGHT=300></APPLET></HTML>
```

Miután lefordítottuk és futtattuk az appletet, az applet ablakában a 14.10. ábrán láthatóhoz hasonlónak kell megjelennie:



14.10. ábra. A könyv osztály létrehozása és használata

EGYSZERŰ KONSTRUKTOR FÜGGVÉNY KÉSZÍTÉSE

A *konstruktor függvény* osztálymetódus, amelynek ugyanaz a neve, mint magának az osztálynak. Ha az aktuális osztálynak *könyv* a neve, akkor a konstruktor függvénynek is *könyv* lesz a neve. Ugyanígy, ha van egy *mozi* nevű osztályunk, akkor az abban lévő konstruktor függvénynek is *mozi* lesz a neve. Ha az osztályunk definiál konstruktor függvényt, akkor a Java automatikusan meghívja azt minden olyan esetben, amikor létrehozunk ilyen osztálytípusú objektumot.

Az alábbi utasítások egy *könyv* nevű osztályt hoznak létre. Az osztálydefiníció tartalmaz egy *könyv* nevű konstruktor függvényt, amely kezdeti értékeket rendel az objektumhoz. A konstruktor függvény nem tud értéket visszaadni; ennek ellenére a függvény deklarálásához nem kell használnunk a *void* kulcsszót. Egyszerűen csak ne adjuk meg a visszatérési típust. Az appleten belül ugyanúgy kell definiálnunk a konstruktor függvényt, mint bármely más osztálymetódust:

```
class konyv
  {
    public String Cim;
    public double Ar;
    public String Szerzo;
    public konyv(String Cim, double Ar, String Szerzo)
      {
```

```
        this.Cím = Cím;
        this.Ar =Ar;
        this.Szerzo = Szerzo;
      }

    public void CimMegjelenito(Graphics g)
      {
        g.drawString("Cím: " + Cim, 5, 10);
        g.drawString("Ár: " + Ar, 5, 35);
        g.drawString("Szerzõ: " + Szerzo, 5, 60);
      }
  }
```

Az alábbi utasítások a *KönyvKonstruktor.java* appletet hozzák létre:

```
import java.awt.*;
import java.applet.*;

class konyv
  {
    public String Cim;
    public double Ar;
    public String Szerzo;
    public konyv(String Cim, double Ar, String Szerzo)
      {
        this.Cim = Cim;
        this.Ar =Ar;
        this.Szerzo = Szerzo;
      }

    public void CimMegjelenito(Graphics g)
      {
        g.drawString("Cím: " + Cim, 5, 10);
        g.drawString("Ár: " + Ar, 5, 35);
        g.drawString("Szerzõk: " + Szerzo, 5, 60);
      }
  }

public class KonyvKonstruktor extends Applet
  {
    public void paint(Graphics g)
      {
        konyv Webkonyv = new konyv("A Web programozása", 49.95,
        "Jamsa, Lalani, Weakly");
        Webkonyv.CimMegjelenito(g);
      }
  }
```

Az alábbi HTML bejegyzések beírásával hozzuk létre a *KönyvKonstruktor.html* fájlt, hogy futtathassuk az appletet:

```
<HTML><TITLE>KonyvKonstruktor Applet</TITLE>
<APPLET CODE="KonyvKonstruktor.class" WIDTH=400 HEIGHT=300></APPLET></HTML>
```

Figyeljük meg, hogy az appletben az egyes objektumok deklarálása után zárójelek között vannak felsorolva az objektumok kezdeti értékei, akárcsak egy függvényhívásnál. Amikor konstruktor függvényeket használunk, a paramétereket az objektum deklarálásakor adjuk át a függvénynek:

```
konyv Webkonyv = new konyv("Web programozás", 49.95, "Jamsa, Lalani, Weakly");
```

Megjegyzés: A C++ programozási nyelvtől eltérően a Java nyelv nem használ destruktor metódusokat.

BETŰTÍPUSOK HASZNÁLATA

Java appleteken belül gyakran használhatjuk a *Graphics* osztály *Font* (betűtípus) objektumát. A *Font* objektumot használva megválaszthatjuk a betűk típusát, méretét és egyéb jellemzőit (mint például vastagított vagy dőlt betű). Fontos megjegyeznünk, hogy a Java betűtípusokkal kapcsolatos műveletei csak az applet ablakában, nem pedig a böngésző ablakában lévő betűtípusokra vonatkoznak.

Egy appleten belül a betűtípus beállításához először a *new* operátorral létre kell hozni egy *Font* objektumot, majd a *Graphics* osztály *setFont* metódusával vehetjük használatba a betűtípust. Ahhoz, hogy a kimenetet a választott betűtípussal jeleníthessük meg, az appletnek meg kell hívnia a *Graphics* osztály *drawString* függvényét. Az itt következő, *FontObjektum.java* nevű applet azt mutatja be, hogy miként használja a Java a *Font* objektumot:

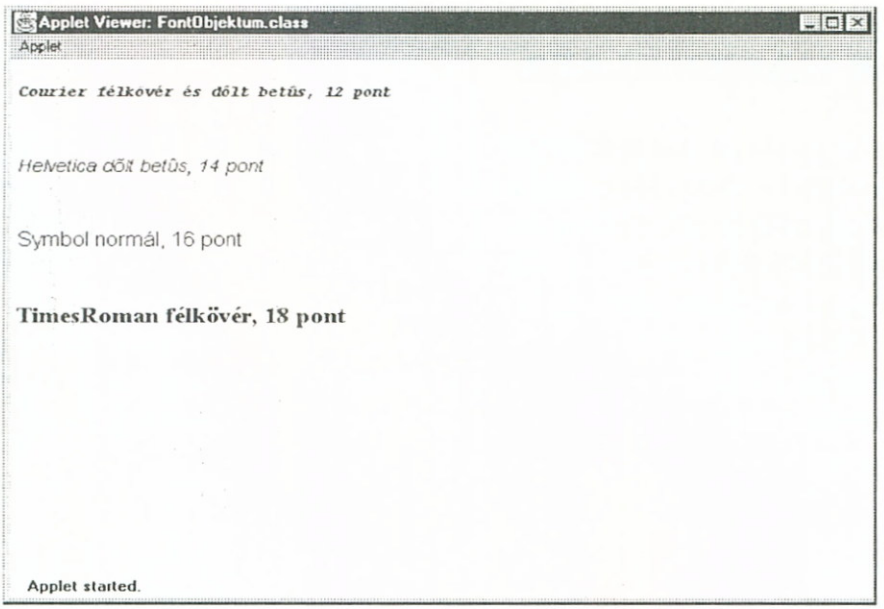
```
import java.awt.*;
import java.applet.*;

public class FontObjektum extends Applet
  {
    public void paint(Graphics g)
      {
        Font Courier = new Font("Courier", Font.BOLD + Font.ITALIC, 12);
        Font Helvetica = new Font("Helvetica", Font.ITALIC, 14);
        Font Symbol = new Font("Symbol", Font.PLAIN, 16);
        Font TimesRoman = new Font("TimesRoman", Font.BOLD, 18);

        g.setFont(Courier);
        g.drawString("Courier félkövér és dõlt betûs, 12 pont", 5, 30);

        g.setFont(Helvetica);
        g.drawString("Helvetica dõlt betûs, 14 pont", 5, 90);

        g.setFont(Symbol);
```

```
        g.drawString("Symbol normál, 16 pont", 5, 150);

        g.setFont(TimesRoman);
        g.drawString("TimesRoman félkövér, 18 pont", 5, 210);
    }
}
```

Az alábbi HTML bejegyzések beírásával hozzuk létre a *FontObjektum.html* fájlt, hogy futtathassuk az appletet:

```
<HTML><TITLE>FontObjektum Applet</TITLE>
<APPLET CODE="FontObjektum.class" WIDTH=400 HEIGHT=300></APPLET></HTML>
```

Amikor futtatjuk az appletet, a képernyőn a 14.11. ábrán láthatóhoz hasonlónak kell megjelennie.



14.11. ábra. A Graphics osztály Font objektumának használata

A *Graphics* osztály az appletek számára a betűtípusok beállítása mellett a színek megválasztását is lehetővé teszi. A *drawString* függvény által megjelenítendő szöveg színének megadásához a *Graphics* osztály *setColor* metódusát kell használni. Az itt következő *BetűSzínek.java* nevű applet a *setColor* metódust használja arra, hogy a Java előre definiált színeivel jelenítsen meg néhány sornyi szöveget:

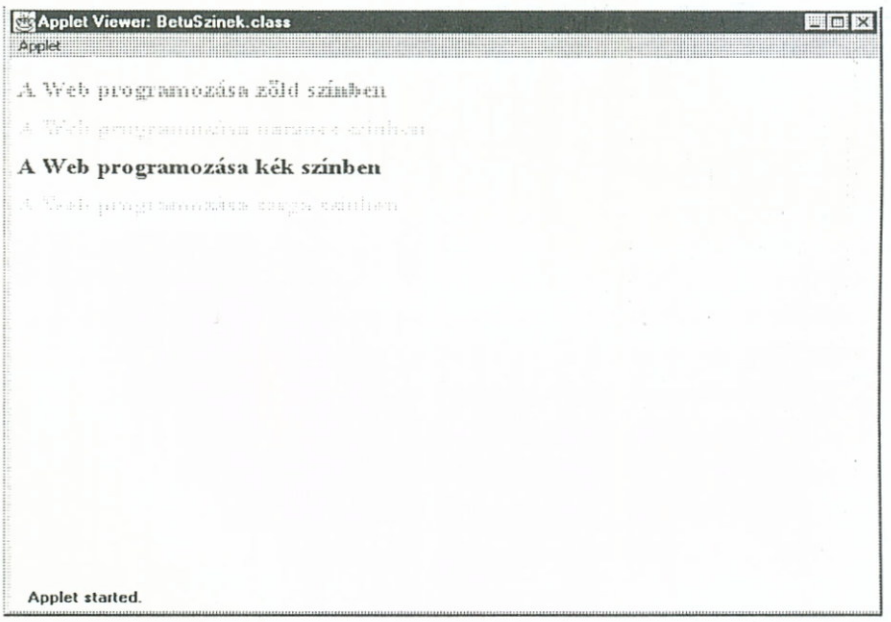
```
import java.awt.*;
import java.applet.*;

public class BetuSzinek extends Applet
  {
    public void paint(Graphics g)
      {
        Font font = new Font("TimesRoman", Font.BOLD, 18);
```

```
        g.setFont(font);

        g.setColor(Color.green);
        g.drawString("A Web programozása zöld színben", 5, 30);

        g.setColor(Color.orange);
        g.drawString("A Web programozása narancs színben", 5, 60);

        g.setColor(Color.blue);
        g.drawString("A Web programozása kék színben", 5, 90);

        g.setColor(Color.yellow);
        g.drawString("A Web programozása sárga színben", 5, 120);
    }
}
```

Az alábbi HTML bejegyzések beírásával létre jön a *BetűSzínek.html* fájl, hogy futtathassuk az appletet:

```
<HTML><TITLE>BetuSzinek Applet</TITLE>
<APPLET CODE="BetuSzinek.class" WIDTH=400 HEIGHT=300></APPLET></HTML>
```

Amikor futtatjuk az appletet, a képernyőn a 14.12. ábrán láthatóhoz hasonlónak kell megjelennie:



14.12. ábra. Színes szöveg megjelenítése a setColor metódus segítségével

HTML PARAMÉTEREK ÁTADÁSA JAVA APPLETNEK

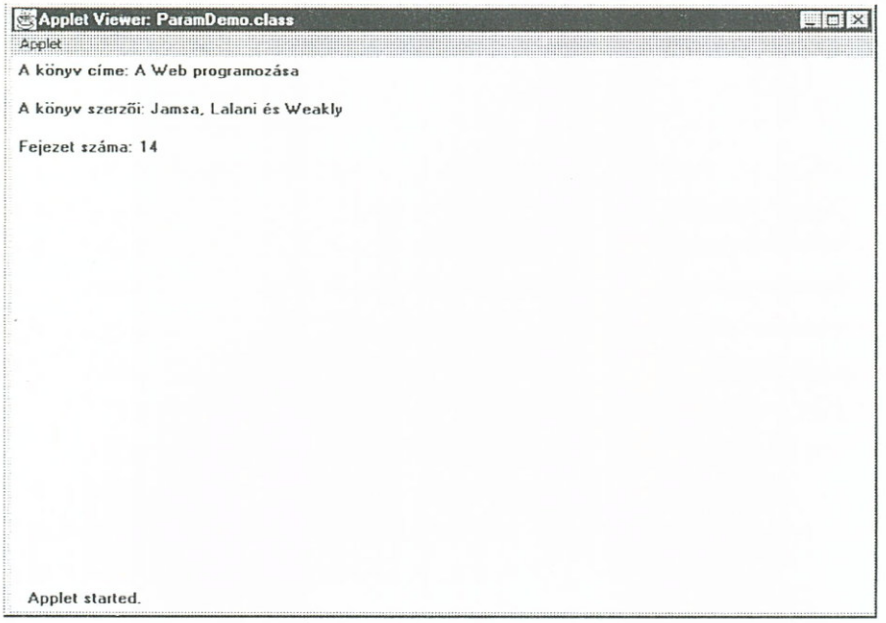
A HTML fájlok legegyszerűbben az *<Applet>* elem *PARAM* attribútumával adhatnak át paramétereket Java appleteknek. Java appletből a *PARAM* attribútumértékeket a *getParameter* függvény segítségével érhetjük el úgy, hogy a kívánt bejegyzést karakterlánc-paraméterként adjuk meg. Tegyük fel például, hogy a *ParamDemo.html* nevű fájl a következő bejegyzéseket tartalmazza:

```
<HTML><TITLE>Paraméter Demo Applet</TITLE>
<APPLET CODE="ParamDemo.class" WIDTH=300 HEIGHT=200>
<PARAM NAME=KonyvCim Value="A Web programozása">
<PARAM NAME=KonyvSzerzok Value="Jamsa, Lalani és Weakly">
<PARAM NAME=Fejezet Value="14">
</APPLET></HTML>
```

Amint látható, mindegyik PARAM attribútum egy *nevet* (Name) és egy *értéket* (Value) ad meg. Az itt következő Java applet a *getParameter* függvény segítségével megjeleníti mindegyik paraméter értékét. Ha az <APPLET> elem nem ad meg paramétert, akkor a *getParameter* függvény a null értéket adja vissza. A *ParamDemo.java* applet a következő kódot tartalmazza:

```
import java.awt.*;
import java.applet.*;

public class ParamDemo extends Applet
  {
    String KonyvCim;
    String Szerzok;
    String Fejezet;

    public void init()
      {
        KonyvCim = getParameter("KonyvCim");
        if (KonyvCim == null)
        KonyvCim = "Ismeretlen";

        Szerzok = getParameter("KonyvSzerzok");
        if (Szerzok == null)
        Szerzok = "Ismeretlen";

        Fejezet = getParameter("Fejezet");
        if (Fejezet == null)
        Fejezet = "Ismeretlen";
      }

    public void paint(Graphics g)
      {
        g.drawString("A könyv címe: " + KonyvCim, 5, 15);
        g.drawString("A könyv szerzői: " + Szerzok, 5, 45);
        g.drawString("Fejezet száma: " + Fejezet, 5, 75);
      }
  }
```

Ha futtatjuk az appletet, akkor a képernyőn a 14.13. ábrán láthatóhoz hasonlónak kell megjelennie:



14.13. ábra. Applet HTML paramétereinek megjelenítése

JAVA ESEMÉNYEK

A Java appleteket sok más Windows alapú programhoz hasonlóan események vezérlik. Ilyen események például az olyan műveletek, amelyeket a felhasználók billentyűzetről vagy az egérrel hajtanak végre. Ha ilyen események következnek be, akkor a böngésző speciális függvényeket, ún. *eseménykezelőket* hív meg. Az appleteinken belül ezeknek a különleges függvényeknek a megvalósításával érhetjük el, hogy az appletünk kezelje az ilyen eseményeket. A fejezet következő részében két, gyakran előforduló eseményt vizsgálunk meg közelebbről: a billentyűeseményeket és az egéreseményeket.

BILLENTYŰESEMÉNYEK

Minden olyan esetben, amikor egy applet futása közben a felhasználó lenyom egy billentyűt, a Java meghívja a *keyDown* függvényt. Hasonlóképpen, amikor a felhasználó felenged egy billentyűt, a Java meghívja a *keyUp* függvényt. A Java mindkét esetben átad a függvénynek egy *Event* nevű paramétert (event = esemény), amely többek között a billentyűzet állapotát határozza meg (például, hogy le volt-e nyomva az ALT vagy a SHIFT), és egy második paramétert, amely a lenyomott billentyűt tartalmazza. Ha az applet nem definiál olyan függvényeket, amelyek ezt a két eseményt kezelnék, akkor az ezzel kapcsolatos feladatokat maga a Java végzi el.

Ha az applet tartalmaz billentyűeseményeket kezelő függvényt, akkor a függvénynek a *true* (igaz) értéket kell visszaadnia a Javának, jelezve, hogy megtörtént az esemény feldolgozása. Ezért az appletekben a *keyDown* és a *keyUp* függvényeket *boolean* típusú függvényként kell definiálnunk:

```
public boolean keyUp(Event esemeny, int LenyomottBillentyû)
  {
    // A billentyûeseményt feldolgozó utasítások
  }
```

```
public boolean keyDown(Event esemeny, int LenyomottBillentyû)
  {
    // A billentyûeseményt feldolgozó utasítások
  }
```

Az itt következő, *BillEsemeny.java* nevű applet a *keyUp* és a *keyDown* függvények használatát szemlélteti. Minden esetben, amikor a felhasználó lenyom és felenged egy billentyűt, a függvények megjelenítik az eseményt leíró üzenetet:

```
import java.awt.*;
import java.applet.*;

public class BillEsemeny extends Applet
  {
    String BillentyuLeEsemeny = null;
    String BillentyuFelEsemeny = null;

    public boolean keyUp(Event event, int betu)
      {
        BillentyuFelEsemeny = "BillentyûFel: " + (char) betu;
        repaint();
        return(true);
      }

    public boolean keyDown(Event event, int betu)
      {
        BillentyuLeEsemeny = "BillentyûLe: " + (char) betu;
        repaint();
        return(true);
      }

    public void paint(Graphics g)
      {
        if (BillentyuFelEsemeny != null)
          g.drawString(BillentyuFelEsemeny, 5, 45);

        if (BillentyuLeEsemeny != null)
          g.drawString(BillentyuLeEsemeny, 5, 75);
      }
  }
```

Az alábbi HTML bejegyzések beírásával hozzuk létre a *BillEsemeny.html* fájlt, hogy futtathassuk az appletet:

```
<HTML><TITLE>BillEsemény Applet</TITLE>
<APPLET CODE="BillEsemeny.class" WIDTH=200 HEIGHT=150></APPLET></HTML>
```

EGÉRESEMÉNYEK

Hasonlóképpen ahhoz, ahogyan egy böngésző eseményt generál, amikor a felhasználó lenyom egy billentyűt, a böngésző akkor is generál eseményt, ha a felhasználó valamilyen egérműveletet végez. Ha például a felhasználó elhúzza az egeret, akkor a Java meghívja a *mouseDrag* függvényt. Amikor a felhasználó lenyomja az egér gombját, a Java meghívja az applet *mouseDown* függvényét. Ugyanígy, ha a felhasználó felengedi az egér gombját, a Java a *mouseUp* függvényt hívja meg.

A Java minden ilyen „egéreseménnyel" kapcsolatos függvény meghívásakor átadja a függvénynek az egérmutató – az esemény bekövetkezésekor érvényes – *x* és *y* koordinátáját. Az egérmutató *x* és *y* koordinátáját a képernyőn lévő objektumok koordinátáival összehasonlítva az applet meg tudja állapítani, hogy a felhasználó a képernyőn lévő valamelyik objektumra kattintott-e.

A Java az egérműveleteket olyan eseményeknek tekinti, amelyekre az appletnek vagy magának a Javának válaszolnia kell. Ha az appletünk válaszol egy egéreseményre, akkor az eseményt feldolgozó függvénynek a *true* (igaz) értéket kell visszaadnia a *return* utasítás segítségével.

A következő, *EgerMuveletek.java* nevű applet a *mouseUp*, *mouseDown* és a *mouseDrag* függvényeket definiálja. Mindegyik függvény megjeleníti az egér mutatójának azokat az *x* és *y* koordinátáit, ahol az egéresemény bekövetkezett:

```
import java.awt.*;
import java.applet.*;

public class EgerMuveletek extends Applet
  {
    String EgerLeEsemeny = null;
    String EgerFelEsemeny = null;
    String EgerHuzasEsemeny = null;

    public boolean mouseUp(Event esemeny, int x, int y)
      {
        EgerFelEsemeny = "Gomb felengedése: " + x + "," + y;
        repaint();
        return(true);
      }

    public boolean mouseDown(Event esemeny, int x, int y)
      {
        EgerLeEsemeny = "Gomb lenyomása: " + x + "," + y;
        repaint();
        return(true);
      }

    public boolean mouseDrag(Event esemeny, int x, int y)
      {
        EgerHuzasEsemeny = "Egér elhúzása: " + x + "," + y;
        repaint();
        return(true);
      }

    public void paint(Graphics g)
```

```
    {
      if (EgerHuzasEsemeny != null)
        g.drawString(EgerHuzasEsemeny, 5, 15);

      if (EgerFelEsemeny != null)
        g.drawString(EgerFelEsemeny, 5, 45);

      if (EgerLeEsemeny != null)
        g.drawString(EgerLeEsemeny, 5, 75);
    }
}
```

Amint látható, az applet boolean típusúként definiálja a *mouseUp, mouseDown* és a *mouseDrag* függvényeket. Az applet mindegyik függvényen belül létrehoz egy *String* típusú karakterlánc-változót, ami az egéresemény *x* és *y* koordinátáit tartalmazza. Ezt követően az applet a *repaint* függvény segítségével újrarajzolja az ablakának tartalmát.

Az alábbi HTML bejegyzések beírásával hozzuk létre az *EgérMűveletek.html* fájlt, hogy futtathassuk az appletet:

```
<HTML><TITLE>EgérMûveletek Applet</TITLE>
<APPLET CODE="EgerMuveletek.class" WIDTH=300 HEIGHT=150></APPLET></HTML>
```

PÁRBESZÉDABLAKOK HASZNÁLATA JAVA APPLETEKBEN

Említettük már, hogy a programozók interaktív appleteket tudnak készíteni. A felhasználók közreműködésének megkönnyítése céljából a Java széleskörűen támogatja a különböző gombokat, jelölőnégyzeteket, menüket és beviteli mezőket tartalmazó párbeszédablakok (panelek) használatát. Leegyszerűsítve egy Java applet megjelenít egy sor párbeszédablak-objektumot, például egy gombsort, a gombok mindegyikéhez hozzárendel egy speciális eseménykezelő függvényt, majd a felhasználó választásától függően különböző műveleteket végez. Így például az itt következő, *Gombok.java* nevű applet egy sor választógombot jelenít meg az applet ablakában. Amikor a felhasználó kiválaszt ezek közül egyet, az applet reagál az eseményre, és egy szöveges üzenet megjelenítésével igazolja vissza a választott gombot.

```
import java.awt.*;
import java.applet.*;

public class Gombok extends Applet
  {
    String GombEsemeny = null;

    CheckboxGroup ValtoGomb = new CheckboxGroup();

    public void init()
      {
        add(new Checkbox("Hétfõ", ValtoGomb, true));
```

```
        add(new Checkbox("Kedd", ValtoGomb, false));
        add(new Checkbox("Szerda", ValtoGomb, false));
        add(new Checkbox("Csütörtök", ValtoGomb, false));
        add(new Checkbox("Péntek", ValtoGomb, false));
        add(new Checkbox("Szombat", ValtoGomb, false));
        add(new Checkbox("Vasárnap", ValtoGomb, false));
    }

    public void GombKezelo(Checkbox kor)
    {
        GombEsemeny = kor.getLabel() + " " + kor.getState();
        repaint();
    }

    public boolean action(Event esemeny, Object objektum)
    {
        if (esemeny.target instanceof Checkbox)
        GombKezelo((Checkbox)esemeny.target);

        return(true);
    }

    public void paint(Graphics g)
    {
        if (GombEsemeny != null)
        g.drawString("Gombesemény: " + GombEsemeny, 5, 100);
    }
}
```

Az alábbi HTML bejegyzések beírásával hozzuk létre a *Gombok.html* fájlt, hogy futtathassuk az appletet:

```
<HTML><TITLE>Gombok Applet</TITLE>
<APPLET CODE="Gombok.class" WIDTH=300 HEIGHT=150></APPLET></HTML>
```

Ha futtatjuk az appletet, akkor a képernyőn a 14.14. ábrán látható, választógombokat tartalmazó párbeszédablak jelenik meg:

14.14. ábra. Választógombok feldolgozása Java appletben

GRAFIKUS KÉPEK BETÖLTÉSE ÉS MEGJELENÍTÉSE

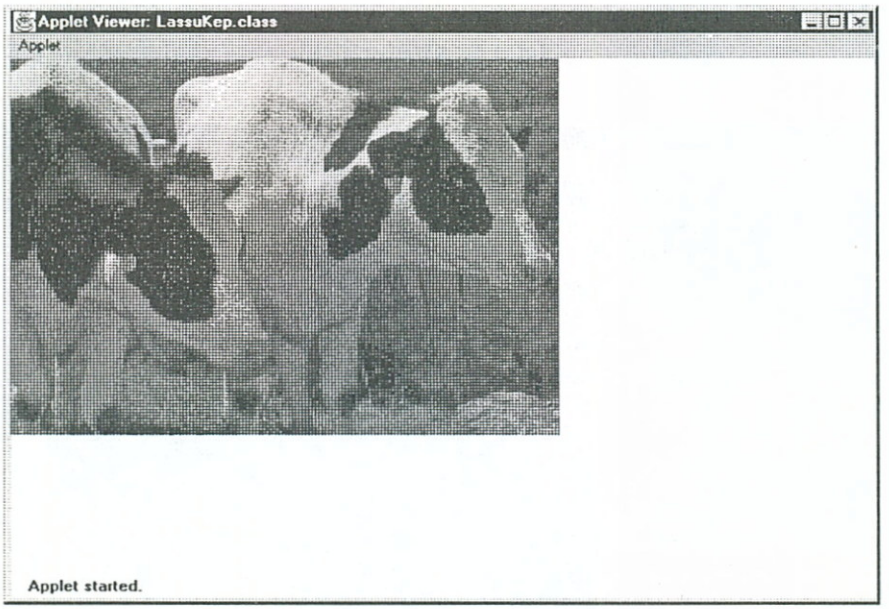
Bárhol járunk a Weben, tapasztalhatjuk, hogy a Web-helyek tervezői sűrűn alkalmaznak különböző képeket a háttérben, hogy érdekesebbé tegyék az oldalaikat. Számos Java applet is megjelenít valamilyen grafikát az ablakában. Egyes esetekben ezek a grafikák – mintegy hátteret képezve – teljesen kitöltik az applet ablakát. Más esetekben az appletek kisebb (vagy lehetőség szerint méretezhető) képeket jelenítenek meg. Az itt következő, *LassuKep.java* nevű applet a *getImage* és a *drawImage* függvény segítségével egy nagy méretű képet tölt be az applet ablakába. (A képet tároló *Cows.gif* fájlt az első kötetben levő CD-ROM tartalmazza.)

