GNU/Linux segédprogramok használata
1.0

Terék Zsolt terek@cs.bme.hu

2002. július 9.

A módosított változat közzétételéért felelős személyek:

Sári Gábor saga@lme.linux.hu

Javítások: Sári Gábor

Szerző

Terék Zsolt terek@cs.bme.hu

Szakmai lektor

Nyelvi ellenőrzés

Sári Gábor saga@tux.hu

Formázás (LaTeX)

Sári Gábor saga@tux.hu
Kósa Attila atkosa@shinwa.hu

Előzmények

Segédprogramok használata

A kiadás éve: 2002.

Szerző

Terék Zsolt terek@cs.bme.hu

Szakmai lektor

Nyelvi ellenőrzés

Sári Gábor saga@tux.hu

Formázás (LaTeX)

Sári Gábor saga@tux.hu
Kósa Attila atkosa@shinwa.hu

Tartalomjegyzék

1. fejezet

Segédprogramok használata

1.1. Bevezetés

Jelen füzet bevezetést kíván nyújtani a Linux operációs rendszerekben elérhető segédprogramok használatának megismeréséhez. Főként szövegfeldolgozásra használható programozási nyelvek és eszközök ezek, amelyek a hetvenes évek UNIX rendszereinek tervezésekor születtek. Különböző változataik terjedtek el, amelyeket végül a POSIX szabvány létrehozása során egyesítettek. A Linuxba a GNU változatok kerültek, amelyek teljes egészében kompatibilisek a POSIX szabvánnyal, és néhol kiegészítéseket is tartalmaznak.

A GNU projekt célja, hogy ingyenesen, bárki számára hozzáférhető szoftver csomagot hozzanak létre. Ezt az Free Software Foundation (FSF) szervezet irányítja. Programjaik különböző rendszerekben is hozzáférhetők, sőt, azok forráskódját is bárki megnézheti, módosíthatja, felhasználhatja. Egyedüli megszorítás, hogy a módosított változatot is ugyanilyen feltételekkel mindenki számára elérhetővé kell tenni. Ez a GNU Public Licence (GPL). A GNU egyébként egy rekurzív rövidítés: GNU is Not Unix.

A második fejezet a reguláris kifejezéseket ismerteti. Ezek a matematikai formális nyelvészet eredményeinek felhasználásával rendkívül kényelmes, jól használható segédeszközt jelentenek szöveges adatok vizsgálatához, elemzéséhez és feldolgozásához. Valamennyi segédprogram kihasználja a reguláris kifejezések nyújtotta lehetőségeket.

A harmadik fejezet az `ed` nevű szövegszerkesztőt mutatja be. Ez némileg eltér a szövegszerkesztők mai fogalmától, de annak idején nem is pontosan ilyen célok vezérelték kiagyalóit. Nehézkesnek és nyakatekertnek tűnhet elsőre, de ha a megfelelő helyen és a megfelelő feladatokra használják, jó szolgálatot tehet. Ezért is nélkülözhetetlen bemutatása egy általános ismertetőben.

A negyedik fejezetben az `ed`-re meglehetősen hasonlító, de alkalmazási céljaiban különböző szövegfeldolgozó segédprogram, a `sed` használatával foglalkozunk. A stream editort, vagyis a folyamszerkesztőt a UNIX-szerű rendszerek számos szkriptje is gyakran segítségül hívja, ha egyszerűbb szövegkonverziók végrehajtása szükséges. Amellett, hogy hasznos, még rendkívül szellemes is lehet egy-egy `sed` nyelven írt, pár karakterből álló programocska.

Az ötödik fejezet az `awk` programozási nyelv részleteibe vezeti be az olvasót. Az `awk` nyelv is alapjába véve szövegmanipulációs célokra lett tervezve, de azért itt már

komolyabb programozási szerkezetek is vannak, amelyek legtöbbje a C nyelvből lehet ismerős. Ugyanakkor sokat „örökölt” a `sed` és az `ed` szemléletéből is.

A hatodik rész a `Perl` bemutatására hivatott. A `Perl` mára már univerzális programozási nyelvvé fejlődött: modularizált, objektum-orientált és akár grafikus felület programozására is alkalmas. Átfogó ismertetése túlmutat ezen füzet keretein, mindössze azon aspektusait vizsgáljuk, amelyek az előző fejezetek ismeretében könnyen és gyorsan megérthetők.

A fejezetek felépítésére jellemző, hogy először általános áttekintést adnak a bemutatott dologról. Az ezután következő részek, amelyek akár referenciaként is szolgálhatnak, sorban bemutatják az adott segédprogram funkcióit. Ez első olvasásra nehezebbnek tűnhet, de ez nem baj, esetleg többször átfutva összeáll majd a kép. Végül alkalmazási példákkal zárjuk a fejezetet, amelyek az előzőleg bemutatott anyag gyakorlatba ültetését illusztrálják. A példák mellé fűzött megjegyzések segíthetnek azoknak, akik a részletes ismertető előtt szívesebben böngészik a működő szkripteket.

1.2. Sztringek és speciális karakterek

Ha egy karaktersorozatot szeretnénk egy programnak megadni, azt nem biztos, hogy pontosan ugyanúgy kell leírnunk. Tekintsük a parancssori programindítás példáját. Ennek hagyományos formája:

```
programnév par1 par2 ... parn
```

vagyis a program neve után szóközzel elválasztva, néhány paramétert sorolunk fel. A paraméterek karaktersorozatok, vagyis sztringek, ám nem tartalmazhatnak tetszőleges karaktereket. Pontosabban, nem olyan egyszerűen.

Az itt vázolt szabályok a `bash` parancsértelmezőre vonatkoznak. Ez talán a legelterjedtebb shell, ezért ennek a karakterkezelését mutatjuk be. Máshol is nagyon hasonló módon bánnak a speciális karakterekkel, legalábbis az irányelvek ugyanezek.

Képzeljük el azt a helyzetet, hogy programunk első paramétereként egy több szóból álló sztringet szeretnénk megadni. Ha ezt

```
programnév első paraméter
```

alakban írjuk, honnan kellene tudnia a parancsértelmezőnek, hogy nem két paraméterrel hívjuk a programot, ahol az első az a szó, hogy első, a második pedig az, hogy paraméter? A helyzet nyilván nem egyértelmű, viszont a bash valahogy mindig értelmezi, méghozzá következetesen: a szóköz elválasztja a paramétereket. Valamilyen más, cselesebb módon kell megfogalmazni eredeti akaratunkat.

Az egyik karakternek speciális jelentése van. Ez, csakúgy mint legtöbb más nyelvben a backslash, vagyis a fordított perjel: \. Ha ezt a karaktert olvassa a `bash`, akkor tudni fogja, hogy sem ez, sem a következő karakter nem saját magát jelöli, hanem valamilyen „metajelentése” van. Az angol nyelvből átvéve szokás ezeket escape-szekvenciáknak nevezni (A backslash-t és az azt követő karaktert együttvéve.) Egy backslash és egy szóköz együttvéve olyan szóközt jelent, amely része a paraméternek. Tehát az előbbi parancssor helyesen:

```
programnév első\ paraméter
```

Arra a kérdésre, hogy miként lehet backslash karaktert írni egy paraméterbe, egyszerű a válasz: két backslash egy „igazi” backslash-t jelent. Ezen kívül a \n az újsor karakter megfelelője, a \t pedig a tabulátoré. Ha olyan karakter szerepel a backslash után, amelyhez nem tartozik különleges escape-szekvencia, akkor egyszerűen az adott karakter kerül a helyére.

Ugyancsak elterjedt konvenció, hogy backslash-re egy sor végén azt jelenti, hogy a következő sort is tekintse az értelmező ezen sor folytatásának. (Természetesen a jelző \\ karakter elvételével.)

A shell számára a dollárjel is különleges jelentéssel bír. Segítségével hivatkozhatók a megfelelő nevű környezeti változók. Ezek általában arra használhatók, hogy a programok a rendszer konfigurációjáról információt szerezzenek. Például $PATH egy olyan sztringet tartalmaz, amelyben kettősponttal elválaszta fel vannak sorolva azon könyvtárak, ahol a futtatható programok nagy része elhelyezkedik. Természetesen, ha a dollárjelet szeretnénk használni különleges értelmezés nélkül, egy backslash eléírásával ezt is megtehetjük.

Paraméterek specifikálását segíthetik az idézőjelek használata. Az egyik fajta a kettős idézőjel ("). Két ilyen jel közé írt karaktereket egyetlen paraméternek tekinti a shell, tehát így is megoldható a szóköz használata:

```
programnév "első paraméter" második
```

Persze maguk az idézőjelek nem részei annak a sztringnek, amit paraméterként megkap a hívott program. Ilyen macskakörmök (double quotes) között érvényesek az escape-szekvenciák, vagyis a backslash speciális jelentésű. Ennek köszönhetően lehet például " karaktert írni a paraméterbe anélkül, hogy az értelmező azt hinné, ez már a lezáró macskaköröm.

```
programnév "Egy idézet:\n"Minden egér szereti a sajtot.\""
```

Ez valójában egy kétsoros sztringnek felel meg, amely második sora idézőjellel kezdődik és végződik.

Egyszeres idézőjelek, aposztrófok (’) is alkalmasak szöveg keretezésére. A legfontosabb eltérés azonban az, hogy ebben már nem érvényes a \ vagy a $ karakterek speciális jelentése. Minden karakter „saját magáért” áll a szövegben. Ennek következtében nincs mód aposztróf használatára, hiszen az mindenképp záró aposztrófként kerül értelmezésre. Viszont cserébe nem kell figyelni a szóközökre, különleges karakterekre. Ha több soros paramétert akarunk megadni a parancssorban, ENTER-t is használhatunk:

```
programnév ’Több soros paraméter,
> enter nyomása után sem lesz
> vége a parancssornak,
> csak ha lezárjuk egy aposztróffal.’ második
```

A sorok elejére a > jelet a shell írta ki, jelezve, hogy többsoros parancsot adunk ki.

Még egy nagyon érdekes tulajdonság hátra van. Ha fordított idézőjelek (backquotes: ‘) közé írt szöveg szerepel valahol, a shell azt futtatandó parancsként értelmezi. Végrehajtja, majd a futás során kapott kimenetet helyettesíti a ‘jelekkel határolt részre. Például a `programnév ‘ls‘` az `ls` program kimenetével, vagyis az aktuális könyvtár tartalmának listájával, mint paraméterekkel hívja meg az adott programot. Ez ekvivalens azzal, hogy `programnév dir1/ dir2/ file1 file2 file3` Makrószerűségről van szó, tehát ha a behelyettesített szöveg szóközöket tartalmaz, az para-

méterelválasztóként kerül értelmezésre. Ha egyetlen paraméterként szeretnénk a teljes listát átadni, akkor azt `programnév ‘‘‘ls‘‘‘` formában kell írni, mivel ez megfelel az alábbi formának: `programnév "dir1/ dir2/ file1 file2 file3"` Nyilvánvalóan egyszeres idézőjelek között a fordított aposztróf is elveszti különleges hatását.

1.3. Reguláris kifejezések

Az egyes segédprogramok szintaktikájának megismerése előtt néhány szót kell ejteni a reguláris kifejezésekről. A reguláris kifejezés egy karaktersorozat, amely egy szóhalmazt jelöl. Például a `*` a csak a betűt tartalmazó szavakat jelenti.

A reguláris kifejezésekben bizonyos karakterek jelentése speciális. Ha azonban mégis egy ilyen karaktert szeretnénk használni, akkor egy fordított per jelet (backslash, \) kell eléírni. Nyilvánvalóan magára a backslash karakterre annak megduplázásával hivatkozhatunk: \\.

1.4. Alapvető műveletek

A legegyszerűbb reguláris kifejezések egyetlen betűből állnak, a . (pont) pedig tetszőleges karaktert jelent, kivéve a sorvéget. (A sorvég is egy speciális karakter a rendszer számára.) Ha most `reg1` és `reg2` reguláris kifejezés, ezekből összetettebb kifejezéseket a következő szabályok segítségével építhetünk:

reg1reg2 A konkatenáció, vagyis egymás után írás olyan szavaknak felel meg, amelyeket a `reg1` és a `reg2` által leírt szóhalmazok egy-egy szavának egymás után írásával kaphatunk. Például `a.` az `a` és a `.` kifejezések konkatenáltja, tehát olyan kétbetűs szavak halmaza, amelyek első karaktere `a`, a második pedig tetszőleges.

reg1|reg2 Ezzel a két halmaz unióját képezzük, tehát akár az egyik, akár a másik kifejezés által leírt szavak megfelelnek. Például az `ab|a` halmaz két szót tartalmaz, ab-t és a-t.

reg1* Az ismétlés operátorával minden olyan szót megkaphatunk, amely tetszőleges reg1-beli szavak egymás után írásával kapható. Az ismétlések száma nulla is lehet, tehát ennek a halmaznak eleme az üres szó. Például `a*b*` olyan szavakat jelöl, amelyekben néhány `a` betűt néhány `b` követ. Megengedett az is, hogy csak `a`, vagy csak `b` betűt tartalmazzon a szó.

reg1+ Az előzőhöz nagyon hasonló művelet, de itt az ismétlések száma legalább egy. Tehát az `a+b+` halmaz szavai mindenképp `a`-val kezdődnek, és `b`-re végződnek.

reg1? Ide `reg1` szavain kívül az üres szó tartozik, tehát a `?` operátorral az előtte álló kifejezést opcionálissá tehetjük.

A reguláris kifejezések gyakorlása érdekében gondoljuk végig az alábbiak azonosságát: `reg*, reg+?, reg?+`.

Az említett operátorok közül az ismétlők (*, +,?) műveleti elsőbbség szempontjából erősebbek a konkatenációnál, míg a leggyengébb az alternálás (|). Az azonos

erősségű operátorok balról jobbra értékelődnek ki. Gömbölyű zárójelekkel felülbírálhatók az előbbi szabályok.

Az ismétlődések számát tovább szabályozhatjuk. Ha `reg` egy kifejezés, akkor `reg{m}` jelentése az, hogy `reg` pontosan m-szer ismétlődik. Alsó és felső korlát adható az ismétlődések számára: `reg{m,n}` legalább m, legfeljebb n ismétlést ír elő. A felső korlát elhagyása (a vessző megtartásával) tetszőleges m-nél nagyobb ismétlődést megenged.

1.5. Karakterosztályok

További kényelmi lehetőséget jelent a karakterosztályok használata. Szögletes zárójelek ([és]) között megadott karaktersorozat bármelyik benne szereplő betűnek felel meg. Kötőjellel intervallumokat is megadhatunk, tehát [abcde] és [a-e] ugyanaz. Például [A-Z][A-Za-z]+ olyan legalább kétbetűs szavakat ír le, amelyek kezdőbetűje nagy, a folytatásban pedig nagy- és kisbetűk egyaránt előfordulhatnak.

Ha a karakterosztály kalap-jellel kezdődik (^), akkor a jelentés az ellentettje lesz: azok a karakterek felelnek meg a kifejezésnek, amelyek nem szerepelnek benne, intervallumként sem. Tehát a[^a]*a olyan szavakat jelent, amelyek a-val kezdődnek és végződnek, de a belsejükben nincs ilyen karakter.

Karakterosztályokon belül előre definiált csoportokat használhatunk. Ezek:

alfanumerikus karakterek, azaz betűk, számok és az aláhúzás ([:alnum:]), betűk ([:alpha:]), vezérlő karakterek ([:cntrl:]), számjegyek ([:digit:]), kisbetűk ([:lower:]), nagybetűk ([:upper:]), nyomtatható, tehát nem vezérlő karakterek ([:print:]). Fontos, hogy ezek önállóan nem karakterosztályok, csak ha maguk is szögletes zárójelek között fordulnak elő. Így például [0-9] ugyanaz, mint [[: digit:]], de [a-zA-Z] abban különbözik [[:alpha:]]-tól, hogy az előbbi csak az angol ábécé betűit ismeri.

Szögletes zárójelek között három karakternek van speciális jelentése. Ha ezeket saját maguk jelentésére szeretnénk haszni, akkor ^ ne legyen az első karakter,] viszont csak a legelső helyen kerül karakterként, és nem bezáró zárójelként értelmezésre. A kötőjel legyen a legutolsó. Például []^-] szabályos karakterosztály.

1.6. Reguláris kifejezések használata

A reguláris kifejezés karaktersorozatok egy halmazát jelenti. Azt mondjuk, hogy egy r reguláris kifejezés minta (pattern) illeszkedik (matches) egy karakterláncra, ha az r által meghatározott halmaz szavainak bármelyike előfordul az említett karakterláncban. Így például akármi illeszkedik minden olyan karaktersorozatra, amelyikben előfordul az akármi részsorozatként.

Előfordulhat, hogy egy szövegre többféleképpen is illeszthető egy reguláris kifejezés. Attól lesz egyértelmű az illeszkedés, hogy mindig a legbaloldali, vagyis a legelső illesztés az érvényes. Ha ilyenből is több van, ezek közül a leghosszabb számít. Például a bc* minta a abccbbc szövegre a második karaktertől kezdve pontosan három karakteren keresztül illeszkedik.

Az illesztésen szigoríthatunk úgy is, hogy előírjuk, az előfordulás csak a szöveg elején vagy csak a végén történhet. Ha a reguláris kifejezés kalap jellel kezdődik, az illesztés csak a szöveg elején kezdődhet, míg ha dollár jelet írunk egy kifejezés végére, akkor sikeres illeszkedés esetén a részsorozat a szöveg végéig tart. Példaként tekintsük az a+ kifejezés módosításainak lehetőségét:

a+ mindenre illeszthető, amiben előfordul legalább egy a betű

^a+ csak az a-val kezdődő szövegekre illeszthető

a+$ az a-ra végződő karakterláncokra illik

^a+$ a csupa a betűből álló szavak felelhetnek meg

A `grep` segédprogram használata teljes egészében a reguláris kifejezésekre épül. A szabványos bemenetén olvasott fájl minden sorára megvizsgálja, illeszkedik-e a paraméterként kapott minta. Amely sorok esetében sikerült az illesztés, azokat kiírja a kimenetre. Persze önmagában ez a program gyakorlatilag használhatatlan, de más egyszerű parancsokkal együtt rendkívül általános feladatokat végezhet.

Egyszerű példaként keressük meg azon fájlokat, amelyek neve b betűvel kezdődik és ugyanígy végződik! Az `ls` parancs kiírja az összes állomány nevét az aktuális könyvtárból, ezt szűrjük a grep segítségével, alkalmazva a `^b.*b$`. (Emlékeztetőül: a pont bármilyen karakterre illeszkedik.)

```
ls | grep ^b.*b$
```

1.7. Változatok, kiterjesztések

A reguláris kifejezések szintaktikájának sajnos számos, különböző változata is elterjedt. Ezért fontos, hogy adott helyzetben ellenőrizzük, a kifejezést értelmező program melyik „nyelvjárást” ismeri. Erre a legalkalmasabbak a megfelelő manual oldalak. Ugyancsak ott kaphatunk részletes információt az alkalmazható kiterjesztésekről.

A különböző változatok között néhány régebbi fordítva kezeli a zárójelezést és zárójel karaktereket. Ez azt jelenti, hogy a gömbölyű és a kapcsos zárójel hagyományos karakternek tekintendő, és speciális karakterként (amikor is műveleti elsőbbségek miatt zárójelezünk, vagy korlátozott számú ismétlést írunk elő) backslash-t kell eléjük írni. A stream editor (`sed`) ezt az értelmezést követi.

Sok helyen további kényelmi lehetőségekkel találkozhatunk, illetve bizonyos kiterjesztések is léteznek, amelyekkel az egyszerű reguláris kifejezések adta lehetőségnél több is megfogalmazható. A legfontosabb ilyen kiterjesztés a hivatkozás. A zárójelek nemcsak a műveletek sorrendjének szabályozására alkalmazhatók, hanem a reguláris kifejezés illesztésekor a szöveg egy részére illeszkedik a zárójelben lévő részkifejezés. Az első kilenc zárójel által illesztett részek hivatkozhatunk vissza a következőkkel: `$1, $2,..., $9`.Ezzel ismétlődést írhatunk elő az illesztett szövegben. Például `\^(.*)$1$` egyszerűen azokra a szövegekre illik, amelyek valamilyen szöveg megismétlésével kaphatók (baba, dada).

A `sed` ettől eltérően, dollár helyett backslash karaktereket használ a visszahivatkozásra.

1.8. Példák

Néhány egyszerű reguláris kifejezés a hozzáfűzött magyarázattal segíthet a megértésben.

a*b nulla vagy több a, melyeket egy b követ

a?b nulla vagy egy a karakter, melyet egy b követ

a+b+ legalább egy a, melyet legalább egy b karakter követ

.* egy akármilyen sor összes karaktere

.+ illeszkedik minden nemüres sorra (mivel legalább egy nem sorvég karakternek kell lennie az illeszkedő szövegben)

^main.*.* olyan sorokra illeszkedik, melyek main szóval kezdődnek, és szerepel bennük egy nyitó-csukó zárójel pár valahol

^# a # karakterrel kezdődő sorok, melyek legtöbb szkript-nyelvben a kódhoz fűzött megjegyzést jelentik

$ backslash karakterre végződő sorok (két \ kell, mert speciális jelentésű karakterről van szó)

^.0,15A Ez olyan sorokra illeszkedik, amelyek tizenhat karaktere között van A betű. A reguláris kifejezések illeszkedésénél „legbaloldalibb, leghosszabb” szabálya miatt ilyenkor az illeszkedés az 16 karakter utolsó A betűjéig tart. Tehát a .0,15 rész olyan hosszúra nyúlik, amilyenre csak tud, ha több illeszkedés is lehetséges.

2. fejezet

Az ed szövegszerkesztő

Az `ed` egy sor szemléletű szövegszerkesztő, amely segítségével létrehozhatunk, módosíthatunk szöveges fájlokat mind interaktív módon, mind szkriptből. Mivel ez a Unix rendszerek eredeti szövegszerkesztője, szinte mindenhol hozzáférhető. Ennek ellenére legtöbb esetben valamely teljes képernyős szövegszerkesztőt célszerűbb használni. Kivételt jelentenek az olyan feladatok, ahol például sok fájlon kell olyan módosításokat végezni, amelyeket nehézkes volna interaktívan megcsinálni, viszont könnyen algoritmizálhatók.

Az `ed` két alapvető működési móddal rendelkezik, ezek a parancs és a beviteli módok. Parancs módban megadott utasítás hatására, amennyiben szöveget szúrunk be, beviteli módba vált. Beviteli módban maradunk addig, amíg végül egy mindössze egyetlen pontot tartalmazó sort gépelünk be. Ilyenkor visszakerülünk parancs módba.

Például az `a` parancs (append) az aktuális sor után szúrja a megadott szöveget.

```
ed
a
Ezeket a sorokat
a pufferbe írja.
Egészen eddig tart.
.
,p
Ezeket a sorokat
a pufferbe írja.
Egészen eddig tart.
Az ,p utasítással kilistázhatjuk a puffer teljes tartalmát.
```

A parancsok általános formája a következő:

```
[cím [,cím]] parancs [paraméterek]
```

Egy vagy két cím megadásával a parancs végrehajtásának helyét, illetve intervallumát adhatjuk meg. Amennyiben nem adunk meg címzést, az aktuális sor vagy intervallum lesz az, amin a művelet végrehajtásra kerül. A parancs egyetlen betűből áll, amelyet paraméterek követhetnek.

Az aktuális sor, amelyen a címzés nélküli parancsok alapértelmezés szerint dolgoznak, a műveletek végrehajtása után megváltozik. Általában a legutoljára megváltoztatott sor lesz aktuális, kivéve azokban az esetekben, amikor törlésről van szó. Az egyes parancsok leírásánál erre még kitérünk. Közvetlenül indítás után a beolvasott fájl utolsó sora lesz az aktuális.

2.1. Címzés

Az ed szövegszerkesztőben a címzéssel egy sort jelölhetünk ki a pufferben. Ezzel adjuk meg, hogy a parancsok a szöveg mely részein végezzenek módosításokat.

A parancs előtt vesszővel vagy pontosvesszővel elválasztott címlista utolsó egy vagy két címét használja fel a parancs attól függően, hogy egy soron, vagy egy intervallumon dolgozik. Az alábbi módokon adhatunk meg címeket:

. A puffer éppen aktuális sora.

$ A puffer utolsó sora.

n A puffer n-dik sora, ahol n 0 és $ közötti érték.

- vagy ^ Az aktuális sort megelőző sor. Hatása -1-gyel egyező, és többszörösen alkalmazható.

-n vagy ^n Az aktuális sort n-nel megelőző sor, ahol n nemnegatív egész.

+ A következő sor. Ekvivalens +1-gyel és többszörösen alkalmazható.

+n vagy szóköz n Az aktuális utáni n-dik sor, ahol n nemnegatív egész. Szóközt követő szám ugyanígy értelmezhető.

, vagy % A puffer első sorától az utolsóig tartó intervallum. Ugyanaz, mint 1,$

; Az aktuális sortól az utolsó sorig tartó intervallum. Ugyanaz, mint .,$

/re/ Az aktuális sortól indulva a következő olyan sor, amelyre illeszkedik a megadott reguláris kifejezés. A puffer végére érve a keresés az elején folytatódik. Az előző keresés megismételhető megadásával.

?re? Az aktuális sortól kezdve visszafelé indulva az első olyan sor, amelyre illeszkedik a megadott reguláris kifejezés. A puffer elejére érve a keresés a legutolsó sortól folytatódik, ily módon körbe-körbe is lépkedhetünk a pufferben. Az előző keresés megismételhető megadásával. Tehát ha egy bonyolult reguláris kifejezés alapján keresünk, majd még egyszer megismételnénk ezt a keresést, nem kell újra begépelni az előző kifejezést, hanem egyszerűen megismétli azt.

'b Az előzőleg k paranccsal elhelyezett könyvjelző által mutatott hely, ahol b a könyvjelzőt azonosító kisbetű.

2.2. Parancsok

Minden parancs egyetlen karakterből áll, amely után paramétereket adhatunk át. Ha több sorba akarjuk tördelni a paramétereket, az utolsó sor kivételével mindegyik backslash karakterre kell, hogy végződjön. Ezzel olyan hatás érhető el, mintha az összes sor egyetlen parancssor volna, kivéve persze a sorvéget jelző \ karaktereket.

Általában egyetlen parancs szerepel egy sorban. Kivétel lehet néhány, amelyek megengedik p, l vagy n hozzáírását a parancshoz, mellyel a végrehajtás után információt szerezhetünk az utolsó módosított sorról.

Az ed a következő parancsokat ismeri. A parancsok megadása előtt zárójelben feltüntettük az alapértelmezésként használt címet vagy intervallumot, amelyet a parancs használ, ha a felhasználó semmit sem ad meg.

(.)a (append) A megcímzett sor után írja azt a szöveget, amelyet a parancs végrehajtása után beviteli módban megadunk. Lehetséges 0 cím megadása, ilyenkor az első sor elé kerül a beszúrt szöveg. A végrehajtás után az aktuális sor az utolsó beszúrt sor lesz.

(.,.)c (change) A megcímzett sor vagy intervallum törlődik, és helyére az előző parancsnál leírtakhoz hasonlóan bekerül a beviteli módban megadott szöveg. Az aktuális sor az utolsó beszúrt sor lesz.

(.,.)d (delete) Törli a megadott sorokat. Ha az aktuális sor is a töröltek között van, akkor a törölt részt megelőző sor lesz aktuális, egyébként nem változik.

e fájlnév (edit) Megnyitja a megadott fájlt, a pufferbe helyezi, annak előző tartalmát felülírva. Ha nem mentettük a puffert az utolsó módosítás óta, figyelmeztetést kapunk.

e !parancs A puffer tartalmát felülírja a megadott parancs végrehajtásakor a szabványos kimenetre írt adatokkal.

E fájlnév Feltétel nélkül megnyitja a megadott fájlt. Az e-hez hasonló, de itt elvesznek a nem mentett módosítások.

f fájlnév Az alapértelmezés szerinti fájlnév beállítása.

(1,$)g/re/ parancs-sorozat Minden sorra, amelyre a megadott reguláris kifejezés illeszkedik, végrehajtja a parancs-sorozat utasításait úgy, hogy a végrehajtás előtt az aktuális sor az illeszkedő sor lesz. Végül az utolsó módosított sor válik aktuálissá.

A parancs-sorozat minden utasítása külön sorban kell, hogy szerepeljen, és az utolsót kivéve minden sornak backslash (\) karakterre kell végződnie. Tetszőleges parancs megengedett, kivéve `g, G, v és V`. Üres sor a parancs-sorozatban megfelel egy p parancsnak. Ha az ed-et `-G` opcióval indítottuk, akkor az üres sorok +1p-nek felelnek meg.

(1,$)G/re/ Interaktív módon szerkeszti a megcímzett rész azon sorait, amelyekre illeszkedik a megadott reguláris kifejezés. Minden egyes találat esetén az aktuális sor arra a sorra állítódik, és a felhasználó megadhatja az azon a soron végrehajtandó parancsot az előzőekben látott parancs-sorozat formájában. A `G` parancs befejeztével az utoljára módosított sor válik aktuálissá.

Újsor karakter üres parancs-sorozatnak felel meg, míg egyetlen & beírásával az előző nem üres parancs-sorozat ismételhető meg.

H Hibaüzenetek kiírását változtatja. (igen/nem között)

h (help) Az utolsó hiba magyarázatát írja ki.

(.)i (insert) Szöveget szúr be az aktuális sor elé. Az aktuális sor a legutolsó beírt sor lesz.

(.,.+1)j (join) Egyesíti a megadott sorokat. A kijelölt rész törlődik, és helyette az egyetlen sorként, a törölt sorok egyesítése kerül be. (Sorvég jelek törlődnek.) Az így kapott sor lesz aktuális.

(.)kb Elhelyez egy könyvjelzőt, amelyet a kis b betűvel jelöl. Erre hivatkozhatunk címzésnél 'b formában. A könyvjelző nem törlődik amíg az általa mutatott sort nem töröljük, vagy felül nem írjuk magát a könyvjelzőt. Mivel a könyvjelzőket egyetlen karakter azonosítja, amely csak kisbetű lehet, összesen 26 könyvjelző használható egyszerre. (Csak az angol ábécé betűi megengedettek.)

(.,.)l (list) Kiírja a megadott sorokat, dollárjellel jelezve azok végét. Ha a képernyőre nem fér ki egy sor, azt új sorban folytatja, de a törést backslash karakterrel jelzi. A 128-nál nagyobb kódú karakterek (pl. ékezetes magyar betűk) backslash után oktális kódjukkal kerülnek a kiírásra. Ha a terminál képernyő alsó sorát eléri, újsor karakterre vár a további írásig. Aktuálissá válik a legutoljára kiírt sor.

(.,.)m(.) (move) A megadott intervallumban lévő sorokat a paraméterként kapott címmel jelzett sor mögé mozgatja. Az utoljára módosított sor lesz aktuális. Ha a paraméterként kapott cím 0, akkor a puffer elejére kerül a mozgatott szöveg.

(.,.)n (numbered print) A megadott sorokat sorszámozva kiírja. Hasonlóan a többi kiíró parancshoz, az utoljára megjelenített sor lesz aktuális.

(.,.)p (print) A megadott sorokat kiírja mindenféle egyéb karakter nélkül. Ha a terminál képernyő alsó sorát eléri, újsor karakterre vár a további írásig.

P A parancsmódban a sor elején megjelenő prompt kijelzését változtatja igen/ nem között. Hacsak indításkor nem adtunk meg prompt szöveget -p szöveg formában, alapértelmezés szerint nem jelenik meg semmi.

q (quit) Kilépés az ed-ből. Módosított puffer esetén figyelmeztetést kapunk.

Q Feltétel nélküli kilépés.

($)r fájlnév (read) A megadott fájlt beolvassa és a megcímzett sor után beszúrja a pufferbe. Ha volt alapértelmezett fájlnév, akkor az marad, különben az új fájl neve lesz alapértelmezett. Ha nem adunk meg fájlnevet, az alapértelmezett fájl kerül beszúrásra.

($)r ! parancs Az előzőhöz hasonlóan, a parancs végrehajtásának szabványos kimenete kerül a puffer megadott helyére. Az alapértelmezett fájlnév változatlan marad, az aktuális sor az utolsó beszúrt sor lesz.

(.,.)s/re/csere/, (.,.)s/re/csere/g, (.,.)s/re/csere/n (substitution) A megadott sorok közül azokban, melyekre illeszkedik a re reguláris kifejezés, az illeszkedő részt kicseréli a csere alatt megadott szövegre. Ebben lehetnek hivatkozások az illesztett reguláris kifejezés részkifejezéseire \1, \2, ..., \9 formában. A teljes illesztett részre hivatkozhatunk & jellel.

A parancs utáni g hatására nem csak az első, hanem az összes illeszkedés esetén megtörténik a csere. Ha pedig egy számot írunk n helyére, akkor az annyiadik illeszkedésnél hajtódik végre a csere.

Hibajelzést kapunk, ha nem történik illeszkedés egy sorra sem. Az aktuális sor az utoljára módosításra kerülő sor lesz. Ez talán a leghatékonyabb művelet, amelyik a teljes képernyős szövegszerkesztőkből hiányzik. A `sed` és a perl is tartalmazza lényegében ugyanezt az utasítást, kisebb módosításokkal. A példáknál részletesen bemutatjuk használatát.

(.,.)s Megismétli az előző cserét.

(.,.)t(.) (transfer) Átmásolja a megadott részt a paraméterként kapott címre. Hasonló m-hez, de nem törli a forrást.

u (undo) Az előző művelet érvénytelenítése. A g, G, v és V műveleteket egyetlen műveletnek tekinti. Hatását egy újabb u visszavonja.

(1,$)v/re/ parancs-sorozat Működése hasonló g-hez, kivéve, hogy a nem illeszkedő sorokon hajtja végre a műveleteket.

(1,$)V/re/ Interaktívan végzi ugyanazt, mint v, analóg módon, mint G.

(1,$)w fájlnév A megadott fájlba írja a kijelölt részeket. A fájl előző tartalma elvész. Ha nem volt alapértelmezés szerinti fájlnév, akkor az felveszi a megadott értéket, különben változatlan marad. Ha nem adunk meg fájlnevet, az alapértelmezett fájlba ír.

(1,$)wq A w parancs és egy utána beírt q hatásával egyező.

(1,$)w !parancs Elindítja a parancs-ot, és a kijelölt sorokat annak bemenetére küldi.

(1,$)W fájlnév Hozzáírás történik a megadott fájlhoz.

(.)x A vágólap tartalmát beilleszti a megadott helyre.

(.,.)y (yank) A kijelölt részeket a vágólapra másolja.

(.+1)z n Az megadot sortól kezdve n sort kiír, és az utolsóra állítja az aktuális mutatót. Ha nem adunk meg n-et, akkor az ablak méretének megfelelő szám lesz az alapértelmezés.

! parancs Végrehajtja a megadott parancsot az sh parancsértelmező segítségével. Ha a parancs első karaktere !, azt az előző paranccsal helyettesíti, ha pedig százalékjel szerepel a parancsban, azt az aktuális fájlnévre cseréli. Valódi százalékjelet pedig % segítségével írhatunk.

A futás eredménye a képernyőre kerül, annak befejeztét egy egyetlen ! jelet tartalmazó sor jelzi. Az aktuális fájlnév és az aktuális sor változatlan.

(.,.)# A # jel utáni rész megjegyzés (komment), vagyis az `ed` figyelmen kívül hagyja. Ha előtte pontosvesszővel lezárt cím volt, az lesz az aktuális sor.

($)= A megcímzett sor számát írja ki.

(.+1) Kiírja a megcímzett sort, majd az lesz az aktuális.

Több helyen láthattuk, hogy fájlműveleteknél, ha ott felkiáltójellel kezdődik a paraméter, akkor nem fájlnévként értelmeződik a paraméter, hanem futtatható parancsként. A parancs lefutása alkalmával annak szabványos kimenetére írt adatok kerülnek a pufferbe olvasás esetén, illetve annak szabványos bemenetére ír az ed, amennyiben kiírás volt a művelet. Például

```
r !ls
```

azt jelenti, hogy hajtsd végre az ls parancsot, vedd annak kimenetét, vagyis az aktuális könyvtár fájljainak listáját, és olvasd be a pufferbe.

2.3. Reguláris kifejezés sajátosságok

A bemutatott reguláris kifejezés szintaktikához képest a következő eltérések vannak. A `?, +, {, }, (, )` jelek megelőző backslash nélkül saját magukat jelentik, speciális jelentésük \ prefix esetén van csak.