```
import java.awt.*;
import java.applet.*;

public class LassuKep extends Applet
  {
    Image kep;

    public void init()
      {
        kep = getImage(getCodeBase(), "Cows.gif");
      }

    public void paint(Graphics g)
      {
        g.drawImage(kep, 0, 0, this);
      }
  }
```

A következő HTML bejegyzések beírásával hozzuk létre a *LassuKep.html* fájlt, hogy futtathassuk az appletet:

```
<HTML><TITLE>LassuKep Applet</TITLE>
<APPLET CODE="LassuKep.class" WIDTH=600 HEIGHT=400></APPLET></HTML>
```

Ha futtatjuk az appletet, akkor a képernyőn a 14.15. ábrán láthatóhoz hasonlónak kell megjelennie:



14.15. ábra. Kép megjelenítése egy applet ablakában

Az appletnek azért *LassuKep.java* a neve, mert azzal az eljárással, amelyet az applet a kép betöltéséhez használ, a böngésző csak lassan és „darabosan" képes az ábra megjelenítésére. A Java programozók a *kétszeres pufferelés* nevű eljárást használják arra, hogy kiegyenlítsék a megjelenítés darabosságát.

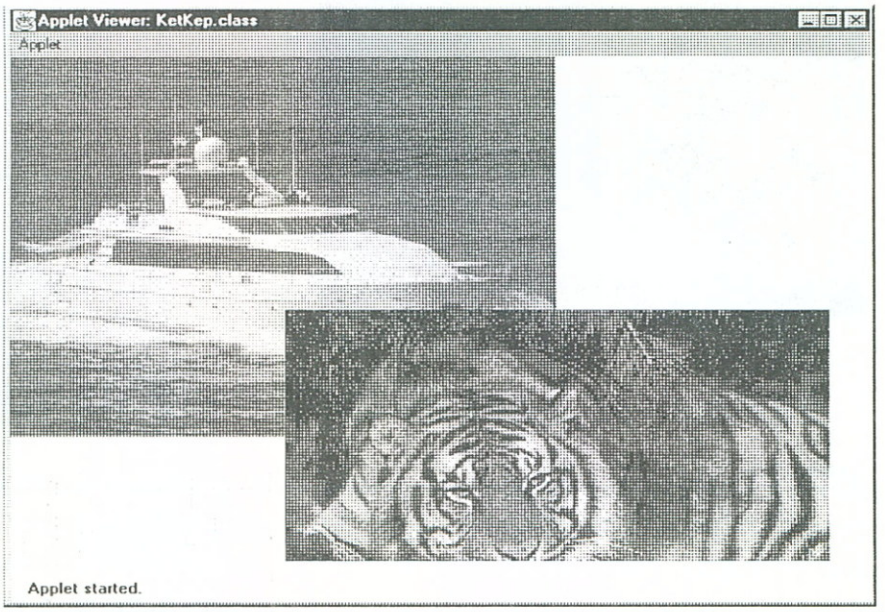
A következő, *KetKep.java* nevű applet két képet tölt be és jelenít meg *(Boat.gif* és *Tiger.gif):*

```
import java.awt.*;
import java.applet.*;

public class KetKep extends Applet
  {
    Image ElsoKep;
    Image MasodikKep;

    public void init()
      {
        ElsoKep = getImage(getCodeBase(), "Boat.gif");
        MasodikKep = getImage(getCodeBase(), "Tiger.gif");
      }

    public void paint(Graphics g)
      {
        g.drawImage(ElsoKep, 0, 0, this);
        g.drawImage(MasodikKep, 200, 200, this);
      }
  }
```

Az alábbi HTML bejegyzések beírásával hozzuk létre a *KetKep.html* fájlt, hogy futtathassuk az appletet:

```
<HTML><TITLE>KetKep Applet</TITLE>
<APPLET CODE="KetKep.class" WIDTH=600 HEIGHT=400></APPLET></HTML>
```

Az applet futtatásakor a képernyőn az 14.16. ábrán láthatóhoz hasonlónak kell megjelennie:



14.16. ábra. Két kép megjelenítése egy applet ablakában

Amint látható, az applet a *drawImage* függvényt használja annak az *x* és *y* koordinátának a megadására, amelytől kezdődően az applet a képet megjeleníti.

GRAFIKUS KÉPEK KÉTSZERES PUFFERELÉSE

Ha futtattuk az előző két appletet, akkor tapasztalhattuk, hogy mennyire lassan jelennek meg a képek az applet ablakában. A megjelenítés javítása céljából sok applet használja a *kétszeres pufferelés* nevű eljárást. Induláskor a Java applet betölti a képet a memóriába, és csak akkor jeleníti meg a képet az ablakában, miután befejeződött a betöltés. A következő *GyorsKep.java* nevű applet ezt a kétszeres pufferelés eljárást használja a *Cows.gif* kép betöltéséhez:

```
import java.awt.*;
import java.applet.*;

public class GyorsKep extends Applet
  {
    Image kep;
    boolean KepBetoltve = false;

    public void init()
      {
        kep = getImage(getCodeBase(), "Cows.gif");
        Image ErnyonKivuliKep = createImage(size().width, size().height);
```

```
        Graphics ErnyonKivuliGr = ErnyonKivuliKep.getGraphics();
        ErnyonKivuliGr.drawImage(kep, 0, 0, this);
      }

    public void paint(Graphics g)
      {
        if (KepBetoltve)
          {
            g.drawImage(kep, 0, 0, null);
            showStatus("Kész");
          }
        else
        showStatus("Kép betöltése folyamatban");
      }

    public boolean imageUpdate(Image img, int infoflags, int x, int y, int w, int h)
      {
        if (infoflags == ALLBITS)
          {
            KepBetoltve = true;
            repaint();
            return false;
          }
        else
        return true;
      }
  }
```

Az applet az *init* függvényen belül deklarál egy *ErnyőnKívüliKép* nevű *Image* objektumot, amelyhez hozzárendeli a *createImage* függvény eredményét. Ezt követően a függvény létrehozza az *ErnyőnKívüliGr* nevű *Graphics* objektumot, amelyhez hozzárendeli a képernyőn kívül létrejövő kép *grafikus környezetét*. Egy Java appletben a grafikus környezet olyan tulajdonságokat ad meg, mint az aktuális betűtípus és a képernyő színe.

Utolsó lépésként az applet a *drawImage* függvény segítségével megrajzolja a képet. Mivel azonban az applet a képet a képernyőn kívül rajzolja meg, a kép nem jelenik meg azonnal az applet ablakában:

```
public void init()
  {
    kep = getImage(getCodeBase(), "Cows.gif");
    Image ErnyonKivuliKep = createImage(size().width, size().height);
    Graphics ErnyonKivuliGr = ErnyonKivuliKep.getGraphics();
    ErnyonKivuliGr.drawImage(kep, 0, 0, this);
  }
```

A *drawImage* függvényhívás negyedik paramétereként a *this* kulcsszót használja. A függvény ezzel a paraméterrel utasítja a Javát az *ImageObserver* interfész használatára, ami azt jelenti, hogy az applet ezen az interfészen keresztül valósítja meg (implementálja) az *imageUpdate* (képfrissítés) függvényt a következők szerint:

```
public boolean imageUpdate(Image img, int infoflags, int x, int y, int w, int h)
  {
    if (infoflags == ALLBITS)
      {
        KepBetoltve = true;
        repaint();
        return false;
      }
    else
    return true;
  }
```

Vagyis az *imageUpdate* függvény teszi lehetővé az applet számára, hogy az applet megállapíthassa, betöltődött-e teljesen a memóriába a kép. Minden esetben, amikor egy applet meghívja a *drawImage* függvényt, az létrehoz egy szálat, amelyik meghívja az *imageUpdate* függvényt. A szál mindaddig folytatja az *imageUpdate* függvény hívását, amíg a függvény a *false* érték visszaadásával nem jelzi, hogy megtörtént a kép betöltése.

Az *imageUpdate* függvény második paramétere, az *infoflags* teszi lehetővé az applet számára, hogy figyelemmel kísérhesse, a kép mekkora része töltődött be már a memóriába. Amikorra a paraméter értéke azonos lesz az ALLBITS értékkel, a kép teljes egészében betöltődött a memóriába. Miután a kép bekerült a memóriába, a függvény a *KépBetöltve* változóba a *true* értéket írja és meghívja a *repaint* függvényt, amely frissíti az applet ablakát. Végül a függvény a *false* értéket adja vissza, amivel megállítja a *drawImage* végrehajtási szálat abban, hogy tovább hívja az *imageUpdate* függvényt.

Ha megvizsgáljuk az applet paint függvényét, akkor láthatjuk, hogy ennek végrehajtását a *KépBetöltve* változó vezérli. Ha a változó értéke *true*, akkor a *paint* függvény a *drawImage* függvény segítségével megjeleníti a képet az alábbi módon:

```
public void paint(Graphics g)
  {
    if (KepBetoltve)
      {
        g.drawImage(kep, 0, 0, null);
        showStatus("Kész");
      }
    else
    showStatus("Kép betöltése folyamatban");
  }
```

Amint látható, a *paint* függvény úgy hívja a *drawImage* függvényt, hogy annak null a negyedik paramétere, és ezzel megakadályozza, hogy a *drawImage* függvény hívja az *imageUpdate* függvényt. Mivel a kép már a memóriában van, a kép azonnal megjelenik az applet ablakában.

A következő, *KetGyors.java* applet ugyancsak *kétszeres pufferelléssel* tölt be két képet a memóriába, mielőtt megjelenítené azokat:

```
import java.awt.*;
import java.applet.*;

public class KetGyors extends Applet
  {
    Image ElsoKep;
    Image MasodikKep;

    boolean KepBetoltve = false;
    boolean ElsoBetoltve = false;
    boolean MasodikBetoltve = false;

    public void init()
      {
        ElsoKep = getImage(getCodeBase(), "Boat.gif");
        MasodikKep = getImage(getCodeBase(), "Tiger.gif");

        Image ErnyonKivuliKep = createImage(size().width, size().height);

        Graphics ErnyonKivuliGr = ErnyonKivuliKep.getGraphics();
        ErnyonKivuliGr.drawImage(ElsoKep, 0, 0, this);
        ErnyonKivuliGr.drawImage(MasodikKep, 200, 200, this);
      }

    public void paint(Graphics g)
      {
        if (KepBetoltve)
          {
            showStatus(" ");
            g.drawImage(ElsoKep, 0, 0, null);
            g.drawImage(MasodikKep, 200, 200, null);
          }
        else
        showStatus("Képek betöltése folyamatban");
      }

    public boolean imageUpdate(Image img, int infoflags, int x, int y, int w, int h)
      {
        if (infoflags == ALLBITS)
          {
            if (img == ElsoKep)
             ElsoBetoltve = true;
            else
             MasodikBetoltve = true;

            KepBetoltve = (ElsoBetoltve && MasodikBetoltve);

            if (KepBetoltve)
             repaint();
```

```
            return false;
          }
        else
        return true;
      }
  }
```

Amint látható, az applet két boolean típusú változó *(ElsőBetöltve, MásodikBetöltve)* segítségével figyeli, hogy betöltődött-e már mindkét kép a memóriába. Miután mindkét kép betöltődött, az applet megjeleníti azokat az ablakában.

HANG BETÖLTÉSE ÉS LEJÁTSZÁSA

A Java segítségével egy applet könnyen lejátszhat hangfájlt. A Java jelenleg csak az AU formátumú audiófájlok lejátszását támogatja, a WAV formátumú fájlokét nem. Ezért azoknak, akik WAV fájlokat szeretnének lejátszani, be kell szerezniük olyan szoftvert, amelyik a WAV formátumot AU formátumra alakítja át. A Java appletek a Java *AudoClip* osztályát használják hangfájlok betöltésére és lejátszására. A következő, *JavaHang.java* nevű applet szemlélteti az audióosztály használatát:

```
import java.awt.*;
import java.applet.*;

public class JavaHang extends Applet
  {
    public void paint(Graphics g)
      {
        AudioClip audioKlip = getAudioClip(getCodeBase(), "SoundDemo.au");

        g.drawString("Java hangbemutató!", 5, 15);
        audioKlip.loop();
      }
  }
```

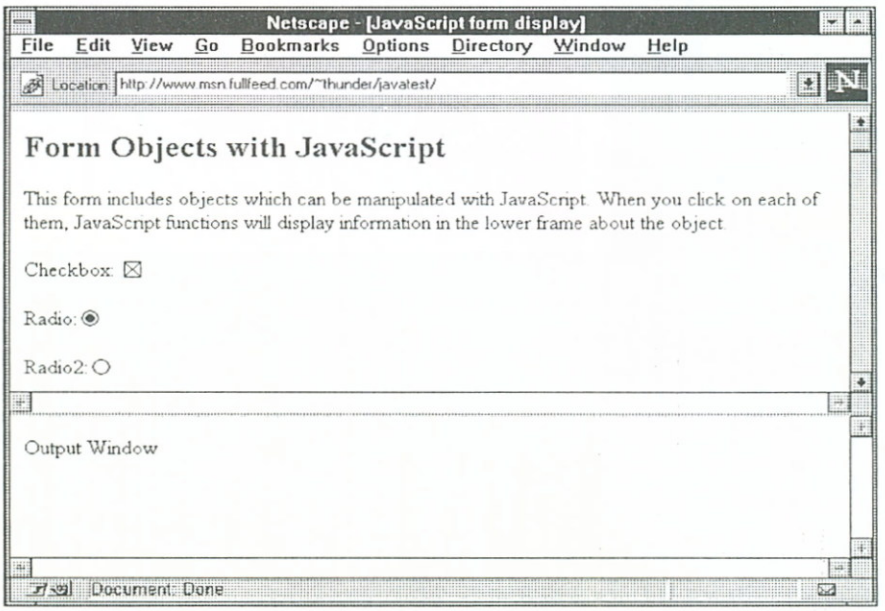
Amint látható, az applet betölti a *SoundDemo.au* nevű fájlt (a fájl megtalálható a könyvhöz beszerezhető CD-n), és az *audioKlip.loop* metódus segítségével ismétlődően lejátssza a hangfájlt mindaddig, amíg nem zárjuk be az applet ablakát. Kísérletezzünk az applettel például úgy, hogy az *audioKlip.loop* metódus helyett az *audioKlip.play* metódust használjuk arra, hogy az applet csak egyszer játssza le a hangfájlt. A *JavaHang* applet futtatásához készítsük el az alábbi bejegyzéseket tartalmazó *JavaHang.html* fájlt:

```
<HTML><TITLE>JavaHang Applet</TITLE>
<APPLET CODE="JavaHang.class" WIDTH=400 HEIGHT=300></APPLET></HTML>
```

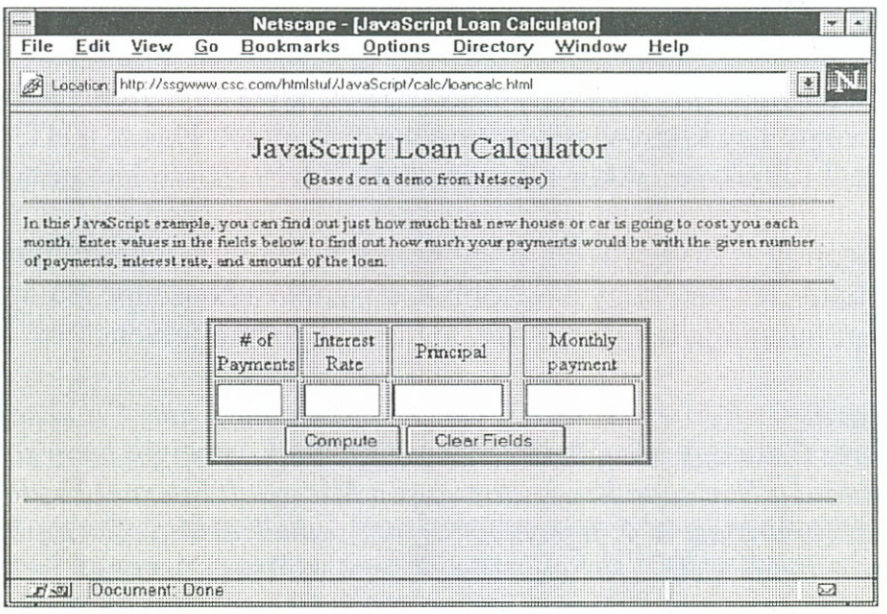
A JAVASCRIPT NYELV

Mint ismeretes, a Java programozási nyelv, amelynek segítségével grafikát, hangot, párbeszédablakot és egyéb objektumokat tartalmazó animációs Web-oldalakat készíthetünk. A hangsúly itt azon van, hogy a Java programozási nyelv. Java appletek készítéséhez programozási szakismeretekre van szükségünk.

Azért, hogy a Web-tervezők (Webmesterek) és programozók közötti szakismeretbeli különbségeket át lehessen hidalni, a Netscape létrehozta a JavaScript nyelvet. A Perl nyelvhez hasonlóan a JavaScript is parancsnyelv, amelyet a tervezők interaktív Web-oldalak készítéséhez használhatnak. A tervezők a JavaScript segítségével például a 14.17. ábrán láthatóhoz hasonló űrlapokat vagy akár a 14.18. ábrán látható interaktív számológépet tartalmazó Web-oldalt is elkészíthetik.



14.17. ábra. Interaktív űrlap készítése JavaScript nyelven



14.18.ábra. Interaktív számológép készítése JavaScript nyelven

A fejezet következő részeiben megismerkedünk a JavaScript alapjaival. Miután áttanulmányoztuk ezeket, képesek leszünk interaktív Web-oldalak megírására JavaScript nyelven. Más programo-

zási nyelvekkel szemben, mint például a Java vagy a J++, a JavaScript egyik előnye, hogy nincs szüksége fordítóprogramra. Ahhoz, hogy a felhasználó egy JavaScript fájlt használhasson, csak böngészőre van szüksége, vagyis egy olyan programra, amely feltehetően valamennyi felhasználó rendelkezésére áll.

Ha ki szeretnénk próbálni a JavaScript nyelvet, akkor nyissunk meg egy egyszerű szövegszerkesztőt, ami akár a Windows Jegyzetfüzete (Notepad) is lehet, és készítsünk el egy ASCII alapú HTML fájlt. A HTML fájlba a fejezet további részében bemutatásra kerülő bármelyik JavaScript utasítást beírhatjuk.

HOVÁ KERÜL A JAVASCRIPT

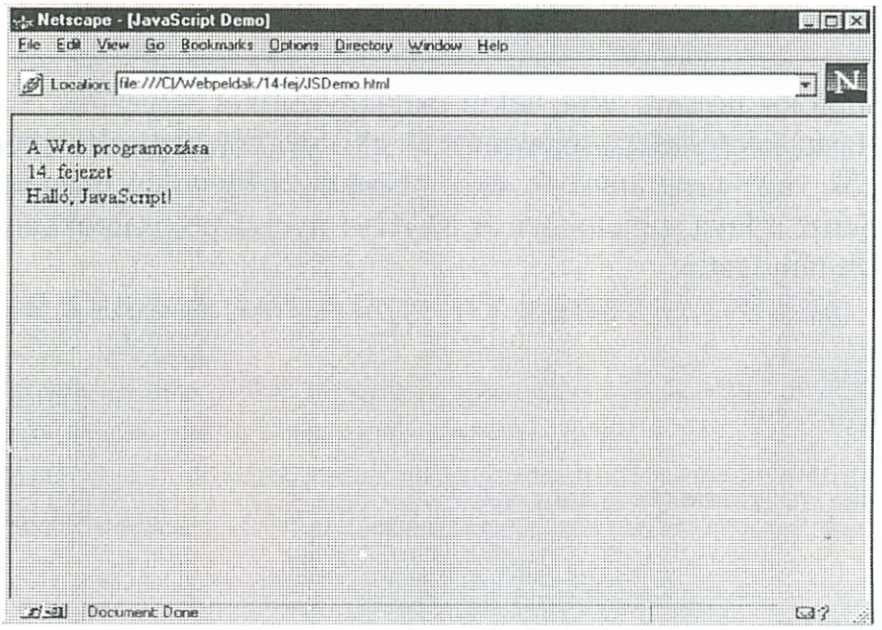
A Perlhez hasonlóan a JavaScript is parancsnyelv. Eltérően azonban a Perl scripttól, ami egy kiszolgálón fut, a JavaScript nyelven készült parancsfájlokat a böngésző futtatja. A JavaScript segítségével úgy készíthetünk script fájlokat, hogy JavaScript utasításokat helyezünk el egy HTML fájlban. Az itt következő, *JSDemo.html* nevű fájl a HTML <SCRIPT> elemét használja arra, hogy néhány egyszerű JavaScript utasítást helyezzen el a fájlban:

```
<HTML>
<HEAD><TITLE>JavaScript Demo</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">

<!- A JavaScript kód elrejtése
document.write("A Web programozása<BR>");
document.write("14. fejezet<BR>");
document.write("Halló, JavaScript!<BR>");
// A kód elrejtésének vége ->

</SCRIPT></BODY></HTML>
```

A script fájl a *document.write* függvény segítségével küld egy üzenetet a böngészőnek, és a
 elemmel hajt végre egy kocsi-vissza/soremelés műveletet. Ha ezt a HTML fájlt a Netscape-Navigator programmal töltjük be, akkor a képernyőn a 14.19. ábrán láthatóhoz hasonló ablaknak kell megjelennie.

A Javához hasonlóan itt is érvényes az, hogy nem mindegyik böngésző támogatja a JavaScript nyelvet. Ha viszont feltételezzük, hogy a legtöbb felhasználó a Netscape Navigatort vagy a Microsoft Internet Explorert használja böngészéshez, akkor nyugodtan felvehetünk JavaScript fájlokat a Web-oldalainkra.

Megjegyzés: A Javától eltérően a JavaScript használatához nincs szükség fordítóprogramra vagy fejlesztőkészletre. Csak egy böngészőre van szükségünk, amelyik megérti a JavaScript fájlokat, mint amilyen például a Netscape Navigator vagy a Microsoft Internet Explorer. Létrehozhatunk olyan HTML fájlokat (amint azt a 4. fejezetben tettük), amelyek JavaScript bejegyzéseket tartalmaznak. A böngészőn belül URL-ként egy **file://c:\valamilyen_útvonal** *helykijelölést megadva megnyithatjuk a fájlokat és végrehajthatjuk a bennük lévő JavaScript utasításokat.*



14.19. ábra. Üzenetek megjelenítése JavaScript segítségével

JAVASCRIPT PARANCSOK MEGJELENÍTÉSÉNEK KIKAPCSOLÁSA HTML MEGJEGYZÉSEKKEL

Mielőtt részleteiben megvizsgálnánk a JavaScript nyelvet, tisztában kell lennünk azzal, hogy a tervezők hogyan használják a HTML megjegyzéseket JavaScript utasítások közrefogásához. A 4. fejezetből emlékezhetünk arra, hogy a tervezők egy HTML dokumentumban a következő formátumot használják a megjegyzésekhez:

```
<!- Ez egy megjegyzés
<!- A megjegyzés vége ->
```

Amint majd látni fogjuk, a tervezők egy <SCRIPT> elemen belül a JavaScript utasításokat HTML megjegyzésekkel veszik közre az alábbi módon:

```
<SCRIPT>
<!- A megjegyzés kezdete, de a megjegyzések itt nem fejezõdnek be

JavaScript utasítások helye

// Vége a megjegyzéseknek ->
</SCRIPT>
```

Ha közelebbről megvizsgáljuk a HTML utasításokat, akkor megfigyelhetjük, hogy az első megjegyzés nem fejeződik be (nem zárják le a –> karakterek). A HTML dokumentumokban ez a helyes formátum, ezt használjuk.

A tervezők a JavaScript utasításokat HTML megjegyzésben helyezik el. Így azok a böngészőprogramok, amelyek nem támogatják a JavaScript nyelvet, figyelmen kívül hagyják a JavaScript fájlt, mert a böngésző az utasításokat megjegyzéseknek tekinti.

JAVASCRIPT MEGJEGYZÉSEK

Amikor JavaScript fájlokat hozunk létre, akkor megjegyzéseket helyezünk el a script fájlon belül, amelyek az egyes utasítások feladatát magyarázzák el. Ilyen megjegyzések beírásával segíthetjük, hogy azok a tervezők vagy programozók, akik olvassák a fájlunkat, megértsék annak célját és működését. A JavaScript a Java appletekben és a C/C++ nyelvű programokban is használatos megjegyzésformátumokat támogatja:

```
/* Ez egy megjegyzés */
// Ez is egy megjegyzés
/* Ilyen módon
     többsoros megjegyzéseket írhatunk */
```

A <SCRIPT> ELEM

Az 5. fejezetben röviden már említettük, hogy egy HTML fájlon belül arra használhatjuk a <SCRIPT> elemet, hogy megadjuk egy script dokumentum URL-jét, valamint magukat a script utasításokat. A következő bejegyzés például egy JavaScript dokumentum URL-jét adja meg:

```
<SCRIPT LANGUAGE= "JavaScript"
SRC="http:\www.jamsa.com\vmilyenscript.js"></SCRIPT>
```

Az alábbi <SCRIPT> elem olyan JavaScript utasításokat tartalmaz, amelyeket a böngészőnek végre kell hajtania:

```
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript kód elrejtése
document.write("A Web programozása<BR>");
// A kód elrejtésének vége ->

</SCRIPT></BODY></HTML>
```

JAVASCRIPT KARAKTERLÁNCOK

Amint tudjuk, egy karakterlánc 0 vagy több ASCII karakterből áll. JavaScript dokumentumban a karakterláncokat egyszeres vagy kétszeres idézőjelek közé kell tenni. Egy karakterlánc alábbi kétféle megadása egymással egyenértékű:

```
VmilyenLánc = 'Halló Világ!';
VmilyenMásLánc = "Halló Világ!";
```

Mint majd látni fogjuk, a plusz (+) operátor segítségével két karakterláncot egy karakterlánccá foghatunk össze:

```
Üzenet = 'Halló ' + "Világ!";
```

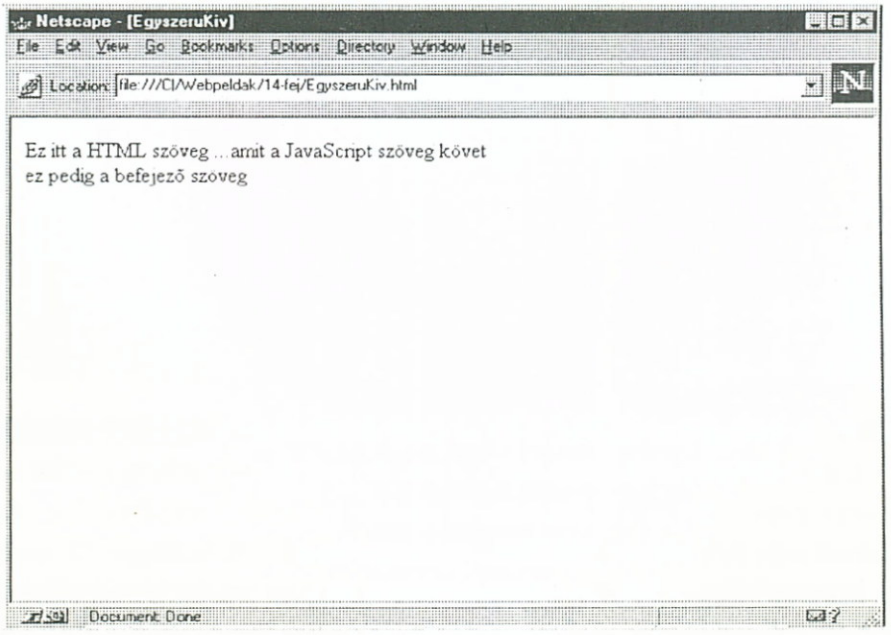
JavaScript karakterláncban ún. ESC karaktersorozatokat is elhelyezhetünk, mint például újsor karakter (\n), tab karakter (\t), lapdobás karakter (\f) stb.

EGYSZERŰ KIÍRATÁS A JAVASCRIPT SEGÍTSÉGÉVEL

Valamely HTML dokumentumon belül felváltva használhatunk HTML bejegyzéseket és JavaScript utasításokat is szövegek megjelenítéséhez. Ahhoz, hogy a JavaScript segítségével jelenítsünk meg valamilyen szöveget, a dokumentumnak a *document* objektum *write* metódusát kell használnia az alábbiak szerint.

```
<HTML>
<HEAD><TITLE>EgyszeruKiv</TITLE></HEAD>
<BODY>
Ez itt a HTML szöveg
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
document.write("...amit a JavaScript szöveg követ<BR>");
// A kód elrejtésének vége ->
</SCRIPT>
ez pedig a befejezõ szöveg
</BODY></HTML>
```

Amint látjuk, az *EgyszerűKiv.html* nevű fájlban lévő utasítások a HTML segítségével és a *document.write* utasítással jelenítenek meg szöveget. Ha a Netscape Navigatort futtatva értelmezzük ezeket az utasításokat, akkor a képernyőn a 14.20. ábrán láthatóhoz hasonlónak kell megjelennie:



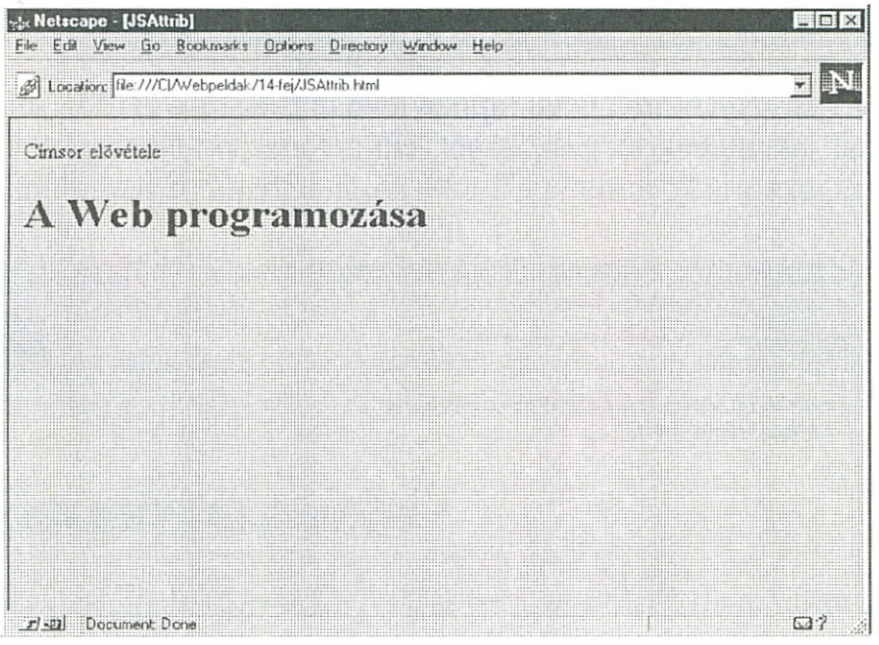
14.20. ábra. HTML és JavaScript kivitel megjelenítése

JavaScript fájlon belül a *document* objektum az aktuális HTML dokumentumnak felel meg. Ha a script fájl a kivitel megjelenítéséhez a *document.write* metódust használja, akkor a böngésző ehhez az aktuális dokumentumban lévő kivitelt veszi figyelembe.

A JavaScript kivitelének és a HTML elemek kivitelének koordinálása céljából a *document.write* metódus a dokumentum aktuális betűtípus-jellemzőit használja (méret, vastagítás, döntés stb.). Ha például a HTML dokumentum lehetővé teszi a vastagított vagy dőlt betűs szöveget, akkor a böngésző a *document.write* metódus kimenetének visszaadásához ezeket a jellemzőket fogja használni. A következő, *JSAttrib.html* fájl azt illusztrálja, hogy a betűjellemzők hogyan befolyásolják a JavaScript kimenetét:

```
<HTML>
<HEAD><TITLE>JSAttrib</TITLE></HEAD>
<BODY>
Címsor elõvétele <H1>
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
document.write("A Web programozása");
// A kód elrejtésének vége ->
</SCRIPT></H1>
</BODY></HTML>
```

Ha a Netscape Navigatort futtatva értelmezzük ezeket az utasításokat, akkor a képernyőn a 14.21. ábrán láthatóhoz hasonlónak kell megjelennie:

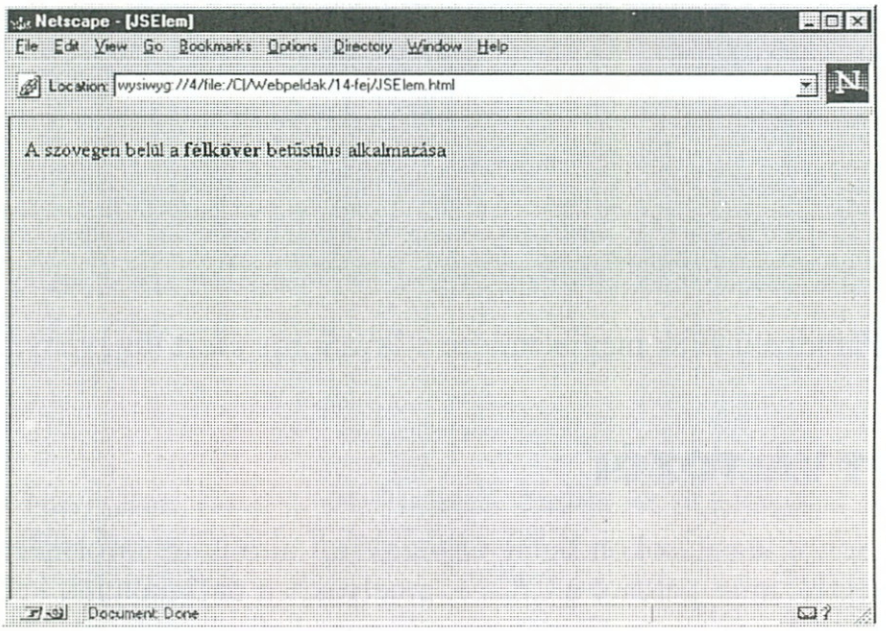


14.21.ábra. A böngészők a HTML aktuális betűtípus-beállításait használják a *document.write* metódus kimenetének megjelenítéséhez

Továbbmenve, ha olyan HTML elemeket, mint például <B> vagy <I> illesztünk be a *document.write* metódus kimenetébe, akkor a böngésző a kimenet megjelenítésekor alkalmazni fogja az ezeknek megfelelő betűjellemzőket. A következő, *JSElem.html* fájl például ilyen elemet használ arra, hogy a *document.write* metódus kimenetén belül megváltoztasson egy betűjellemzőt:

```
<HTML>
<HEAD><TITLE>JSElem</TITLE></HEAD>
<BODY>
A szövegen belül a
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
document.write("<B>");
// A kód elrejtésének vége ->
</SCRIPT>
félkövér
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
document.write("</B>");
// Kód elrejtésének vége ->
</SCRIPT>
betûstílus alkalmazása
</BODY></HTML>
```

Ha a Netscape Navigatort futtatva értelmezzük ezeket az utasításokat, akkor a képernyőn a 14.22. ábrán láthatóhoz hasonlónak kell megjelennie:



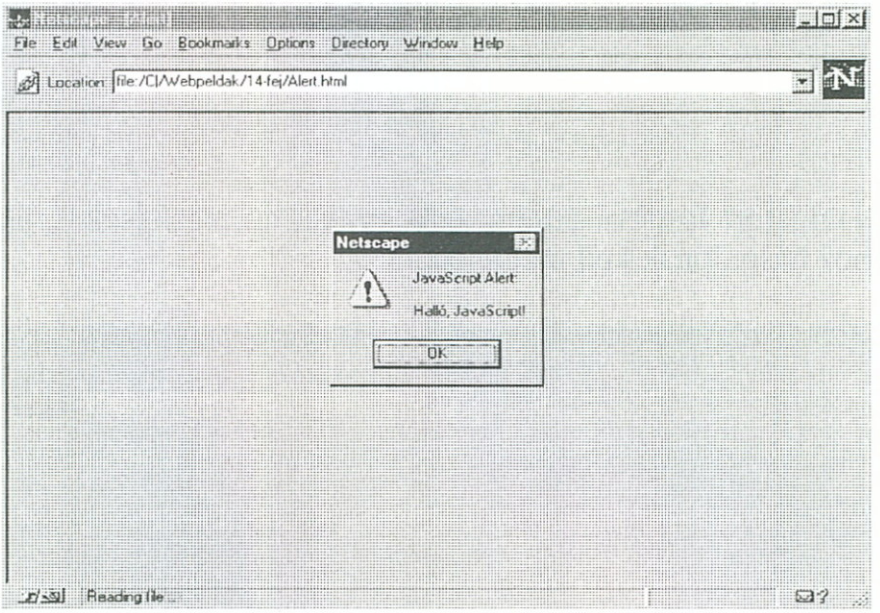
14.22. ábra. Betűjellemzők beállítása a *document.write* metódus segítségével

EGYSZERŰ ÜZENETABLAKOK KÉSZÍTÉSE

Amint az előbb láttuk, a JavaScript *document* objektumának *write* és *writeln* metódusa segítségével egy script fájl szöveges kimenetet tud létrehozni egy HTML dokumentumban. A script fájl feladatától függően előfordulhatnak olyan esetek, amikor egy párbeszédablakban valamilyen üzenetet szeretnénk megjeleníteni a felhasználó számára. Ilyen párbeszédablak megjelenítéséhez az *alert* függvényt használhatjuk. A következő, *Alert.html* fájl az *alert* függvényt használja ahhoz, hogy a *Halló, JavaScript!* üzenetet jelenítse meg egy párbeszédablakban:

```
<HTML>
<HEAD><TITLE>Alert</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
alert("\nHalló, JavaScript!");
// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Netscape Navigatort futtatva értelmezzük ezeket az utasításokat, akkor a képernyőn a 14.23. ábrán láthatóhoz hasonlónak kell megjelennie:



14.23. ábra. Párbeszédablak megjelenítése a JavaScript *alert* függvénye segítségével

A JAVASCRIPT VÁLTOZÓI

A script fájl a JavaScript utasításokon belül változókban tárolja az információkat. Eltérően más programozási nyelvektől, mint például a C/C++, amelyekben definiálnunk kell a változók típusát *(int, float, char stb.)*, a JavaScript változók típusát nem kell megadnunk. Ehelyett a JavaScript utasításokat végrehajtó böngésző állapítja meg az egyes változók típusát a változók használata alapján.

A JavaScript változók nevének betűvel vagy az aláhúzás (_) karakterrel kell kezdődniük. Továbbá a JavaScript változók nevei „érzékenyek" a kis- és a nagybetűs írásmódra, ami azt jelenti, hogy a JavaScript más és más változónak tekinti a *nagy*, a *Nagy* és a *NAGY* nevű változókat.

Script fájlon belül egy változót a *var* kulcsszóval, majd a változó nevének megadásával deklarálunk. Az alábbi utasítások például különböző változókat deklarálnak:

```
var KonyvCim, FejezetSzam;
var Kiado;
var x, y, z;
```

Script fájlban a változók deklarálásakor az értékadó operátor (=) segítségével egyúttal kezdeti értékeket is rendelhetünk a változókhoz az alábbiak szerint:

```
var KonyvCim = "A Web programozása", FejezetSzam = 14;
var Kiado = "Jamsa Press";
var x = 1, y = 2, z = 3;
var a, b, c = 3; // csak a c változó kap kezdeti értéket
```

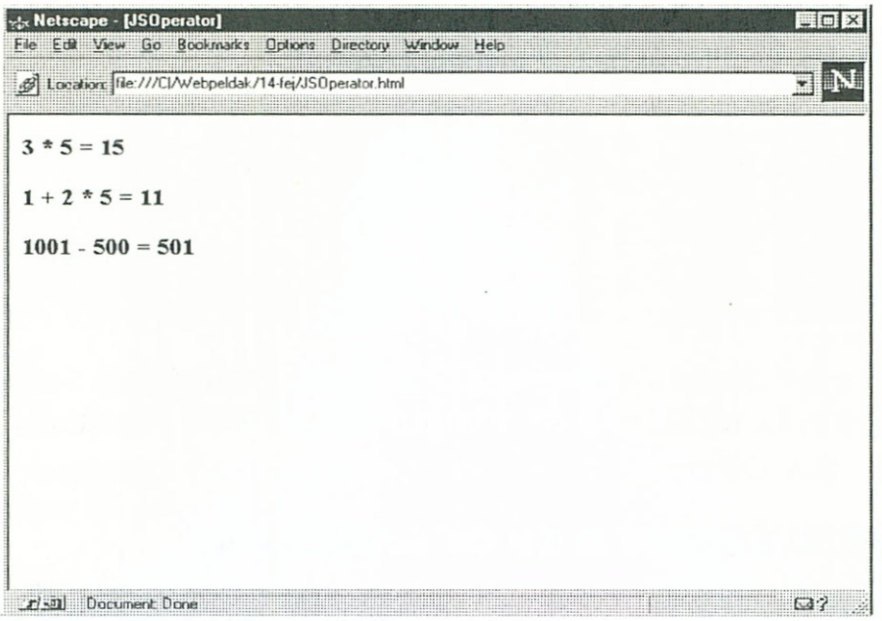
Mint látni fogjuk, a JavaScript lehetővé teszi objektumok definiálását. Script fájlon belül karakterláncok kezeléséhez nyújt segítséget a JavaScript beépített *String* objektuma.

A JAVASCRIPT MŰVELETEI ÉS OPERÁTORAI

Egy script fájlon belül számos olyan operátor (műveleti jel) használható, mint amilyenekkel C/C++ vagy Java programokban találkozhatunk. Az alábbi, *JSOperatorok.html* nevű fájl a leggyakrabban előforduló JavaScript műveletek és operátorok használatát mutatja be:

```
<HTML>
<HEAD><TITLE>JSOperator</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
document.write("<H3>3 * 5 = " + (3 * 5) + "</H3>");
document.write("<H3>1 + 2 * 5 = " + (1 + 2 * 5) +
"</H3>");
document.write("<H3>1001 - 500 = " + (1001 - 500) +
"</H3>");
// Kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Netscape Navigatort futtatva értelmezzük ezeket az utasításokat, akkor a képernyőn a 14.24. ábrán láthatóhoz hasonlónak kell megjelennie:



14.24. ábra. A JavaScript leggyakoribb műveleteinek használata

Más programozási nyelvekhez hasonlóan a JavaScript műveleteihez is meghatározott sorrendiség (precedencia) tartozik. A 14.6. táblázat a JavaScript operátorait sorolja fel precedenciájuk szerinti sorrendben a legmagasabbtól a legalacsonyabbig haladva:

Operátor	Neve	Példa
.	Tagválasztó	objektum.tag_neve
[]	Index	tömb[elem]
()	Függvényhívás	kifejezés(paraméterek)
+	Postfix inkrementáló	változó++
+	Prefix inkrementáló	++változó
--	Postfix dekrementáló	változó--
--	Prefix dekrementáló	--változó
~	1-es komplemens	~kifejezés
!	NOT művelet jele	!kifejezés
*	Szorzás	kifejezés * kifejezés
/	Osztás	kifejezés / kifejezés
%	Modulo	kifejezés % kifejezés
+	Összeadás	kifejezés + kifejezés
-	Kivonás	kifejezés – kifejezés
<<	Bitenkénti léptetés balra	kifejezés << kifejezés
>>	Bitenkénti léptetés jobbra	kifejezés >> kifejezés
>>>	Nullával feltöltő jobbra léptetés	kifejezés >>> kifejezés
<	Kisebb mint	kifejezés < kifejezés
>	Nagyobb mint	kifejezés > kifejezés
<=	Kisebb vagy egyenlő	kifejezés <= kifejezés
>=	Nagyobb vagy egyenlő	kifejezés >= kifejezés

Operátor	Neve	Példa
==	Egyezőség	kifejezés == kifejezés
!=	Nem egyezőség	kifejezés != kifejezés
&	Bitenkénti AND	kifejezés & kifejezés
^	Bitenkénti kizáró OR	kifejezés ^ kifejezés
|	Bitenkénti OR	kifejezés | kifejezés
&&	Logikai AND	kifejezés && kifejezés
||	Logikai OR	kifejezés || kifejezés
?:	if-else	(boolean_kifejezés) ? igaz_kifejezés: hamis_kifejezés
operátor=	Értékadás	változó *= kifejezés;

14.6. táblázat. A JavaScript operátorai a precedenciájuk sorrendjében

SZÖVEG BEKÉRÉSE A FELHASZNÁLÓTÓL

Amint látni fogjuk, űrlapok használatával bármilyen típusú adatbevitelre (beírás beviteli mezőbe, jelölőnégyzetek bejelölése stb.) felkérhetjük a felhasználót. A script fájl rendeltetésétől függően előfordulhat, hogy mindössze csak egyetlen soros szöveget várunk a felhasználótól. Ilyen esetekben a script fájl a *prompt* függvény segítségével megjeleníthet egy párbeszédablakot, amelyben felszólítja a felhasználót az adat bevitelére, majd a függvény visszaadja a bevitt adatot. Az itt következő, *Prompt.html* nevű fájl a *prompt* függvény használatát mutatja be:

```
<HTML>
<HEAD><TITLE>Prompt</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
var nev, kor;
nev = prompt("Írja be a nevét:", "");
kor = prompt("Írja be az életkorát:", "");
document.write("<H3>Az Ön neve: " + nev + "</H3>");
document.write("<H3>Életkora: " + kor + "</H3>");
// Kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Netscape Navigatort futtatva értelmezzük ezeket az utasításokat, akkor a képernyőn a *prompt* függvény minden egyes meghívásakor egy párbeszédablaknak kell megjelennie. A *prompt* függvény második paramétere az alapértelmezett értéket adja meg.

AZ EVAL FÜGGVÉNY

A script fájl rendeltetésétől függően sokszor lehet szükség arra, hogy a script egyszerű számításokat végezzen – például kiszámítsa egy előre beírt értékre a hozzá tartozó forgalmi adót. Számítások

végzéséhez a script meghívhatja az *eval* függvényt. A következő, *JSKalk.html* nevű fájl a *prompt* és az *eval* függvény kombinálásával megjeleníti annak a kifejezésnek (mint például 5+3*4/2) az eredményét, amelyet a felhasználó írt be a nála megjelent párbeszédablak mezőjébe:

```
<HTML>
<HEAD><TITLE>JSKalk</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
eredmeny = eval(prompt("Írjon be egy kifejezést:", ""));
document.write("<H3>Az eredménye: " + eredmeny + "</BR>");
// Kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

A JAVASCRIPT FOGLALT SZAVAI

Más programozási nyelvekhez hasonlóan a JavaScript nyelvben is van egy sor foglalt szó, amelyek nem használhatók névazonosítóként, például változók vagy függvények neveként. A JavaScript foglalt szavait a 14.7. táblázat sorolja fel.

abstract	boolean	break	byte	case
catch	char	class	const	continue
default	do	double	else	extends
false	final	finally	float	for
function	goto	if	implements	import
in	instanceof	int	interface	long
native	new	null	package	private
protected	public	return	short	static
super	switch	synchronized	this	throw
throws	transient	true	try	var
void	while	with		

14.7. táblázat. A JavaScript foglalt szavai

A JAVASCRIPT PROGRAMVEZÉRLŐ SZERKEZETEI

A JavaScript nyelv ugyanazokat a programvezérlő szerkezeteket használja, mint amelyeket a C/C++ és a Java. A 14.8. táblázat röviden bemutatja a JavaScript programvezérlő szerkezeteit.

Programszerkezet	Formátum
if	if (feltétel) utasítás;

Programszerkezet	Formátum
if else	if (feltétel) utasítás; else utasítás;
for	for (kifejezés; feltétel; kifejezés) utasítás;
for in	for (var objektumban) utasítás;
while	while (feltétel) utasítás;
with	with (objektum) utasítás
switch	switch (kifejezés) { case 1: utasítás; break; case 2: utasítás; break; default: utasítás; }
goto	goto címke;

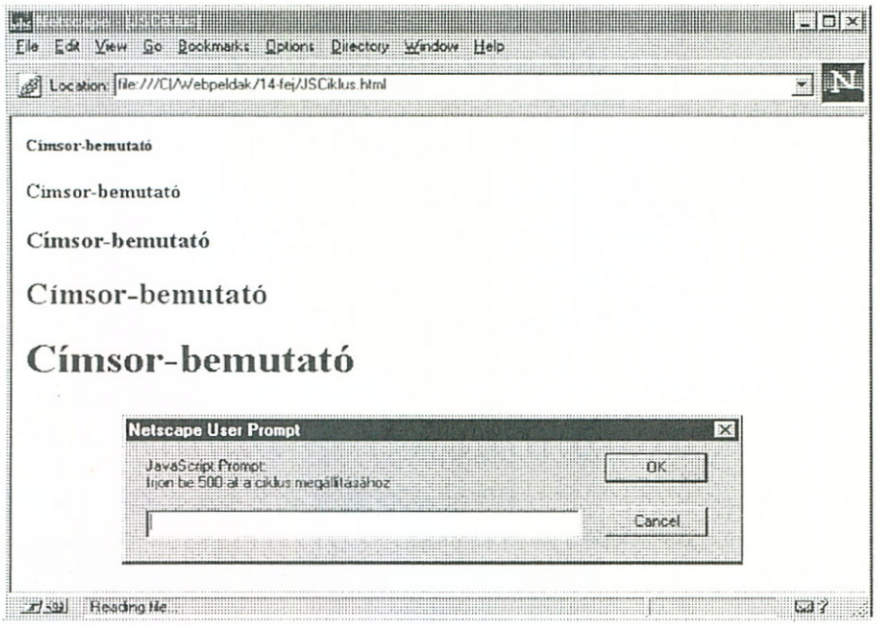
14.8. táblázat. Programvezérlő szerkezetek a JavaScript nyelvben

Az itt következő, *JSCiklusok.html* nevű fájl a JavaScript néhány ciklusvezérelt programszerkezetére mutat példát:

```
<HTML>
<HEAD><TITLE>JSCiklus</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
var i;
for (i = 5; i > 0; i-)
    document.write("<H" + i + "> Címsor-bemutató</H" + i +">");

szam = "";
while (szam != "500")
  {
    szam = prompt("Írjon be 500-at a ciklus megállításához", "");
  }

// Kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

A *for* ciklus a HTML <H5> és <H1> közötti címsor elemeinek használatát mutatja be. A *while* ciklus felszólítja a felhasználót, hogy írja be az 500-as számot. A *while* ciklus karakterláncot hasz-

nál a bevitel vizsgálatára, hogy a script fájlnak ne kelljen arra figyelnie, hogy a felhasználó numerikus értéket írt-e be. Amikor a felhasználó a helyes számot írja be, a ciklus futása befejeződik. Ha a Netscape Navigatort futtatva értelmezzük ezeket az utasításokat, akkor a képernyőn a 14.25. ábrán láthatóhoz hasonlónak kell megjelennie:



14.25. ábra. JavaScript ciklusvezérlő programszerkezeteivel előállított kimenetek megjelenítése

A JAVASCRIPT FÜGGVÉNYEI

Ahhoz hasonlóan, ahogyan a Java és a C/C++ nyelven programozók függvényeket használnak a nagyobb feladatok kisebb, jobban kezelhető részekre bontásához, a JavaScript nyelvben is használhatunk függvényeket a programok egyszerűsítéséhez. A Java függvényeihez hasonlóan a script fájlok is átadhatnak paramétereket a függvényeknek, a függvények pedig értékeket adhatnak vissza a hívójuknak. A Java és a JavaScript függvények között az a különbség, hogy a JavaScript nyelvben a függvénydefiníció elé kötelezően be kell írni a *function* kulcsszót, és nem kell megadnunk a függvény visszatérési értékének, valamint a paramétereknek a típusát:

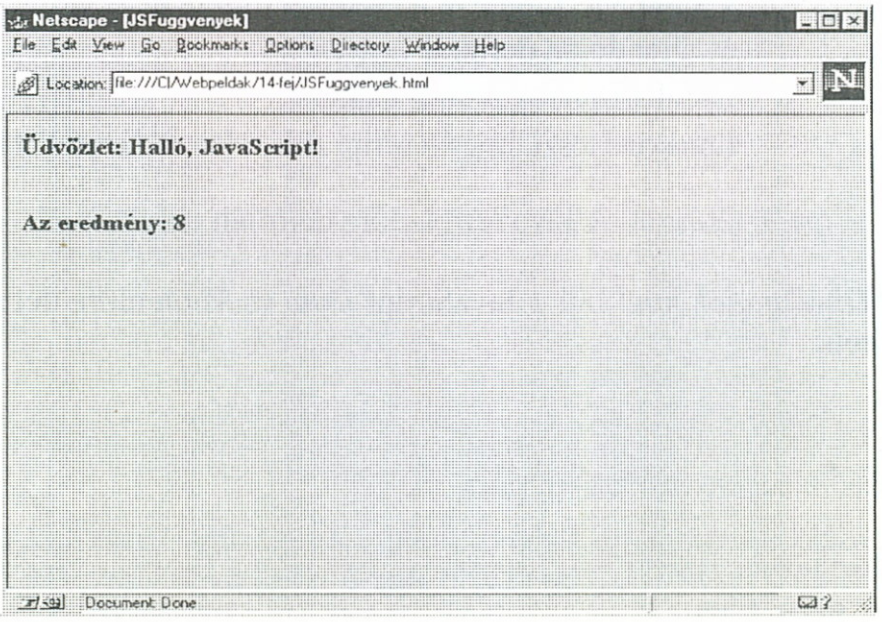
```
function VmilyenNev(elso_parameter, masodik_parameter)
  {
    // utasítások
    return(eredmeny);
  }
```

A következő, *JSFüggvények.html* nevű fájl a JavaScript függvényeinek használatát illusztrálja:

```
<HTML>
<HEAD><TITLE>JSFuggvenyek</TITLE></HEAD>
<BODY>
<H3>Üdvözlet:
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
```

```
function Hello()
  {
    document.write("Halló, ");
  }

function UdvVege(udvozlet)
  {
    document.write(udvozlet);
  }

Hello();                                    // a függvény hívása
UdvVege("JavaScript!");
// Kód elrejtésének vége ->
</SCRIPT>
</H3><BR>
<H3>Az eredmény:
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
function szamit(a, b)
  {
    return(a + b);
  }

document.write(szamit(3, 5));
// Kód elrejtésének vége ->
</SCRIPT>
</H3></BODY></HTML>
```

Amint látható, a HTML dokumentum két script kódot tartalmaz, amelyek közül az első kód két, a második pedig egy függvényt definiál. Ha a Netscape Navigatort futtatva értelmezzük a dokumentumban lévő utasításokat, akkor a képernyőn a 14.26. ábrán láthatóhoz hasonlónak kell megjelennie:



14.26. ábra. A JavaScript függvények használatának illusztrálása

A JAVASCRIPT TÖMBJEI

Bizonyára ismert, hogy egy tömb (mátrix) olyan adatszerkezet, amely lehetővé teszi, hogy a program egyetlen változóban több, azonos típusú értéket tároljon (például 100 tanuló nevét). A JavaScript tömbjei a Javában és a Perlben használatos tömbök keverékei. A Javához hasonlóan a tömb elemeit 0-tól kezdődően indexelhetjük, azaz a tömb első eleme az *elem[0]*. Ugyanakkor a tömbelemeket a Perlhez hasonlóan nem numerikus értékkel is indexelhetjük, hogy asszociatív tömböt hozzunk létre (mint például *alkalmazott,,Feri"*).

A JAVASCRIPT OBJEKTUMAI

A Java programozási nyelvhez hasonlóan a JavaScript is objektumorientált, ami azt jelenti, hogy a JavaScript is támogatja az objektumok használatát. Ezek az objektumok itt is azokat az értékeket határozzák meg, amelyeket egy változó tárolni tud, valamint azokat a műveleteket, amelyeket a program a változón el tud végezni. Amint a fejezet előző, a Javáról szóló részében említettük, azokat az osztály alapú függvényeket, amelyek egy objektumon végeznek műveleteket, *metódusoknak* nevezik. A 14.9. táblázat a JavaScript előre definiált objektumait sorolja fel. Ezek ismerete segítheti a JavaScript objektumainak könnyebb megértését.

Objektum	Rendeltetése
Date	Az aktuális rendszerdátumot adja vissza
Document	Lehetővé teszi, hogy egy script adatokat írjon egy HTML oldalra
Form	Lehetővé teszi, hogy egy script adatokat gyűjtsön és jelenítsen meg
History	Lehetővé teszi, hogy a felhasználó hozzáférjen az előzmények listájához
Location	Lehetővé teszi, hogy a script kezelje az aktuális URL-t
Math	Különböző fontos matematikai függvényeket és állandókat definiál
String	Karakterláncokat tárol és kezel
Windows	Lehetővé teszi, hogy a script ablakokat és kereteket készítsen

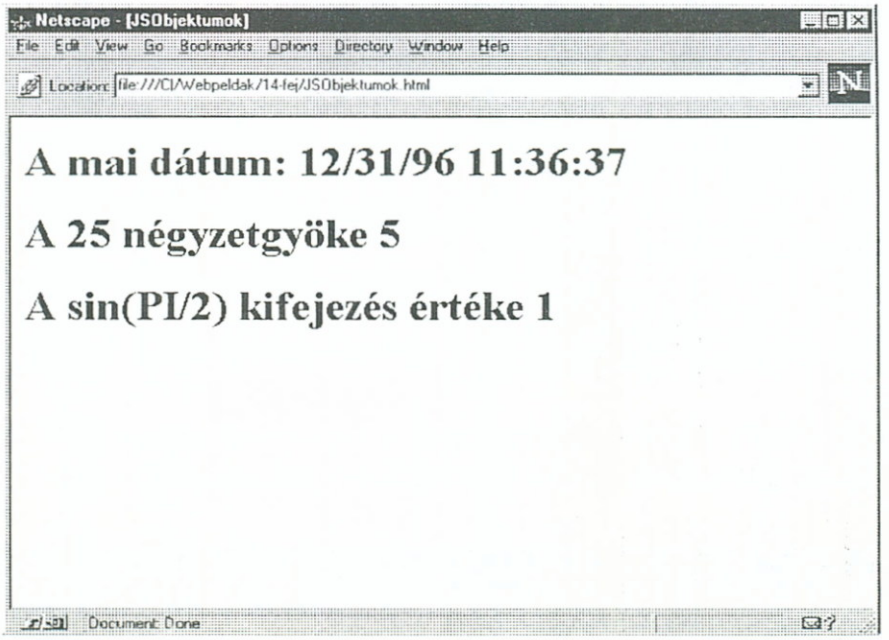
14.9. táblázat. A JavaScript előre definiált objektumai

A következő, *JSObjektumok.html* fájl a JavaScript *Date* és *Math* előre definiált objektumainak használatát szemlélteti:

```
<HTML>
<HEAD><TITLE>JSObjektumok</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
MaiNap = new Date();
```

```
document.write("<H1>A mai dátum: " + MaiNap.toLocaleString() + "</H1>");

document.write("<H1>A 25 négyzetgyöke " + Math.sqrt(25) + "</H1>");
document.write("<H1>A sin(PI/2) kifejezés értéke " + Math.sin(Math.PI / 2)
+ "</H1>");
// Kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Netscape Navigatort futtatva értelmezzük a dokumentumban lévő utasításokat, akkor a képernyőn a 14.27. ábrán láthatóhoz hasonlónak kell megjelennie:



14.27. ábra. A JavaScript előre definiált objektumainak használata

SAJÁT JAVASCRIPT OBJEKTUMOK LÉTREHOZÁSA

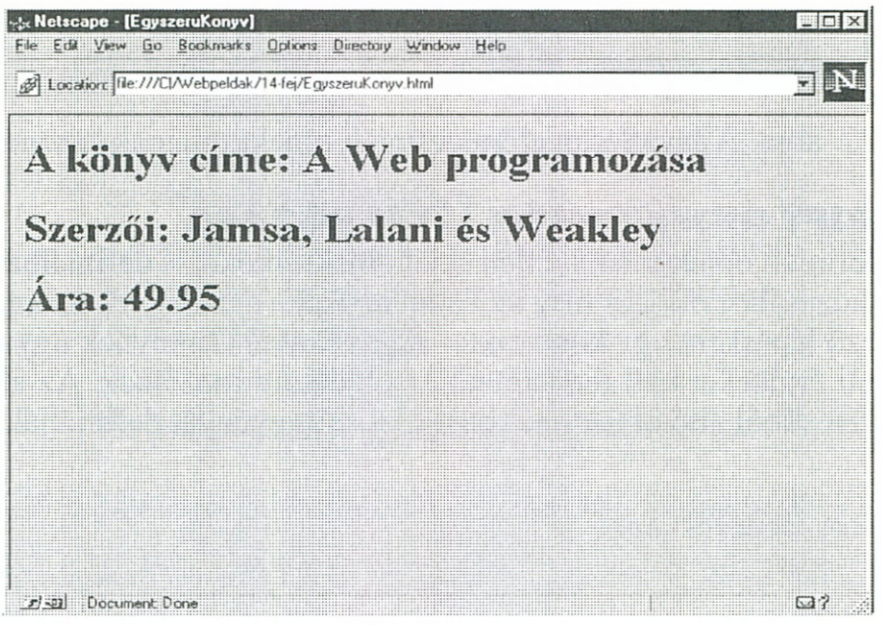
A JavaScript előre definiált objektumai mellett saját objektumokat is definiálhatunk egy script kódban. Objektum létrehozásához először meg kell adni azokat a mezőneveket, amelyeket az objektum tárolni fog. Következő lépésként egy konstruktor függvénnyel (amelynek ugyanaz a neve, mint az objektumnak) kezdeti értékeket kell rendelnünk az objektum mezőihez. Az alábbi utasítások egy *Könyv* nevű, három mezőből álló objektum konstruktor függvényét definiálják:

```
function Konyv(cim, szerzo, ar)
  {
    this.cim = cim;
    this.szerzo = szerzo;
    this.ar = ar:
  }
```

A függvény a *this* kulcsszó segítségével rendel kezdeti értékeket az objektum mezőihez. A következő, *EgyszerűKönyv.html* nevű fájl a *Könyv* objektum használatát szemlélteti:

```
<HTML>
<HEAD><TITLE>EgyszeruKonyv</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése>
function Konyv(cim, szerzo, ar)
  {
    this.cim = cim;
    this.szerzo = szerzo;
    this.ar = ar;
  }

WebKonyv = new Konyv("A Web programozása", "Jamsa, Lalani és Weakley", 49.95);

document.write("<H1>A könyv címe: " + WebKonyv.cim + "</H1>");
document.write("<H1>Szerzõi: " + WebKonyv.szerzo + "</H1>");
document.write("<H1>Ára: " + WebKonyv.ar + "</H1>");

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Netscape Navigatort futtatva értelmezzük a dokumentumban lévő utasításokat, akkor a képernyőn a 14.28. ábrán láthatóhoz hasonlónak kell megjelennie:

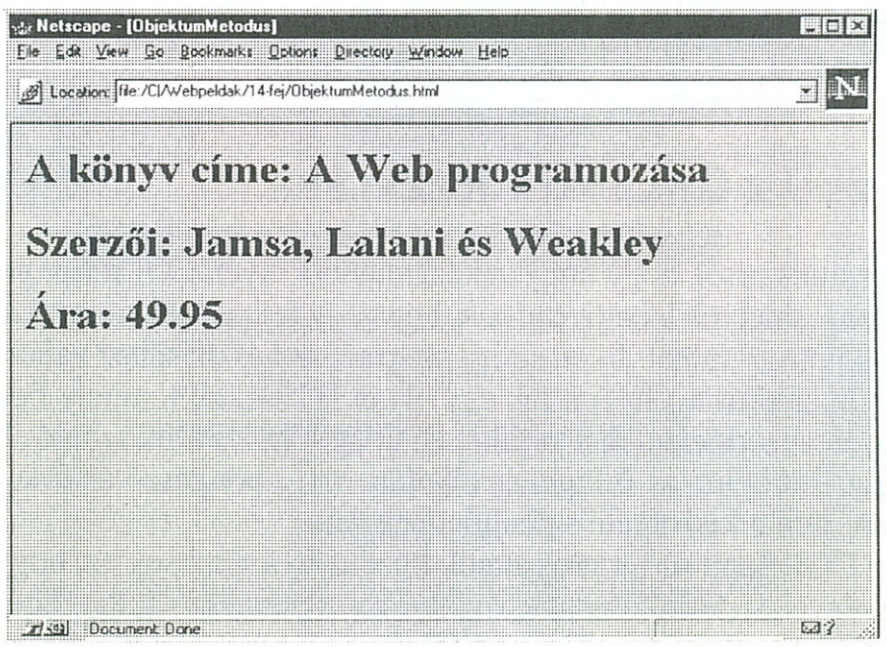


14.28. ábra. Objektummezők használatának szemléltetése

Az előbb bemutatott *Könyv* objektum meglehetősen egyszerű volt olyan értelemben, hogy csak adatmező-értékeket definiált. Más szavakkal, az objektum nem definiált metódusokat (függvényeket), amelyek műveleteket végeztek volna az adatokon. Metódust úgy rendelhetünk egy objektumhoz, hogy egyszerűen definiáljuk a metódust (mint egy JavaScript függvényt), majd hozzárendeljük az objektum megfelelő mezőnevéhez. A következő, *ObjektumMetódus.html* nevű fájl az adat- és metódusmezők használatát mutatja be a *Könyv* objektumon keresztül:

```
<HTML>
<HEAD><TITLE>ObjektumMetodus</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!- A JavaScript elrejtése
function adat_megjelenito()
  {
    document.write("<H1>A könyv címe: " + this.cim + "</H1>");
    document.write("<H1>Szerzõi: " + this.szerzo + "</H1>");
    document.write("<H1>Ára: " + this.ar + "</H1>");
  }

function Konyv(cim, szerzo, ar)
  {
    this.cim = cim;
    this.szerzo = szerzo;
    this.ar = ar;
    this.adat_megjelenito = adat_megjelenito;
  }

WebKonyv = new Konyv("A Web programozása", "Jamsa, Lalani és Weakley", 49.95);
WebKonyv.adat_megjelenito();

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Netscape Navigatort futtatva értelmezzük a dokumentumban lévő utasításokat, akkor a képernyőn a 14.29. ábrán láthatóhoz hasonlónak kell megjelennie:



14.29. ábra. Példa objektum metódusainak használatára

A JAVASCRIPT ESEMÉNYEI

Windows alapú vagy böngésző alapú környezetben a programoknak gyakran kell reagálniuk olyan eseményekre, mint egy billentyű lenyomása vagy az egér mozgatása. Az ilyen események kezeléséhez a JavaScript fájlok tipikusan egy függvényt definiálnak, amelynek utasításai feldolgozzák az eseményt. A JavaScript különböző eseményeket definiál, amelyekre a script fájl reagálni tud. A 14.10. táblázat röviden bemutatja a JavaScript eseményeit.

Esemény	Leírása
blur	Akkor fordul elő, amikor a felhasználó az egérrel az aktuális mezőn kívülre kattint
change	Akkor fordul elő, amikor a felhasználó megváltoztat egy értéket egy űrlapon
click	Akkor fordul elő, amikor a felhasználó ez egérrel egy csatolásra vagy az űrlap valamelyik mezőjére kattint
focus	Akkor fordul elő, amikor a felhasználó kijelöl egy elemet egy űrlapon
load	Akkor fordul elő, amikor a böngésző betölti az oldalt
mouseover	Akkor fordul elő, amikor a felhasználó az egér mutatóját áthúzza egy csatoláson
select	Akkor fordul elő, amikor a felhasználó kijelöl egy mezőt az űrlapon
submit	Akkor fordul elő, amikor a felhasználó elküldi az űrlapot
unload	Akkor fordul elő, amikor a felhasználó másik HTML oldalra lép

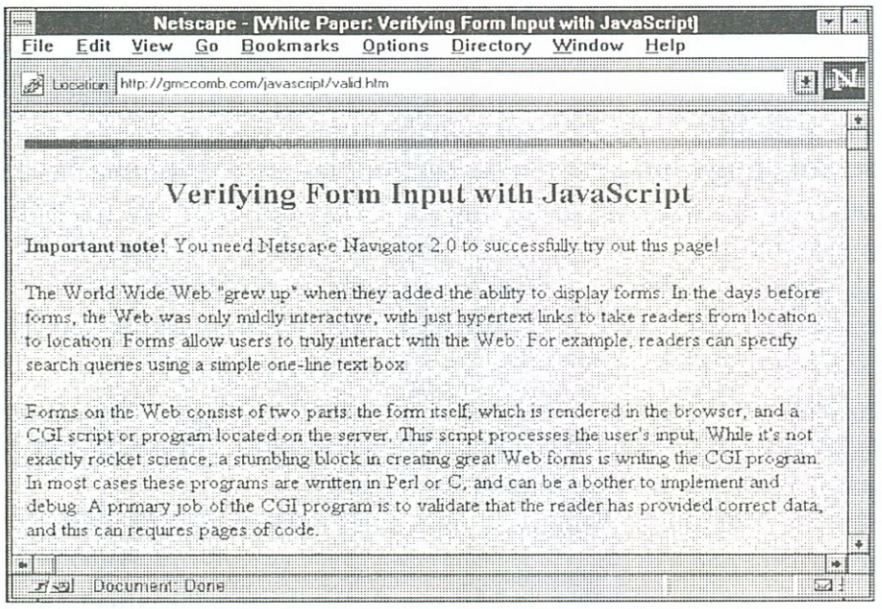
14.10. táblázat. Események, amelyekre a JavaScript alapú script fájlok reagálni tudnak

Amint láttuk, a Javában az eseménykezelőt függvény valósítja meg, mint amilyen például a fejezet első részében bemutatott *keyDown* és *keyUp*. Ezzel szemben a JavaScript nyelvben az eseménykezelő egy script kód.

JAVASCRIPT ÉS AZ ŰRLAPOK KEZELÉSE

A 11. fejezetben olvastunk arról, hogy hogyan készíthetünk olyan CGI script programokat (C/C++, Perl és más nyelveken), amelyek lehetővé teszik, hogy egy felhasználó interaktív módon együttműködjön egy kiszolgáló programmal. Ilyen script programok készítéséhez valamelyik programozási nyelv segítségével (dinamikusan) felépítünk egy olyan HTML fájlt, amely beviteli mezőket, jelölőnégyzeteket és egyéb vezérlőelemeket tartalmazó űrlapot tesz a felhasználó elé. Miután a felhasználó a szükséges adatokat beírta az űrlapra és elküldte azt, a böngésző az űrlap adatait elküldi egy speciális programnak, ami a kiszolgáló számítógépen fut.

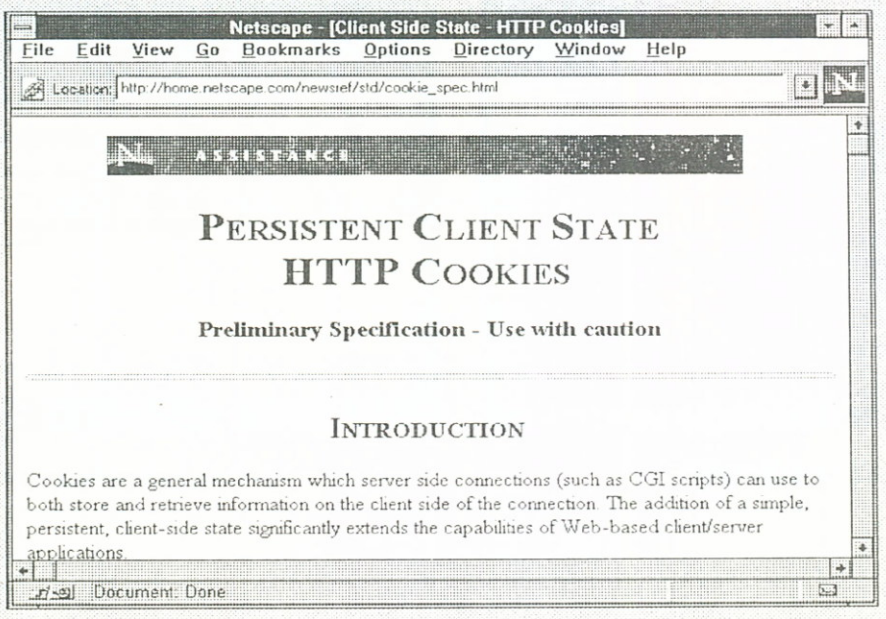
A JavaScript és a CGI alapú script programok közötti fő különbség az, hogy a JavaScript segítségével olyan függvényeket definiálhatunk, amelyeket a böngésző futtat, hogy feldolgozza a felhasználó által bevitt adatokat. Ugyanakkor a CGI alapú script fájlokat a kiszolgálón lévő programok futtatják. Ha további információkat szeretnénk megtudni arról, hogy miként használhatjuk a JavaScript nyelvet CGI-szerű űrlapok készítéséhez, akkor látogassuk meg a következő Web-helyet: *http://gmccomb.com/javascript/valid.htm* (lásd a 14.30. ábrát).



14.30. ábra. CGI-szerű script programok készítése JavaScript segítségével

COOKIE-K („SÜTIK")

A böngészőkkel kapcsolatban kevésbé ismert dolog, amiről talán még sok tapasztalt Web-szörföző sem tud, hogy egy Web-hely arra utasíthatja a böngészőnket, hogy bizonyos információkat tároljon a merevlemezünkön egy úgynevezett *cookie* fájlban. Tegyük fel például, hogy meglátogatjuk a Jamsa Press Web-oldalát. Ha hosszabb időt töltünk el ott a Java könyvek tallózásával, akkor a Web-hely utasíthatja a böngészőnket, hogy tárolja a „Java Books" (Java könyvek) karakterláncot a Web-hely cookie fájljában (ami a merevlemezünkön jön létre). Ha a későbbiekben ismét felkeressük a Jamsa Press helyét, akkor a hely speciális műveleteket végezhet annak az ismeretnek a birtokában, hogy érdeklődünk a Java könyvek iránt. Ha több helyet keresünk fel, akkor mindegyik hely (vagyis mindegyik HTML dokumentum) tárolhatja a merevlemezünkön a maga *cookie* fájlját. További információkat kaphatunk a *http://home.netscape.com/newsref/std/cookie_spec.html* címen (lásd a 14.31. ábrát).



14.31. ábra. Információk megtekintése a Web *cookie* fájljairól

ÖSSZEFOGLALÁS

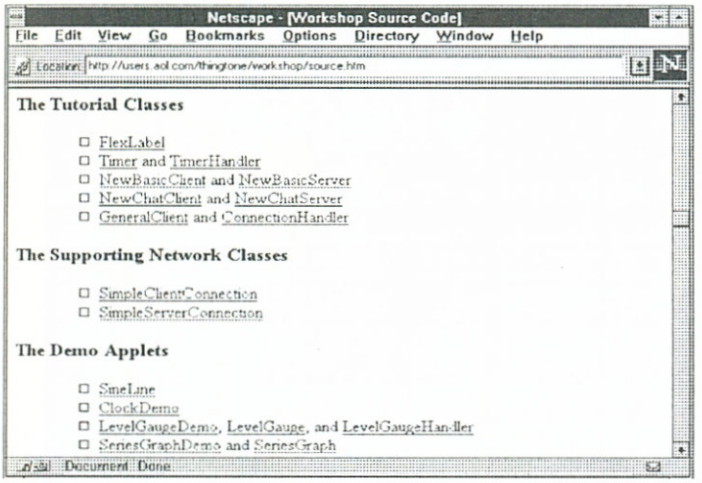
Ebben a fejezetben arról olvashattunk, hogy miként készíthetünk animációs Web-helyeket Java programozási nyelven. Röviden olvashattunk a Java olyan változatairól is, mint a J++, a Latte és a Café. Azt is megtudhattuk, hogy miképpen készíthetünk parancsfájlokat a JavaScript nyelv segítségével. A 15. fejezetben azt tanulhatjuk meg, hogy a VBScript nyelv és az ActiveX vezérlőelemek segítségével hogyan hozhatunk létre interaktív Web-helyeket. Mielőtt azonban rátérnénk erre a fejezetre, győződjünk meg arról, hogy tisztában vagyunk az alábbiakkal:

- A Web-oldaltervezők munkájának támogatására a Sun Microsystems Inc. kifejlesztette a World Wide Web programozásához használható Java programozási nyelvet.
- A programozók a Java segítségével kisebb alkalmazásokat (appleteket) készíthetnek, amelyekkel hangot, zenét, animációt és interaktív kapcsolatokat integrálhatnak egy Web-helyre.
- A Java nagyon hasonlít a C++ programozási nyelvhez.
- A könyvhöz beszerezhető CD tartalmazza a Java fejlesztőkészletét (JDK), amelyet a Sun ingyenesen terjeszt a programozók között.
- A Visual J++ – korábbi nevén Jakarta – a Microsoft által megvalósított Java programozási környezet.
- A Latte a Borland által kidolgozott Java integrált programfejlesztői környezet.
- Az applet olyan Java nyelvű program, amelyet böngésző tud végrehajtani. Az önálló Java program olyan program, amely a felhasználó rendszerén fut és böngészővel nem futtatható.
- A Java kifejlesztői úgy csökkentették annak a veszélyét, hogy a programozók vírusprogramokat írhassanak Java nyelven, hogy korlátozták azokat a műveleteket (például a lemezre írást), amelyeket egy Java applet végre tud hajtani.
- A JavaScript olyan, HTML alapú parancsnyelv, amely a nem programozó Web-tervezők számára is lehetővé teszi, hogy használhassák a Java számos képességét.
- A JavaScript és más parancsnyelvek, mint például a Perl közötti elsődleges különbség az, hogy a JavaScript utasításait a böngésző, és nem a kiszolgálón található program hajtja végre.

A JAVA NYELVVEL FOGLALKOZÓ FONTOSABB HELYEK

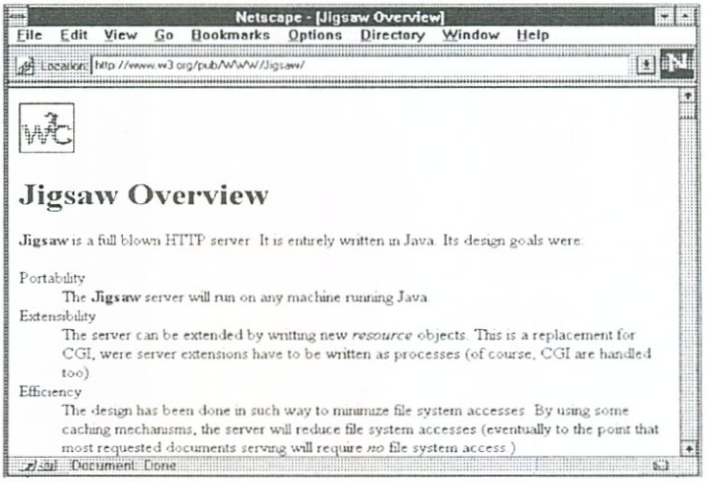
A következő Web-helyek segíthetnek abban, hogy részletesen olvashassunk a Java nyelv specifikációjáról, további speciális információkat kaphassunk a JavaScript, a HotJava, a Visual J++ nyelvekről és sok minden másról. A helyeket kiindulási pontként is használhatjuk a Web felfedezéséhez.

Workshop Source Code



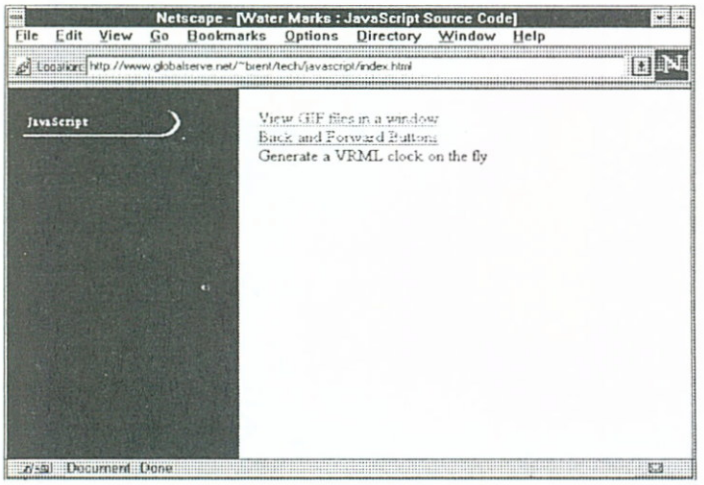
http://users.aol.com/thingtone/workshop/source.htm

Jigsaw Overview



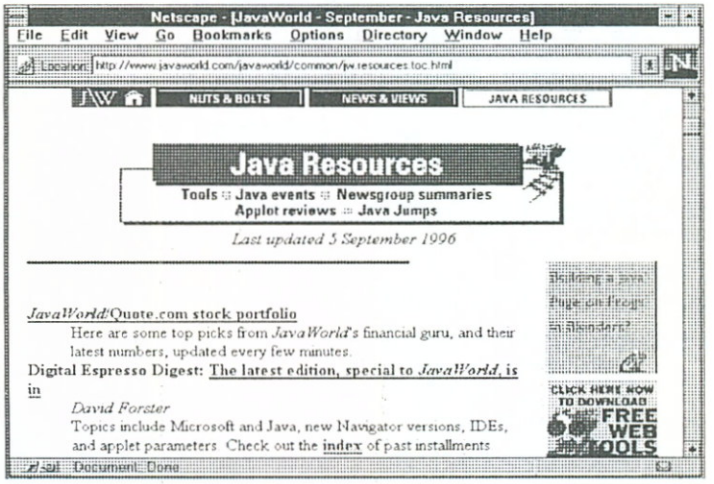
http://www.w3.org/pub/WWW/Jigsaw/

Water Marks: JavaScript Source Code



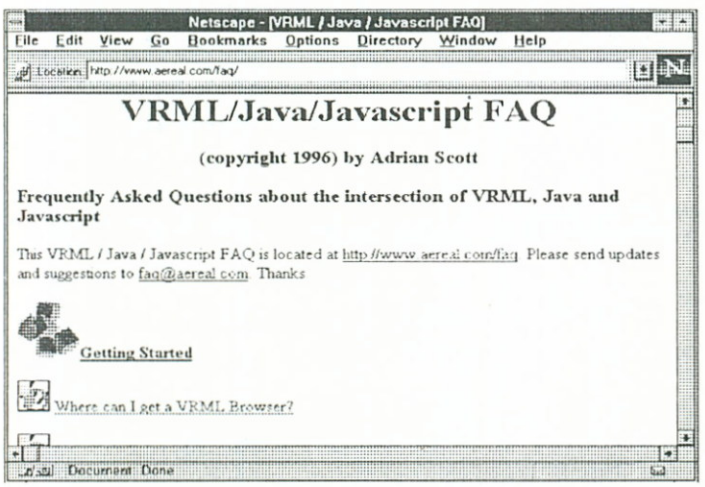
http://www.globalserve.net/~brent/tech/javascript/index.html

Java World – Java Resources



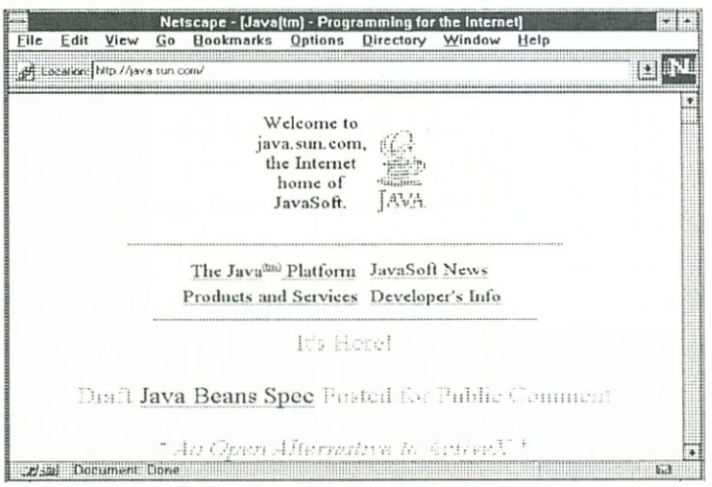
http://www.javaworld.com/javaworld/common/jw.resources.toc.html

VRML/Java/Javascript FAQ



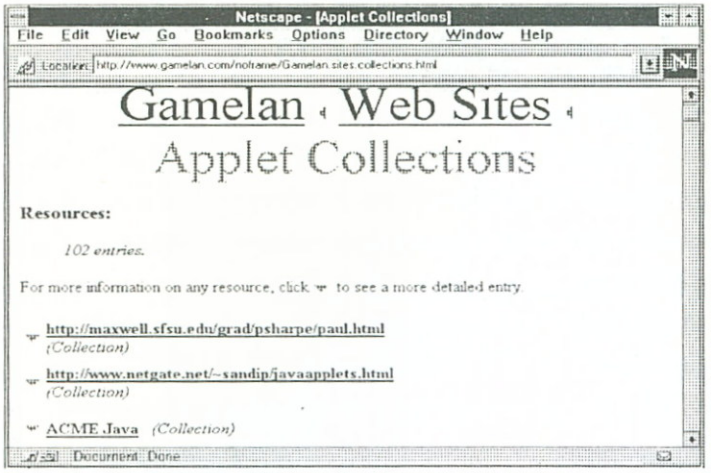
http://www.aereal.com/faq/

Java: Programming for the Internet



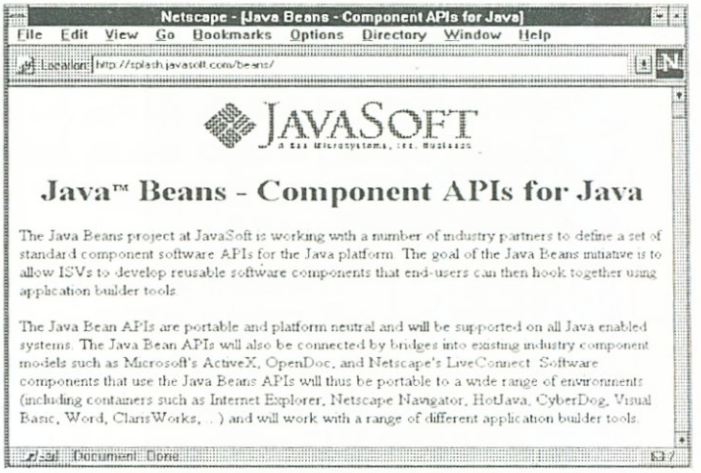
http://java.sun.com/

Applet Collections



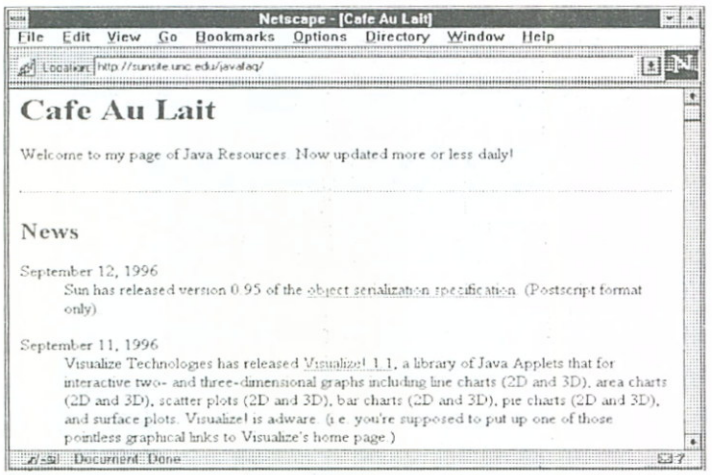
http://www.gamelan.com/noframe/Gamelan.sites.collections.html

Java Beans – Component APIs



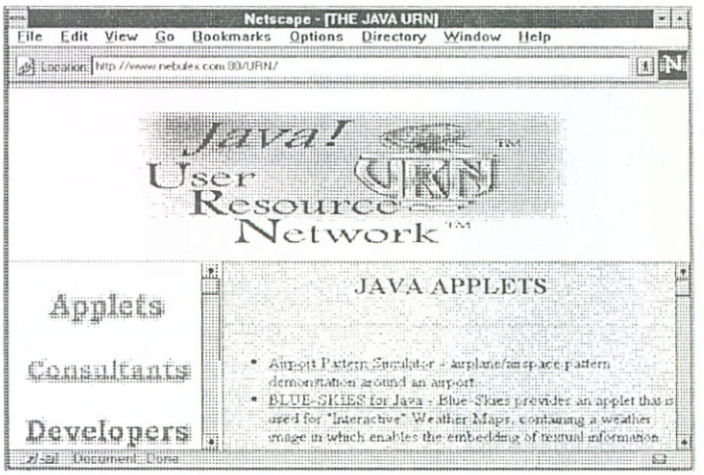
http://splash.javasoft.com/beans/

Cafe Au Lait (Java Resources)



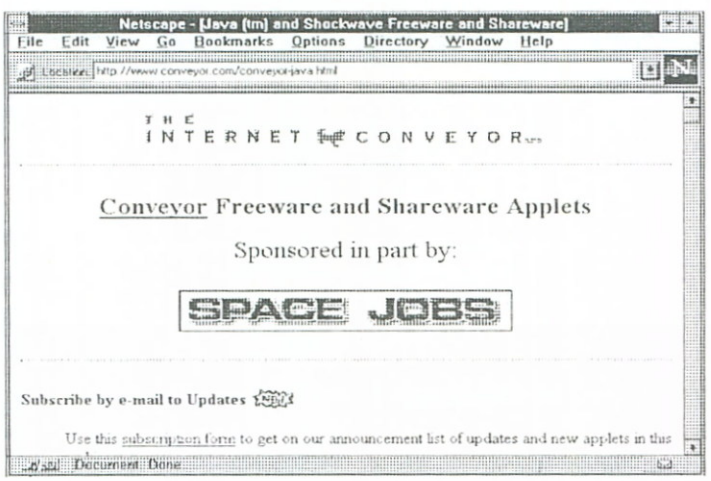
http://sunsite.unc.edu/javafaq/

The Java URN



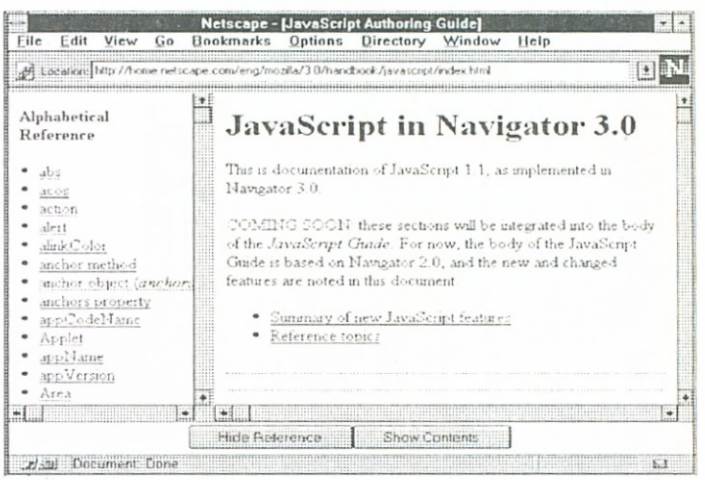
http://www.nebulex.com:80/URN/

Java Freeware and Shareware



http://www.conveyor.com/conveyor-java.html

JavaScript Authoring Guide



http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html

15. FEJEZET

A WEB PROGRAMOZÁSA VBSCRIPT ÉS ACTIVEX SEGÍTSÉGÉVEL

Aki figyelemmel kísérte a „böngészők csatáját", az tudhatja, hogy eleinte a Netscape Navigator volt az egyetlen olyan böngésző, amelyet a felhasználók a Weben való szörfözéshez használtak. A Netscape valóban annyira egyeduralkodó volt, és annyira elterjedt, hogy szakmai megfigyelők már arról cikkeztek, hogy a Microsoft „elaludt a volánnál", és a szoftverek területén hátrányba kerül a Netscape-pel szemben. A Microsoft válaszul erre a kihívásra elkészítette a Microsoft Internet Explorer névre keresztelt böngészőjét, amely képességeit tekintve megfelelt, helyenként pedig meg is haladta a Netscape hasonló tulajdonságait.

Ezzel egy időben a Microsoftnak egy másik kihívással is szembe kellett néznie: a Sun Microsystems által kifejlesztett Java programozási nyelvvel. Többek között annak köszönhetően, hogy a Sun ingyenesen terjesztette (és terjeszti ma is) a saját megfogalmazása szerint „a Web leghatékonyabb programozási nyelvét", rövid idő alatt tekintélyes pozícióra tett szert a programozási nyelvek piacán. Most is elhangzottak olyan vélekedések, miszerint a Sun „állva" hagyta a Microsoftot, és rövidesen uralni fogja piacnak ezt a szegmensét. Válaszul a Microsoft elkészítette a Visual J++ nyelvet (erről a 14. fejezetben volt szó) és a VBScript nyelvet. Ez utóbbinak az a fő erőssége, hogy használni tudja az OLE (objektumok csatolása és beágyazása) objektumokat, vagy újabb nevükön az ActiveX vezérlőelemeket.

E fejezet célja, hogy az itt található ismeretek birtokában azonnal használni tudjuk a VBScript nyelvet és az ActiveX vezérlőelemeket. A fejezet továbbá olyan Web-címeket is felsorol, ahol más, fontos információkhoz juthatunk. A fejezet végére érve érteni fogjuk az alábbiakat:

- A VBScript a Microsoft által kifejlesztett és a Visual Basic programozási nyelvre épülő parancsnyelv.
- A JavaScript nyelvhez hasonlóan VBScript utasításokat is beágyazhatunk egy HTML fájlba, amelyeket aztán a böngésző végrehajt.
- Jelenleg csak a Microsoft Internet Explorer böngésző érti meg a VBScript nyelvet.
- A VBScript erősségét az adja, hogy képes hozzáférni OLE objektumokhoz (az úgynevezett ActiveX vezérlőelemekhez).
- A VBScript segítségével multimédiás Web-oldalakat készíthetünk (hang, animáció és video).
- A Perlhez hasonlóan VBScript nyelven is készíthetünk CGI-alapú alkalmazásokat.

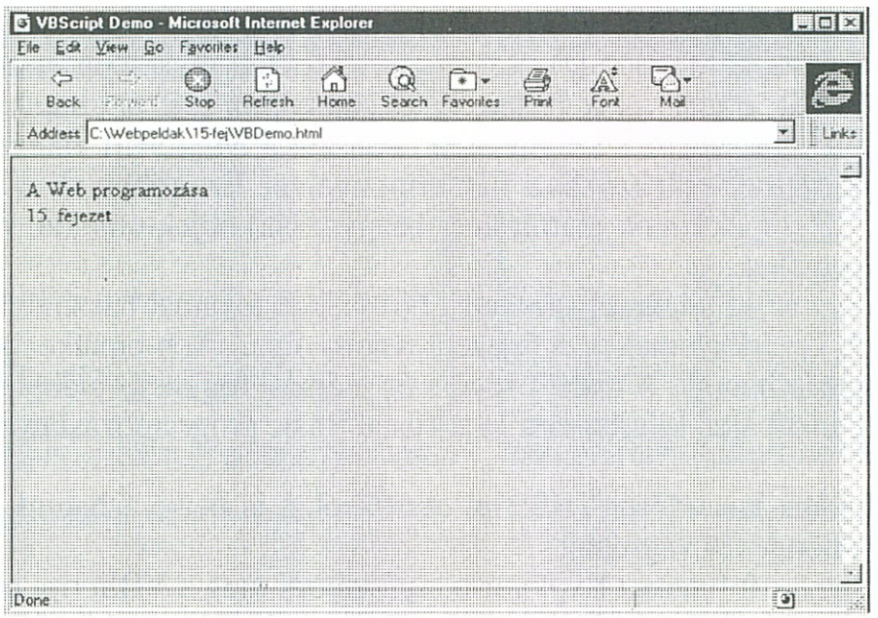
A VBSCRIPT NYELV ÁTTEKINTÉSE

A JavaScripthez és a Perlhez hasonlóan a VBScript is parancsnyelv. A VBScript nyelven megírt script fájlokat a böngésző futtatja (ebben ugyancsak megegyezik a JavaScripttel, ugyanakkor eltér a Perltől – a Perl nyelvű fájlok ugyanis a kiszolgálón futnak). Jelenleg csak a Microsoft böngészője, az Internet Explorer érti meg a VBScript fájlokat. Mint látni fogjuk, a VBScript a Visual Basic programozási nyelv részhalmaza, ami egyúttal azt is jelenti, hogy a VBScript nem tartalmazza a Visual Basic nyelv valamennyi képességét.

VBScript nyelven úgy készíthetünk script fájlt, hogy a megfelelő VBScript utasításokat egyszerűen beágyazzuk egy HTML dokumentumba. A következő, *VBDemo.html* nevű fájl például a HTML <SCRIPT> elemét használja arra, hogy néhány egyszerű VBSript utasítást ágyazzon be a fájlba:

```
<HTML>
<HEAD><TITLE>VBScript Demo</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

document.write("A Web programozása<BR>")
document.write("15. fejezet<BR>")

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha az Internet Explorer segítségével beolvassuk ezt a HTML fájlt, akkor a böngésző ablakában a 15.1. ábrán láthatóhoz hasonló jelenik meg:



15.1. ábra. Megjelenítés képernyőn VBScript fájl futtatásával

Megjegyzés: A JavaScripthez hasonlóan a VBScript használatához sincs szükség fordítóprogramra vagy fejlesztőkörnyezetre. VBScript fájlok futtatásához elegendő csak a Microsoft Internet Explorer böngésző. A 4. fejezetben tanultak szerint készítsük el a HTML fájlt, és vegyük bele a megfelelő VBScript utasításokat. A böngésző Address mezőjébe URL-ként adjuk meg a fájl elérési útját **fájl://c:\elérési_út** *alakban. Az Enter lenyomására a böngésző megnyitja a fájlt, és végrehajtja a benne lévő VBScript utasításokat.*

A VBSCRIPT ÉS A JAVASCRIPT ÖSSZEHASONLÍTÁSA

A 14. fejezetben a JavaScript nyelvet arra használtuk, hogy olyan, interaktív script fájlokat írjunk, amelyek bizonyos eseményekre, például egérkattintásra vagy billentyűlenyomásra reagálnak. Néhány korábbi parancsnyelvhez, mint például a Perl vagy a C++, képest a JavaScript egyik előnye, hogy lehetővé teszi a böngésző bizonyos fokú közreműködését egy script

fájl végrehajtásában. A dolog lényege úgy is megfogalmazható, hogy a Perl vagy a C++ script fájlok programokként a kiszolgálón helyezkednek el, és a böngészőnek csak az a dolga, hogy elküldje a kitöltött űrlapot. Ilyen módon a böngésző csak passzív szereplőként vesz részt a script fájl végrehajtásában.

A JavaScript nyelvhez hasonlóan a VBScript is lehetővé teszi, hogy utasításokat adhassunk meg, amelyeket a „VBScript nyelvet értő" böngészők végre tudnak hajtani. Egyszerűbb script fájlok írásához nagyon jól használhatjuk akár a JavaScript, akár a VBScript nyelvet. Bonyolultabb fájlok esetén azonban tapasztalni fogjuk, hogy a VBScript nyelvnek az a képessége, amelynek révén együtt tud működni ActiveX objektumokkal, jelentős előnyhöz juttatja a VBScript nyelven programozókat a JavaScript nyelvű programozókkal szemben. A JavaScript nyelvhez képest a VBScript használatának jelenleg az az árnyoldala, hogy csak nagyon kevés platform támogatja a VBScript fájlokat és az ActiveX objektumokat. Ezért ha olyan feladatunk van, amelyet a Windowson kívüli környezetben kell megoldanunk, akkor be kell érnünk a hordozható JavaScript fájlokkal.

A VBSCRIPT ÉS A PERL ÖSSZEHASONLÍTÁSA

A 12. és a 13. fejezetben láttuk, hogyan hozhatunk létre CGI script fájlokat Perl programozási nyelven. A Perl beépített reguláris kifejezéseinek köszönhetően nagyon jól használható szövegfeldolgozásra. Láttuk, hogy amikor a felhasználó egy egérkattintással elküldi a CGI-alapú űrlapot, akkor a böngésző az űrlap adatait egy, a kiszolgáló rendszerében futó programnak adja át. Ez a program a Perl nyelvű script fájl. Ezt a script fájlt a kiszolgáló futtatja úgy, hogy a parancssorban és a környezeti változókban átadja az adatokat a fájlnak. Miután a script fájl feldolgozta az adatokat, elkészít egy HTML dokumentumot, amit a kiszolgáló visszaküld a böngészőnek. Ebben a modellben az adatok tényleges feldolgozását nem a kiszolgáló és nem a böngésző, hanem egy harmadik résztvevő végzi, „aki" normál esetben a kiszolgáló rendszerében található.

A VBScript nyelv segítségével viszont olyan script fájlokat készíthetünk, amelyek az ügyfélnél végeznek el bizonyos feldolgozásokat. Ennek egyik előnye, hogy csökken a hálózaton folyó tevékenység (miáltal javul a hálózat teljesítménye) és csökken a kiszolgálóra háruló teher (azzal, hogy nem kell üzeneteket küldözgetnie az ügyfél és egy harmadik program között).

Könnyen belátható, hogy az olyan nyelvek, mint a C++, Java, de még a Visual Basic fordítóprogramjainak is tekintélyes mennyiségű szöveget kell feldolgozniuk (amikor elemzik a forráskód sorait). Ezért a fordítóprogramokban van olyan kód, amely meghatározott minták szerint keres szövegegyezőségeket (megkeresi például a nyelv szintaxisának megfelelő kulcsszavakat). Mivel a Microsoftnak olyan szoftverei vannak, amelyek a szövegek feldolgozásához bőségesen használják a reguláris kifejezéseket, joggal várható, hogy a Microsoft beépíti ezeket a szövegfeldolgozó képességeket a VBScript nyelvbe, hogy jobb pozíciót foglaljon el a Perl nyelvvel folytatott versenyben.

WEB-OLDALI ÉS KISZOLGÁLÓ-OLDALI SCRIPT FÁJLOK

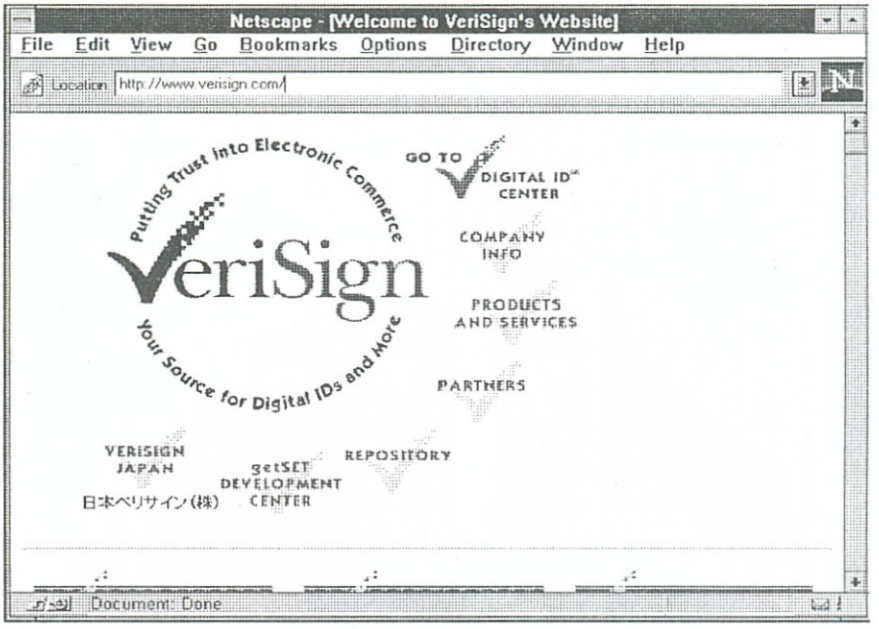
Amint tudjuk, egy script fájl nem más, mint programutasítások listája, amelyben az utasítások valamilyen meghatározott feladatot végeznek el. Amikor más programozókkal script fájlokról beszélünk, akkor az egyik legelső tisztázandó kérdés, hogy a szóban forgó script fájl *Web-oldali* vagy *kiszolgáló-oldali*-e. A 12. és a 13. fejezetben a Perl nyelvvel kapcsolatban olvasottakra visszaemlékezve tudjuk, hogy a Perl script fájlok a kiszolgáló rendszerében helyezkednek el, vagyis ezek kiszolgáló-oldali script fájlok. Amikor a felhasználó meghív egy kiszolgáló-oldali script fájlt, akkor a kiszolgáló számítógépe futtatja azt, és a script fájl valamilyen módon kapcsolatba lép a felhasználóval. Ezzel ellentétben egy Web-oldali olyan script fájl, amelynek az utasításai azokban a HTML bejegyzésekben helyezkednek el, amelyek összeállítják a Web-oldalt. Amikor a felhasználó egy Web-oldali script fájlt hív meg, akkor a böngésző futtatja a fájlt.

A VBSCRIPT ÉS A HÁLÓZATI BIZTONSÁG

A 14. fejezetben olvastunk arról, hogy a Java kifejlesztői a különböző tervezési szempontok mérlegelésekor nagyon nagy hangsúlyt helyeztek a biztonságra. Láttuk, hogy a Java fejlesztői felelősségteljes döntéseket hoztak arra vonatkozóan, hogy milyen képességeket építsenek be a Javába, és milyeneket hagyjanak ki belőle. A Java appletjei biztonságosak, mert a fejlesztők eléggé erősen korlátozták a nyelv lehetőségeit.

A Microsoft fejlesztőmérnökeit a VBScript megtervezése során az a szándék vezette, hogy ez a nyelv is legalább annyi biztonságot nyújtson, mint a JavaScript. Eleinte a VBScript tervezői tudták is tartani magukat a Javának a „ha kétséges, akkor hagyjuk ki" alapgondolatára épülő biztonsági modelljéhez, miszerint ha egy elgondolás potenciális veszélyt hordozhat magában (mint például az, hogy a script fájlok legyenek képesek I/O műveletek végzésére), akkor azt egyszerűen kihagyják a nyelvből. Azt is tudjuk azonban, hogy a VBScript tudásának jó részét annak köszönheti, hogy képes az ActiveX objektumok integrálására. Mint látni fogjuk, egy ActiveX objektum egy OLE vezérlőelem, amely a saját fájljában helyezkedik el. Egy ActiveX objektum használatához a script fájlnak a fájlból le kell töltenie a vezérlőelemet a böngésző merevlemezére, majd futtatnia kell az objektum metódusait. Minden olyan esetben, amikor egy biztonságos program (mint amilyen egy VBScript fájl) valamilyen „külső" programot futtat, fennáll annak a veszélye, hogy a külső programban lévő objektumok nem biztonságosak, netán rosszindulatúak!

Az ilyen veszélyek elkerülése céljából a kereskedelmi forgalomban lévő ActiveX objektumokat a jövőben el kell látni egy külső szervezet által kibocsátott igazolással, ami tanúsítja, hogy az objektum kódja biztonságos és „jól nevelt". Az igazolás alapján a kód egy *digitális aláírást* (digital signature) kap, ami mintegy „pecsétként" engedélyezi forgalomba hozatalát. Így a böngésző már a külső objektum meghívását megelőzően ellenőrizheti, hogy rendelkezik-e az objektum ilyen aláírással és ettől függően folytathatja a munkáját. A digitális aláírások használatáról a *http://www.verisign.com/* Web-helyen kaphatunk további információkat (lásd a 15.2. ábrát).

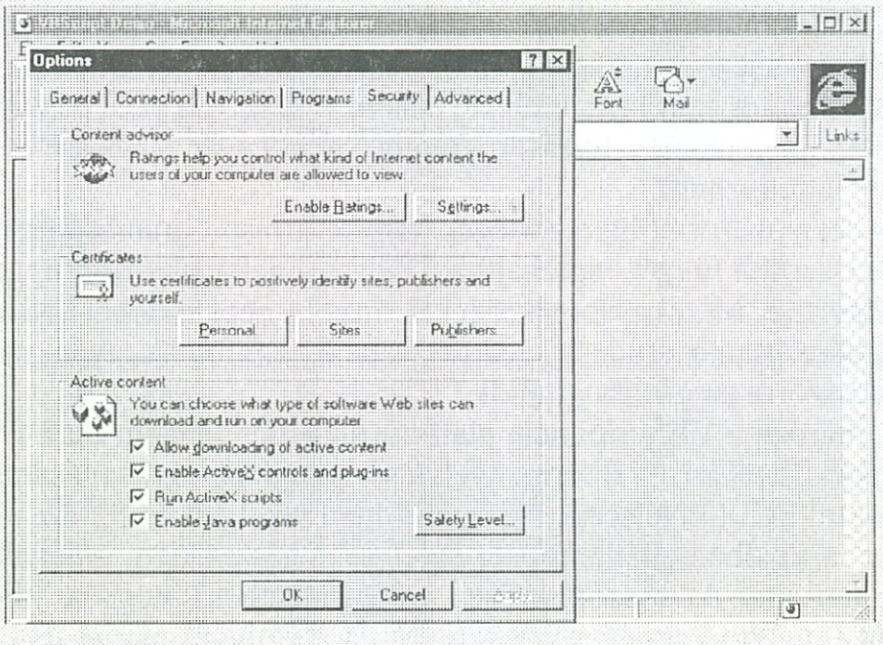


15.2. ábra. Információk a digitális aláírásokról

A MICROSOFT INTERNET EXPLORER BÖNGÉSZŐ BIZTONSÁGÁNAK BEÁLLÍTÁSA

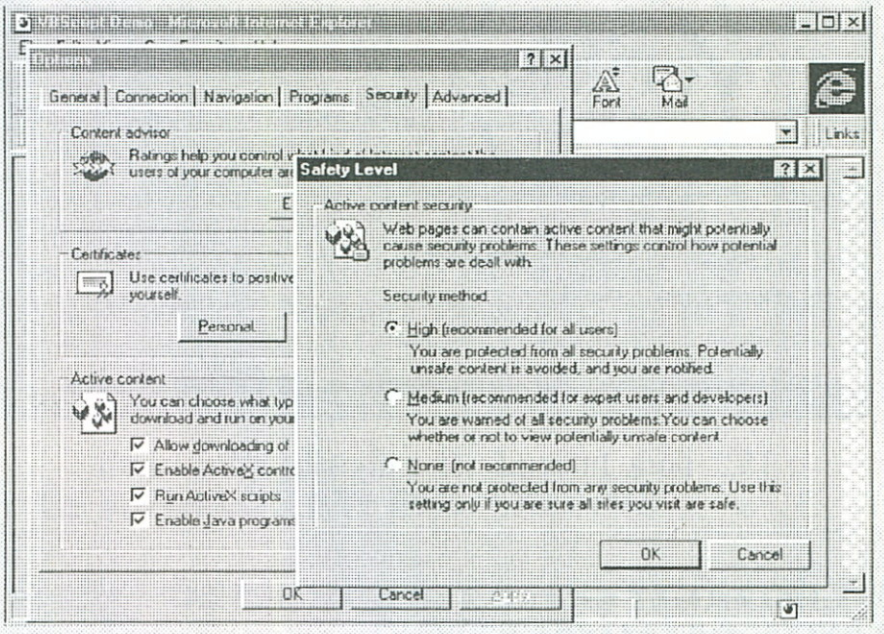
Ha a Microsoft Internet Explorer böngészőt használjuk, akkor egyéni igényeink szerint állíthatjuk be a böngésző biztonságát olyan tekintetben, hogy támogassa-e az ActiveX objektumokat, igényelje-e a digitális aláírásokat, támogasson-e speciális aláírásokat stb. A böngésző biztonsági beállításait az alábbiak szerint végezhetjük el:

1. A böngésző View (Nézet) menüjében válasszuk az Options (Beállítások) parancsot. A böngésző ekkor megjeleníti az Options (Beállítások) párbeszédablakot.
2. Az Options párbeszédablakban válasszuk a Security (Biztonság) fület. A böngésző megjeleníti a 15.3. ábrán látható Security űrlapot.



15.3. ábra. A Microsoft Internet Explorer Options párbeszédablakának Security űrlapja

3. A Security űrlapon jelöljük be a kívánt beállításokat, majd kattintsunk a Safety Level (Biztonsági szint) gombra. A böngésző megjeleníti a 15.4. ábrán látható Safety Level párbeszédablakot, amelyben beállíthatjuk a kívánt biztonsági szintet.



15.4. ábra. A Safety Level párbeszédablak

A VBSCRIPT NYELV NEM ESZKÖZ- ÉS PLATFORMFÜGGETLEN

A 14. fejezetben megismerkedtünk a Java programozási nyelvvel, amely részben platformfüggetlenségének köszönhetően rendkívüli népszerűségre tett szert. Mint láttuk, a Java platformfüggetlensége lehetővé teszi, hogy a felhasználók ugyanazt az appletet több különböző rendszerben is futtassák anélkül, hogy a programozóknak mindegyik platformhoz külön-külön le kellene fordítaniuk a forráskódot. Másként megközelítve ez azt jelenti, hogy ha létrehozunk egy Java appletet, akkor a felhasználók ugyanezt az appletet Windows, Mac vagy akár Unix környezetben is tudják futtatni. Ha az adott feladatot nem Java nyelven, hanem valamilyen más, például C++ programozási nyelven írtuk volna meg, akkor a forrásprogramból mindegyik rendszerhez más és más végrehajtható fájlt kellene létrehoznunk.

A Java úgy éri el ezt a platformfüggetlenséget, hogy a forrásprogramból virtuális gépi kódot állít elő (nem pedig processzortól függő, végrehajtható fájlt). Ennek a platformfüggetlenségnek a hátránya a sebesség csökkenése. Mivel a böngészőnek a virtuális gépi kódot át kell alakítania a mindenkori processzor számára érthető kóddá, a Java programok nem futnak olyan gyorsan, mint a kifejezetten az adott processzorra megírt programok.

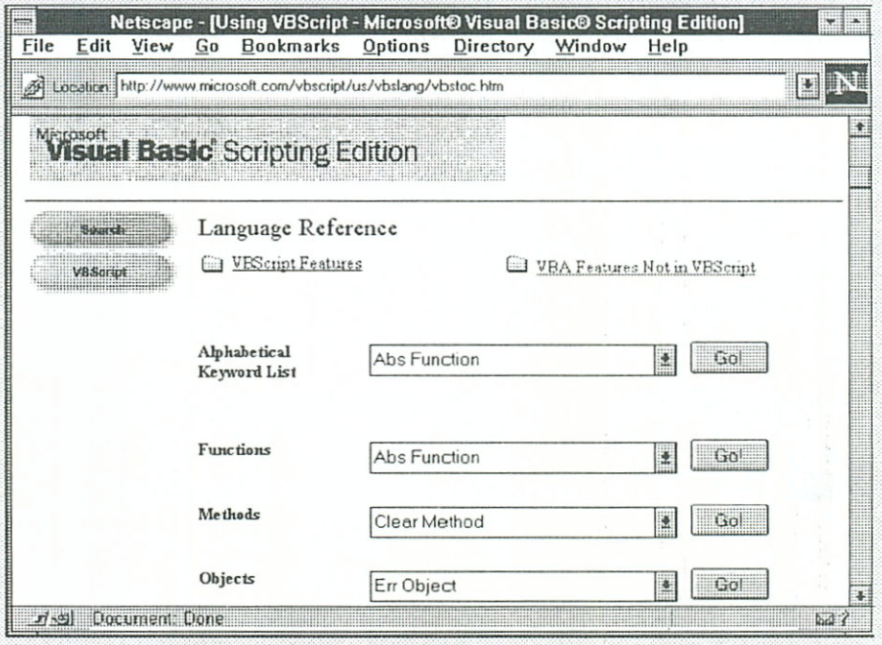
A Java nyelvtől eltérően a VBScript nem platformfüggetlen, hanem csak Windows alapú rendszerekben használható. Továbbá a VBScript nyelvű script fájlok végrehajtásához a Microsoft Internet Explorert is futtatni kell (a Netscape Navigator nem alkalmas erre). Az elképzelhető, hogy a Netscape Navigatort a jövőben ellátják olyan szoftverkiegészítőkkel, amelyek alkalmassá teszik VBScript fájlok futtatására. Az viszont erősen valószínűsíthető, hogy a VBScript – legalábbis belátható időn belül – a Windows környezeten (95 és NT) belül marad. Ha viszont figyelembe vesszük azt a tényt, hogy a felhasználók döntő többsége Windows rendszerben dolgozik, akkor az a körülmény, hogy a VBScript fájlok ezt a környezetet igénylik, és nem platformfüggetlenek, iga-

zából nem tekinthető szigorú megszorításnak. A Windows környezet teszi lehetővé egyébként még azt is, hogy az ehhez szorosan kötődő ActiveX objektumok révén növelhető legyen a teljesítmény.

A VBSCRIPT NYELV SPECIFIKÁCIÓJÁNAK LETÖLTÉSE

A VBScript nyelv számos, más programozási nyelvekben is megtalálható szerkezetet tartalmaz. Az ActiveX objektumok hasznosítása révén pedig a VBScript nyelv kerek egésszé tehető.

A fejezet célja, hogy bemutassa, hogyan készíthetők és futtathatók VBScript fájlok és ActiveX objektumok. Ennek során megismerkedünk azokkal az alapdolgokkal, amelyekre egyszerűbb VBScript appletek írásához szükségünk lesz. Amint egyre összetettebbé válnak az appleteink, olyan témakörökkel is foglalkoznunk kell, mint az ActiveX objektumok, a hálózat biztonsága, multimédiás támogatás stb. Ezekben a témakörökben az egyik legjobb (és ingyenes) információforrás a VBScript Reference, amely a Web *http://www.microsoft.com/vbscript/us/vbslang/vbstoc.htm* címén található (lásd a 15.5. ábrát).



15.5. ábra. A VBScript Language Reference

A <SCRIPT> ELEM

Az 5. fejezetben röviden említettük, hogy a HTML oldalak tervezői a <SCRIPT> elem segítségével adják meg a script utasításokat tartalmazó script fájlok URL-jét. Az alábbi bejegyzés például egy VBScript fájl URL-jét adja meg:

```
<SCRIPT LANGUAGE= "VBSript"
SRC="http:\www.jamsa.com\vmilyenscript.VBS"></SCRIPT>
```

Ehhez hasonlóan a következő <SCRIPT> elem határozza meg azokat a VBScript utasításokat, amelyeket a böngészőnek végre kell hajtania:

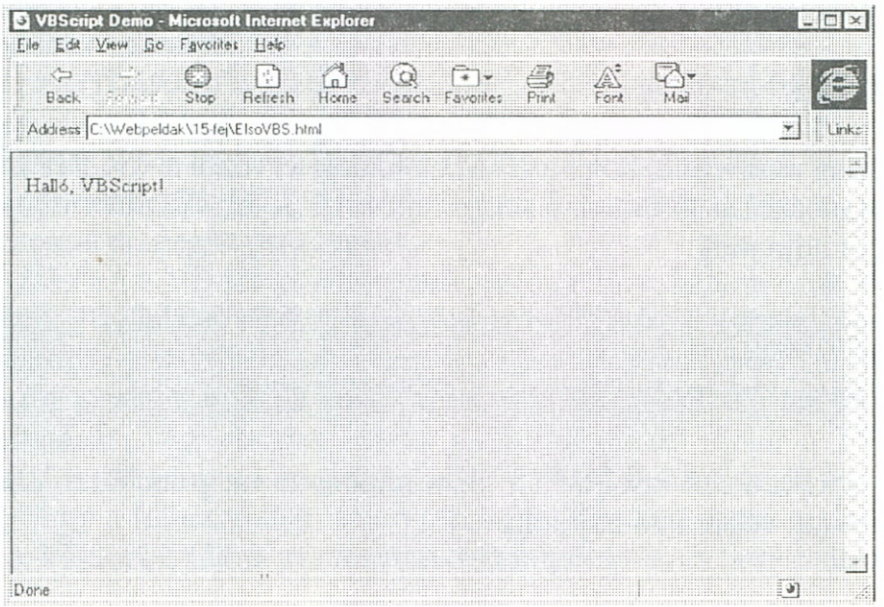
```
<HTML>
<HEAD><TITLE>VBScript Demo</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

document.write("Halló, VBScript!<BR>")

// A kód elrejtésének vége ->
</SCRIPT> </BODY></HTML>
```

AZ ELSŐ VBSCRIPT APPLETÜNK

A JavaScripttel azonos módon használhatjuk a VBScript nyelvet, hogy script utasításokat helyezzünk el egy HTML dokumentumba. Az alábbi, *ElsőVBS.html* fájl egy egyszerű VBScript fájlt állít elő:

```
<HTML>
<HEAD><TITLE>Elso VBScript</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

document.write("Halló, VBScript!")

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a fenti dokumentumot a Microsoft Internet Explorer segítségével beolvassuk, akkor a böngésző ablakában megjelenik a *Halló, VBScript!* üzenet a 15.6. ábrán látható módon:



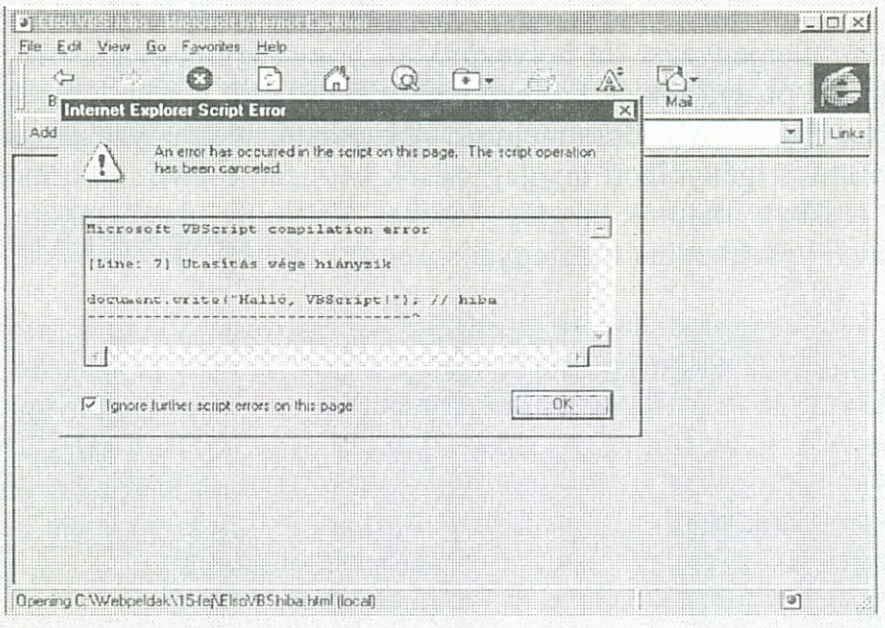
15.6. ábra. A *Halló, VBScript!* üzenet megjelenítése a Microsoft Internet Explorer ablakában

A VBSCRIPT SZINTAKTIKAI HIBÁI

Amikor VBScript nyelvű script fájlokat írunk, akkor minden más programozási nyelvhez hasonlóan ennek során is be kell tartanunk a nyelv meghatározott szabályait (szintaxisát). Ha egy script fájl futtatása során a böngésző hibával találkozik, akkor megjelenít egy párbeszédablakot, amelyben leírja a hiba jellegét. Tegyük fel, hogy a 14. fejezet egyik JavaScript nyelvű példáját követve tévedésből pontosvesszőt tettünk a *document.write* utasítás végére az alábbi módon:

```
<HTML>
<HEAD><TITLE>Elso VBScript</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

document.write("Halló, VBScript!");// hiba

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Amikor a Microsoft Internet Explorer segítségével beolvassuk ezeket az utasításokat, akkor a böngésző egy párbeszédablakban a 15.7. ábrán látható hibaüzenetet jeleníti meg. Ilyen esetben a HTML dokumentum szerkesztésével javítsuk ki a hibát.