Újdonságot jelentenek a szavak elejére, illetve szavak végére való illeszkedést megkönnyítő < és > kombinációk. Az előbbi szavak első betűjére, míg az utóbbi a szavakat követő első karakterre illeszkedik. (Szavak a betűkből valamint aláhúzásjelből álló összefüggő karaktersorozatok.) További kiegészítések a következők.

; ´ Feltétel nélkül illeszkednek a sor elejére (̀) illetve a végére (́).

_ Szavak elejére és végére illeszkedik (nulla karakter hosszan).

\B Szavak belsejére illeszkedik 0 karakter hosszan.

\w Egy szó tetszőleges karakterére illeszkedik.

\W Szavakon kívüli karakterekre illeszkedik.

2.4. Indítás parancssorból

Az `ed` a következő paramétereket fogadja el:

```
ed [-] [-Gs] [-p szöveg] [fájlnév]
```

-G Kompatibilitási problémák miatt használható. Hatására a `G, V, f, l, m, t` és `!!` parancsok működnek másként.

-, -s Diagnosztikai üzenetek megszűntetése. Akkor van rá szükség, ha szkriptből használjuk az ed-et, és nem interaktívan.

-p szöveg A parancs módban a sor elején lévő prompt szöveget állíthatjuk be.

A megadott fájlnév a beolvasandó fájl neve. Ha felkiáltójellel kezdődik, akkor azt a shell által végrehajtató parancsnak értelmezi ed és a futtatásakor megjelenő kimenetet dolgozza fel. Az alapértelmezés szerinti fájlnév akkor állítódik be, ha nincs végrehajtás, vagyis a megadott fájlnév nem ! jellel kezdődik.

2.5. Példák

Egy interaktív szerkesztés illusztrálja az `ed` használatát. A kommentek a # jellel kezdődő sorokban vannak.

```
$ed
# Üres pufferrel kezdünk. Ehhez írunk hozzá az a paranccsal.
a
Edvárd király, angol király
Léptet fakó lován:
Hadd látom, úgymond, mennyit ér
A wales-i tartomány.
.
# Visszaértünk parancs módba. Elmentjük a szöveget.
w bardok
100
```

```
q
A mentés után az ed kiírta a puffer méretét, vagyis hogy hány karakteres a
szöveg. Most javítsuk ki a helyesírási hibákat
$ed bardok
100
,p
Edvárd király, angol király
Léptet fakó lován:
Hadd látom, úgymond, mennyit ér
A wales-i tartomány.
# Az utolsó sor az aktív.
1
Edvárd király, angol király
s/Edvárd/Edward/
p
Edward király, angol király
# A wales sem úgy kell
/wales-/s//velsz/
,p
Edward király, angol király
Léptet fakó lován:
Hadd látom, úgymond, mennyit ér
A velszi tartomány.
# Mentsük a javításokat
w
99
q
```

3. fejezet

sed, a folyamszerkesztő

A `sed` (stream editor), vagyis a folyamszerkesztő egy nem interaktív szövegszerkesztő. Segítségével egyszerű módosításokat végezhetünk a bemeneten érkező folyamon. (Ez lehet egy fájl, vagy adatok egy pipe-ból.) A működése bizonyos mértékig hasonlít az olyan szövegszerkesztőkhöz, amelyek megengedik a szkriptből történő vezérlést (mint például az `ed`), de a `sed` a bemenet minden sorát legfeljebb egyszer dolgozza fel, következésképpen hatékonyabb. Szöveg feldolgozás pipe belsejében a `sed` egyedi tulajdonsága, ez különbözteti meg őt a hasonló típusú szövegszerkesztőktől.

A `sed` a következő paramétereket fogadja el:

-V, –version A program verziószámát és egyéb információkat íratjuk ki.

-h, –help Rövid ismertetőt kaphatunk a paraméterezési szabályokról, alapvető működésről.

-n, –quiet, –silent Az alapértelmezés szerinti, minden működési ciklus utáni kiíratást tiltja.

-e szkript, –expression=szkript A szkript-et, mint parancsot fűzze hozzá a feldolgozást végző parancs-sorozathoz.

-f szkript-fájl, –file=szkript-fájl A megadott fájl tartalmát fűzze hozzá a feldolgozást vezérlő parancs-sorozathoz.

Ha nem adunk meg más paramétert, alapértelmezésként a szabványos bemenetet dolgozza fel a program. Amennyiben egy vagy több fájlnév szerepel a paraméterek között, azokat sorra veszi a `sed`. Ezek között utalhatunk - jelöléssel a szabványos bemenetre, mint fájlra.

Ha nagyon rövid szkriptet írunk, az alábbi paraméterezési forma is megfelel:

```
sed szkript fájlnév1 fájlnév2 ...
```

Tehát amennyiben nincsenek opciók, az első paraméter maga a szkript kell, hogy legyen, utána jöhetnek a feldolgozandó fájlok nevei.

A `sed` működését egy parancsokból álló sorozat (program) vezérli, amelyet a -e és a -f paraméterekkel megadott utasítások illetve programrészletek összefűzésével kapunk. Ha egy sorba több parancsot írunk, azokat pontosvesszővel (;) kell elválasztani.

Külön sorba kerülnek azok a parancsok, amelyeket külön -e opció mellett adtunk meg. Hívásra is mutatunk példákat később.

A parancs-sorozat parancsokból áll, amelyek három részre bonthatók. Az első a cím vagy címtartomány, amely az adott parancs végrehajtását engedélyezi vagy tiltja. A második rész egyetlen karakter, maga a parancs. Az utolsó rész a parancs paramétere, amely műveletenként különböző.

A sed rendelkezik egy munkaterülettel és egy vágólappal. A működés ciklusokban történik, ami a következő bemeneti sornak a munkaterületre való írásával kezdődik. Ekkor a sed sorra veszi a parancs-sorozat egyes parancsait és egyenként megvizsgálja a címzésüket. Ha egy parancs címzése engedélyezi az aktuális soron a parancs végrehajtását, akkor ez megtörténik. Hacsak az adott parancs nem rendelkezik másképp, a működés a parancs-sorozat következő elemén folytatódik, a már módosított munkaterületen tovább dolgozva. (Léteznek a vezérlést befolyásoló parancsok is.) Fontos, hogy a címzés vizsgálata nem a módosított munkaterület szerint történik, hanem az eredeti bemeneti sor alapján.

A vágólapot néhány egyszerű parancs kiadásával használhatjuk. Segítségükkel a munkaterület tartalma a vágólapra menthető, máskor visszamásolható, esetleg a két terület tartalma felcserélhető.

A működési ciklus befejezésével (ez zárja le egy bemeneti sor feldolgozását) a munkaterület tartalma alapértelmezés szerint a kimenetre kerül. Ez felülbírálható a -n opció használatával, amikor is csak akkor kerül adat a kimenetre, ha azt a p paranccsal explicite kérjük.

3.1. Címzés

A címzés segítségével sort vagy sorokat jelölhetünk ki. A címzés végére írt felkiáltójel a jelentést az ellenkezőjére állítja: eszerint a címzés pontosan azon sorokat jelöli ki, amelyeket a felkiáltójel nélküli címzés nem jelölt ki.

Ha nem adunk meg címzést, az az összes sort kijelöli. Alapvetően a következő címzési módok használhatók:

szám A megadott sorszámú sort jelöli ki. Ha több bemeneti fájl van, akkor sed folyamatosan számolja a sorokat, tehát a második fájl első sora nem jelölhető ki egyessel.

első ~lépésköz A megadott sortól kezdve, lépésköz közönként minden sort kijelöl. Például 1~2 a páratlan sorokra utal, míg 3~3 a hárommal oszthatóakra.

$ A legutolsó sora az utolsó bemeneti fájlnak.

/regkif/, %regkif% Azon sorokat jelöli ki, amelyekre a megadott reguláris kifejezés illeszkedik. Ha a / jeltől eltérő zárójeleket szeretnénk használni, arra is lehetőség van. A második formában a százalékjel helyett tetszőleges más karakter használható. Ilyenkor természetesen a kifejezés belsejében nem kell a per jel elé backslash-t tenni. A határoló karaktert azonban nem tudjuk önállóan használni. Általában nem nehéz találni olyan karaktert, amely nem szerepel a reguláris kifejezésünkben. Célszerű tehát azt választani határoló karakterként. Például keressünk egy szövegben olyan részeket, ahol két / között számok vannak:

```
//[0-9]*//
```

Ebben a formában eléggé átláthatatlan a kifejezés, helyette sokkal jobb az alábbi:

```
\@/[0-9]*/@
```

Ezeknél sokkal durvább esetek is vannak a gyakorlatban. Amikor sok \ és / karakter szerepel egy kifejezésben, azt szokás „ferde fogpiszkáló szindrómának” nevezni.

/regkif/I, %regkif%I A kifejezés után írt nagy I betű hatására a kis- és nagybetűk megkülönböztetése nélkül próbálja illeszteni a reguláris kifejezést (case-insensitive).

Címtartomány megadható, ha két címzést adunk meg vesszővel elválasztva. Ha a második címzés reguláris kifejezés, annak vizsgálata csak az első címzést követő sornál kezdődik.

3.2. Parancsok

3.2.1. Gyakori parancsok

A `sed` használata során a következő parancsokra minden bizonnyal szükség lesz.

Megjegyzés kezdetét jelzi, amely egészen a sor végéig tart. Bizonyos implementációk csak teljes soros kommenteket engedélyeznek, de ez nem jellemző.

s/re/csere/jelző A `re` reguláris kifejezést megpróbálja illeszteni a munkaterületre. Ha ez sikerült, az illeszkedő részt kicseréli a csere alatt megadott szövegre. Ebben lehetnek hivatkozások az illesztett reguláris kifejezés részkifejezéseire \1, \2, ..., \9 formában. A teljes illesztett részre hivatkozhatunk & jellel.

Mivel a munkaterület (a legtöbb alkalmazással ellentétben) többsoros lehet, itt a reguláris kifejezés tartalmazhat újsor karakterre történő hivatkozást (`\n`).

Az `s` parancs utáni jelzők a `g, p, szám, w` valamint `I` közül kerülhetnek ki. Hasonlóan az `ed`-nél látottakhoz, a `g` arra utasítja a programot, hogy ne csak az első illeszkedés esetén végezze el a cserét, hanem a módosítás helye után folytatva, ameddig még lehet. A szám hozzáírásával az annyiadik illeszkedésnél lehet kieszközölni a cserét. Ha `p` előfordul a jelzők között, a munkaterület tartalma a csere után a kimenetre kerül. `I` a reguláris kifejezés kis- és nagybetűktől független illesztését kéri, míg `w fájlnév` a megadott fájlba írja a munkaterület tartalmát.

q A működés befejezése további feldolgozás nélkül. A munkaterület tartalma kiíródik, amennyiben az automatikus kiírás nincs tiltva. Használatára példa a `head` parancs megvalósítása `sed`-del:

```
sed 10q fájlnév
```

Az egyetlen parancs akkor hajtódik végre, amikor a címe aktivizálódik, vagyis a 10-dik sorban. Addig alapértelmezés szerint kiírás történik. A 10-dik sor `q` művelete kilépés előtt még kiírja azt a sort is.

d Törli a munkaterületet. Vezérlésmódosító hatással is rendelkezik: új ciklus kezdődik, vagyis az aktuális sor feldolgozásának vége, a következőt olvassa be a bemenetről. Általában arra használható, hogy egy címzés esetén ne folytatódjon annak a sornak a feldolgozása, hanem kiírás nélkül, azonnal lépjünk a következő sorra, az utasításokat elölről kezdve. Például:

```
sed -e '$d'
```

az utolsó sort nem írja ki.

p Ez a parancs a munkaterület tartalmát a kimenetre írja. Szinte mindig a `-n` opcióval együtt használatos. Nem definiált, hogy ha az automatikus kiíratás engedélyezett, akkor a `p` parancs használata után mégegyszer kikerüljön-e a munkaterület a ciklus végén, vagy sem (megvalósításfüggő).

n Amennyiben az automatikus kiíratás engedélyezett (alapértelmezés), kiírja a munkaterület tartalmát. Függetlenül attól, hogy volt-e kiírás, betölti a következő sort a munkaterületre, annak eddigi tartalmát felülírva. A vezérlés a következő parancsnál folytatódik, tehát nem kezdődik elölről a parancs-sorozat feldolgozása. Intuitív jelentése: vedd a következőt (next).

parancsok Parancsokat csoportosít közös címzéshez, amely a nyitó zárójel előtt szerepel.

3.2.2. Ritkábban használt parancsok

Valószínűleg ritkábban van szükség ezekre a funkciókra, de olykor rövid szkriptek írásakor is hasznunkra lehetnek.

y/forrás-karakterek/cél-karakterek/ (A / karakter tetszőleges másikra cserélhető egy `y` parancson belül.)

A forrás-karakterek és a cél-karakterek ugyanannyi karaktert kell, hogy tartalmazzon a behelyettesítések elvégzése után (\ jellel védett karakterek, escape-szekvenciák). A parancs hatására, ha a forrás-karakterek valamelyike szerepel a munkaterületen, az a neki megfelelő cél-karakterre cserélődik. A „neki megfelelő” `a` hely sorszáma szerint értendő. Például `y/abc/ABC/` az `a, b, c` karaktereket nagybetűs párjukra cseréli.

a szöveg A szöveg új sorban kell, hogy kezdődjön. Több soron át is folytatódhat, de az utolsót kivéve mindegyiknek backslash-re kell végződnie. A parancs hatására az aktuális ciklus befejeztével, vagyis a következő bemeneti sor beolvasása előtt a megadott szöveg kiíródik.

i szöveg Az előzőhöz hasonló parancs, de a szöveg kiírásra azonnal sor kerül.

c szöveg A megcímzett tartomány eldobása után kiírja a megadott szöveget, majd új ciklus kezdődik. Ha nem csak egy sort címzünk meg, hanem egy intervallumot, akkor az `a` rész törlődik, és helyette bekerül a megadott szöveg, amelyet az `a` parancshoz hasonlóan, új sorba kell írni.

= A megcímzett sor számát írja ki újsor karakterrel lezárva.

l Kiírja a munkaterületet egyértelmű formátumban. Ez azt jelenti, hogy a nem nyomtatható karakterek (köztük az ékezetesek és a \ is) C-stílusban, backslash-sel és kódukkal kerülnek ki, hosszú sorok törésénél \ jelzi a törés helyét, valamint minden sor végére $ kerül.

r fájlnév A megadott fájl tartalma az aktuális ciklus végén a kimenetre kerül (vö. a). Ha nem sikerül a fájl olvasása, nem jelez hibaüzenetet, hanem azt üresnek tekinti.

w fájlnév A munkaterület tartalmát a megadott fájlba írja (A fájlt létrehozza ha nem létezik, illetve felülírja, ha az létezett.).

D Törli a munkaterületet az első újsor karakterig bezárólag. Ha maradt még szöveg a munkaterületen, újrakezdi a ciklust újabb sor beolvasása nélkül. Egyébként új ciklus kezdődik új beolvasással.

N Újsort fűz a munkaterület végére, majd hozzáírja a következő beolvasott sort. Ha nincs több bemenet, tehát nem sikerül újabb feldolgozandó sort behoznia, a program futása befejeződik, vagyis kilép.

P Kiírja a munkaterületet az első újsor karakterrel bezárólag.

h A vágólapra másolja a munkaterület tartalmát (copy).

H Újsort ír a vágólap tartalmához, majd hozzáfűzi a munkaterület tartalmát.

g A munkaterületre másolja a vágólap tartalmát.

G Újsor karaktert ír a munkaterület végére, majd hozzáírja a vágólap tartalmát is. Kezdetben a vágólap üres. Ezt használja ki az alábbi parancs is, amelyik a bemenetén érkező sorok közé egy-egy újsort szúr be:

```
sed G fájlnév
```

x Felcseréli a vágólap és a munkaterület tartalmát.

3.2.3. További parancsok

Ha ezekre a parancsokra van szükség, felmerül a gyanú, hogy általánosabb programozási nyelvet kellene inkább használni, mint például a `perl`-t. Esetenként, ha mégis a `sed` mellett maradunk, az alábbiak segítségével már igen szövevényes szkriptek készíthetők.

: címke Megjelöli a helyet egy címkével, amelyre a vezérlést a `b` vagy a `t` paranccsal irányíthatjuk. Műveleti hatása nincs.

b címke A vezérlés feltétel nélkül a címkével megadott helyre kerül. Analóg egyes nyelvek goto utasításával.

t címke Feltételes ugrás a megadott címkére. Akkor történik ugrás, ha az utolsó beolvasás vagy az előző `t` parancs végrehajtása óta volt sikeres művelet.

3.3. Reguláris kifejezés sajátosságok

A `sed`-es reguláris kifejezések némileg eltérnek az előzőekben bemutatott szintaktikától, viszont a különbségek hasonlítanak az `ed`-nél látottakra. Bizonyos speciális karakterek itt is backslash után nyerik el különleges jelentésüket. Ezek a következők: |, +, ?, (,), {, }. Viszont annak következtében, hogy a `sed` a munkaterületen végzi a műveleteket, illetve a reguláris kifejezés illesztéseket, ami ugye több sorból is állhat, az újsor karakter értelmezése egy kicsit más. A pont (.) újsor karakter is lehet, és \n-ként bárhol hivatkozhatunk újsor jelre. A munkaterület végén lévő újsor karakterre nem illeszkedik a \n. A ^ és a $ operátorok a munkaterület sorainak elejére illetve a végére illeszkednek.

A tab karaktert illetően az egyes `sed` megvalósítások között eltérés van, közülük több nem ismeri fel \t alakban.

3.4. Példák

A `sed` szkriptek általában nagyon rövidek, szellemesek és emiatt egy kicsit nehezen érthetők. Az itt bemutatott példákat ezért részletes magyarázat kíséri, amely a megértésüket hivatott segíteni. A példák az egyes parancsok legapróbb tulajdonságait is kihasználják, ezért nem árt, ha az olvasásuk során visszatekintünk a parancsok leírásához.

Amikor nem szerepel bemeneti fájl megadva, a `sed` a szabványos bemenetén várja a sorokat, és azt a szabványos kimenetre írja. Legtöbb példa az egyszerűség kedvéért ezt a formát használja.

A parancsok ismertetésénél szerepelt a

```
sed G
```

példa, amely a bemenet minden sora után beszúr egy üres sort, kihasználva, hogy a vágólap üres. Ennek a parancsnak az ellentettje, vagyis amikor a bemenet minden második sorát, amelyek tudvalevően üresek, kiszűrjük, tehát csak a páratlanadik sorokat írjuk ki:

```
sed 'n;d'
```

Ekkor az első parancs következtében kiíródik a beolvasott sor, majd a következővel folytatjuk a `d` parancsnál. Ez eldobja a munkaterületet, és újrakezdi a ciklust, immáron a harmadik sorral az elején. Innen folytatódik a harmadik sor kiírásával, a negyedik eldobásával, stb. Látható, hogy a páratlan sorok kiíródnak, a párosakat pedig eldobja a program. Megoldható lett volna ugyanez bonyolultabb címzéssel is (1~2 a páratlan, 2~2 a páros sorokat címzi), de ilyen bonyolítás nem szükséges. Vegyük észre, hogy ebben a példában aposztrófok közé írtuk az utasításokat. Ha csak `sed n;d`-t írtunk volna, akkor a shell két parancsnak értelmezi: az egyik `sed n`, a másik pedig `d`, ami szinonima az `ls`-re. Ez mindenképp rossz, hiszen amit mi a `sed` parancs paraméterének szántunk, azt a `bash` kettévágta. Ha bizonytalanok vagyunk, mindig tegyük ki az idézőjeleket a biztonság kedvéért.

Egy olyan példa következik, amelyik megszámozza a sorokat.

```
sed = | sed 'N;s/\n//'
```

Itt már két `sed` hívás szerepel. Az első minden sor elé kiírja annak sorszámát, de ez egy önálló soron szerepel. A kimenet így néz ki:

```
1
első sor
2
második
3
harmadik
...
```

Ez kerül a következő sed bemenetére, amelyik minden ciklusban a beolvasott sor mellé a munkaterületre fűzi a következő sort (N), majd közöttük az újsor karaktert törli (s/\n/ /), és ezt írja ki. Ezt a problémát azért nem lehet egyetlen `sed` hívással megoldani, mert az = parancs azonnal a kimenetre írja a sorszámot egy újsor karakterrel együtt. Ha a munkaterületre írná, ott helyben el lehetne végezni a további manipulációt. Nagyon komplikált megoldást lehetne az = parancs nélkül készíteni a következő gondolatmenet szerint: A vágólapon tartanánk az előző sor sorszámát. Minden beolvasáskor felcserélnénk a munkaterület tartalmát a vágólapéval, és a munkaterületen megnövelnénk a számot eggyel (Ez bonyolult, sok s/// utasítással megoldható.). Ehhez hozzáfűznénk a vágólapot, majd kiírnánk, és végül az aktuális sorszámot visszaírnánk a vágólapra. Ez nem praktikus, a pipe-os megoldás sokkal jobb. Mindössze a `sed` lehetőségeinek illusztrálása végett említettük.

DOS alatt két karakter jelzi a sorok végét, míg a UNIX-szerű rendszerekben csak egy. Ha DOS-os fájlokat szeretnénk UNIX-osítani, az alábbi szkript megteszi:

```
sed 's/.\$//'
```

A DOS-os sorvégek CR/LF típusúak (kocsivissza és soremelés karakterek). Ezekből a CR-t kell eltüntetni, ami UNIX-os értelmezésben a sor utolsó karaktere. Ezt egyszerűen törli a megadott szkript.

Összetettebb példa következik. A bejövő fájl sorait jobbra illesztjük, vagyis annyi szóközt szúrunk be, hogy minden sor 79 karakterből álljon.

```
sed -e :a -e 's/\^.\{1,78\}\$/ &/;ta'
```

Két paraméterben adtuk meg a szkriptet, gyakorlatilag ez a következőképp néz ki a szkript-darabok összefűzése után, minden utasítást külön sorba írva:

```
:a
s/\^.\{1,78\}\$/ &/
ta
```

Az első sor egy ugrási címke, semmi módosító hatása nincs. A második sor, amennyiben a munkaterület hossza 78-nál nem nagyobb, beszúr az elejére egy szóközt (Az & jel a teljes illesztett részt szúrja be). A harmadik sor `t` parancsa, amennyiben sikeres `s` művelet történt, visszaugrik az `a` címkére, azaz a szkript elejére. Egy bemeneti soron addig tart a ciklus működése, amíg végül eléri a munkaterület hossza a 79-et. Ekkor az `s` nem helyettesít, így `t` sem ugrik vissza. A végére érve kiíródik a

jobbra igazított sor. A parancssort írhattuk volna az alábbi formában is:

```
sed ':a;s/\^.\{1,78\}\$/ &/;ta'
```

A jobbra igazítás megértése után a középre igazítás már sokkal könnyebben megy:

```
sed -e :a -e 's/\^.\{1,77\}\$/ & /;ta'
```

Itt amíg a munkaterület tartalma 78 karakternél rövidebb, addig egy-egy szóközt mindig beszúrunk előre és hátra is. Világos, hogy végül középre igazodik az eredeti sor. Egy másik módszer a jobbra igazításból indul ki:

```
sed -e :a -e 's/\^.\{1,77\}\$/ &/;ta' -e 's/\( *\)\1/\1/'
```

Mindössze utána a munkaterület elején lévő, szóközökből álló hosszú részt megfelezi. Itt a \(*\)\1 reguláris kifejezést illeszti, amely első részét megismétli a \1. Ez abban különbözik az előző megoldástól, hogy míg ott végül minden sor 78 vagy 79 karakter hosszú, itt a sorok jobb oldalán nem lesznek szóközök.

A `q` parancs ismertetésénél megmutattuk, hogy a `head` segédprogram hogyan valósítható meg a `sed`-del. Most annak párja, a `tail` következik, amely a bemenetére küldött fájl utolsó 10 sorát írja csak ki:

```
sed -e
:a -e '\$q;N;11,\$D;ba'
```

Ha a fájl végére érünk, a munkaterületet kiírva kilép a szkript. Egyébként beolvas még egy sort és a munkaterülethez fűzi. Ha már elértük a 11. sort, törli a munkaterület első sorát. A végrehajtás az első utasításnál folytatódik. Ezzel a módszerrel a munkaterületen tartjuk az utolsó 10 sor tartalmát.

A grep segédprogramot is emulálhatjuk a sed-del:

```
sed -n '/regkif/p'
sed '/regkif/!d'
```

Az első változat letiltja az automatikus kiírást, és csak a címzésre illeszkedő sorokat írja ki. A második a címzésre illeszkedő sorok esetében megakadályozza az automatikus kiírást: a `d` parancs a munkaterületet törli, majd a következő ciklusba kezd.

Az elektronikus levelek szabványos formátumában a fejlécet a törzs követi. A fejlécben szerepelnek olyan információk, mint a feladó, a címzett, a tárgy és a levél továbbítási információi. Ezek mind egy-egy sort foglalnak el, ahol a sor első szava a jelentés (pl. To:, From:, stb.), utána kettőspont áll, majd az érdemi információ következik. A levél törzse maga a tartalom, amit egy üres sor választ el a fejléctől. Az alábbi `sed` programok a bemenetükre érkező levél formátumú fájlból kiválasztják a fejlécet illetve a törzset. A fejléc kiválasztásához mindössze az első üres sorig kell mindent kiírni, utána semmit, vagyis ki lehet lépni:

```
sed '/\^\$/q'
```

A törzs ezután kezdődik, vagyis az első üres sorig mindent ki kell törölni:

```
sed '1,/\^\$/d'
```

A levél tárgyát kiválasztani a sor elején lévő Subject: felirat nélkül a következőképpen lehet:

```
sed '/\^Subject:*/!d; s///;q'
```

Minden sor, amely nem Subject: sztringgel kezdődik, törlődik. Az első olyan, amelyik így kezdődik, az `s///` parancshoz kerül. Itt üres reguláris kifejezésre illesztünk, ami az előző reguláris kifejezés megismétlése lesz, tehát a / Subject: */ által illesztett részt töröljük. Ez pont a Subject: felirat az utána következő szóközökkel együtt. Mivel úgyis egyetlen ilyen sor van, a `q` paranccsal kiléphetünk. A válaszcím általában a From: által jelzett cím, ám ha Reply-to: is van a fejlécben, a válaszokat oda kell küldeni. Ezt az alábbi szkript állapítja meg:

```
sed '/\^Reply-To:/q; /\^From:/h; /./d;g;q'
```

Működése során ha Reply-to:-val találkozik, azt kiírja és azonnal kilép. Ha From:-ot talál, azt a vágólapra másolja. Minden más sort figyelmen kívül hagy (`d`). Az üres sort elérve, amely a fejléc végét jelzi és nem illeszkedik a `d` címzésére, a vágólapon lévő From: sort a munkaterületre írja, majd kilép. Ezt helyes, mert amennyiben elérte a fejléc végét, és előbb nem lépett ki, nem volt Reply-to: sor. Mivel From: sor mindenképp kellett, hogy legyen, annak tartalma a vágólapon van, azt kell kiírni kilépéskor.

Végül két nehezebb szkript következik. Az első a beolvasott sorokat fordított sorrendbe írja a kimenetre, tehát az első sort utoljára, a másodikat az utolsó előtti helyen, és így tovább.

```
sed '1!G;h;\$!d'
```

Az első sor kivételével mindegyik beolvasáskor végrehajt egy `G` parancsot. Emlékeztetőül, ez a vágólap tartalmát írja a munkaterület végére egy új sort kezdve. Az így kapott szöveget visszamásolja a vágólapra (`h`), és hacsak nem az utolsó sort dolgozta éppen fel, a következőre lép kiírás nélkül. Az utolsó beolvasott sornál kiírja az eddig felhalmozott adatokat. Ha a működést átgondoljuk, világossá válik, hogy gyakorlatilag minden közbülső lépés során a beolvasott sort a vágólapon lévő szöveg elé írja. Ez végeredményül a megfordított sor-sorrendet adja.

A másik szkript minden beolvasott sor betűit fordított sorrendben írja vissza a kimenetre:

```
sed '/\n/!G;s/\(.\)\(.*\n\)/&\2\1/;//D;s/.//'
```

Egy sor feldolgozása hosszú folyamat. Ezalatt a munkaterület két sorból áll, az első a fel nem dolgozott részt tartalmazza, a második pedig a feldolgozottat. Minden lépésben az első sor első karaktere a második sor elejére kerül. Kezdetben az első sorban van a teljes szöveg, végül pedig megfordítva a másodikban lesz teljesen. A szkript első utasítása csak közvetlenül az első beolvasást követően lesz érvényes: ilyenkor beilleszti az üres vágólapot, vagyis két sort készít az előbb leírtaknak megfelelően.

A második parancs, egy `s/.../.../` a következőt teszi: a munkaterület első sorára illeszt egy reguláris kifejezést (\n-ig), amely két részkifejezésből áll. Ezek közül \1 az első karakter, \2 pedig a másodiktól a sor vége jelig terjedő rész. Tehát \2-be még a sorvég jel is beleértendő. Ezeket más sorrendbe írja vissza, de előtte megismételi a teljes illesztett részt (&). Gyakorlatilag az eredeti két sor közé beszúrja \2\1-et. Mivel \2 újsorra végződik, ez önállóan áll majd egy soron, \1 pedig az eredetileg másodikként szereplő sor elejére kerül. A most három sorból álló munkaterület második két sora annyiban különbözik az eredeti két sortól, hogy annak legelső karaktere a következő sorba került.

Ezután a `D` törli az első sort. Címzése //, ami az előző reguláris kifejezés megismétlése, vagyis amit `s` első fele használ. Ha még volt karakter az első sorban, akkor a `D` végrehajtódhatott és új beolvasás nélkül elölről kezdődhet a parancsok feldolgozása. Egyébként `s//` következik: ez törli az első karaktert, amely egy újsor jel volt. A munkaterületen már csak a megfordított szöveg van, amely kiíródik és folytatódik a munka a következő beolvasott sorral.

4. fejezet

Az awk nyelv

4.1. Áttekintés

Az awk egy szövegfeldolgozó programozási nyelv. Nevét kitalálói nevének kezdőbetűit egybeírva kapta: Alfred V. Aho, Peter J. Weinberger és Brian W. Kernighan.

Egy awk program (szkript) utasításokból áll. Minden egyes utasítás egy minta és egy akció részre osztható:

```
minta { akció } minta { akció } minta { akció } ...
```

Futtatáskor a rendszer a beolvasott fájl egyes soraira egyenként megvizsgálja a minták illeszkedését. Ha egy minta megfelelt, akkor a neki megfelelő akció végrehajtódik. A vizsgálat alapértelmezés szerint a következő utasításnál folytatódik. Az eddigiek alapján ez a nyelv nagyon hasonlít a sed-nél megismert szövegmanipuláló nyelvhez. Ahogy az a következőkben bemutatásra kerül, az awk esetében mind a minták, mind az akciók felépítése sokkal összetettebb lehet, így általánosabb funkciók is megvalósíthatók awk nyelven megírt szkriptekkel.

Az utasítások akció részében a C nyelvhez hasonló vezérlési szerkezeteket használhatunk, és léteznek változók is, amelyekre mind a minta, mind az akció részekben hivatkozhatunk. A komoly ki- és beviteli parancsok segítségével akár más alkalmazások futtatásával pipe-okat hozhatunk létre, és asszociatív tömbök gyorsíthatják és egyszerűsíthetik az adatfeldolgozást. Beépített függvények teszik lehetővé a számítások elvégzését, illetve a karakterláncok átalakítását is. Ezen túl a felhasználó saját függvényeket is írhat, amelyek rekurzív hívása is megengedett.

A feldolgozás teljesen sor-orientált. Az éppen beolvasott sorra a $0 változó hivatkozik. Az awk ezt automatikusan oszlopokra darabolja a munka megkönnyítése végett. Alapértelmezés szerint a feldarabolás az üres (szóköz vagy tab) karakterek mentén történik. A sor elején és végén lévő üres karaktereket nem veszi figyelembe, és a szomszédosakat is egynek tekinti. Az egyes oszlopokra $n alakban hivatkozhatunk, ahol n az oszlop száma. Tehát ha a beolvasott sor `alma körte barack meggy`, akkor $2 értéke körte. A feldarabolás miként jét mi is beállíthatjuk az FS változó segítségével, erről később szó lesz.

Valójában a bemeneti fájl soronkénti feldolgozása is konfigurálható, helyesebb volna rekordokra bontásról beszélni. Mivel azonban az esetek legnagyobb részében soronként dolgozzuk fel a bemenetet, ebben az ismertetőben végig az alapértelmezett

működésnek megfelelően sorokat használunk rekordok helyett. A rekordokra való darabolást egyébként az RS változó szabályozza.

Ha nem adunk meg valahol mintát, akkor az akció minden sorra végrehajtódik. Amennyiben az akció rész marad el, alapértelmezés szerint a megfelelő sorok egyszerűen kiíródnak. Így például az egyszerű

```
{ print \$0 }
```

szkript az összes beolvasott sort a kimenetre írja.

Az awk programok futtatása két különböző módon történhet. Ha a szkript rövid, azt a parancssorba írhatjuk első paraméterként. A rákövetkező paraméterek, amennyiben fájlnevek, a bemeneti fájlokat jelentik. Ha nem adunk meg bemeneti fájlt, az awk a szabványos bemeneten várja a feldolgozandó információt.

```
awk '{ print \$0 }' fájl1 fájl2...
```

Ha hosszabb szkriptet futtatunk, vagy esetleg többször is szeretnénk azt használni, egy fájlba kell írni a szkriptet, ahonnan az

```
awk -f szkripfájl fájl1 fájl2 ...
```

sorral működtethetjük.

A nyelv szabályainak átfogó ismertetése után példák illusztrálják azok használatát.

4.1.1. Az awk nyelv elemei

Minták

Az awk nyelv sokkal általánosabb sorcímzési mechanizmussal rendelkezik, mint a sed vagy az ed. A minta mezőben a program változóit is vizsgálhatjuk, illetve több minta megfeleléséből logikai műveletekkel állíthatunk elő összetett mintákat.

Alapkomponensként ezek a minták használhatók:

/regkif/ Megfelel a minta, ha a megadott reguláris kifejezés illeszkedik a beolvasott sorra. Valójában ez az egyszerűsített írásmódja a `\$0 \~{ } /regkif/` kifejezésnek.

kifejezés Megfelelő, ha a kifejezés értéke számként nem nulla, vagy sztringként nem üres. Változók, függvényhívások és összehasonlítások is szerepelhetnek egy kifejezésben (lásd alább).

BEGIN, END Speciális minták, amelyek az első sor beolvasása előtt (BEGIN), illetve a befejezést közvetlenül megelőzően (END) illeszkednek. Csak önmagukban használhatók, tehát nem szerepelhetnek összetett mintákban.

üres minta Minden sorra illeszkedik.

minta1, minta2 A minta1 első előfordulásától kezdve, a minta2 első előfordulásáig tartó szakaszt jelöli ki.

Az alábbi példában megszámoljuk a bemeneti fájlban előforduló gyümölcsöket:

```
BEGIN { print "Gyümölcsök számolása:" }
/alma|barack|dinnye/ { gyumolcs=gyumolcs+1 }
END { print gyumolcs }
```

Ha figyelmesek vagyunk, feltűnhet hogy nem a gyümölcsök előfordulásának számát kapjuk meg így, hanem azon sorok számát, amelyekben legalább egy gyümölcs neve előfordul. Ez azért van, mert az `awk` soronként dolgozza fel a bemenetet, és hiába szerepel egy sorban több gyümölcs, attól a változót növelő akció csak egyszer hajtódik végre az említett sor következtében. Az eredeti problémára a megoldás az volna, ha az akció részben egy helyett az előfordulások számával növelnénk meg a gyümölcs változót.

Kifejezések

Konstans kifejezés három féle lehet az `awk`-ban. Numerikus konstansokat vagy tízes számrendszerbeli alakjukkal adunk meg normál formában, vagy exponenciális formában. Például 105, 1.05e+2 is ugyanazt jelenti. A sztring konstansokat dupla idézőjelek között kell megadni, például „szöveg”. A harmadik, a reguláris kifejezés konstans, amelyet a szokásos formában, perjelek közé kell írni:

```
/a*b/.
```

Önállóan egy reguláris kifejezés konstans ugyanazt jelenti, mintha `\$0 \~ /regkif /` lenne odaírva. A hullámvonal (tilde) operátor az előtte álló változóra illeszteni próbálja a mögötte álló reguláris kifejezést. Mivel $0 az aktuálisan beolvasott sor, ezzel arra illesztjük a reguláris kifejezést. Egy ilyen illesztés eredménye 0 vagy 1 aszerint, hogy sikerült-e. Például `siker = /alma/` után a siker változó értéke akkor lesz 1, ha `\$0 \~ /alma/` illeszthető.