15.7. ábra. Egy szintaktikai hiba párbeszédablaka

VBSCRIPT PARANCSOK MEGJELENÍTÉSÉNEK KIKAPCSOLÁSA HTML MEGJEGYZÉSEKKEL

Mielőtt részleteiben megvizsgálnánk a VBScript nyelvet, tisztában kell lennünk azzal, hogy a tervezők hogyan használják a HTML megjegyzéseket VBScript utasítások közrefogásához. A 4. feje-

zetből emlékezhetünk arra, hogy a tervezők egy HTML dokumentumban a következő formátumot használják a megjegyzésekhez:

```
<!- Ez egy megjegyzés
A megjegyzés vége ->
```

Amint majd látni fogjuk, a tervezők egy <SCRIPT> elemen belül a VBScript utasításokat HTML megjegyzésekkel veszik közre az alábbi módon:

```
<SCRIPT>
<!- A megjegyzés kezdete, de a megjegyzések itt nem fejeződnek be

VBScript utasítások helye

Vége a megjegyzéseknek ->
</SCRIPT>
```

Próbáljuk meg beolvasni a böngészőnkkel ezeket a HTML megjegyzéseket! Ha az utasításokat, például *<SCRIPT LANGUAGE="VBScript">*, olyan böngészővel olvassuk be, amelyik nem érti a VBScript nyelvet, akkor a böngésző ablakában nem jelenik meg semmi.

A VBSCRIPT MEGJEGYZÉSEI

A VBScript kódokban megjegyzéseket helyezünk el, hogy megmagyarázzuk az egyes utasítások célját. Ilyen módon más programozók könnyebben megérthetik a szándékainkat. Amint tudjuk, a VBScript utasításokat egy HTML dokumentum belsejében, megjegyzésekként helyezzük el, de maga ez a dokumentum is igényelheti megjegyzések beírását. HTML dokumentumba a <!– *Megjegyzés szövege* –> alakban írhatunk megjegyzéseket. VBScript utasításokon belül egy egyszeres aposztróf (') beírásával készíthetünk megjegyzéseket az alábbiak szerint:

```
<HTML>
<HEAD><TITLE>Megjegyzések VBScript</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése
DIM pontszamok_atlaga                    ' Átlagos eredmény
DIM index                                ' Indextömb
DIM pontszam(5)                          ' Pontszámok
DIM sum                                  ' Pontszámok összege

sum = 0                                  ' Pontszámok összegének nullázása

' Pontszámok bekérése a felhasználótól
For index = 0 to 4
 pontszam(index) = InputBox("Írja be a pontszámot " & index + 1)
Next
```

```
' A pontszámok összegzése
For index = 0 to 4
 sum = sum + pontszam(index)
Next

pontszamok_atlaga = sum / 5
document.write("A pontszámok átlaga: " & pontszamok_atlaga)

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

A VBSCRIPT KARAKTERLÁNCAI

Amint tudjuk, egy karakterlánc 0 vagy több ASCII karakterből áll. A VBScript nyelvben a karakterláncokat dupla idézőjelek közé kell tenni:

```
VmilyenLanc = "Halló, világ!";
MasikLanc = "Halló, VBScript!";
```

Mint majd látni fogjuk, a konkatenáló (&) operátor segítségével egymáshoz fűzhetünk két karakterláncot úgy, hogy a kettőből egy lesz:

```
Uzenet = "Halló, " & "VBScript!";
```

A VBSCRIPT VÁLTOZÓI

Más programozási nyelvekhez hasonlóan a VBScript nyelven megírt script fájlok is változókat használnak az információk tárolására. Hasonlóképpen a script fájlok általában a változók típusát is megadják. Egy script fájlon belül mindegyik változónak saját, a többitől különböző névvel kell rendelkeznie. A VBScript nyelvű változóknak betűvel kell kezdődniük. Egy VBScript programon belül a DIM utasítással deklarálhatunk változókat, megadva a változó nevét. A böngésző ezzel a névvel azonosítja a változót egy fájlon belül. Az alábbi utasítások különböző változókat deklarálnak:

```
DIM Szamlalo, Napok, Evek
DIM KedvencEtel
DIM x, y, z
```

Miután deklaráltuk a változókat, az értékadó operátor (=) segítségével kezdeti értékeket rendelhetünk hozzájuk (inicializálás):

```
Szamlalo = 1
Napok = 7
Evek = 3
KedvencEtel = "Csokoládé"
x = 1
y = 4
z = 2
```

A VBSCRIPT PROGRAMOK NEM ÉRZÉKENYEK A KIS- ÉS NAGYBETŰKRE

A Java és a JavaScript nyelvtől eltérően a VBScript nem érzékeny a kis- és a nagybetűs írásmódra. A VBScript nyelv például az alábbi két utasítást azonosnak tekinti:

```
document.write("Halló, világ!")
DOCUMENT.WRITE("Halló, világ!")
```

Ugyanígy, a VBScript az alábbi két változónevet azonosnak tekinti:

```
konyvCim = "A Web programozása"
KONYVCIM = "A Web programozása"
```

A VBSCRIPT VÁLTOZÓTÍPUSAI

Más programokhoz hasonlóan a VBScript nyelvű programok (appletek) is változókban tárolják az értékeket a végrehajtásuk során. Mindegyik változónak meghatározott típusúnak kell lennie, mint például *int* (egész szám), *single* (lebegőpontos szám) stb. Általánosságban egy változó típusa egyrészt azt az értékkészletet határozza meg, amelynek elemeit az illető változó tárolni tudja (az *integer* típusú változók például egész számokat tudnak tárolni), másrészt azokat a műveleteket, amelyeket a program a változón végre tud hajtani (például a program két számot össze tud szorozni, de két karakterlánc szorzása nem értelmezhető). A 15.1. táblázat a VBScript változótípusait sorolja fel:

Változó típusa	Tárolható érték
boolean	igaz (true) vagy hamis (false) érték
byte	a –128 … +127 tartományba eső számok
date	dátum- és időértékek
double	az 1.7×10^{-308}...1.7×10^{308} tartományba eső számok
int	a –32,768 … 32,767 tartományba eső számok
long	a –2,147,483,648 … 2,147,483,647 tartományba eső számok
single	a -3.4×10^{-38} … 3.4×10^{38} tartományba eső számok
string	alfanumerikus értékek
unsigned	a 0 … 65,535 tartományba eső számok

15.1. táblázat. A VBScript változótípusai

NÉVADÁSI SZOKÁSOK A VBSCRIPT NYELVBEN

Amikor egy script fájlban neveket rendelünk a változókhoz, a VBScript standard előtagjait használva kifejezőbbé, és más programozók számára könnyebben érthetővé tehetjük a változóneveket. Ezekkel a három karakterből álló előtagokkal megadhatjuk az illető változó típusát. Így például az olyan változó neve elé, amelyik karakterláncot tartalmaz, az *str* előtagot írhatjuk az alábbiak szerint:

```
strKonyvCim = "A Web programozása"
```

Ehhez hasonlóan, ha egy változó dupla pontosságú lebegőpontos számot tárol, akkor a változó neve elé a *dbl* előtagot írhatjuk:

```
dblKonyvAr = 49.95
```

A 15.2. táblázat a VBScript nyelvben használatos változónév-előtagokat sorolja fel.

Előtag	A változó típusa
bln	boolean
byt	byte
dtm	dátum vagy idő
dbl	duplapontosságú (double)
err	hibaállapot
int	integer (egész szám)
lng	long integer
obj	objektum
sng	egyszeres pontosságú (single)
str	karakterlánc (string)
v	változó

15.2. táblázat. A VBScript nyelv változónév-előtagjai

VÁLTOZÓK IMPLICIT ÉS EXPLICIT DEKLARÁLÁSA

Amikor egy script fájlban a DIM utasítással deklarálunk egy változót, csak a nevét kell megadnunk, a típusát nem. Amikor a VBScript fordítóprogram találkozik egy DIM utasítással, nem foglal helyet a memóriában a változó számára. A fordítóprogram (csakúgy, mint azok a programozók, akik olvassák a kódunkat) egyszerűen csak tudomásul veszi, hogy szándékunkban áll ilyen nevű változó használata. A fordítóprogram csak akkor foglal helyet a memóriában a változó számára, amikor azt először használja. Attól függően, hogy a script fájl a változót az adott környezetben hogyan használja, a fordítóprogram megállapítja a változó típusát (egész számú, dupla pontosságú, karakterlánc stb.), és ennek megfelelő memóriaterületet foglal le a számára.

Tartsuk magunkat ahhoz a jó programozási szokáshoz, hogy a kód elején DIM utasításokkal deklaráljuk valamennyi változónk nevét. Ilyen módon más programozók is könnyebben

megérthetik az egyes változók használatát. Ha DIM utasítással deklarálunk egy változót, akkor *explicit* deklarálásról beszélünk (a fordítóprogramnak és más programozóknak kifejezetten a tudomásukra hozzuk, hogy az illető változót használni akarjuk). Sajnos a VBScript nyelv az *implicit* deklarálást is támogatja, azaz egy kódon belül bárhol kezdhetünk használni egy változót anélkül, hogy azt a program elején egy DIM utasítással deklaráltuk volna.

Sok programozó kényelmesnek tartja az implicit deklarálás lehetőségét, mondván, hogy „ha majd szükségem lesz egy változóra, akkor létrehozom és használom". Az implicit deklarálásokkal kapcsolatban az a probléma, hogy olyan hibákat okozhatnak, amelyek oka nehezen felderíthető. Tekintsük például az alábbi, *VBVégtelen.html* nevű HTML dokumentumot, amelyben a *While* utasítás végtelen ciklust hoz létre:

```
<HTML>
<HEAD><TITLE>Vegtelen ciklus</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

Dim Szamlalo
Szamlalo = 1

While (Szamlalo <= 10)
  document.write(Szamlalo & " ")
  Szamlal = Szamlalo + 1
Wend

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Amint sejthetjük, a script fájl célja az 1 és 10 közötti számok megjelenítése egy HTML dokumentumban. Ha viszont futtatnánk ezt a fájlt, akkor azt látnánk, hogy az végtelen ciklusban futna, és állandóan az 1 értéket jelenítené meg. Ha közelebbről szemügyre vesszük a *While* utasítást, akkor láthatjuk, hogy a ciklus két változót használ: az egyiknek *Szamlalo*, a másiknak *Szamlal* a neve (az utóbbi végéről hiányzik az „o" betű). Mivel a kód elején csak a *Szamlalo* változó lett deklarálva, a VBScript a – nyilvánvalóan elírásból származó – *Szamlal* változó előfordulásakor létrehozza az ilyen nevű implicit változót.

Ha el akarjuk kerülni, hogy az implicit változók ilyen hibákat okozhassanak, akkor kapcsoljuk ki a VBScript nyelvnek ezt a képességét. Ehhez mindössze az alábbi sort kell beírnunk a script fájl elejére:

```
Option Explicit
```

Ha ilyen beállítással futtatjuk le az előző, *VBVégtelen* nevű fájlt, akkor a VBScript fordítóprogram szintaktikai hibát jelez, amikor az előzetesen nem deklarált *Szamlal* nevű változóval találkozik.

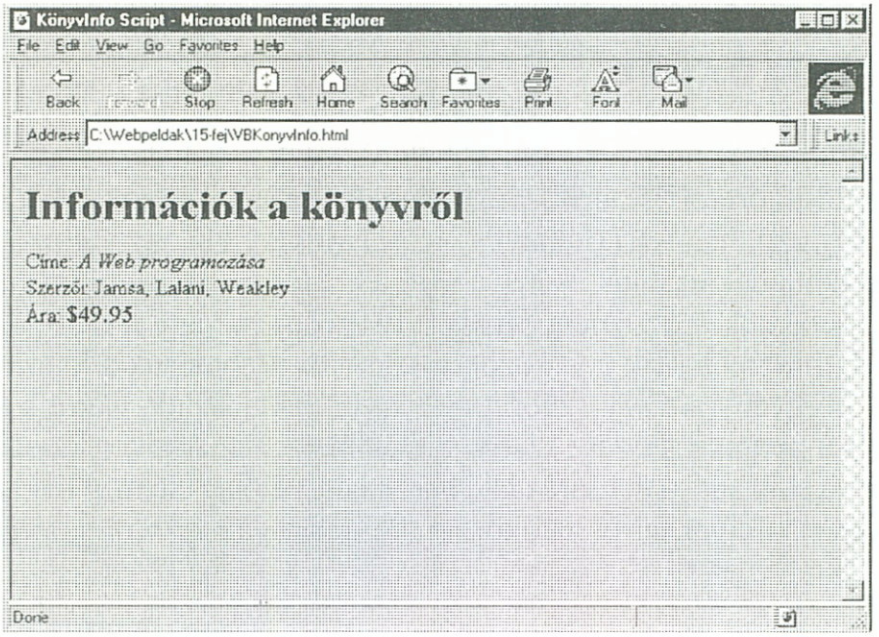
A VÁLTOZÓK HATÓKÖRE

Valamely változó *hatóköre* egy programon belül azokat a helyeket határozza meg, ahol a változó neve jelentést hordoz (ismerik a függvények és a szubrutinok). A VBScript a hatókörök kétféle típusát használja: *helyi* (local) és *script* hatókört. Ha egy változót egy függvényen vagy szubrutinon belül deklarálunk, akkor a változó hatóköre arra a függvényre vonatkozóan helyi lesz. Más szavakkal ez azt jelenti, hogy a függvényen vagy a szubrutinon kívüli programutasítások nem használhatják a változót, mert nincs tudomásuk a létezéséről. Ha viszont egy változót egy függvényen vagy szubrutinon kívül deklarálunk, akkor a változónak script lesz a hatóköre, és a program tetszése szerint használhatja azt. Amikor létrehozunk változókat, akkor a változó neve előtt álló, a típusára utaló hárombetűs előtag elé még beírhatjuk az *s* betűt és az aláhúzás karaktert, amivel script hatókört rendelünk a változóhoz: *s_intOldalSzám*.

EGYSZERŰ KIÍRATÁS LÉTREHOZÁSA VBSCRIPT NYELVEN

Amint tudjuk, a HTML dokumentumok egyrészt szöveget, másrészt olyan nyelvi elemeket tartalmaznak, amelyek meghatározzák, hogy a böngésző miképpen jelenítse meg az adott szöveget vagy grafikát. A script fájlok a VBScript *document.write* függvénye segítségével a kimenetüket beírják az aktuális dokumentumba, amit aztán a böngésző megjelenít a képernyőn. A most következő, *VBKönyvInfo.html* nevű fájl HTML bejegyzéseket használ arra, hogy a *document.write* függvény információkat jelenítsen meg erről a könyvről az aktuális HTML dokumentumban:

```
<HTML>
<HEAD><TITLE>KönyvInfo Script</TITLE></HEAD>
<BODY>
<H1>Információk a könyvrõl</H1>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

document.write("Címe: <I>A Web programozása</I><BR>")
document.write("Szerzõi: Jamsa, Lalani, Weakley<BR>")
document.write("Ára: <Big>$49.95</Big>")

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

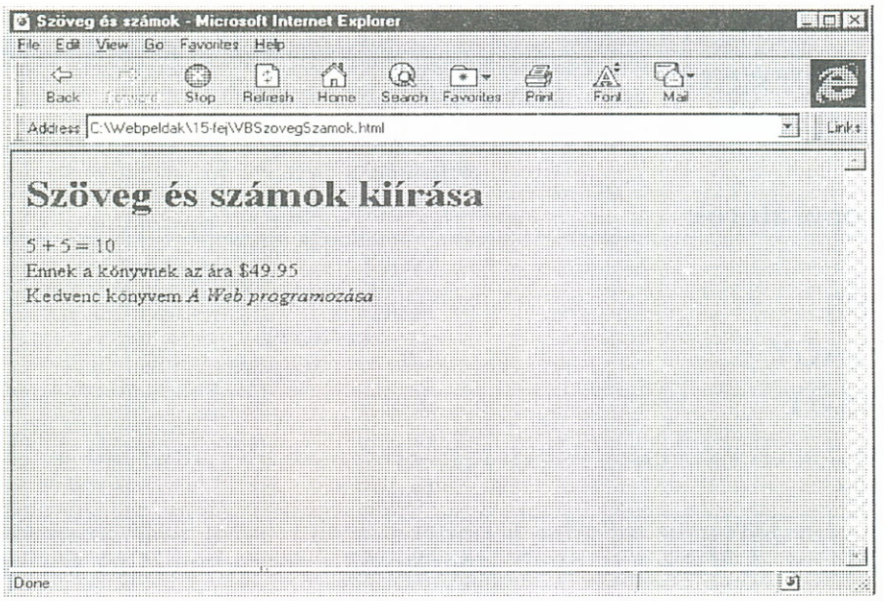


15.8. ábra. Szöveg egyszerű kiíratása a *document.write* függvénnyel

Sok esetben szükség lehet arra, hogy ugyanazon kiíráson belül szöveget és számot kombináljunk egymással. Ehhez csak annyit kell tennünk, hogy a szöveghez hozzáfűzzük (konkatenáljuk) a numerikus adatokat. Az alábbi, *VBSzövegSzamok.html* nevű fájlban a HTML bejegyzések szöveget és numerikus adatokat kombinálnak egy *document.write* függvényhíváson belül:

```
<HTML>
<HEAD><TITLE>Szöveg és számok</TITLE></HEAD>
<BODY>
<H1>Szöveg és számok kiírása</H1>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

Dim Ar
Ar = 49.95

document.write("5 + 5 = " & 5 + 5 & "<BR>")
document.write("Ennek a könyvnek az ára $" & Ar & "<BR>")
document.write("Kedvenc könyvem <I>A Web programozása</I>")

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Figyeljük meg, hogyan használja a script fájl a *document.write* függvényhíváson belül a HTML elemeket a szöveg dőlt betűssé tételére! Hasonló módon használhatók az egyéb HTML elemek is a szöveg formázásához. Ha a Microsoft Internet Explorer segítségével beolvassuk ezeket a bejegyzéseket, akkor a böngésző ablakában a 15.9. ábrán láthatóhoz hasonló jelenik meg:



15.9. ábra. Szöveg és számok megjelenítése a *document.write* függvénnyel

Megjegyzés: A rosszindulatú programok potenciálisan veszélyeztetik a rendszerünket. E veszélyek csökkentése céljából a VBScript nyelv a Java és a JavaScript nyelvhez hasonlóan nem teszi lehetővé, hogy a fájlok írási és olvasási műveleteket végezzenek.

A VBSCRIPT OPERÁTORAI

VBScript fájlokban sűrűn használunk olyan kifejezéseket, amelyek összeadásokat, kivonásokat, szorzásokat és más műveleteket végeznek változókon, numerikus állandókon, függvények által visszaadott értékeken stb. A VBScript nyelv a más programozási nyelvekben használt operátorok többségét ismeri. Más programozási nyelvekhez hasonlóan a VBScript is precedenciát rendel az operátoraihoz, és ennek alapján határozza meg a műveletek elvégzési sorrendjét. A 15.3. táblázat a VBScript operátorait sorolja fel a precedenciájuk sorrendjében a legmagasabbtól a legalacsonyabbig.

Aritmetikai operátorok	
Hatványozás	^
Negálás	-
Osztás és szorzás	*,/
Maradékszámítás (modulo)	%
Összeadás és kivonás	+, –
Karakterláncok konkatenálása	&

Összehasonlító operátorok	
Egyezőség	=
Különbözőség	<>
Kisebb mint	<
Nagyobb mint	>
Kisebb vagy egyenlő	<=

Összehasonlító operátorok	
Nagyobb vagy egyenlő	>=
Objektumegyezőség	is

Logikai operátorok	
Negálás	Not
Logikai ÉS	And
Logikai VAGY	Or
Kizáró VAGY	Xor
Ekvivalens	Eqv
Implikáció	Imp

15.3. táblázat. A VBScript nyelv operátorai a precedenciájuk sorrendjében

A VBSCRIPT PROGRAMSZERKEZETEI

Bonyolultabb script fájlokban általában elkerülhetetlen, hogy különböző programszerkezetekkel (mint például *If Else* utasításokkal vagy *For* ciklussal) vezéreljük a fájl végrehajtását. A VBScript programszerkezetei nagyon hasonlítanak a más programozási nyelvekben, mint például a C/C++ vagy a Java nyelvben meglévő programszerkezetekhez. A 15.4. táblázat egy-egy példát mutat be ezekre a programszerkezetekre.

Programszerkezet	Formátum
If	If (feltétel) utasítás End If
If Else	If (feltétel) utasítás Else utasítás End If
If ElseIf	If (feltétel) utasítás ElseIf utasítás End If
Do Until	Do Until (feltétel) utasítás Loop
Do While	Do While (feltétel) utasítás Loop
For	For kifejezés inicializálása [lépésköz kifejezés] utasítás

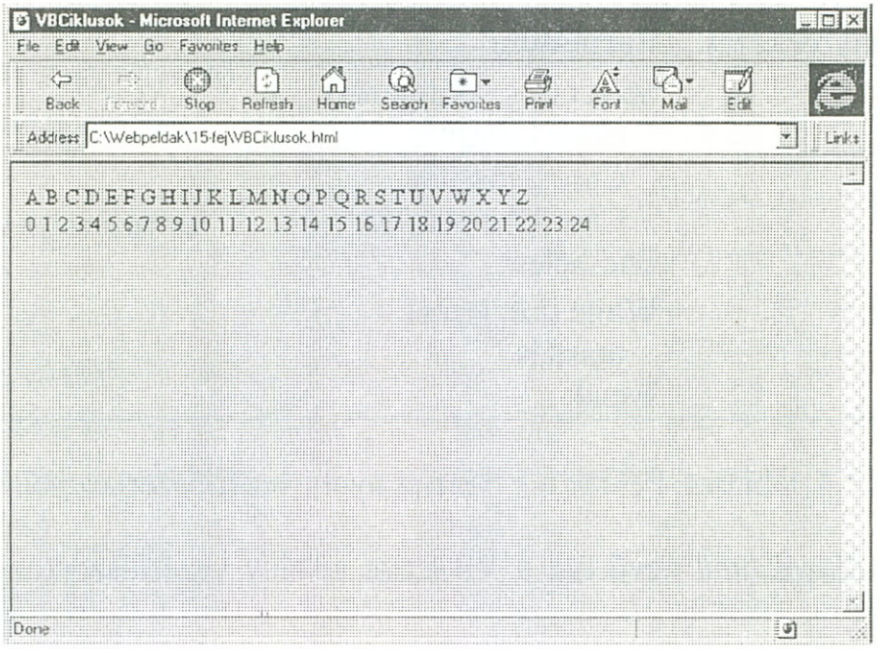
Programszerkezet	Formátum
Select	Select Case (kifejezés) Case 1 utasítás Case 2 utasítás Case Else utasítás End Select
While	While (kifejezés) utasítás Wend

15.4. táblázat. Példák a VBScript nyelv vezérlőszerkezeteire

Az alábbi, *VBCiklusok.html* nevű fájl a VBScript néhány vezérlőszerkezetének használatára mutat példát:

```
<HTML>
<HEAD><TITLE>VBCiklusok</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

Dim Betu
Dim Index

For Betu = 65 To 90
  document.write(Chr(Betu) & " ")
Next

document.write("<BR>")

Index = 0

Do While (Index < 25)
  document.write(Index & " ")
  Index = Index + 1
Loop

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Microsoft Internet Explorer segítségével beolvassuk ezeket a HTML utasításokat, akkor a böngésző ablakában a 15.10. ábrán láthatóhoz hasonló jelenik meg:



15.10. ábra. Különböző VBScript vezérlőszerkezetek használata

A VBSCRIPT NYELV TÖMBJEI

Más programozási nyelvekhez hasonlóan a VBScript is lehetővé teszi, hogy azonos típusú értékeket egyetlen, tömbnek nevezett változóban tároljunk. Egyik tömbben tárolhatjuk például 100 tanuló érdemjegyeit, egy másik tömbben pedig a tanulók neveit. Azt is megtehetjük, hogy ugyanabban a tömbben tároljuk az érdemjegyeket is (amelyek numerikus értékek), és a neveket is (amelyek karakterláncok).

VBScript nyelvben egy tömbváltozót a DIM utasítással deklarálhatunk. A deklarációban kell megadnunk azt is, hogy hány elemet tároljon a tömb. Az alábbi deklaráció például 100 érdemjegy tárolására alkalmas tömb számára foglal helyet a memóriában:

```
DIM intErdemJegy(99)
```

Valamely tömbön belül indexértékek segítségével érhetjük el az egyes elemeket. A VBScript tömbjeinek első elemére a 0 indexérték mutat. A fenti példa alapján ennek a hivatkozásnak az *intErdemJegy(0)* az alakja. A tömb utolsó eleméhez az *intErdemJegy(99)* hivatkozással férhetünk hozzá.

Egydimenziós tömbök (értékek oszlopa) mellett többdimenziós tömbök is használhatók a VBScript nyelvben. Az alábbi utasítás például olyan tömböt hoz létre, amelyben elhelyezhetők a tic-tac-toe néven ismert kiszorítós játék 3 x 3-as táblájának mezői:

```
DIM int_TicTacToe(2, 2)
```

Ha áttanulmányozzuk a fejezet későbbi részében ismertetésre kerülő függvényeket és eljárásokat, akkor látni fogjuk, hogy a VBScript nyelv egy sor olyan, beépített függvényt tartalmaz, amelyek segítségével megállapíthatjuk és megváltoztathatjuk egy tömb méretét, törölhetjük a tömb tartalmát stb.

A VBSCRIPT NYELV FÜGGVÉNYEI ÉS SZUBRUTINJAI

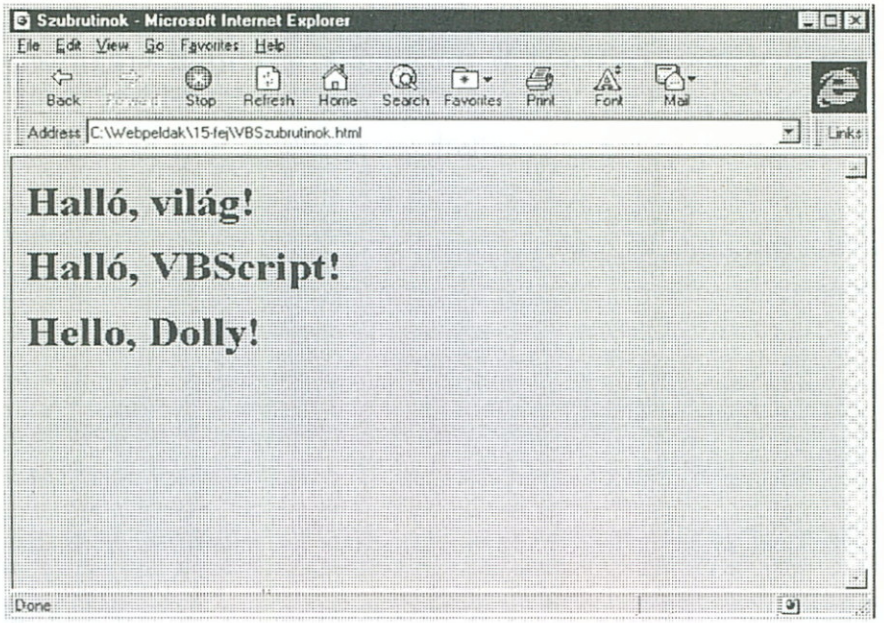
A programozók az összetettebb feladatokat általában kisebb, könnyebben kezelhető részfeladatokra, úgynevezett függvényekre és szubrutinokra bontják. Egy függvény abban különbözik egy szubrutintól, hogy az előbbi általában visszaad valamilyen értéket. Mind a *függvényeknek*, mind a *szubrutinoknak* értékeket (paramétereket) adhatunk át feldolgozás céljából. Az alábbi utasítások a VBScript nyelv szubrutinjainak és függvényeinek formátumát szemléltetik:

```
Sub szubrutin_neve[(opcionális paraméterek)]
  ' utasítások helye
End Sub

Function név[(opcionális paraméterek)]
  ' utasítások helye
  név = eredmény
End Function
```

Minden függvénynek és szubrutinnak saját, a többitől eltérő névvel kell rendelkeznie (aminek betűvel kell kezdődnie). A szögletes zárójelek azt jelzik, hogy a paraméterek megadása nem kötelező. Az alábbi, *VBSzubrutinok.html* nevű HTML fájl a VBScript szubrutinjainak használatát mutatja be:

```
<HTML>
<HEAD><TITLE>Szubrutinok</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

sub vilag
  document.write("<H1>Halló, világ!</H1>")
end sub

sub VBScript
  document.write("<H1>Halló, VBScript!</H1>")
end sub

sub titkos(Uzenet)
  document.write("<H1>Hello, " & Uzenet & "!" & "</H1>")
end sub

call vilag
call VBScript
call titkos("Dolly")

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Microsoft Internet Explorer segítségével beolvassuk ezeket a HTML utasításokat, akkor a böngésző ablakában a 15.11. ábrán láthatóhoz hasonló jelenik meg:



15.11. ábra. Szubrutinok használata a VBScript nyelvben

Röviden említettük már, hogy a függvények annyiban különböznek az szubrutinoktól, hogy az előzőek értéket adnak vissza a függvény hívójának. Tegyük fel például, hogy van egy *ÁtlagÉrték* nevű függvényünk, amely két érték átlagát adja vissza. A script fájlon belül a visszaadott értéket hozzárendelhetjük egy változóhoz, feltételként használhatjuk, vagy akár meg is jeleníthetjük az alábbiak szerint:

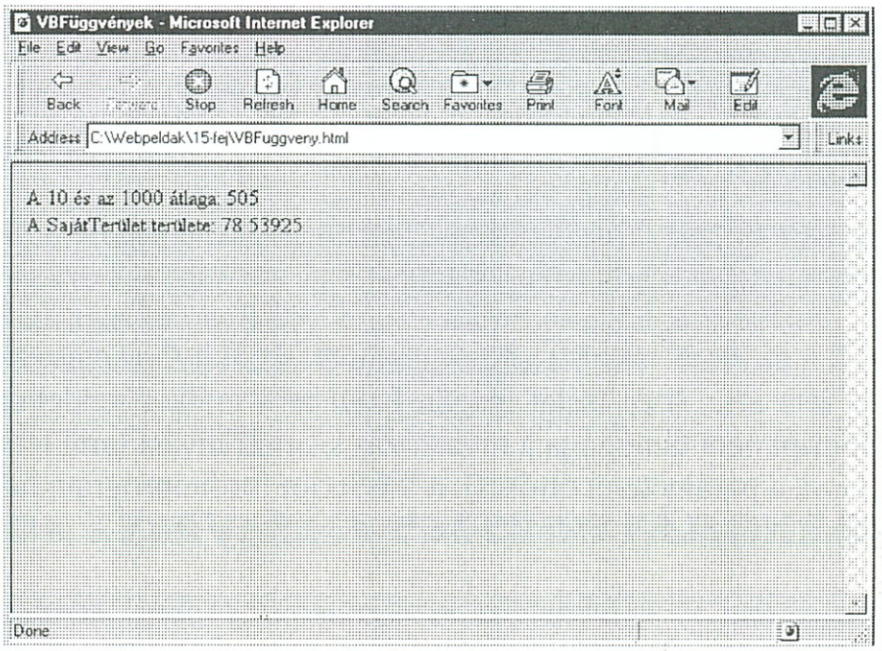
```
atlag = AtlagErtek(100, 333)            ' hozzárendelés egy változóhoz
if (AtlagErtek(150, 200) < 145)         ' használata feltételben
document.write("Az átlagérték: " &
    AtlagErtek(12, 1112)                ' megjelenítése függvénnyel
```

Az alábbi, *VBFüggvény.html* nevű HTML fájl a VBScript nyelv függvényeinek használatát mutatja be:

```
<HTML>
<HEAD><TITLE>VBFüggvények</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

Function AtlagErtek(a, b)
  AtlagErtek = (a + b) / 2.0
End Function

Function KorTerulet(sugar)
  KorTerulet = 3.14157 * sugar * sugar
End Function
```

```
DIM SajatTerulet

document.write("A 10 és az 1000 átlaga: " & AtlagErtek(10, 1000))

SajatTerulet = KorTerulet(5)

document.write("<BR>A SajátTerület területe: " & SajatTerulet)

// A kód elrejtésének vége ->
</SCRIPT></BODY></HTML>
```

Ha a Microsoft Internet Explorer segítségével beolvassuk ezeket a HTML utasításokat, akkor a böngésző ablakában a 15.12. ábrán láthatóhoz hasonló jelenik meg:



15.12. ábra. A VBScript nyelv függvényeinek használata

Amint látható, az *ÁtlagÉrték* függvény értékként két paramétert kap. A JavaScript nyelvhez hasonlóan a VBScript nyelvben sem tudja megváltoztatni egy függvény vagy szubrutin egy paraméter értékét. Más szavakkal ez azt jelenti, hogy a VBScript függvényei és szubrutinjai csak az *érték szerinti* (by value) hívást támogatják, a *hivatkozás szerinti* (by reference) hívást nem.

PARAMÉTEREK ÁTADÁSA VBSCRIPT FÜGGVÉNYEKNEK

Az előbb említettük, hogy a VBScript függvényei és szubrutinjai nem tudják megváltoztatni egy paraméter értékét. Ha meg akarunk győződni arról, hogy a VBScript valóban az értékek szerint adja át a paramétereket a függvényeknek és szubrutinoknak (úgy, hogy eközben a függvény és a szubrutin nem változtatja meg a paramétert), akkor kísérletezzünk a következő, *VBÉrtékSzerint.html* nevű fájlban lévő HTML utasításokkal:

```
<HTML>
<HEAD><TITLE>Érték szerinti paraméterek</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">

<!-A VBscript kód elrejtése
sub ParameterValtoztato(UjErtek)
  document.write("A ParaméterVáltoztató szubrutinba az " & ertek & " érték
      került" & "<BR>")
  UjErtek = 1001
  document.write("A ParaméterVáltoztató szubrutinban az " & ertek & " érték
      maradt" & "<BR>")
end sub

Dim ertek
ertek = 1
Call ParameterValtoztato(ertek)
document.write("Visszatérés a szubrutinból az " & ertek & " értékkel")

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Amint látható, a fájl a az *érték* változót adja át paraméterként a *ParaméterVáltoztató* szubrutinnak, amely kísérletet tesz arra, hogy saját magán belül más értéket rendeljen a paraméteréhez. Mivel azonban a VBScript nem támogatja a hivatkozás szerinti hívást (ami lehetővé tenné, hogy a szubrutin megváltoztassa a paraméterei értékét), a VBScript fordítóprogram hibát jelez, amikor az értékadással találkozik.

KILÉPÉS SZUBRUTINBÓL AZ EXIT PARANCCSAL

Amikor a VBScript meghív egy függvényt vagy szubrutint, akkor a szubrutin végrehajtása legtöbb esetben az első utasításával kezdődik, majd sorban végrehajtódnak az utasításai egészen az utolsóig. Attól függően, hogy a szubrutin vagy a függvény hogyan használ feltételes utasításokat (mint amilyen például az *if* vagy a *select*), előfordulhat, hogy a VBScript nem hajtja végre a szubrutin valamennyi utasítását. Ennek ellenére a szubrutin végrehajtásának a szubrutin elején kell kezdődnie, és a végén kell befejeződnie. Másként fogalmazva ez azt jelenti, hogy a szubrutinban egyetlen belépési pontnak (az elején) és egyetlen kilépési pontnak (a végén) kell lenni.

A feladattól függően azonban az is előfordulhat, hogy a szubrutin végrehajtását nem a végén, hanem valamely más, közbenső pontján szeretnénk megállítani. Ha például a szubrutin komoly problémával találkozik, akkor azt szeretnénk, hogy a program azonnal kilépjen a szubrutinból. Az azonnali kilépéshez a szubrutinnak vagy a függvénynek az *Exit* eljárást kell meghívnia az alábbi módon:

```
<HTML>
<HEAD><TITLE>Exit bemutatása</TITLE></HEAD>
```

```
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése
sub ExitMinta
document.write("<H1>Halló, kilépek a szubrutinból!</H1>")

if (true) then
Exit sub
end if
document.write("Ez a sor már nem fog megjelenni<BR>")
end sub

document.write("Most következik az ExitMinta szubrutin hívása<BR>")
call ExitMinta
document.write("Ez már nem az ExitMinta szubrutin")

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Azért, hogy más programozók számára megkönnyítsük kódjaink olvasását és megértését, lehetőség szerint csak egyetlen be- és egyetlen kilépési pontja legyen a szubrutinjainknak. A legtöbb esetben átírható egy kód úgy, hogy más programszerkezetekkel, például *If-Else* utasításokkal kiküszöbölhető legyen az *Exit* használata.

SZUBRUTINOK SZERVEZÉSE VBSCRIPT FÁJLOKBAN

Hosszabb és összetettebb script fájlokban minden bizonnyal sűrűn kell használnunk függvényeket és szubrutinokat. A fájl jobb szervezése céljából előnyös lehet, ha az összes szubrutint és függvényt a fájl egyetlen részén gyűjtjük össze. Ilyen módon áttekinthetőbbé válnak a fájljaink, és gyorsabban megtalálhatók a függvények és a szubrutinok. Az alábbi, *VBSzubCsoport.html* nevű HTML fájl szemlélteti, hogyan csoportosíthatjuk az összes szubrutint és függvényt egyetlen helyre:

```
<HTML>
<HEAD><TITLE>Szubrutinok csoportosítása</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A VBScript kód elrejtése

sub TermekInfo
  document.write("Cím: A Web programozása<BR>")
  document.write("Szerzõk: Jamsa, Lalani, Weakley<BR>")
  document.write("Ár: $49.95<BR><BR>")
end sub
```

```
sub Roviditesek
  document.write("HTTP: Hypertext Transfer Protocol<BR>")
  document.write("HTML: Hypertext Markup Language<BR>")
  document.write("ISAPI: Internet Server API")
  document.write("<BR><BR>")
end sub

sub HalloVilag
  document.write("Halló, VBScript világ!<BR>")
end sub

// A kód elrejtésének vége ->
</SCRIPT>
<H1>Itt valamilyen HTML szöveg állhat</H1>

<SCRIPT LANGUAGE="VBScript">
<!- A kód elrejtése

call HalloVilag
call TermekInfo
call Roviditesek

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

EGYSZERŰ PÁRBESZÉDABLAKOK MEGJELENÍTÉSE

A fejezet eddigi részében láttuk, hogyan használható a *document.write* metódus arra, hogy szöveget illesszünk egy HTML fájlba. A script fájl céljától függően előfordulhat, hogy valamilyen üzenetet szeretnénk megjeleníteni egy párbeszédablakban a felhasználó számára. Ilyen esetben a *MsgBox* függvényt használhatjuk az alábbi módon:

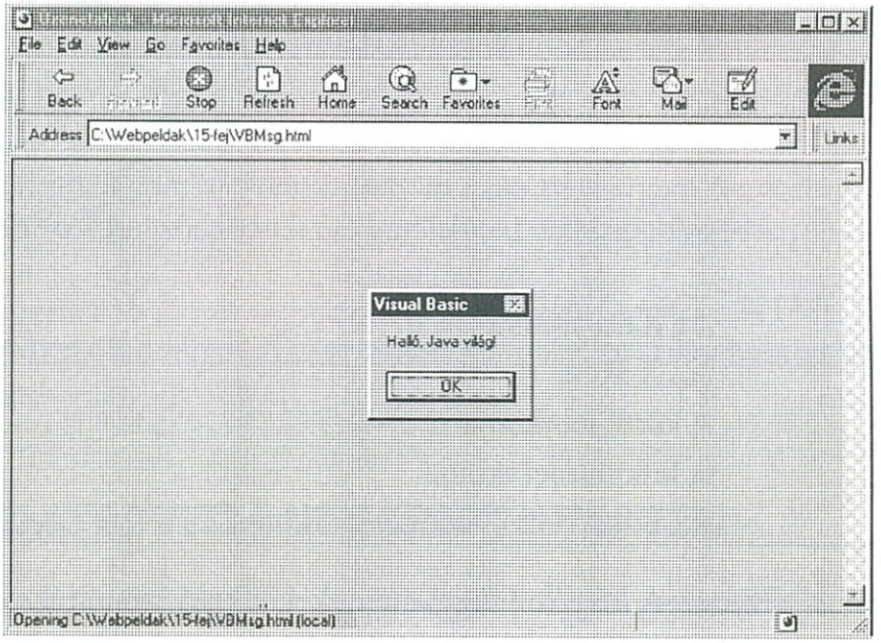
```
MsgBox "Halló, VBScript!"
```

Az alábbi, *VBMsg.html* nevű HTML fájl szemlélteti a *MsgBox* függvény használatát. A függvény üzenetet jelenít meg egy párbeszédablakban a felhasználó számára:

```
<HTML>
<HEAD><TITLE>Üzenetablak</TITLE></HEAD>
<SCRIPT LANGUAGE="VBScript">
<!-A kód elrejtése

MsgBox("Halló, Java világ!")

// A kód elrejtése ->
```

```
</SCRIPT>
</BODY></HTML>
```

Ha a Microsoft Internet Explorer segítségével beolvassuk ezeket a HTML utasításokat, akkor a böngésző ablakában a 15.13. ábrán láthatóhoz hasonló jelenik meg:



15.13. ábra. A *MsgBox* használata üzenet párbeszédablakban történő megjelenítéséhez

Előfordulhatnak olyan esetek, amikor szöveget és számokat kell kombinálnunk ugyanabban a párbeszédablakban. Ehhez a konkatenáció operátorát kell használnunk. Az alábbi, *VBBoxSzamok.html* nevű HTML fájlban a HTML utasítások szöveget és számokat kombinálnak a *MsgBox* kimenetében:

```
<HTML>
<HEAD><TITLE>Üzenetablak</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A kód elrejtése

MsgBox("A 3+5 mûvelet eredménye " & 3 + 5 & "!")

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

A *MsgBox* függvény az előbb bemutatottaknál jóval nagyobb tudású: különböző paraméterek átadásával befolyásolhatjuk a tartalmát és megjelenését. Ha például a párbeszédablak OK gombját Cancel, Yes, No stb. gombokra akarjuk cserélni, akkor a 15.5. táblázatban felsorolt értékek valamelyikét kell átadnunk a *MsgBox* függvénynek (zárójelben azok a gombnevek állnak, amelyek a Windows 95 magyar nyelvű változatának használata esetén jelennek meg).

Érték	A gomb típusa
0	OK gomb
1	OK és Cancel gomb (OK, Mégse)
2	Abort, Retry és Ignore gomb (Megszakítás, Ismét, Tovább)
3	Yes, No és Cancel gomb (Igen, Nem, Mégse)
4	Yes vagy No gomb (Igen, Nem)
5	Retry és Cancel gomb (Ismét, Mégse)

15.5. táblázat. A *MsgBox* függvénynek átadható gombértékek

A *MsgBox* gombtípusainak megváltoztatása mellett ikonok is megjeleníthetők a párbeszédablakban. A 15.6. táblázat a párbeszédablakban megjeleníthető ikonokhoz tartozó értékeket sorolja fel.

Érték	Az ikon típusa
16	Az X ikon
32	A kérdőjel ikonja
48	A felkiáltójel ikonja
64	Az információ i betűjének ikonja

15.6. táblázat. A *MsgBox* függvénynek átadható ikonértékek

Ha használni akarjuk a gombok és az ikonok értékeit, akkor egy második paramétert is át kell adnunk a *MsgBox* függvénynek, amelynek értéke a kívánt gomb és ikon értékének az összege. Ezen túlmenően még egy harmadik paramétert is átadhatunk a függvénynek, amivel a párbeszédablak címsorában megjelenő szöveget adhatjuk meg:

```
eredmeny = MsgBox("szöveg", gomb + ikon, "A címsor szövege")
```

Ha az *MsgBox* függvénnyel megjelenítünk egy vagy több gombot, akkor a függvény a felhasználó által választott gombnak megfelelő értéket adja vissza. A 15.7. táblázat felsorolja a *MsgBox* függvénynek azokat az értékeit, amelyeket a függvény a különböző gombok választásakor ad vissza.

Visszatérési érték	A választott gomb
1	OK gomb
2	Cancel gomb
3	Abort gomb
4	Retry gomb
5	Ignore gomb
6	Yes gomb
7	No gomb

15.7. táblázat. A *MsgBox* függvény visszatérési gombértékei

A következő, *VBMsgBoxIkon.html* nevű HTML fájl az ikonok és gombok *MsgBox* függvényen belüli használatát szemléltetik:

```
<HTML>
<HEAD><TITLE>Üzenetablak</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A kód elrejtése

eredmeny = MsgBox("Annyira izgatott vagyok!", 48 + 0, "Izgalmas címsor")
valasz = MsgBox("Kérdezhetek valamit?", 32 + 4, "A kérdés címsora")
valasztas = MsgBox("Melyik gombot választja?", 64 + 2, "Néhány gomb...")

// A kód elrejtésének vége ->
</SCRIPT></BODY></HTML>
```

Ha a Microsoft Internet Explorer segítségével beolvassuk ezeket a HTML utasításokat, akkor a fájl különböző *MsgBox* párbeszédablakokat jelenít meg, amelyek bemutatják a különböző gombok és ikonok használatát.

EGYSZERŰ ADATBEVITEL VBSCRIPT NYELVEN

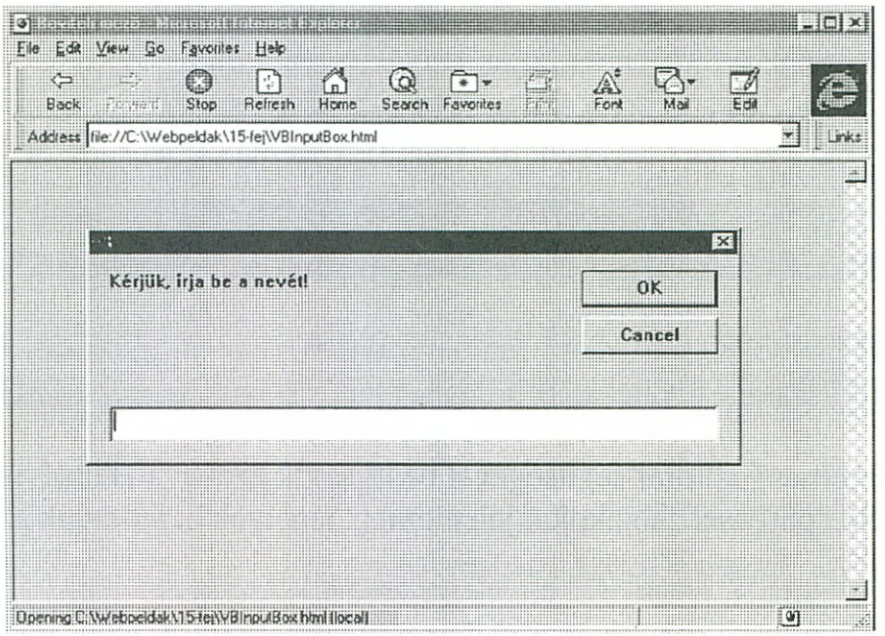
Amint láttuk, a VBScript *MsgBox* függvénye segítségével nagyon egyszerűen jeleníthetünk meg üzeneteket egy párbeszédablakban. A script fájloknak esetenként azonban arra is szükségük lehet, hogy a felhasználótól kérjenek be egysoros információkat. Ilyen esetben a script fájlnak az *InputBox* függvényt kell meghívnia. Az *InputBox* függvény segítségével a script fájl azt is megjelenítheti a párbeszédablakban, hogy milyen jellegű információt kér a felhasználótól. Miután a felhasználó beírta a kívánt információt és lenyomta az Enter billentyűt, az *InputBox* a felhasználó által beírt adattal tér vissza. A függvény visszatérési értéke hozzárendelhető egy változóhoz, vagy kifejezésben is használható:

```
eredmeny = InputBox("Felhasználói prompt")
```

Az alábbi, *VBInputBox.html* nevű HTML fájl az *InputBox* függvény segítségével kér be adatot a felhasználótól:

```
<HTML>
<HEAD><TITLE>Beviteli mezõ</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A kód elrejtése

DIM eredmeny

eredmeny = InputBox("Kérjük, írja be a nevét!");
document.write("<BR>Üdvözöljük, " & eredmeny & "!")

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Microsoft Internet Explorer segítségével beolvassuk ezeket a HTML utasításokat, akkor a böngésző ablakában a 15.14. ábrán láthatóhoz hasonló jelenik meg:



15.14. ábra. Adatbevitelre felszólító párbeszédablak

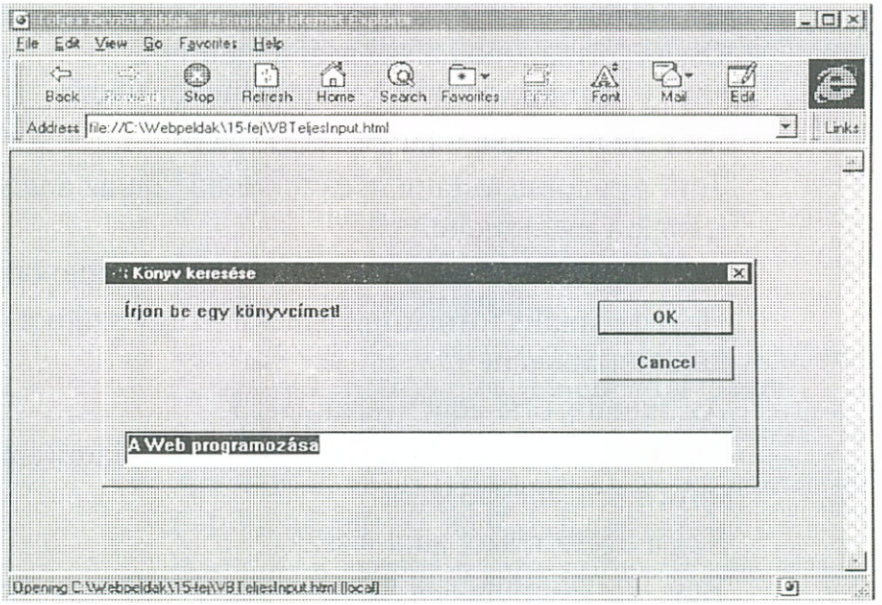
Bonyolultabb script fájlokban további paramétereket is átadhatunk az *InputBox* függvénynek, amelyekkel befolyásolhatjuk a párbeszédablak megjelenésének módját és helyét:

```
eredmeny = InputBox("prompt", "cím",
   "alapértelmezett válasz", x koordináta, y koordináta,
   "Súgó fájl", "Súgó fájl kontextus")
```

Az alábbi, *VBTeljesInput.html* nevű HTML fájl szemlélteti az *InputBox* paramétereinek használatát:

```
<HTML>
<HEAD><TITLE>Teljes beviteli ablak</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="VBScript">
<!- A kód elrejtése

DIM eredmeny

eredmeny = InputBox("Írjon be egy könyvcímet!", "Könyv keresése", "A Web
 programozása", 1500, 3000)

document.write("<BR>" & eredmeny)

// A kód elrejtésének vége ->
</SCRIPT>
</BODY></HTML>
```

Ha a Microsoft Internet Explorer segítségével beolvassuk ezeket a HTML utasításokat, akkor a böngésző ablakában a 15.15. ábrán láthatóhoz hasonló jelenik meg:



15.15. ábra. Teljes beviteli párbeszédablak megjelenítése

A VBSCRIPT BEÉPÍTETT FÜGGVÉNYEI

Amint láttuk, a VBScript nyelv lehetővé teszi a függvények és szubrutinok használatát. A programozási munka egyszerűsítése céljából a VBScript nyelv „beépített" függvényeket bocsát a rendelkezésünkre, amelyek közvetlenül használhatók általános programozási feladatokhoz. E függvények többsége hasonlít a más programozási nyelvek standard függvénytáraiban megtalálható függvényekhez. A beépített függvényeket a 15.8. táblázat sorolja fel.

Függvény neve	Feladata
abs	Kifejezés abszolút értékét adja vissza
asc	Karakterlánc első karakterének ASCII kódját adja vissza
ascB	Karakterlánc első karakterének bájtértékét adja vissza
atn	Kifejezés árkusz tangensét adja vissza
cBool	Boole-értékké alakít át egy változót
cByte	Egy 0...255 tartományban lévő számot bájtértékké alakít át
cDbl	Számot duplapontosságú lebegőpontos értékké alakít át
chr	Egy karakter ASCII kódját adja vissza, ami egy adatbájtnak felel meg
chrB	Bájtértéket ad vissza, ami megfelel egy ASCII karakterkódnak
cInt	Számot integer értékké (egész számmá) alakít át
cLng	Számot hosszú integerré alakít át
cos	Radiánban megadott szög koszinuszát adja vissza
cSng	Számot egyszeres pontosságú lebegőpontos értékké alakít át

Függvény neve	Feladata
date	Az aktuális rendszeridőt adja vissza
dateSerial	Az aktuális dátumhoz viszonyított dátumot adja vissza
dateValue	Olyan formátumba alakítja át a dátumot, amely összehasonlításoknál jobban kezelhető
day	Az aktuális napot adja vissza (1–31)
erase	Törli egy tömb tartalmát, és felszabadítja az általa lefoglalt memóriahelyet
fix	Szám egész részét adja vissza visszatérési állapotkódként
hex	Hexadecimális formátumba alakít át egy számot karakterláncként
hour	Az aktuális órát adja vissza (0–23)
inputBox	Párbeszédablakban kéri a felhasználót valamilyen információ megadására
int	Szám egész részét adja vissza
inStr	Rész-karakterláncnak az egész karakterláncon belüli kezdő pozícióját (indexét) adja vissza
inStrB	Hasonló az inStr függvényhez azzal a különbséggel, hogy egy bájtérték helyét adja vissza
isArray	Igaz értéket ad vissza, ha az adott változó tömb, ellenkező esetben a hamis értékkel tér vissza
isDate	Igaz értéket ad vissza, ha az adott változó dátumot tartalmaz
isEmpty	Igaz értéket ad vissza, ha az adott változó még nem lett inicializálva, ellenkező esetben a hamis értékkel tér vissza
isError	Igaz értéket ad vissza, ha az adott változó hibakódot tartalmaz, ellenkező esetben a hamis értékkel tér vissza
isNull	Igaz értéket ad vissza, ha az adott változó értéke nulla, ellenkező esetben a hamis értékkel tér vissza
isNumeric	Igaz értéket ad vissza, ha az adott változó numerikus értéket tartalmaz, ellenkező esetben a hamis értékkel tér vissza
isObject	Igaz értéket ad vissza, ha az adott változó OLE objektumnak felel meg
lBound	Az adott tömb alsó korlátját (indexértékét) adja vissza
Lcase	Kisbetűsre alakít át egy karakterláncot
Left	Karakterlánc részláncát adja vissza; a részlánc a karakterlánc bal szélén kezdődik és n karakterből áll
Len	Karakterláncban lévő karakterek számát adja vissza
LenB	Egy változóban tárolandó bájtok számát adja vissza
Log	Az adott szám természetes logaritmusát adja vissza
Ltrim	Levágja egy karakterlánc bal széléről a vezető szóközöket, és az így kapott karakterláncot adja vissza
Mid	Karakterlánc belsejében lévő részláncot adja vissza

Függvény neve	Feladata
Minute	Az aktuális percet adja vissza (0–59)
MsgBox	Üzenetet jelenít meg egy párbeszédablakban
Now	Az aktuális dátumot és időt tartalmazó karakterláncot adja vissza
Oct	Oktális formátumba alakít át egy számot karakterláncként
Right	Karakterlánc részláncát adja vissza; a részlánc a karakterlánc jobb szélén kezdődik és n karakterből áll
Rnd	Véletlen számot generál
Rtrim	Levágja egy karakterlánc jobb széléről a szóközöket, és az így kapott karakterláncot adja vissza
Second	Az aktuális másodpercet adja vissza (0–59)
Sgn	Adott érték előjelét adja vissza egész számként (–1, ha negatív, 0, ha 0, 1, ha pozitív)
Sin	Radiánban megadott szög szinuszát adja vissza
Space	Adott számú karaktert tartalmazó karakterláncot ad vissza
Sqr	Adott érték négyzetgyökét adja vissza
StrComp	Egész értéke ad vissza, amely megadja, hogy két karakterlánc azonos-e, az első kisebb, mint a második stb.
String	Olyan karakterláncot ad vissza, amely az adott karaktert n-szer tartalmazza
Tan	Radiánban adott szög tangensét adja vissza
Time	Az aktuális rendszeridőt tartalmazó karakterláncot adja vissza
TimeSerial	Az aktuális időhöz viszonyított időt adja vissza
TimeValue	Olyan formátumba alakítja át az időt, amely összehasonlításoknál jobban kezelhető
Trim	Levágja egy karakterlánc elejéről és végéről a szóközöket, és az így kapott karakterláncot adja vissza
Ubound	Az adott tömb felső korlátját (indexértékét) adja vissza
VarType	A változó típusát jelző értéket adja vissza
Weekday	A hét aktuális napjának megfelelő egész értéket adja vissza (0 = vasárnap, 6 = szombat)
Year	Az aktuális évet adja vissza

15.8. táblázat. A VBScript nyelv beépített függvényei

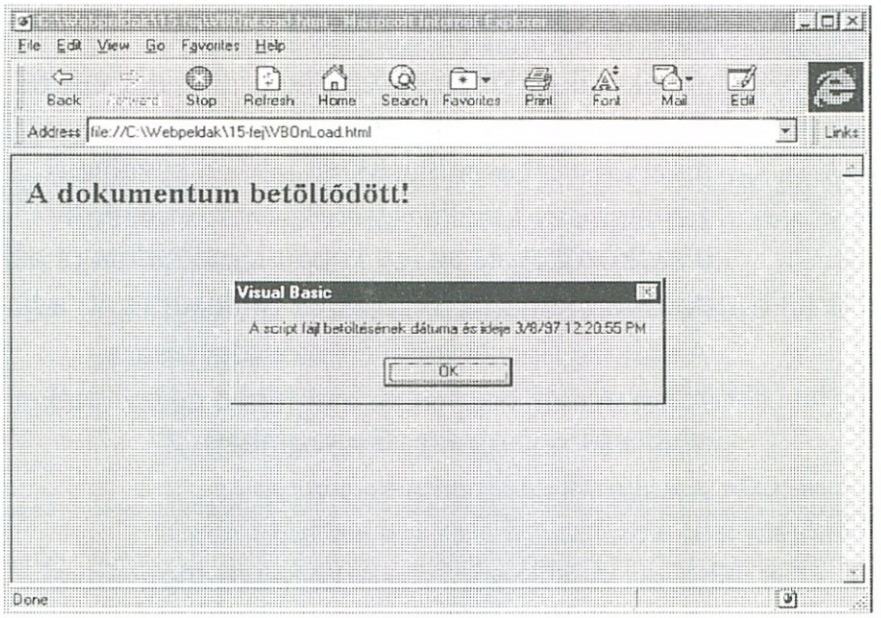
A VBSCRIPT ESEMÉNYEI

Az előző fejezetekben már többször szóba került, hogy a Windows alapú programok eseményvezéreltek, ami azt jelenti, hogy ezeknek a programoknak gyakran kell válaszolniuk (reagálniuk) a fel-

használó különböző műveleteire. Ha például a felhasználó az egérrel rákattint egy objektumra, akkor a Windows programok többsége egy speciális, eseménykezelőnek nevezett függvényt hív meg, amely reagál az eseményre. A VBScript nyelvben szubrutinok segítségével definiálhatunk ilyen eseménykezelőket. Az alábbi, *VBOnLoad.html* nevű fájl utasításai az *OnDocumentLoading* nevű különleges függvény meghívásával válaszolnak az *OnLoad* eseményre. Ez az esemény akkor fordul elő, amikor a böngésző első alkalommal tölti be a dokumentumot:

```
<HTML>
<HEAD><TITLE>Dokumentum betöltése esemény</TITLE></HEAD>

<SCRIPT LANGUAGE="VBScript">
<!- A kód elrejtése

Sub OnDocumentLoading
  MsgBox("A script fájl betöltésének dátuma és ideje " & Date & " " & Time)
End Sub

// A kód elrejtésének vége ->
</SCRIPT>

<BODY LANG="VBScript" ONLOAD="OnDocumentLoading">

<H2>A dokumentum betöltõdött!</H2>
</BODY></HTML>
```

Ha a Microsoft Internet Explorer segítségével beolvassuk ezeket a HTML utasításokat, akkor a böngésző ablakában a 15.16. ábrán láthatóhoz hasonló jelenik meg:



15.16. ábra. Válaszolás az eseménykezelő segítségével az *OnLoad* eseményre

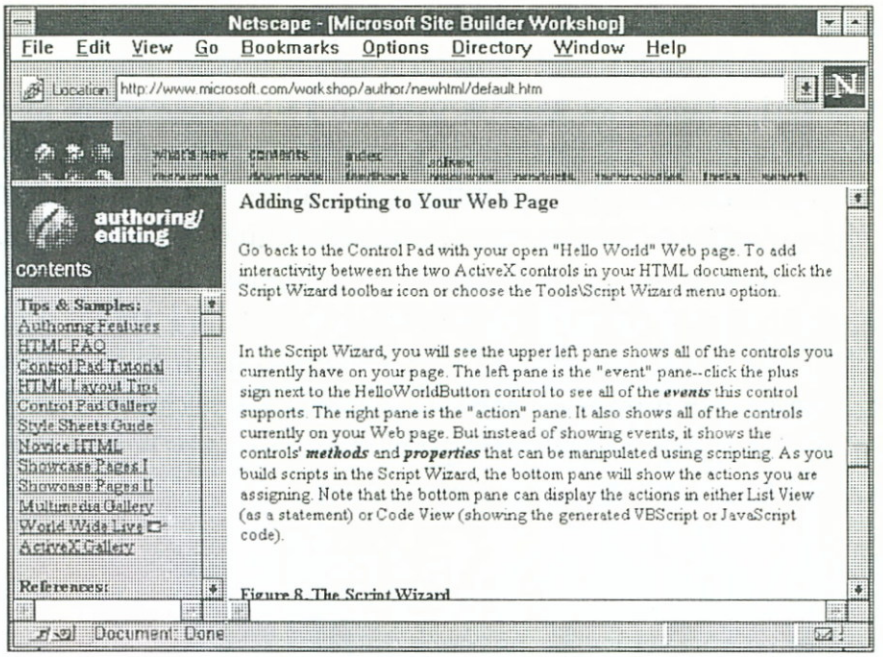
Amint látjuk, amikor a böngésző futtatja a script fájl betöltéskezelőjét (onload handler), még éppen folyamatban van a dokumentum készítése. Mivel a dokumentum még nem létezik, a script fájl nem tudja használni a *document.write* függvényt a kimenetének képernyőn történő megjele-

nítésére. A fájl ezért kénytelen a *MsgBox* függvényt meghívni, hogy ennek az üzenetablakába írja ki a fájl indításának dátumát és idejét.

A VBScript nyelvű script fájloknak nagyon gyakran kell válaszolniuk különböző egéreseményekre. Előfordulhat például, hogy a felhasználó egy hiperhivatkozásra, egy jelölőnégyzetre, a küldésnek megfelelő gombra (Submit) vagy valami másra kattint. Az ilyen egérkattintási események kezelése céljából a fájl egy vagy több szubrutint definiálhat, amelyek válaszolnak az eseményekre. Az egyes objektumokra vonatkozó egérkattintások kezelőjének definiálásához a program olyan eljárásokat hoz létre, amelyeknek a neve *VmilyenObjektum_onclick* alakú. Az alábbi, *VBEgérKattintás.html* nevű HTML fájl utasításai például egy *Kapcsoló* nevű jelölőnégyzethez hoznak létre egy kattintáskezelőt. Minden alkalommal, amikor a felhasználó rákattint a jelölőnégyzetre, a script fájl megváltoztatja a jelölőnégyzet aktuális állapotát (bejelölt vagy bejelöletlen), és egy számmal kiírja, hogy a felhasználó hányszor kattintott a négyzetre:

```
<HTML>
<HEAD><TITLE>VBScript egérkattintás-kezelõ</TITLE></HEAD>

<SCRIPT LANGUAGE="VBScript">
<!- A kód elrejtése

DIM KattSzamlalo
KattSzamlalo = 0

Sub Kapcsolo_ONCLICK
  KattSzamlalo = KattSzamlalo + 1
  MsgBox("Eddig " & KattSzamlalo & " alkalommal kattintott a jelölõnégyzetre!")
End Sub

// A kód elrejtésének vége ->
</SCRIPT>
<BODY LANG="VBScript">
<INPUT TYPE="CHECKBOX" NAME="Kapcsolo">Kapcsoló<BR>
</BODY></HTML>
```

Bonyolultabb, összetettebb script fájlokban nagyon gyakran kell kezelnünk eseményeket. A VBScript eseményekkel kapcsolatban speciális információkhoz juthatunk a *http://www.microsoft.com/workshop/author/newhtml/default.htm* címen (lásd a 15.17. ábrát).



15.17. ábra. Érdekességek VBScript eseményekről

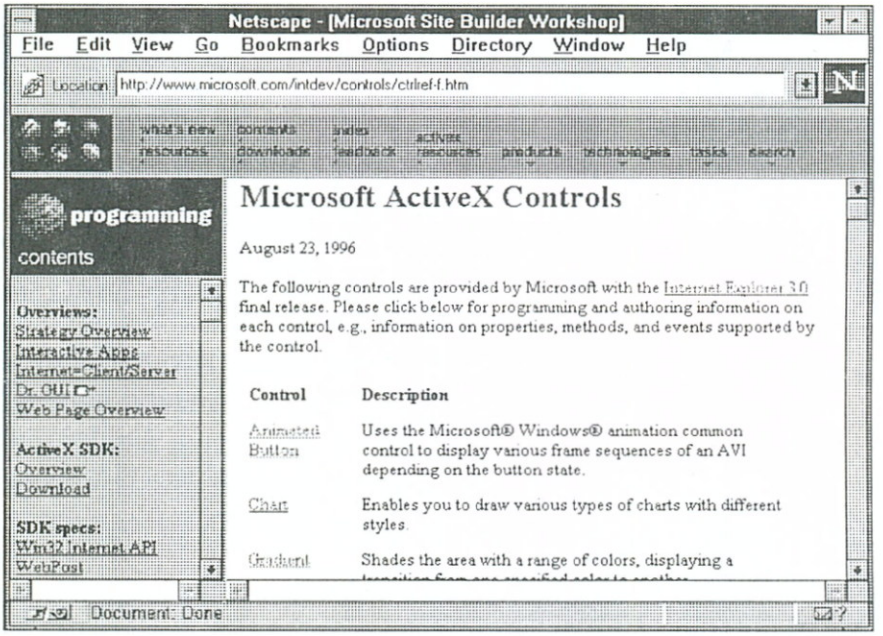
ACTIVEX OBJEKTUMOK

A fejezetben már többször említettük azt az ActiveX-nek nevezett technológiát, amelynek köszönhetően a VBScript nyelv magasan kiemelkedik az egyéb, hasonló célú script nyelvek közül. Úgy képzeljük el az ActiveX vezérlőt, mint egyfajta OLE (object linking and embedding, magyarul: objektumok csatolása és beágyazása) objektumot. Röviden fogalmazva egy OLE objektum olyan osztály-függvénytár, amely egy objektum kezeléséhez szükséges komplett függvényhalmazt tartalmaz. Mivel egy OLE objektum kódja már létezik, a programozók könnyen és gyorsan bevehetik és használhatják az objektumokat az alkalmazásaikban. A Windows programok már évek óta széles körben használják az OLE objektumokat, amelyek segítségével könnyebben megoldhatók a bonyolult feladatok.

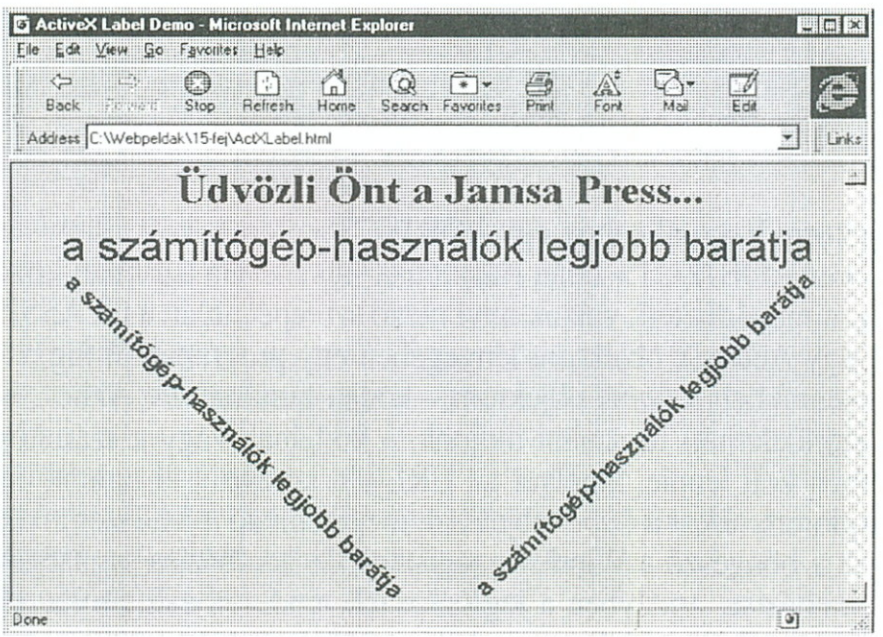
Ami a VBScript nyelvet illeti, az OLE objektumok, vagy ahogyan ezeket manapság nevezik, az ActiveX vezérlőelemek közvetlen módon lehetővé teszik, hogy a programozók kibővítsék a böngészők képességeit, és ezáltal még nagyobb tudású Web helyeket lehessen készíteni. A jövőben a programozók ActiveX vezérlőelemeket fognak használni VRML objektumok kezeléséhez, egy hely titkosítási és biztonsági kérdéseivel kapcsolatos feladatok megoldásához, hitelkártyás tranzakciók biztonságos lebonyolításához, multimédiás animációkhoz és sok minden máshoz. Teljes körű áttekintést kaphatunk az ActiveX vezérlőelemekről, ha felkeressük a *http://www.microsoft.com/intdev/controls/ctrlref-f.htm* helyet (lásd a 15.18. ábrát).

Az ActiveX technológia megértésének egyik legjobb módja, ha programpéldákat tanulmányozunk. Mi itt most a *Label* nevű ActiveX objektumot fogjuk használni, ami a Microsoft Internet Explorer telepítésekor kerül be a rendszerünkbe. (A fájlt gyorsan megtalálhatjuk, ha az .OCX kiterjesztésű fájlokat keressük a merevlemezünkön. A fájl neve *ielabel.ocx*)

A *Label* vezérlőelem lehetővé teszi, hogy a script fájl egy Web-oldal különböző helyein, adott betűtípussal, betűstílussal, méretben stb. jelenítsen meg szöveget. A 15.19. ábra például azt szemlélteti, hogy miként használható a *Label* vezérlőelem a *Jamsa Press... a számítógép-használók legjobb barátja* szöveg különböző helyen, különböző betűtípusokkal és szögekben történő megjelenítéséhez.



15.18. ábra. Információk beszerzése az ActiveX technológiáról



15.19. ábra. Szöveg megjelenítése a *Label* vezérlőelem segítségével

Ez a Web-oldal egy <H1> címsor-elemet és három *Label* objektumot használ a szöveg megjelenítéséhez. Az alábbi, *ActXLabel.html* nevű HTML fájl tartalmazza a címkék (labels) megjelenítéséhez szükséges HTML utasításokat:

```
<HTML>
<HEAD><TITLE>ActiveX Label Demo</TITLE></HEAD>
<BODY><H1 ALIGN=CENTER>Üdvözli Önt a Jamsa Press...</H1>

<OBJECT
CLASSID="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2"
id=MyLabel1 width=600 height=50 align=center>
<PARAM NAME="caption" VALUE="a számítógép-használók legjobb barátja">
```

```
<PARAM NAME="FontSize" VALUE=24>
</OBJECT>

<OBJECT
CLASSID="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2"
id=MyLabel2 width=300 height=250 align=left>
<PARAM NAME="caption" VALUE="a számítógép-használók legjobb barátja">
<PARAM NAME="angle" VALUE="315">
<PARAM NAME="FontSize" Value="14">
<PARAM NAME="FontBold" Value="1">
</OBJECT>

<OBJECT
CLASSID="clsid:99B42120-6EC7-11CF-A6C7-00AA00A47DD2"
id=MyLabel3 width=300 height=250 align=LEFT>
<PARAM NAME="caption" VALUE="a számítógép-használók legjobb barátja">
<PARAM NAME="angle" VALUE="45">
<PARAM NAME="FontSize" Value="14">
<PARAM NAME="FontBold" Value="1">
</OBJECT>
</BODY></HTML>
```

Egy ActiveX objektum használatához egy <OBJECT> elemet kell elhelyezni a HTML fájlban. Első ránézésre egy ilyen <OBJECT> talán riasztónak látszik. Ha azonban közelebbről szemügyre vesszük, akkor könnyen megérthetjük valamennyi mező szerepét, talán a CLASSID elem kivételével.

Amint említettük, egy ActiveX objektum lényegében egy programozói függvénytár, amely különböző objektumok – esetünkben a *Label* objektum – kódját tartalmazza. Ahhoz, hogy egy ilyen függvénytárt (ActiveX vezérlőelemet) használni tudjunk, a kódjának vagy a számítógépünkön, vagy egy olyan távoli rendszerben kell lennie, amelyet az URL-jén keresztül el tudunk érni. Ha telepítettük a Microsoft Internet Explorert, akkor a *Label* objektum megtalálható a rendszerünkben. A helyben telepített vezérlőelem helyét a böngésző a Windows regisztrációs adatbázisa segítségével találja meg (ez a Windowsnak az az adatbázisa, amelyben a legfontosabb rendszerbeállításokat tárolja). A CLASSID mezőben szereplő számjegyek hosszú sora a *Label* vezérlőelem regisztrációs száma. A rendszerünkbe telepített minden egyes vezérlőelemnek egyedi, másoktól különböző regisztrációs száma van. A vezérlőelemhez tartozó dokumentáció általában tartalmazza a vezérlőelemmel együtt használandó számot.

A CLASSID mezőt követően az <OBJECT> elemben különböző attribútumok értékei adhatók meg, mint például az objektum helye, mérete stb. Mindegyik ActiveX objektumnak saját attribútumkészlete van. Az attribútumok beállításait és a használható paramétereket ugyancsak az objektum dokumentációja tartalmazza.

Kísérletezzünk egy kicsit ezzel a HTML dokumentummal *(ActXLabel.html)*. Változtassuk meg a feliratot, a betűméretet és más jellemzőit. Néhány percnyi kísérletezést követően bár bizonyára nem fogjuk olyan ijesztőnek tartani az ActiveX objektumot.

ACTIVEX VEZÉRLŐELEMEK LELŐHELYEI

Mivel a Microsoftnak komoly érdekei fűződnek az ActiveX technológia terjesztéséhez, ideális információforrás az ActiveX vezérlőelemek tekintetében. Mellette még számos más szoftver-

fejlesztő cég is kínálja az ActiveX objektumait korlátozott „kipróbálási időre". Jó néhány ilyen helyet megtalálhatunk, ha a *http://www.yahoo.com* címen elérhető „keresőgéppel" az „ActiveX" szót kerestetjük. Végül valamelyik programozási nyelven, például Visual C++ nyelven saját ActiveX vezérlőelemeket is készíthetünk.

Az ActiveX *Label* vezérlőeleme mellett a *Timer* (időzítő) vezérlőelem is megtalálható a rendszerünkben, ha telepítettük a Microsoft Internet Explorer böngészőt. A most következő HTML fájl ezt a *Timer* vezérlőelemet használja egy egyszerű időzítési esemény kiváltására. Amikor betöltjük a dokumentumot, megjelenik az „Üdvözli Önt ..." szövegű üzenet, majd az időzítő megkezdi az 5 másodperces visszaszámlálását. Miután letelt a beállított idő, az „Üdvözli Önt...a Jamsa Press" szöveg jelenik meg. Az *ActiveXTimer.html* fájl az alábbi utasításokat tartalmazza:

```
<HTML><HEAD><TITLE>ActiveX Timer Demo</TITLE>
</HEAD><BODY>
<H1 ALIGN=CENTER>Üdvözli Önt...</H1>

<OBJECT
CLASSID="clsid:59ccb4a0-727d-11cf-ac36-00aa00a47dd2"
id=TimerEvent>
<PARAM NAME="Interval" VALUE="5000">
<PARAM NAME="Enabled" VALUE="-1">
</OBJECT>

<SCRIPT LANGUAGE="VBSCRIPT">
<!- A kód elrejtésének kezdete

sub TimerEvent_timer
  document.write("<H1 ALIGN=CENTER>Üdvözli Önt a...Jamsa Press</H1>")
  TimerEvent.Enabled = 0 ' Az idõzítõ kikapcsolása
end sub
//A kód elrejtésének vége->
</SCRIPT></BODY></HTML>
```

Amint látható, amikor letelik a beállított idő, a script fájl meghívja a *TimerEvent_Timer* szubrutint, hogy kezelje az eseményt. A szubrutin az eseményre egyszerűen az *Üdvözli Önt a...Jamsa Press* üzenet megjelenítésével reagál. Ezt követően a szubrutin az *Enabled* mező hamis értékre állításával kikapcsolja az időzítőt. Ha a script fájl nem kapcsolná ki az eseményt, akkor az időzítő alapbeállítás szerint automatikusan ismételné magát újra és újra.

ÖSSZEFOGLALÁS

Ebben a fejezetben a számítógépes programozás két, nagyon gyors ütemben fejlődő területével ismerkedtünk meg: a VBScript nyelvvel és az ActiveX objektumokkal. Ezeket az eljárásokat használva a Web alapú alkalmazások olyan, nagy teljesítményű szoftvereket vehetnek igénybe, amelyek magán a Windowson belül állnak rendelkezésre. Ahogyan a Microsoft tovább növeli az Internet Explorer tudását, egyre több és több olyan innovatív Web hellyel fogunk találkozni, amelyek

hasznosítják ezt a két eljárást. Ezzel a fejezettel befejezzük a Web programozás területén tett kirándulásunkat. Mielőtt azonban az itt megszerzett tudásunkra építve VRML jelenetsorok tervezésével, CGI script fájlok írásával vagy beépített böngészővel rendelkező alkalmazások készítésével folytatnánk a munkánkat, győződjünk meg arról, hogy értjük az alábbiakat:

- A VBScript a Microsoft által a Web programozásához kifejlesztett script nyelv.
- A Perl nyelvhez hasonlóan VBScript nyelven is készíthetünk CGI alkalmazásokat. Ennél fontosabb azonban, hogy a JavaScript nyelvhez hasonlóan VBScript nyelven is olyan script fájlokat írhatunk, amelyeket a böngészők futtatnak.
- VBScript nyelven úgy írhatunk script fájlokat, hogy VBScript utasításokat helyezünk el egy HTML fájlba, amit azután a böngésző hajt végre.
- A VBScript nyelvű script fájlok egyetlen (de meglehetősen nagy) hátránya, hogy jelenleg csak a Microsoft Internet Explorer böngészője képes a végrehajtásukra.
- A VBScript nyelvű script fájlok tudása hatalmas mértékben megnőtt az ActiveX vezérlőelemek révén – ez utóbbiak az OLE objektumok továbbfejlesztései.

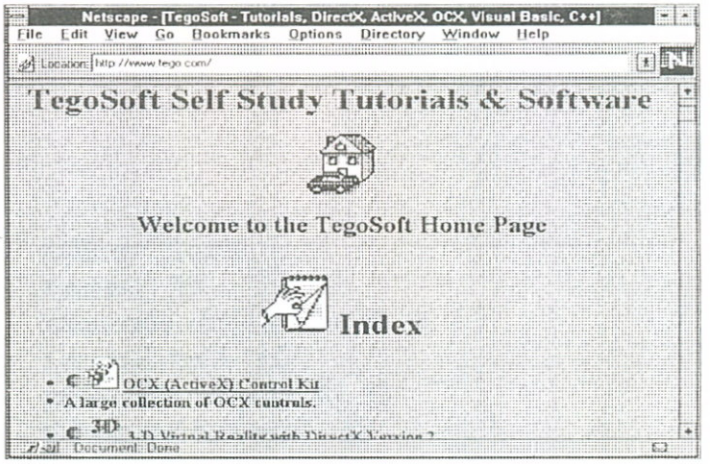
VBSCRIPT NYELVŰ SCRIPT FÁJLOK ÍRÁSÁVAL ÉS ACTIVEX OBJEKTUMOK HASZNÁLATÁVAL FOGLALKOZÓ FONTOSABB HELYEK

A következő Web helyek segíthetnek abban, hogy részletesen olvashassunk a VBScript nyelv specifikációjáról, az ActiveX szoftverfejlesztő készletről, továbbá információkat kapjunk az ActiveX vezérlőelemekről, az Internet Database Connectivity használatáról, VBScript nyelven megírt űrlapokról és sok másról. A helyeket kiindulási pontként is használhatjuk a Web felfedezéséhez.

Welcome to ActiveXtra.com

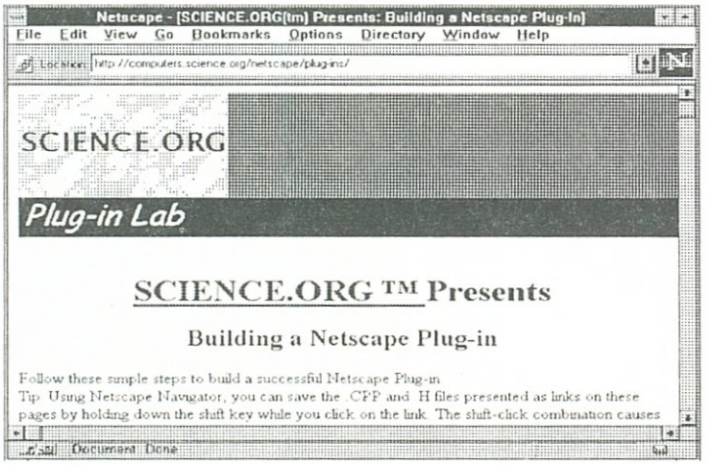
http://www.activextra.com/

TegoSoft – Tutorials, DirectX, ActiveX, OCX, Visual Basic, C++



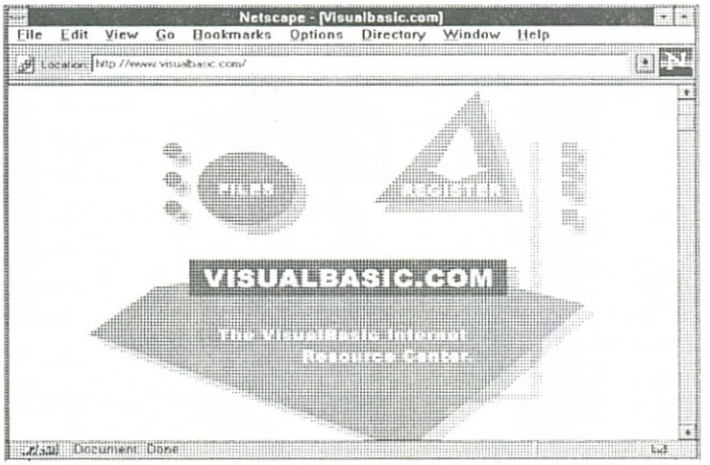
http://www.tego.com/

SCIENCE.ORG(tm) Presents: Building a Netscape Plug-In



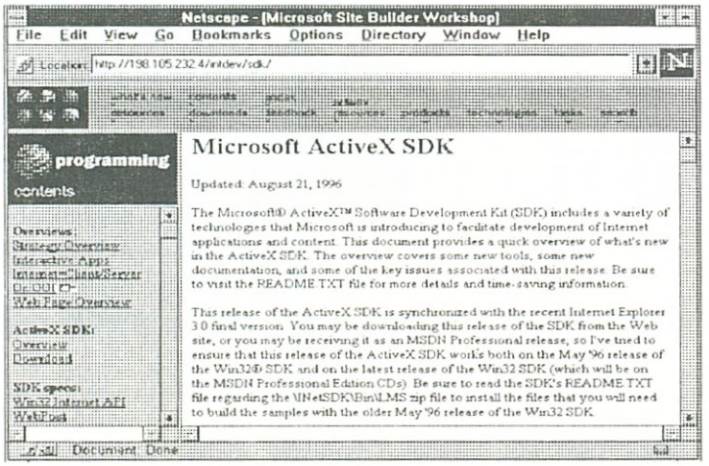
http://computers.science.org/netscape/plug-ins/

Visualbasic.com



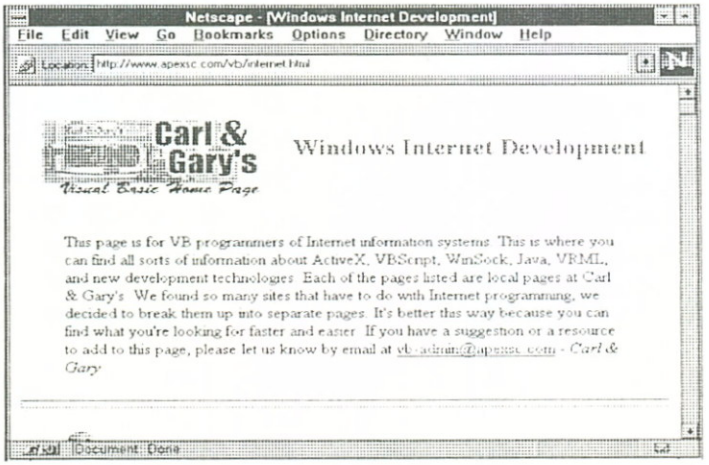
http://www.visualbasic.com/

Microsoft Site Builder Workshop



http://198.105.232.4/intdev/sdk/

Windows Internet Development



http://www.apexsc.com/vb/internet.html

TÁRGYMUTATÓ

E

F

G

H

I

J

K

L

M

S

SZ

T

U, Ü

V

W

TARTALOM

10 FEJEZET

BEVEZETŐ A VRML NYELVBE

11. FEJEZET

CGI PROGRAMOZÁS

12. FEJEZET

A PERL PROGRAMOZÁSI NYELV

13. FEJEZET

CGI SCRIPT FÁJLOK KÉSZÍTÉSE PERL NYELVEN

14. FEJEZET

A WEB PROGRAMOZÁSA JAVA ÉS JAVASCRIPT NYELVEN

15. FEJEZET

A WEB PROGRAMOZÁSA VBSCRIPT ÉS ACTIVEX SEGÍTSÉGÉVEL

AZ ELSŐ KÖTETBEN LEVŐ CD-ROM A KÉT KÖTETBEN SZEREPLŐ VALAMENNYI MINTAPÉLDÁT TARTALMAZZA, VALAMINT EGY JAVA ÉS EGY PERL NYELVŰ PROGRAMOZÁSI KÖRNYEZETET.

AZ ELSŐ KÖTET TARTALMÁBÓL:

FELELŐS KIADÓ KOCSIS ANDRÁS SÁNDOR
A KOSSUTH KIADÓ RT. ELNÖK-VEZÉRIGAZGATÓJA
A KÖTETET KORONCZAI MAGDOLNA SZERKESZTETTE
MŰSZAKI VEZETŐ KUN GÁBOR
MŰSZAKI SZERKESZTŐ PÁNYI BÉLA
TERJEDELME 23,5 (A/5) ÍV
A TÖRDELÉS A KOSSUTH KIADÓBAN KÉSZÜLT
NYOMTA ÉS KÖTÖTTE A SZEKSZÁRDI NYOMDA KFT.
FELELŐS VEZETŐ VADÁSZ JÓZSEF IGAZGATÓ

A WEB PROGRAMOZÁSA II.

Napjainkban a World Wide Web (WEB) az Internet legfrekventáltabb területe. A világhálón kalandozók naponta újabb és újabb Web-helyeket fedezhetnek fel világszerte. A tervezők egymással versenyeznek, ki tud érdekesebb, izgalmasabb, képet, hangot, animációt is magában foglaló Web-oldalakat készíteni. A kereskedelem is felfedezte magának a Web-oldalak interaktivitásában rejlő lehetőségeket. Ez a könyv megismerteti az olvasóval az internetes kommunikálás alapjait, a Web-oldalak készítéséhez használt HTML nyelvet. Bemutatja a különböző böngésző, kiszolgáló és kereső programok működését, és segítséget nyújt elkészítésükhöz. A könyv olvasója megtudhatja, hogy a C++, Perl, JavaScript és VBScript nyelvű fájlok segítségével hogyan készíthetők CGI programok, melyek interaktív kapcsolatot teremtenek a Web-oldalak olvasóival. A Web programozásának olyan új és érdekes területei is megismerhetők a könyvből, mint a 3D objektumok készítésére alkalmas VRML nyelv vagy az ActiveX vezérlőelemek használata.
A megértést számos, a gyakorlatban is kipróbálható példaprogram és ábra segíti.
A programok forráskódjaikkal együtt megtalálhatók a CD-mellékleten, amely a Java és a Perl nyelv fejlesztőkészletét is tartalmazza.

KOSSUTH KIADÓ

ISBN 963-09-3930-4

9 789630 939300