Változók

Az `awk` változói a szkript utasításain keresztül módosíthatók, olvashatók, de a parancssoron keresztül kezdeti értéket is adhatunk nekik. Változók betűket, számokat és aláhúzás karaktereket tartalmazhatnak, de nem kezdődhetnek számjeggyel. A kis- és nagybetűk különbségére érzékeny a nyelv (case sensitive).

A változók hordozhatnak mind numerikus, mind szöveges információt. Hogy éppen hogyan kerül értelmezésre tartalmuk, az az adott környezettől, és a változókhoz belsőleg rendelt típus információtól is függ. A belső típusok közötti konverzió ugyanakkor automatikus. Egy számot tartalmazó változó numerikus környezetben a megfelelő számértéket jelenti, de szöveges környezetben az adott szám tízes számrendszerbeli alakját, mint sztringet. Például legyen `a=2, b=3 és c=4`. Ekkor `(a b) + c` értéke 27. A zárójelben egymás után írtuk az a és b változókat. Az összefűzés sztring művelet, operandusait sztringként kezeli, vagyis egymás mellé írva 24-et ad eredményül. Ehhez van hozzáadva c, és mivel az összeadás numerikus művelet, érték szerint adódik össze a kettő. Ha olyan sztring kerül numerikus környezetbe, amelyik nem értelmezhető számként, akkor helyette a 0 értékkel számol a rendszer. Ezért például tigris + 1 értéke egyszerűen 1.

Valójában minden változóhoz három féle típus valamelyike van belsőleg rendelve, kritikus esetben ennek segítségével értelmezi a rendszer tartalmukat (például összehasonlításkor). Ezek a STRING, a NUMERIC és az STRNUM. Ez az 1992-es POSIX szabványosításkor lett bevezetve. Az új típus, az STRNUM értelme, hogy a rendszer

helyesen tudja kezelni a sztringként leírt számokat, mint például a „ +2”. A használat során a változókhoz rendelt típus nem változik, egyedül akkor, amikor értéket adunk a változónak. A `split()` függvény, illetve az `argv` tömb által érintett változók STRNUM típusúak.

Az összehasonlítási operátorok a következők: <, >, <=, >=, ==, !=. Ezek közül az első négy magától értetődő. Az azonosság vizsgálatához két egyenlőség jelet kell írni, míg a különbözőségnél csak egyet. Ezek az összehasonlítások az operandusaik típusától függően vagy számértékként, vagy sztringként végzik el az összehasonlítást. A második eset azért érdekes, mert ha a „29” és „210” sztringeket lexikografikus rendezés szerint hasonlítjuk össze, akkor „210” kerül ki kisebbként, mivel balról az első különböző karakternél 1 kisebb, mint 9. Ha STRING belső típusú változót hasonlítunk össze más változóval, akkor sztring-összehasonlítás történik, egyébként numerikus.

Hátra van még két speciális összehasonlító operátor ismertetése: ~ és !~ a reguláris kifejezések illeszkedését vizsgálják. Az első igaz értékkel tér vissza, ha a bal oldalán álló kifejezésre illeszthető a jobb oldalán álló reguláris kifejezés, a másik pedig ennek ellentettje.

Műveletek

Aritmetikai műveletek, más programozási nyelvekhez hasonlóan itt is vannak. Az alapműveletek (+,-,*,/) rendelkezésre állnak, de mivel az `awk` lebegőpontos számokkal dolgozik, az osztásnál nincs egészre kerekítés (Erre van az int() függvény.) Ezeken kívül a maradékolás (%) és a hatványozás (^) használható még.

Sztringek összefűzése történik, ha egyszerűen egymás után írjuk őket. Ezt leggyakrabban a print művelet paraméterében tesszük: print "Vezetéknév: " $1 "\t keresztnév: " $2

Változók értékadása az egyenlőségjellel lehetséges. A bal oldalon álló változó a jobb oldalon lévő kifejezés értékét veszi fel ennek hatására. Az `a` változó értékének `b`-vel való növelésére kézenfekvő lehetőség a következő:

```
a = a + b
```

Ilyen és ehhez hasonló értékadások nagyon gyakran előfordulnak. Ezért az aritmetikai műveleteknek bevezették megfelelő párjaikat, amelyek a bal oldalukon álló változón és a jobb oldalukon álló kifejezésen elvégzik az adott műveletet, majd a bal oldali változónak ezt azonnal értékül is adják. Így az előző sor egyenértékű az alábbival:

```
a += b
```

Az összeadás mellett ilyen „egyenlősített” változata van még a kivonásnak, szorzásnak, osztásnak, maradékos osztásnak és a hatványozásnak is.

Az egyszerűsítésekkel még tovább mentek, mivel a programozás során rendkívül gyakori az eggyel való növelés és csökkentés is. Erre való a `++` és a `\-\-` unáris (egyoperandusú) művelet, amelyek a megadott változó értékét eggyel növelik, illetve csökkentik. Aszerint, hogy az operátort a változó elé vagy mögé írjuk, az így kapott kifejezés értéke a változó új, illetve régi értéke lesz. Legyen például `a=5`, majd hajtsuk végre a `b=a++` értékadást. Ennek hatására `b 5` lesz, `a` pedig 6. Ha ehelyett `b=++a`-t írtunk volna, akkor mindkét változó 6-tal lenne egyenlő.

Logikai kifejezések

Bizonyos esetekben a kifejezéseket logikai igaz-hamis értékként kell értelmeznie a rendszernek. Ekkor a következő két szabály érvényes. Ha a változó egy szám, értéke akkor „igaz”, ha nem nulla. Sztring esetében pedig a nem üres karakterlánc lesz igaz értékű. Ennek következtében a „0” sztringkonstans igaz értékű, de „0”+0 már hamis, mivel az utóbbi numerikus jellegű mennyiség.

Boolean-típusú, vagyis igaz-hamis értékű kifejezések között logikai műveleteket végezhetünk.

bool1 && bool2 Logikai „és” művelet; akkor lesz igaz, ha minkét operandusa igaz.

bool1 || bool2 Logikai vagy művelet; akkor igaz értékű, ha legalább az egyik oldalán igaz értékű kifejezés áll.

!bool A megadott kifejezés negáltját állítja elő.

Feltételes kifejezések

Van még egy, rendkívül tömör szerkezet, amellyel célszerű óvatosan bánni, mert könnyen áttekinthetetlenné válhat a program.

```
feltétel ? kif1 : kif2
```

Ennek értelmezése a következő: ha a feltétel kiértékelése igaz értéket ad, vedd kif1-et, egyébként kif2-t.

Műveleti precedenciák

Az ismertetett operátorok között előre definiált műveleti elsőbbségi sorrend van. Ez felülbírálható zárójelezéssel. Használjunk zárójeleket akkor is, ha csak egy kicsit is bizonytalanok vagyunk, mert így megelőzhető a hibás működés.

A következő táblázat összefoglalja a műveletek egymáshoz képesti viszonyát, a legerősebbtől a leggyengébbig haladva. Az egyes szinteken belül, kivéve az értékadásokat, a csoportosítás balról jobbra történik.

...	Zárójelezés.
$	Mező hivatkozás.
++ –	Növelő és csökkentő operátorok, mind a prefix, mind a posztfix változat.
^	Hatványozás.
! + -	Unáris műveletek.
/ %	Szorzás, osztás, maradékképzés.
+ -	Összeadás, kivonás.
szóköz	Sztring összefűzés.
< > <= > != == » |	Összehasonlító valamint átirányító műveletek.
~ !~	Reguláris kifejezésekhez kötődő összehasonlító műveletek.
in	Tömb tagsági művelet.
&&	Logikai ÉS.
||	Logikai VAGY.
?:	Feltételes kifejezés.
= += -= *= /= %= ^=&	Értékadások.

Ki- és bevitel

Az `awk` leginkább a szabványos bemenetet, illetve a bemenetként megadott fájlokat dolgozza fel, és a szabványos kimenetre ír. A szkript számára átlátszó módon történik a bemeneti fájlok olvasása; belülről nem dönthető el, hogy a szabványos bemenetről vagy fájlokból érkezik a feldolgozásra váró adat.

Az előzőekben elmondottak mellett lehetőség van egyéb fájlokból történő olvasásra, oda írásra, sőt, a shell segítségével más programok indítására azok szabványos ki- vagy bemenetét írva vagy olvasva. Ebben a fejezetben ezeket a lehetőségeket taglaljuk.

Az írás legfőképp a `print` művelettel történik. Használatának az alábbi módjai lehetségesek:

print Paraméterek nélkül meghívva az aktuálisan beolvasott sort ($0) írja ki.

print kif1 kif2 ... A megadott kifejezéseket sorban kiírja, újsor karakterrel elválasztva.

print kif1 kif2 ... < fájlvnév, print kif1 kif2 ... « fájlvnév, print kif1 kif2 ... | fájlvnév
A kifejezéseket nem a szabványos kimenetre, hanem a megadott fájlba írja. Az átirányításra a « is használható, ekkor hozzáír a fájlhoz, annak eredeti tartalma nem vész el. Pipe-ba írhatunk, ha a | karaktert használjuk az átirányításra.

Hasonlóan viselkedik a beolvasást támogató getline parancs:

getline A következő sort veszi az alapértelmezésként olvasásra használt fájlból (Ahonnan egyébként is olvassa a sorokat a működési ciklusok során). A $0 változóba kerül a beolvasott rekord.

getline fájlnév A megadott fájlból olvassa a következő sort, és a $0 változóba írja.

getline változó A következő beolvasott sort a megadott változóba teszi, $0 változatlan marad.

getline változó < fájlnév A következő sort az adott fájlból olvassa be, és a megadott változóba teszi, $0 szintén változatlan marad.

Pipe-okat használhatunk beolvasáskor is. Ekkor azonban a futtatandó parancssort kell előre írni, azután következik a `getline`. Például

```
"ls" | getline > nev
```

az aktuális könyvtár következő fájljának nevét a `nev` változóba írja. Pipe-ok és fájlok használata esetén fontos tudni, hogy az `awk` magától nem zárja le a megnyitott állományokat vagy pipe-ot, még a fájl vége jel (EOF) elérése után. Ha tehát elölről szeretnénk olvasni például egy fájlt minden működési ciklus során, akkor magunknak kell a lezárásról gondoskodni. Erre a `close(fájlnév)` parancs alkalmas. Adott esetben fájlnév helyett ugyanúgy írható ide megnyitott pipe neve is.

Szabályozható a beolvasás az alapértelmezés szerinti fájlokból is. A `next` utasítás azonnal a következő működési ciklusba kezd: beolvassa a következő sort és a szkript elejére helyezi a vezérlést. Ennél nagyobb lépést tesz meg a `nextfile` utasítás: ez nem a következő soron, hanem a következő fájlnál folytatja a feldolgozást.

Programok futtatására szolgál a `system()` függvény. A paraméterként kapott parancssort megpróbálja végrehajtani, majd visszatér annak kilépési kódjával (A nem POSIX szabványú rendszerekben ez nem biztos, hogy működik).

A `printf` utasítás formázott kiírást tesz lehetővé, nagyon hasonlít a C nyelvből ismerős `printf`-re. Használata során

```
printf formátum, kif1, kif2, ...
```

alakban írandó, ahol az első paraméter sztring kell, hogy legyen. Ez specifikálja a nyomtatás formátumát, illetve azt, hogy a további paraméterek milyen helyet foglalnak el a kiírt adatok közt. Használata a C nyelvet ismerők számára magától értetődő, a többiek számára a `man awk` parancs adhat megfelelő felvilágosítást.

Vezérlési szerkezetek

Az `awk` szkript akciókat tartalmazó részei több utasítást is tartalmazhatnak, ezeket vagy külön sorba kell írni, vagy pontosvesszővel kell őket elválasztani egy soron belül. A szkript végrehajtása során általában a leírt utasításokat egymás után veszi. Hatékonyabb, összetettebb programok írásakor azonban szükség lehet vezérlést irányító utasítások használatára. Ezek, mint a legtöbb procedurális nyelvben, itt is megtalálhatók.

A feltételes elágazás alapvető fontosságú az `awk` nyelvben is. Alakja

```
if (kifejezés) igaz-ág
 [ else ] hamis-ág
```

(A szögletes zárójelbe írt else-ág nem kötelező, de magukat a szögletes zárójeleket nem kell kiírni sohasem!) A működést úgy befolyásolja, hogy amennyiben a kifejezés kiértékelése igaz logikai értéket ad, az igaz-ág hajtódik végre, egyébként, ha van, a hamis ág. Ha több utasítást szeretnénk valamelyik ágba tenni, kapcsos zárójeleket kell használni.

Példák:

```
if (/alma/) print $0
```

Ha az aktuálisan olvasott sorra illeszkedik az alma reguláris kifejezés, kiírja az egész sort, egyébként nem.

```
 if (/alma/) {
print $0
almak++
}
else next
```

Hasonló az előzőhöz, de ha nincs illeszkedés, azonnal a következő sornál folytatja az olvasást. Emellett minden alkalommal, ha a beolvasott sorban szerepel az `alma` szó, megnöveli az `almak` változót.

Ciklusok szervezésére több lehetőség is van. Ha addig szeretnénk valamilyen műveletet ismételni, amíg egy bizonyos feltétel még teljesül, az alábbi szerkezetet hasznosíthatjuk:

```
while (feltétel-kif)
 törzs
```

Ez a végrehajtást a törzs ismételgeti mindaddig, amíg a feltételként megadott kifejezés kiértékelése igaz logikai értéket ad. Természetesen, ha az első vizsgálatnál a feltétel hamis, egyszer sem lesz végrehajtva az utasítás törzse.

Sokszor legalább egyszer végre kell hajtani az utasítás-blokkot ahhoz, hogy a feltételt megfelelően vizsgálhassuk. Erre való az alábbi szerkezet:

```
do
törzs
while (feltétel-kif)
```

Működése megegyezik az egyszerű while-ciklussal, kivéve az első végrehajtását a törzsnek, amely mindenképp megtörténik.

Változók léptetését megvalósító ciklus készíthető a `for` kulcsszó segítségével.

```
for (kezd; feltétel; léptetés)
 törzs
```

Ha a vezérlés eléri a `for`-t, végrehajtódik a `kezd` részbe írt utasítás. Ezután a `feltétel` vizsgálatával megkezdődik a ciklus. Ha a kiértékelés eredménye igaz, végrehajtódik a `törzs`. Ezután a újra visszakerül a vezérlés a `feltétel` vizsgálatához. Ez a kör egészen addig tart, amíg végül a `feltétel` hamissá válik, és a vezérlés elhagyja a `for` ciklust.

Példaként az alábbi ciklus kiírja 1-től 100-ig az egészeket:

```
for (i=1; i<=100; i++)
 print i
```

A megértés ellenőrzése érdekében gondoljuk végig, hogy az alábbi megoldás ugyanazt eredményezi, mint egy for ciklus:

```
kezd
if (feltétel)
do
  törzs
  léptetés
while (feltéten)
```

A ciklusok irányítására jól használható a `continue` és a `break` utasítás; a szakasz végén erről is szó lesz.

A tömbök elemein futhatunk végig a `for` másik alakját használva (A tömbök részletes ismertetése később jön).

```
for (változó in tömb)
műveletek tömb[változó]-val
```

A megadott változót a rendszer végiglépteti azokon az értékeken, amelyek indexként előfordulnak. Nagyon hasznos szerkezet, az alacsonyabb szintű programozási nyelvek nem tartalmaznak ehhez hasonló, kényelmes megoldásokat.

A `break` kulcsszó segítségével kiléphetünk az aktuális `while`, `do-while` vagy `for` ciklusból. Hasonló környezetben működik a `continue` is, de ez nem szakítja meg a ciklus működését teljesen, hanem azon nyomban újrakezdi a ciklust az elején. Ez a `while` esetében feltétel-ellenőrzéshez vezet, míg a `for`-nál léptetés és ellenőrzés is van. Fentebb megadtunk egy, a `for` működésével megegyező `if-do-while` szerkezetet. Világos, hogy `continue` használata eltérő működéshez vezet.

Változók

A „Kifejezések” szakaszban már részletesen foglalkoztunk a változókkal. Az alábbiakban a tömbökkel ismerkedünk meg, illetve az előredefiniált változók leírását közöljük.

Tömbök

Az `awk` nyelv támogatja az asszociatív tömbök használatát. Ezzel olyan eszköz kerül a kezünkbe, amely nagyban megkönnyítheti a munkát, és megkímélheti a programozót a vacakolástól, amelyet akkor kellene elvégeznie, ha ugyanezt mondjuk C nyelven akarná megírni. Ennek az ára a sebesség csökkenése, de ez csak iszonyatos méretű fájlok esetén volna észrevehető.

A tömb olyan változó, amely mögé szögletes zárójelbe különböző értékeket írva gyakorlatilag különböző skalár (=nem tömb) változókat kapunk. Például

```
  tomb[1]="alma"
tomb[2]="korte"
tomb["paradicsom"]=16
```

Legtöbb programozási nyelv tömbjei csak nem negatív egészekkel indexelhetők. Az `awk`, a `perl`-hez hasonlóan sztring kifejezéseket is megenged indexelésre. Képzeljük el, egy fájlba soronként egy szó szerepel. Szeretnénk statisztikát készíteni,

mégpedig arról, hogy melyik szó hányszor fordul elő a fájlban. Erre alkalmas a következő szkript:

```
   { szavak[\$0]++}
END { for (szo in szavak) print szo ":" szavak[szo] }
```

Tetszőleges szó esetén, a szavak tömbök azzal a szóval indexelve egy önálló változót kapunk (Kezdetben minden numerikus változó nulla értékű, a sztringek pedig üresek). Egy szó számba vétele esetén a tömb neki megfelelő változóját, amely az eddigi előfordulások számát tartalmazza, eggyel megnöveljük. Végül kiírjuk a tömb egyes elemeinek tartalmát, és persze az indexelő elemet, amely a szó volt.

Ahogyan új elemeket szúrhatunk be egy tömbbe, meglévőket törölni is lehet. Erre való a delete utasítás. A paraméterben megadott tömb elemet törli:

```
delete tomb["alma"]
```

Ha az egész tömböt törölni kívánjuk, akkor index nélkül adjuk meg a nevét a delete után: `delete tomb`.

Az is ellenőrizhető, hogy valamilyen index szerepel-e a tömb indexei között. A tartalmazás operátor használatát egy példa illusztrálja:

```
if ("tigris" in szavak)
print "Volt már tigris is:" szavak["tigris"]
```

A többváltozós tömböt úgy lehet elképzelni, hogy egy tömb elemei tömbök. Ekkor két változóval is indexelni kell, hogy megkapjuk a megcímzett skalárt. Ennek a hagyományos módja az, hogy vesszővel elválasztva írjuk a változókat az indexbe: `tablazat[x,y]`. Az `awk` ilyenkor gyakorlatilag összefűzi a megadott változókat, és így egyetlen sztringet kapva, azzal indexel egy egydimenziós tömböt. Az indexek összefűzése a SUBSEP nevű beépített változó értékének közbeékelésével történik. Ez alapértelmezés szerint egy nem nyomtatható karakter, „034”. Ha átállítjuk, figyelni kell az ebből adódó potenciális hibalehetőségekre. Például SUBSEP="" hatására a teszt tömb következő két eleme a várakozásokkal ellentétben ugyanaz a változó:

```
teszt["a@b","c"] és teszt["a","b@c"].
```

Többdimenziós tömbök indexének tartalmazásvizsgálatára is alkalmaz az `in` operátor. Használatának formája:

```
(var1, var2, ... ,varn) in tömb
```

kifejezés visszatérési értéke igaz, ha van ilyen indexe a megadott tömbnek, és hamis egyébként. Ugyancsak lehetőség van `for` ciklussal végigfutni a tömb összes elemén. Ehhez azonban egy kis trükkre van szükség:

```
for (összetett in tömb) {
split(összetett, szétbontott, SUBSEP)
...
}
```

Ilyenkor az összetett nevű sztring típusú változó végigfut a tömb lehetséges indexein. Mivel többdimenziós tömbről van szó, a futó változó SUBSEP-pel elválasztott értékeket tartalmaz. Ezeket szedi szét a `split` függvény, és darabjai a szétbontott tömbbe kerülnek. A `split` függvény részletes ismertetése a „Beépített függvények” szakaszban található meg.

Beépített változók

A legtöbb változót saját céljaira használhatja a programozó az `awk` szkriptek megírása során. Ez alól néhány kivétel van. Bizonyos változók arra használhatók, hogy befolyásolják az `awk` alapértelmezés szerinti működését; segítségükkel finomhangolható, és ezzel hatékonyabbá tehető a feldolgozás.

Először az `awk`-t közvetlenül vagy közvetve irányító változók bemutatása következik:

CONVFMT Ez a sztring szabályozza a számok sztringgé alakítását. Úgy működik, mint az `sprintf` első argumentuma. Alapértelmezés szerinti értéke "%.6g".

FIELDWIDTHS Szóközökkel elválasztott számértékek felsorolásának kell lennie. Ha nem üres, akkor a beolvasott sorok pontosan ilyen széles oszlopokra bomlanak, így adva a $1, $2, stb. változókat. Ha nem adtunk neki értéket, a bontás FS szerint történik.

FS A bemeneti mezők elválasztója. Ha értéke egyetlen karakterből álló sztring, akkor ezzel a karakterrel kerülnek feldarabolásra a beolvasott sorok. Ha üres (""), akkor a beolvasott sor minden karaktere külön mező lesz.

Alapértelmezés szerint egyetlen szóközt tartalmaz az FS változó. Ez egy kivételes eset, ilyenkor szóközök és tabulátorok tetszőleges sorozata elválasztóként működik. Továbbá a sorok elején és végén lévő szóközök és tabulátorok figyelmen kívül maradnak.

Amennyiben értéket adunk az előzőekben ismertetett FIELDWIDTH változónak, az `awk` átvált fix szélességű oszlopokra bontás módra. Ha szeretnénk újból tartalom szerinti mezőkre bontással dolgozni, az FS változónak kell értéket adni. Erre a legegyszerűbb mód az FS=FS utasítás.

Általánosabban, tetszőleges reguláris kifejezés is adható (idézőjelek között) FS-nek értékül. Ekkor a beolvasott sorra való diszjunkt (egymásba nem érő) illeszkedések szabdalják szét azt mezőkre.

IGNORECASE Ha ez a változó nem nulla és nem is üres sztring, vagyis logikai értéke igaz, akkor a reguláris kifejezések illesztésénél a kis- és nagybetűk között nem tesz különbséget a rendszer.

OFMT Számok sztringgé konvertálását szabályozza a `print` utasítás végrehajtása során.

OFS (Output Field Separator) Az egyetlen `print` utasítással a kimenetre írt változók közti szeparátor, kezdeti értéke egyetlen szóköz.

ORS (Output Record Separator) Minden `print` után kiíródik ez a változó. Alapértéke „\n”, vagyis az újsor karakter.

RS (Record Separator) A bemeneti fájl kisebb egységekre darabolását szabályozza. Alapbeállításként egy újsor karaktert tartalmaz, ezért a bemenetet soronként dolgozza fel az awk. Ha az FS-hez hasonló módon ezt átállítjuk, akkor a beolvasott fájlok nem feltétlenül sorokra tagolódnak, hanem ún. rekordokra bomlanak. Az egyértelműség kedvéért ebben a fejezetben mindenütt az alapértelmezést vettük figyelembe, amikor sor-orientált feldolgozóként mutattuk be az awk nyelvet. A működést nem befolyásolja kiszámíthatatlanul, ha a tagolás nem a sorvégeknél történik.

Üres sztringet megadva RS-nek, az üres sorok mentén bomlik rekordokra a bemeneti fájl. Általánosabb szerkezeteket reguláris kifejezések megadásával alakíthatunk ki.

SUBSEP Többdimenziós tömbök indexelése során gyakorlatilag az indexváltozók összefűzésével kapott sztringgel indexel a rendszer. Az összefűzés során az egyes változók közé a SUBSEP karakterlánc ékelődik. Alapértelmezés szerint ez egy nem nyomtatható karakter, tehát nem veszíthetjük el az egyértelműséget (Lásd bővebben a „Tömbök” szakaszt).

Az alábbi változók információt hordoznak. Módosításuk nem célszerű, bár néha megfontolt írásukkal ki lehet használni bizonyos tulajdonságaikat (ezt helyben megemlítjük).

ARGC és ARGV Az ARGV egy tömb, amelyet 0-tól ARGC-1-ig egész számokkal indexelhetünk. A tömb tartalmazza az awk meghívásakor a paraméterként átadott fájlneveket, amelyekből a bemenet formálódik. Ez a példa egyszerűen kiírja ezeket a fájlneveket:

```
BEGIN {
  for (i=0; i<ARGC; i++)
    print ARGV[i]
}
```

Hasonlóan a C nyelvhez, ARGV[0] itt is a futó program neve, általában awk.

ARGIND Az éppen feldolgozás alatt álló bemeneti fájl sorszáma.

Az ARGV [ARGIND] tömbelem pedig e fájl nevét adja meg. Ez utóbbi megegyezik a FILENAME változó tartalmával.

ENVIRON Egy asszociatív tömb, amely segítségével elérhetjük a shell környezeti változóit. Például ENVIRON["PATH"] a futtatható programok elérési útjait adja meg, míg ENVIRON["HOME"] a felhasználó saját könyvtárát. Értékeinek megváltoztatása hatástalan az awk által elindított további folyamatokra nézve.

ERRNO Ha a `getline` vagy a `close` művelet végrehajtása során hiba lép fel, ez a változó tartalmazza annak leírását.

FILENAME A feldolgozás alatt álló fájl neve. Ha a szabványos bemeneten dolgozik az awk, akkor ez „-”. A BEGIN által kijelölt akció végrehajtása során a FILENAME üres, mivel még nem kezdte meg a rendszer a beolvasást.

FNR Az aktuális rekord sorszáma az éppen feldolgozás alatt álló fájlban. Minden új rekord beolvasásakor eggyel növekszik. Új fájlok megkezdésekor nullára állítódik. Ennek megváltoztatását ki lehet használni (Alapértelmezés szerint a rekordok a sorok).

NF Az aktuális sorban (rekordban) lévő mezők száma. Ezek a mezők $n hivatkozással érhetők el, ahol n egy 1 és NF közötti egész szám.

NR Az eddigi teljes feldolgozás alatt beolvasott rekordok száma.

RLENGTH A `match` függvény által illesztett sztring hossza karakterekben kifejezve. Sikertelen illesztésnél -1.

RSTART A kezdő karakter sorszáma a `match` függvény által illesztett résznek, és 0, ha nem sikerült az illesztés.

RT A rekord beolvasásakor az RS által illesztett részt tartalmazza. Értelme akkor van, ha reguláris kifejezést használunk a bemenet rekordokra bontására, különben úgyis egy \n karakter az értéke.

Függvények

A függvények segítségével sok hasznos művelet végezhető a változókon. A függvényeket a nevük után zárójelben, vesszővel elválasztott paraméterekkel lehet meghívni. Némelyek visszatérési értékként adják vissza eredményüket, illetve a végrehajtás sikerére vonatkozó információt.

```
eredmeny=fuggveny(p1,p2)
```

Ha egyetlen paraméter sem szükséges a híváshoz, a zárójeleket akkor is ki kell tenni. Enélkül az `awk` számára nem volna egyértelmű, hogy függvényt akarunk-e hívni, vagy egy változóra hivatkozunk. Fontos, hogy soha ne hagyjunk szóközt a függvény neve és a paramétereket tartalmazó zárójel között!

Beépített függvények

A beépített függvényeket az `awk`-ban mindig használhatjuk, a szkript különböző területeiről, ahol ez szintaktikailag lehetséges. Ezekkel szemben állnak a felhasználó által definiált új függvények, amelyeket csak a definíciót tartalmazó szkriptben használhatunk.

Az alábbi lista a számokkal dolgozó függvényeket veszi sorra:

int(x) A megadott számot a tizedesjegyek elhagyásával egész számmá alakítja. Így a pozitívakat lefelé, a negatívakat pedig felfelé kerekíti.

sqrt(x) Egy nem negatív szám négyzetgyökét adja vissza, egyébként hibát jelez.

exp(x) Exponenciális függvény, az `e` szám megadott hatványát számítja ki.

log(x) Természetes alapú logaritmus, csak pozitív számokra működik helyesen, egyébként hibát jelez.

sin(x), cos(x) Trigonometrikus függvények, ahol a paraméter radiánban értendő.

atan2(x,y) Megadja `y/x` arcus tangensét.

rand() Egy 0 és 1 közötti véletlenszámot ad. Az eloszlás egyenletes, továbbá nulla és egy sohasem jön ki.

srand([x]) A véletlenszám generátort inicializálja a megadott értékkel. Paraméter nélkül meghívva a pillanatnyi dátum és idő segítségével készíti elő a véletlenszám generátort.

Ugyancsak rendkívül hasznosak az alábbi, főként karakterláncok manipulálására, feldolgozására használható függvények.

index(sztring, resz) Az első paraméterként kapott sztring változóban megkeresi a második paraméter sztring előfordulását részsztringként, majd visszatér az első ilyen előfordulás kezdőkarakterének sorszámával. Ha nincs találat, 0-t ad vissza.

length([sztring]) A megadott sztring karakterekben számított hosszát adja vissza. Ha számot adunk meg, akkor azt sztringgé alakítja, és annak a hosszát kapjuk meg. Ha nem adunk meg argumentumot, akkor a $0 hossza lesz az eredmény.

match(sztring, regkif) Reguláris kifejezést próbál illeszteni a sztring változóra. Visszatérési értéke az első illeszkedés kezdete. Sikertelen illeszkedés 0 visszatérési értéket eredményez. Emellett még a beépített RSTART és RLENGTH változókat használja az eredmény visszaadására. RSTART ugyanaz, mint a visszaadott szám, RLENGTH pedig az illesztett rész karakterekben mért hossza.

split(sztring, tömb [,szeparátor]) A megadott sztring változót feldarabolja a `szeparátor` segítségével. Az eredmény a `tömb`-be kerül: az első darab a `tömb[1]`, a második `tömb[2]`, stb. Ha a harmadik paramétert elhagyjuk, az aktuális FS-t használja, csakúgy, mint a beolvasott sor mezőkre bontásánál.

Ha a szeparátor szóköz (" "), akkor szóközök és tabulátorok minden sorozata elválasztó, és a sztring elejéről és végéről is eltünteti ezeket a karaktereket. Ha tetszőleges másik karaktert adunk meg szeparátor-nak, akkor annak minden előfordulása elválasztó hatású lesz. Ha több karakterből álló sztringet adunk meg, az reguláris kifejezésként értelmeződik, és minden illeszkedése a sztring-re egy elválasztást jelent.

sprintf(formátum, kif1, ...) A C nyelv `\printf` függvényéhez hasonló, visszatérési értéke a megadott kifejezés-sorozat formátum szerinti sztringgé alakított változata.

sub(regkif, csere [,eredmény]) A megadott reguláris kifejezés első illeszkedését az eredmény változóban kicseréli a csere által megadott szövegre. A harmadik paraméter nem lehet konstans, mert a függvény megváltoztatja az értékét. Ha nem adunk meg semmit azon a helyen, a változtatások a $0 változót érintik. Visszatérési érték a sikeres változtatások száma.

A csere sztringben a & karakterek a teljes illesztett részre helyettesítődnek. A & karakterhez \& -t kell írni, mivel önálló backslash-t már sztringbe is duplán írandó.

gsub(regkif, csere [,eredmény]) Az előzőhöz hasonló művelet, de itt nem csak az első, hanem az összes további illeszkedés is kicserélődik a megfelelő módon.

gensub(regkif, csere, hogyan [,sztring]) A `sub` és a `gsub` függvények általános változata. A `hogyan` paraméter segítségével specifikálható, mely illeszkedések esetén történjen meg a csere. Ha egy számot adunk meg, az annyiadik illeszkedés lesz kicserélve, míg ha g vagy G betűvel kezdődő sztringet, akkor mindegyik, ahogy azt a `gsub` is teszi.

Vigyázat: ez a függvény már nem módosítja a megadott változót. Az eredményezett sztring a visszatérési érték.

További általánosítást jelent, hogy a csere sztringben hivatkozhatunk a megadott reguláris kifejezés részkifejezéseinek illeszkedésére \1, \2, stb. alakban. Ez teljesen hasonlóan megy, mint a `sed s` parancsánál.

substr(sztring, kezdet [,hossz]) A sztring részsztringjét adja vissza a `kezdet` sorszámú karaktertől kezdődően, `hossz` hosszan. Ha nem adunk meg `hosszat`, a részsztring az eredeti karakterlánc végéig tart majd.

tolower(sztring) A sztring-ben előforduló nagybetűket kicsivé alakítja.

toupper(sztring) A sztring-ben található kisbetűket naggyá alakítja.

A ki és bevitel tárgyalásánál már volt szó az alábbi beépített függvényekről: `close, system`. Az `fflush()` függvény a megadott fájlnévvel azonosított puffer tartalmát azonnal kiírja. Különböző `awk` megvalósítások másként kezelik, implementálják.

A `systime()` függvény visszaadja a rendszer időszámításának kezdete óta eltelt másodpercek számát. Ez POSIX kompatibilis rendszerekben 1970 január 1-e. Az aktuális időt a `strftime` függvénnyel kérdezhetjük le. Paraméterezése hasonlít az ANSI C megfelelő függvényének paraméterezési szabályaira. Ennek részleteibe itt nem megyünk bele, a C változat manual lapján minden megtalálható (`man strftime`). Paraméter nélkül hívva az alábbihoz hasonló eredményt ad:

```
Mon Jul 23 14:19:25 CEST 2001
```

Felhasználói függvények

A felhasználó által definiált függvények a szkriptben tetszőleges helyen, a felsorolt minta akció feldolgozási szabályokon kívül lehet definiálni. Egy ilyen definíció formája:

```
function name(paraméter-lista)
{
  függvény törzse
}
```

A függvény törzsében írjuk le, hogy a megadott paraméterekkel milyen műveleteket végezzen az eljárásunk. A `return` utasítás szolgál a visszatérésre, az utána írt kifejezés lesz függvényünk visszatérési értéke. Például az alábbi függvény a megadott számot a saját, kívánság szerinti formátumunkban írja ki:

```
function myprint(szam)
{
  printf "%6.3g\n", szam
}
```

Az alábbi függvény pedig visszatér a megadott számok szorzatával:

```
function szoroz(a,b)
{
  return a*b
}
```

Az általunk definiált függvényeket pontosan ugyanúgy hívhatjuk meg, mint ahogy azt a beépített függvényekkel tettük.

Reguláris kifejezés sajátosságok

Az első részben bemutatott reguláris kifejezés szintaktika majdnem teljesen megfelel annak, amilyet az `awk` használ. Néhány kiegészítés itt is van, amelyek kényelmesebbé tehetik a munkát.

\w Szavak tetszőleges karakterére illeszkedik, megfelel [[:alnum:]_]-nak.

\W Az előző ellentettje, tehát szavakon kívüli karakterre illeszkedik, ugyanaz, mint [^[:alnum:]_].

\< Szavak elejére, de ott egy nulla hosszúságú részre illeszkedik.

\> A szavak végére illeszkedik, üresen.

\y Mind a szavak elejére, mind azok végére illeszkedik, nulla hosszan.

\B Az előző ellentettjeként szavak belsejében illeszkedik nulla hosszan.

\‘ A buffer elejére illeszkedik.

\’ A buffer végére illeszkedik. Az előzővel együtt csak kompatibilitási okok miatt hagyták meg, ^ és $ mellett nem jelentenek új képességeket.

Az awk indítása

A parancssorról elmondtuk, hogy az első kapcsoló nélküli paramétere a szkript, illetve `-f` kapcsoló esetén a szkriptet tartalmazó fájl, utána pedig a bemenetként feldolgozandó fájlok listája következik.

Lehetőség van az előbb említett paraméterek elé egyéb opciók beszúrására.

-F fs A mező szeparátor (FS) változót állítja be.

-f forrás-fájl Jelzi, hogy a megadott forrás-fájl tartalmazza a futtatandó szkriptet.

-v vált=ért Kezdeti értéket ad a `vált` változónak.

-mf NNN, -mr NNN Memóriakorlátot állít fel. Az `f` jelző a mezők számát maximalizálja a megadott értékkel, míg az `r` a beolvasható rekordok méretét. Csak kompatibilitási okokból tartották meg.

-W opció A POSIX szabvány előírásainak megfelelően kompatibilitási opciók adhatók meg. Ezekkel itt nem foglalkozunk.

Jelzi, hogy a következő argumentum már nem opció, hanem fájlnév, még akkor is, ha kötőjellel kezdődik. Így használhatunk - kezdetű fájlneveket is.

A további paramétereket fájlnévnek tekinti a rendszer a következő kivétellel. Ha a paraméter `vált=ért` alakú, akkor közvetlenül az ezután következő bemeneti fájl beolvasásának megkezdése előtt elvégzi az értékadást. Például

```
awk -f szkript k=1 elso.txt k=2 masodik.txt
```

parancs hatására az elso.txt megkezdése előtt a `k` változót 1-re állítja, a második fájl előtt pedig 2-re.

Példák

Tegyük fel, egy szövegfájlról statisztikát szeretnénk készíteni, amelyből megtudhatjuk, mely szavak milyen gyakorisággal szerepelnek benne. Nagyon jól használhatók ilyenkor az `awk` asszociatív tömbjei.

A módszer legyen a következő. Minden egyes sort szétbontunk szavakra. Ezeket a szavakat sorra vesszük, és segítségükkel megcímezzük a `szavak` tömböt. Így minden szóhoz tartozik egy változó, amelybe az illető szó eddigi előfordulásainak száma van. Ilyenkor ezt a számot megnöveljük. A fájl végére érve a tömb egyes index-elem párjait kiírjuk. Ezt a sort parancs bemenetére küldve rendezett listát kapunk az egyes szavakról.

A `szokincs.awk` szkript így néz ki:

```
{   split(\$0,
sor_szavai, /[\^[:alpha:]]+/ );   for (i in sor_szavai) {
if (length(sor_szavai[i])>=4) {
szavak[sor_szavai[i]]++;     }     } }

END {
  for (szo in szavak) {
    szam = sprintf("%4d",szavak[szo])
    szam = gensub(/ /,"0","g",szam)
    print szam ":" szo
  }
}
```

Itt nincs szükség speciális címzésre, minden sorral ugyanaz történik. Szétbontjuk a `sor_szavai` tömbbe betűktől különböző elválasztó szakaszokkal. Ezen tömb minden elemére, vagyis a sor szavaira megnöveljük a hozzájuk tartozó változókat eggyel, jelezve hogy megint előfordultak (ezt csak a három karakternél hosszabb szavakra tesszük, most csak az az érdekes).

Amint a fájlnak vége, az END cím alatti rész hajtódik végre. Itt a kimenetre írjuk az előfordulások számát és a szavakat, soronként egy-egyet. Ahhoz, hogy ezt a sort parancs megfelelően rendezze, a számokat ugyanolyan hosszan, eléírt nullákkal írjuk ki. Az `sprintf` függvény szóközöket ír a számok elé, amelyeket aztán a `gensub` függvény segítségével nullákra cserélünk.

Lefuttatva a programot jelen könyv fájlján az

```
awk -f szokincs.awk segedprg.sgml | sort
```

paranccsal, megkapjuk a kívánt eredményt. A leggyakoribb szavak a következők:

```
0031:sztring
0033:fájl
0033:illeszkedik
0033:karakter
0037:lesz
0038:csak
0038:után
0040:sort
0045:aktuális
0047:következő
0048:kell
0053:vagy
0057:akkor
0062:kifejezés
0064:reguláris
0069:parancs
0076:első
0076:megadott
```

(Itt most nem mutatjuk a szöveg formázására használt sgml-tag neveket, amelyek sokkal többször fordultak elő, mint bármely szó.) Továbbá a magyar szavak ragozási lehetőségeit is figyelmen kívül hagytuk: bár a „sor” tárgyesete 40-szer is előfordul, lehet, hogy más ragozott vagy ragozatlan formáival együtt sokkal előrébb áll.

5. fejezet

Perl

Ebben a fejezetben röviden vázoljuk a `Perl` nyelv azon részeit, amelyek az előzőekben elsajátított ismeretek birtokában könnyen megérthetők. A `Perl` a `bash`, a `C`, a `sed` és az `awk` nyelvek összevegyítéséből keletkezett, mindegyiktől átvéve annak valamely jól használható részét. A `Perl` interpretált nyelv, tehát nem fordítunk gépi kódot futtatás előtt, hanem az interpreternek megadva a programunkat az értelmezi az utasításokat és azonnal futtatja (léteznek fordítók is, de nem túlságosan elterjedtek). A programkódot általában egy fájlba írjuk, majd a

```
perl forrás.pl
```

utasítással futtathatjuk.

Egy `Perl` program felépítése lényegében olyan, mint az `awk` szkriptek akció része: utasítások sorozata. Tehát nincs címzés, mint a `sed`-nél, vagy az `awk`-nál, hanem a vezérlés a program utasításain lépked végig. Induláskor a program első sorával kezdi a végrehajtást, majd a következőn folytatja, hacsak valamely vezérlő szerkezet másként nem rendeli. Az utasításokat pontosvessző választja el egymástól.

5.1. A Perl nyelv elemei

5.1.1. Változók

`Perl` változó három féle lehet: skalár, lista és asszociatív tömb. Az első és a harmadik nagyon hasonlít az `awk` megfelelő típusaira. A lista lényegében 1-től kezdődő egészekkel indexelt tömb, de leginkább listák összefűzése, listákra való bontás és más olyan listaművelet során használjuk őket, ahol tömb jellegük elhomályosodik. A változók típusának jelzésére prefix karaktert kell használni: $a skalár, @a lista, %a pedig asszociatív tömb (hash).

A skalár változók használata szinte teljesen megegyezik az `awk`-nál látottakkal. Ugyanúgy lehet neki értéket adni, csak a megelőző dollárjelről ne feledkezzünk el!

```
\$szam= 16;
\$masik = "A nyertes szám: \$szam"
print "\$szoveg\n"
```

Látható, hogy az idézőjelek használata pontosan úgy megy, mint a `bash`-nál. Hasonlóan, lehetőség van szimpla idézőjelek, vagyis aposztrófok használatára (’), ezek között nincs behelyettesítés. A fordított aposztrófok (‘) közötti szöveg pedig parancsként végrehajtódik a shell segítségével.

Az előbb említett „idézőjelezéseknek” van általánosabb formája. Ha több soron át tartó szöveget szeretnénk egy változónak értékül adni, azt így oldhatjuk meg:

```
\$szoveg = <<VEGE
Ide jön a szöveg,
egész sok is lehet.
VEGE
```

A két < karakter kijelöli, hogy addig tartson a sztring a következő sortól kezdve, amíg önálló sorként elő nem fordul mégegyszer az utána megadott karaktersorozat (jelen esetben a VEGE).

Listák indexelése szögletes zárójelekkel történik. Például a @dolgok lista első eleme $dolgok[0]. Figyeljük meg, hogy amikor az egész listára hivatkoztunk, @ szerepelt a név előtt. Viszont egy konkrét elemére hivatkozva, amely skalár, a $ kerül a változó neve elé.

Asszociatív tömbök indexelése kapcsos zárójelek segítségével történik. Például legyen %napkelte egy asszociatív tömb. Ekkor egy elemének értéket így adhatunk:

```
$napkelte{’Hétfő’}=’5:15’
```

A listákhoz hasonlóan, a hivatkozott elem már skalár, előtte a $ jelnek kell állni.

Mindenképp szót kell még ejteni a `Perl` egy érdekes tulajdonságáról, az alapértelmezett változóról. Ha bizonyos műveletekhez nem írunk operandusként változót, akkor azok egy ún. alapértelmezés szerinti változón hajtódnak végre. Erre a változóra egyébként explicite is hivatkozhatunk: $_ néven. Tekintsük a következő programrészletet:

```
\$valt = "Ez egy hosszu szoveg...";
\$valt =\~ s/a/ax/;
if (\$valt =\~ /regkif/) {
  \$valt =\~ s/b/bx/;
}
...
\$valt =\~ y/abc/ABC/;
print \$valt;
```

(Nincs sok értelme ebben a formában, de hasonló esetek előfordulhatnak, például ha karakterkódolások között konvertálunk fájlokat.) A lényeg az, hogy ugyanazon a változón sok vizsgálatot és műveletet végzünk. Ilyenkor érdemes használni az alapértelmezett változót:

```
\$\_ = "Ez egy hosszu szoveg...";
s/a/ax/;
if (/regkif/) {
  s/b/bx/;
}
...
y/abc/ABC/;
print;
```

Elhagyva mindenhonnan a $valt változót, végig az alapértelmezett változón dolgoztunk, a hatás ugyanaz. A program áttekinthetőbb és olvashatóbb lett (azok számára, akik hallottak már $_-ról).

Létezik egy speciális érték változóknál, az üres érték (undefined). Ilyen minden használatlan változó, amelyekbe addig nem írtunk semmit, illetve bizonyos függvények is ilyet adhatnak vissza. Feltételes kifejezésben (if) hamis értéket ad egy üres változó vizsgálata.

Reguláris kifejezés műveletek

A `Perlben` használt reguláris kifejezés szintaktika nagyjából megegyezik az `awk`-nál látottakkal. Ugyanúgy a ~= operátor használható reguláris kifejezés illesztésére. Ami érdekességként előjön az az, hogy egy illesztés után a $1, $2, stb. változók sorra tartalmazzák a reguláris kifejezés részkifejezései által illesztett rész-sztringeket:

```
if ( \$a =\~ /([\^x]*)x(.*)/ )
{
  print "\$1 és \$2\n" ;
}
```

Ez a példa az $a változóra illeszt egy mintát. Ha az illesztés sikerül, kiírja a minta két, zárójelekkel megjelölt részmintája által illesztett részt (a minta egyébként az első x betűnél szétbontja a megadott sztringet).

A reguláris kifejezéseken alapuló csere (substitution) is átkerült a `sed`-ből a `Perl`-be, némi kiegészítéssel. Formája:

```
s/regkif/csere/jelzők
```

A jelzők vagy módosítók köre bővül a már megismert változathoz képest (az első négy módosító reguláris kifejezések után is használható /regkif/módosító alakban):

i Kis- és nagybetűk különbségét figyelmen kívül hagyja.

m Többsoros sztringben keresünk.

s Tekintse egyetlen sornak a sztringet.

x Kibővített reguláris kifejezés szintaktika használata. Ennél szóközöket írhatunk a reguláris kifejezésekbe, azokat több sorra tördelhetjük, valamint megjegyzések is szerepelhetnek bennük.

g Általános csere, amikor nem csak az első illeszkedés, hanem az összes alkalmával megtörténik a módosítás. Ugyanaz, mint a `sed` vagy az `awk` esetében.

e A `csere` részt ne hagyományos módon tekintse, hanem mint egy `Perl` kifejezést (expression). Így például számítások végezhetők az illesztett részekkel:

```
\$a= "1x2";
if (\$a =\~ s/([0-9]*)x([0-9]*)/\$1*\$2/e )
{
  print "\$a\n" ;
}
Ez az x karakterrel elválasztott két számot azok szorzatával helyettesíti.
```

Ez az x karakterrel elválasztott két számot azok szorzatával helyettesíti.

Fájlkezelés

Fájl változók léteznek a `Perl` nyelvben. Ezeket általában meg kell nyitni használat előtt. Erre használható az `open` függvény:

```
open(FILE, "fájlnév")
```

Ezután kisebb-nagyobb jelek közé zárva a `fájl változót` egyszerű értékadással olvashatunk ki egy-egy sort a fájlból:

```
$sor = <FILE>
```

Az olvasás sikertelensége (fájl végének elérése) esetén üres értéket kapunk vissza. Végül a `close` függvénnyel zárhatjuk le így:

```
close(FILE).
```

Gyakran a `Perl` programok a szabványos bemenetet és kimenetet használják. Egy sor olvasása a szabványos bemenetről a STDIN fájl változó segítségével lehetséges. Ha soronként szeretnénk egy ciklusban olvasni egészen a fájl végéig, célszerű azt ilyen formában írni:

```
while (\$sor = <STDIN>) {
  ...
}
```

Ilyenkor a ciklus törzsében használhatjuk a $sor változót, amely mindig egy beolvasott sort tartalmaz. Még egyszerűbb a helyzet, ha az alapértelmezés szerinti változót használjuk a beolvasott sor tárolására:

```
while (<STDIN>) {
  ...
}
```

Ekkor a ... helyén a $_ változóval hivatkozható az éppen beolvasott sor, de mivel az alapértelmezett változóról van szó, sok helyen ez a hivatkozás is elhagyható.

Még egy egyszerűsítési lehetőség van. Sokszor kell olyan programot írni, amelyet vagy fájlnevek megadásával paraméterezünk, és akkor azon fájlokat kell feldolgoznia, vagy paraméter nélkül hívjuk és a szabványos bemenetről dolgozik. Ha az üres fájlváltozót használjuk, a Perl a parancssorában megadott, a Perl szkripten kívüli fájlokat adja a programunk bemenetére, vagy a standard bemenetet. Ilyenkor a szkriptben ezekkel nem kell különösebben bajlódni:

```
while (<>) {
  ...
}
```

Az alapértelmezett változó a kívánt sorokon halad végig. Sőt, létezik olyan parancssori opció, amely a `perl` szkript köré írja az itt vázolt szerkezetet.

Példák

Először lássunk néhány hasznos parancssori opciót. Ha a `sed`-hez hasonló módon szeretnénk a `Perl`-t használni, használjuk a `-n` opciót. Ez olyan, mintha a `Perl` programunk az alábbi ciklusba lenne ágyazva:

```
while (<>) {
     ...             \# ide jön a szkript
}
```

Ez a ciklus az alapértelmezett változót iterálja a parancssori paraméterként megadott fájlokon illetve, ha olyan nem volt, a szabványos bemeneten. Nagyon hasonló működést kapunk így ahhoz, ha a `sed -n` formában hívjuk a folyamszerkesztőt. Például

```
perl -n 's/x/y/g; if (/banan/) { print; }' fájl1 fájl2 ...
```

a megadott fájlokat dolgozza fel. Ha a `-p` opciót is használjuk, akkor annak hatására a ciklus végén automatikusan kiíródik az alapértelmezett változó. Jól használható ez egyszerűbb `sed` emulációkra azoknak, akik jobban szeretik a `Perl` reguláris kifejezéseit:

```
perl -p -e szkript [fájl1 fájl2 ... ]
```

Most az `awk`-nál bemutatott szógyakorisági statisztika program `Perl` változatát vizsgáljuk meg:

```
while (<>) {
  @sor_szavak=split(/[\^[:alpha:]+]/);
  for \$szo (@sor\_szavak)        {
      if (length(\$szo)>=4) {
      \$szavak{\$szo}++;
    }
  }
}

for \$szo (keys %szavak) {
  \$szam=\$szavak{\$szo};
  while (length(\$szam)<4) {
    \$szam="0\$szam";
  }
  print "\$szam:\$szo\n";
}
```

A működés teljesen analóg. Minden sor beolvasásakor a `@sor\_szavak` listába kerülnek az adott sor szavai (a `split` utasítás az alapértelmezés szerinti változót bontja). A háromnál több karakterből álló szavaknál a `%szavak` nevű asszociatív tömb (hash) megfelelően indexelt elemét növeljük eggyel.

A feldolgozás befejezésével a `while` ciklus után a `for` végiglépteti a `$szo` változót a `%szavak` hash kulcsain, vagyis a potenciális indexeken. Ezek segítségével megkapjuk az előfordulások számát, amelyet balról nullákkal egészítünk ki. Végül az eredményt kiírjuk.

Mind az `awk`-nál, mind a `Perl` nyelvnél nagy segítségünkre volt a nyelvi támogatás az asszociatív tömbök használatához. Enélkül igen nehezen és kényelmetlenül tudtuk volna megoldani a szavakhoz kapcsolt változók elérését.

GNU Szabad Dokumentációs Licensz 1.1 verzió, 2000 március

0. ELŐSZÓ

Jelen Licensz célja egy olyan kézikönyv, tankönyv, vagy effajta írott dokumentum megalkotása, mely a szó szoros értelmében „szabad”: annak érdekében, hogy mindenkinek biztosítsa a szöveg sokszorosításának és terjesztésének teljes szabadságát, módosításokkal, vagy anélkül, akár kereskedelmi, akár nem-kereskedelmi úton. Másfelől, e Licensz megőrzi a szerző, vagy kiadó munkája elismeréséhez fűződő jogát, s egyúttal mentesíti őt a mások által beiktatott módosítások következményei alól.

Jelen Licensz egyfajta „etalonnak” tekinthető, ami nem jelent mást, mint hogy a dokumentumból származtatott munkák maguk is szabad minősítést kell, hogy kapjanak. E dokumentum egyben a GNU Általános Felhasználói Licensz kiegészítőjeként is szolgál, mely egy a szabad szoftverekre vonatkozó etalon licensz.

E Licenszet a szabad szoftverek kézikönyveiben való használatra alkottuk, hiszen a szabad szoftver egyben szabad dokumentációt is igényel: egy szabad programot olyan kézikönyvvel kell ellátni, mely ugyanazon szabadságokat biztosítja, mint maga a program. Jelen Licensz, mindazonáltal, nem korlátozódik pusztán kézikönyvekre; feltételei tetszőleges tárgykörű írott dokumentumra alkalmazhatók, függetlenül attól, hogy az könyvformában valaha megjelent-e. Mindamellett e Licenszet főként olyan munkákhoz ajánljuk, melyek elsődleges célja az útmutatás, vagy a tájékoztatás.

1. ALKALMAZHATÓSÁG ÉS DEFINÍCIÓK

E Licensz minden olyan kézikönyvre, vagy más jellegű munkára vonatkozik, melyen megtalálható a szerzői jogtulajdonos által feltüntetett figyelmeztetés, miszerint a dokumentum terjesztése jelen Licensz feltételei alapján lehetséges. A „Dokumentum” alább bármely ilyen jellegű kézikönyvre, vagy egyéb munkára vonatkozik. A lakosság minden tagja potenciális licensztulajdonosnak tekinthető, és mindegyikük megszólítása egyaránt „ön”.

A Dokumentum „Módosított Változata” bármely olyan munkára vonatkozik, mely tartalmazza a Dokumentumot, vagy annak elemeit akár szó szerint, akár módosításokkal, és/vagy más nyelvre lefordítva.

A „Másodlagos Szakasz” egy egyedi névvel bíró függelék, esetleg a Dokumentum egy megelőző szakasza, mely kizárólag a kiadóknak, vagy az alkotóknak a Dokumentum átfogó tárgyköréhez (vagy kapcsolódó témákhoz) fűződő viszonyáról szól, és nem tartalmaz semmi olyat, ami közvetlenül ezen átfogó témakör alá eshet. (Ha például a Dokumentum részben egy matematika tankönyv, úgy a Másodlagos Szakaszban nincs lehetőség matematikai tárgyú magyarázatokra.) A fenti kapcsolat tárgya lehet a témakörrel, vagy a kapcsolódó témákkal való történelmi viszony, illetve az azokra vonatkozó jogi, kereskedelmi, filozófiai, etikai, vagy politikai felfogás.

A „Nem Változtatható Szakaszok” olyan speciális Másodlagos Szakasznak számítanak, melyek ilyetén való meghatározását az a közlemény tartalmazza, miszerint a Dokumentum jelen Licensz hatálya alatt lett kiadva.

A „Borítószövegek” olyan rövid szövegrészek, melyek Címlap-szövegként, illetve Hátlap-szövegként kerülnek felsorolásra abban a közleményben, miszerint a Dokumentum jelen Licensz hatálya alatt lett kiadva.

A Dokumentum „Átlátszó” példánya olyan géppel-olvasható változatot jelöl, mely a nyilvánosság számára hozzáférhető formátumban kerül terjesztésre, továbbá melynek tartalma szokványos szövegszerkesztő-programokkal, illetve (pixelekből álló képek esetén) szokványos képmegjelenítő-programokkal, vagy (rajzok esetén) általánosan hozzáférhető rajprogramok segítségével azonnal és közvetlenül megtekinthető, vagy módosítható; továbbá olyan formátumban mely alkalmas a szövegszerkesztőkbe való bevitelre, vagy a szövegszerkesztők által kezelt formátumokba való automatikus átalakításra. Egy olyan, egyébként Átlátszó formátumban készült példány, melynek markupja úgy lett kialakítva, hogy megakadályozza, vagy eltántorítsa az olvasókat minden további módosítástól, nem tekinthető Átlátszónak. A nem „Átlátszó” példányok az „Átlátszatlan” megnevezést kapják.

Az Átlátszóság kritériumainak megfelelő formátumok között megtalálható például a markup nélküli egyszerű ASCII, a Texinfo beviteli formátum, a LaTeX beviteli formátum, az SGML vagy az XML egy általánosan hozzáférhető DTD használatával, és a standardnak megfelelő, emberi módosításra tervezett egyszerű HTML. Az Átlátszatlan formátumok közé sorolható a PostScript, a PDF, a szabadalmaztatott és csak fizetős szövegszerkesztőkkel olvasható formátumok, az olyan SGML vagy XML, melyhez a szükséges DTD és/vagy egyéb feldolgozó eszközök nem általánosan hozzáférhetők, és az olyan gépileg-generált HTML formátum, melyet egyes szövegszerkesztők hoznak létre, kizárólag kiviteli célra.

Egy nyomtatott könyv esetében a „Címlap” magát a címlapot, illetve bármely azt kiegészítő további oldalt jelöl, amely a jelen Licenszben definiált címlap-tartalmak közzétételéhez szükséges. Az olyan formátumú munkáknál, melyek nem rendelkeznek effajta címlappal, a „Címlap” a munka címéhez legközelebb eső, ám a szöveg törzsét megelőző szövegrészeket jelöli.

2. SZÓ SZERINTI SOKSZOROSÍTÁS

Önnek lehetősége van a dokumentum kereskedelmi, vagy nem-kereskedelmi jellegű sokszorosítására és terjesztésére, bármely médiumon keresztül, feltéve, hogy jelen Licensz, a szerzői jogi figyelmeztetés, továbbá a Dokumentumot jelen Licensz hatálya

alá rendelő közlemény minden példányban egyaránt megjelenik, és hogy e feltételeken kívül semmi mást nem tesz hozzá a szöveghez. Nem alkothat olyan technikai korlátokat, melyek megakadályozhatják, vagy szabályozhatják az ön által terjesztett példányok elolvasását, vagy sokszorosítását. Mindazonáltal elfogadhat bizonyos összeget a másolatok fejében. Amennyiben az ön által terjesztett példányok száma meghalad egy bizonyos mennyiséget, úgy a 3. szakasz feltételeinek is eleget kell tennie.

A fenti kritériumok alapján kölcsönbe adhat egyes példányokat, de akár nyilvánosan is közzéteheti a szöveget.

3. SOKSZOROSÍTÁS NAGYOBB MENNYISÉGBEN

Amennyiben 100-nál több nyomtatott változatot tesz közzé a Dokumentumból, és annak Licensze feltételül szabja a Borítószövegek meglétét, úgy minden egyes példányt köteles ellátni olyan borítólapokkal, melyeken a következő Borítószövegek tisztán és olvashatóan fel vannak tüntetve: Címlap-szövegek a címlapon, illetve Hátlap-szövegek a hátlapon. Mindkét borítólapra egyértelműen és olvashatóan rá kell vezetnie a kiadó, vagyis jelen esetben az ön nevét. A címlapon a Dokumentum teljes címének jól láthatóan, továbbá minden egyes szónak azonos szedésben kell megjelennie. Ezen felül, belátása szerint, további részleteket is hozzáadhat a borítólapokhoz. Amennyiben az esetleges módosítások kizárólag a borítólapokat érintik, és feltéve, hogy a Dokumentum címe változatlan marad, továbbá a borítólapok megfelelnek minden egyéb követelménynek, úgy a sokszorosítás ettől eltekintve szó szerinti reprodukciónak minősül.

Abban az esetben, ha a borítólapok bármelyikén megkövetelt szövegrészek túl hosszúnak bizonyulnának az olvasható közzétételhez, úgy csak az elsőként felsoroltakat kell feltüntetnie (amennyi józan belátás szerint elfér) a tényleges borítón, a továbbiak pedig átkerülhetnek a következő oldalakra.

Amennyiben 100-nál több Átlátszatlan példányt tesz közzé, vagy terjeszt a Dokumentumból, úgy köteles vagy egy géppel-olvasható Átlátszó példányt mellékelni minden egyes Átlátszatlan példányhoz, vagy leírni minden egyes Átlátszatlan példányban egy a módosítatlan Átlátszó példányt tartalmazó nyilvános hozzáférésű számítógéphálózat elérhetőségét, ahonnan bárki, anonim módon, térítésmentesen letöltheti azt, egy közismert hálózati protokoll használatával. Ha az utóbbi lehetőséget választja, köteles gondoskodni arról, hogy attól a naptól kezdve, amikor az utolsó Átlátszatlan példány is terjesztésre került (akár közvetlenül ön által, akár kiskereskedelmi forgalomban), a fenti helyen közzétett Átlátszó példány még legalább egy évig hozzáférhető legyen a felhasználók számára.

Megkérjük, ámde nem kötelezzük önt arra, hogy minden esetben, amikor nagyobb példányszámú terjesztésbe kezd, már jóval ezt megelőzően lépjen kapcsolatba a Dokumentum szerzőivel, annak érdekében, hogy megkaphassa tőlük a Dokumentum esetleges felújított változatát.

4. MÓDOSÍTÁS

Önnek lehetősége van a Dokumentum Módosított Változatának sokszorosítására és terjesztésére a 2. és 3. szakaszok fenti rendelkezései alapján, feltéve, hogy a Módosított

Változatot kizárólag jelen Licensz feltételeivel összhangban teszi közzé, ahol a Módosított Változat a Dokumentum szerepét tölti be, ezáltal lehetőséget biztosítva annak terjesztésére és módosítására bárkinek, aki csak hozzájut egy példányához. Mindezen felül, a Módosított Változat az alábbi követelményeknek is meg kell, hogy feleljen:

- A Címlapon (és ha van, a borítókon) tüntessen fel egy a Dokumentumétól, illetve bármely korábbi változatétól eltérő címet (melyeknek, ha vannak, a Dokumentum Előzmények szakaszában kell szerepelniük). Egy korábbi változat címét csak akkor használhatja, ha annak szerzője engedélyezte azt.
- A Címlapon szerzőkként sorolja fel a Módosított Változatban elvégzett változtatásokért felelős személyeket, vagy entitásokat, továbbá a Dokumentum fő szerzői közül legkevesebb ötöt (vagy mindet, ha nincsenek öten).
- A Címlapon a Módosított Változat közzétételéért felelős személyt tüntesse fel kiadóként.
- A Dokumentum összes szerzői jogi figyelmeztetését hagyja érintetlenül.
- Saját módosításaira vonatkozóan is tegyen közzé egy szerzői jogi megjegyzést, a többi ilyen jellegű figyelmeztetés mellett.
- Rögtön a szerzői jogi figyelmeztetéseket követően tüntessen fel egy közleményt, az alábbi Függelék mintájára, melyben engedélyezi a Módosított Változat felhasználását jelen Licensz feltételei alapján.
- A fenti közleményben hagyja érintetlenül a Nem Változtatható Szakaszok és a szükséges Borítószövegek jelen Dokumentum licenszében előírt teljes listáját.
- Mellékelje jelen Licensz egy eredeti példányát.
- Az „Előzmények” szakaszt, illetve annak címét szintén hagyja érintetlenül, emellett adjon hozzá egy új elemet, amely minimálisan tartalmazza a Módosított Változat címét, kiadási évét, továbbá az új szerzők, illetve a kiadó nevét, a Címlapon láthatókhoz hasonlóan. Amennyiben a Dokumentum nem tartalmaz semmiféle „Előzmények” elnevezésű szakaszt, úgy hozzon létre egyet, mely tartalmazza a Dokumentum címét, kiadási évét, továbbá a szerzők, illetve a kiadó nevét, a Címlapon láthatókhoz hasonlóan; majd ezt követően adjon hozzá egy új, a Módosított Változatra vonatkozó elemet, a fentiekkel összhangban.
- Ne tegyen változtatásokat a Dokumentumban megadott Átlátszó példány nyilvános hálózati elérhetőségét (ha van ilyen) illetően, vagy hasonlóképp, a Dokumentum alapjául szolgáló korábbi változatok hálózati helyére vonatkozóan. Ezek az „Előzmények” szakaszban is szerepelhetnek. Csak abban az esetben hagyhatja el egyes korábbi változatok hálózati elérhetőségét, ha azok legkevesebb négy évvel a Dokumentum előtt készültek, vagy ha maga az alkotó engedélyezi azt.
- Bármely „Köszönetnyilvánítás”, vagy „Ajánlások” szakasz címét hagyja érintetlenül, továbbá gondoskodjon arról, hogy azok tartalma és hangvétele az egyes hozzájárulókat, és/vagy az ajánlásokat illetően változatlan maradjon.
- A Dokumentum összes Nem Változtatható Szakaszát hagyja érintetlenül, úgy címüket, mint tartalmukat illetően. A szakaszok számozása, vagy bármely azzal egyenértékű jelölés nem tartozik a szakaszcímek közé.

- Töröljön minden „Jóváhagyás” elnevezésű szakaszt. Effajta szakaszok nem képezhetik részét a Módosított Változatnak.

- Ne nevezzen át semmilyen létező szakaszt „Jóváhagyás”-ra, vagy olyasmire, mely címében a Nem Változtatható Szakaszokkal ütközhet.

Ha a Módosított Változat új megelőző szakaszokat tartalmaz, vagy olyan függelékeket, melyek Másodlagos Szakasznak minősülnek, ám nem tartalmaznak a Dokumentumból származó anyagot, abban az esetben, belátása szerint, e szakaszok némelyikét, vagy akár az összeset nem változtathatóként sorolhatja be. Ehhez nem kell mást tennie, mint felsorolni a szóban forgó címeket a Módosított Változat licenszének Nem Változtatható Szakaszok listájában. E címeknek határozottan el kell különülnie minden egyéb szakaszcímtől.

„Jóváhagyás” elnevezésű szakaszt csak akkor adhat a Dokumentumhoz, ha az kizárólag a Módosított Változatra utaló megjegyzéseket tartalmaz – például mások recenzióira vonatkozóan, vagy hogy egy szervezet a szöveget egy standard mérvadó definíciójaként ismerte el.

Címlap-szöveg gyanánt egy legfeljebb öt szóból álló szövegrészt adhat meg, a Hátlap-szöveg esetén pedig 25 szót fűzhet a Módosított Változat Borítószövegeinek végéhez. Bármely entitás csak és kizárólag egy Címlap- és egy Hátlap-szövegrészt adhat (akár közvetítőn keresztül) a Dokumentumhoz. Ha a dokumentum már eleve rendelkezik Borítószöveggel, akár azért, mert azt korábban ön adta hozzá, vagy mert valaki más önön keresztül gondoskodott erről, abban az esetben nincs lehetőség újabb Borítószöveg hozzáadására; a régit mindazonáltal lecserélheti, abban az esetben, ha annak kiadója egyértelműen engedélyezi azt.

A Dokumentum szerzője/i és kiadója/i jelen Licensz alapján nem teszik lehetővé nevük nyilvános felhasználását egyetlen Módosított Változat támogatása, vagy támogatottsága érdekében sem.

5. KOMBINÁLT DOKUMENTUMOK

Önnek lehetősége van a Dokumentum egyéb, e Licensz hatálya alatt kiadott dokumentumokkal való kombinálására a 4. szakasz módosított változatokra vonatkozó rendelkezései alapján, feltéve, hogy a kombináció módosítás nélkül tartalmazza az eredeti dokumentumok összes Nem Változtatható Szakaszát, és hogy azok mind Nem Változtatható Szakaszként kerülnek felsorolásra a kombinált munka licenszében.

A kombinált munkának jelen Licensz mindössze egy példányát kell tartalmaznia, az egymással átfedésben lévő Nem Változtatható Szakaszok pedig kiválthatók egy összegzett példánnyal. Amennyiben több Nem Változtatható Szakasz szerepelne ugyanazon címmel, ám eltérő tartalommal, úgy alakítsa át minden egyes szakasz címét olyan módon, hogy mögéírja zárójelben az eredeti szerző és kiadó nevét (ha ismeri), vagy egy egyedi sorszámot. Ha szükséges, a Nem Változtatható Szakaszok címeivel is végezze el a fenti módosításokat a kombinált munka licenszében.

A kombinált munkában az eredeti dokumentumok összes „Előzmények” elnevezésű szakaszát össze kell olvasztania, miáltal egy összefüggő „Előzmények” szakasz jön létre; hasonlóképp kell eljárnia a „Köszönetnyilvánítás”, illetve az „Ajánlások” szakaszok tekintetében. Ugyanakkor minden „Jóváhagyás” elnevezésű szakaszt törölnie kell.

6. DOKUMENTUMGYŰJTEMÉNYEK

Önnek lehetősége van a Dokumentumból, illetve bármely egyéb, e Licensz hatálya alatt kiadott dokumentumból gyűjteményt létrehozni, és az egyes dokumentumokban található licenszeket egyetlen példánnyal kiváltani, feltéve, hogy a gyűjteményben szereplő összes dokumentum esetén minden más tekintetben követi jelen Licensz feltételeit, azok szó szerinti sokszorosítására vonatkozóan.

Tetszése szerint ki is emelhet egy meghatározott dokumentumot a gyűjteményből, továbbá terjesztheti azt jelen Licensz feltételei alapján, feltéve, hogy a szóban forgó dokumentumhoz mellékeli e Licensz egy példányát, és minden egyéb tekintetben betartja jelen Licensz előírásait a dokumentum szó szerinti sokszorosítására vonatkozóan.

7. ÖSSZEFŰZÉS FÜGGETLEN MUNKÁKKAL

A Dokumentum és annak származékainak különálló, vagy független dokumentumokkal, illetve munkákkal való összefűzése egy közös tárolási, vagy terjesztési egységen, egészében nem tekinthető a Dokumentum Módosított Változatának, feltéve, hogy az összefűzés nem lesz szerzői jogvédett. Az effajta összefűzés eredményeként „összegzés” jön létre, ám jelen Licensz nem érvényes az abban a Dokumentummal együtt szereplő önálló munkákra, hacsak azok nem a Dokumentum származékai.

Amennyiben a 3. szakasz Borítószövegekre vonatkozó rendelkezései alkalmazhatók a Dokumentum e példányaira, és a Dokumentum a teljes összegzésnek kevesebb, mint egynegyedét teszi ki, úgy a Dokumentum Borítószövegeit olyan módon is el lehet helyezni, hogy azok csak magát a Dokumentumot fogják át. Minden más esetben a teljes összegzés borítólapjain kell feltüntetni a fenti szövegeket.

8. FORDÍTÁS

A fordítás egyfajta módosításnak tekinthető, így hát a Dokumentum lefordított példányai a 4. szakasz rendelkezései alapján terjeszthetők. A Nem Változtatható Szakaszok lefordítása külön engedélyt igényel a szerzői jogtulajdonostól, mindazonáltal közzéteheti a lefordított változatokat is abban az esetben, ha az eredeti Nem Változtatható Szakaszokat is belefoglalja a munkába. E Licensz lefordítására ugyanezek a feltételek érvényesek, vagyis a lefordított változat csak akkor jelenhet meg, ha mellette ott van az eredeti, angol nyelvű Licensz szövege is. Amennyiben eltérés mutatkozna az eredeti változat, illetve a fordítás között, úgy a Licensz angol nyelvű eredetije tekintendő mérvadónak.

9. MEGSZŰNÉS

A jelen Licenszben egyértelműen kijelölt kereteken kívül tilos a Dokumentum bárminemű sokszorosítása, módosítása, allicenszelése, vagy terjesztése. Minden ezzel szembeni sokszorosítási, módosítási, allicenszelési, vagy terjesztési kísérlet a jelen Licenszben meghatározott jogok automatikus megszűnését vonja maga után. Azok a fe-

lek, ugyanakkor, akik önön keresztül jutottak másolathoz, vagy jogosultságokhoz, nem veszítik el azokat, amíg maradéktalanul betartják e Licensz előírásait.

10. JELEN LICENSZ JÖVŐBENI JAVÍTÁSAI

Megtörténhet, hogy a Szabad Szoftver Alapítvány időről időre felülvizsgált és/vagy új verziókat bocsát ki a GNU Szabad Dokumentációs Licenszből. E verziók szellemisége hasonló lesz jelen változatéhoz, ám részleteikben eltérhetnek, új problémák, új aggályok felmerülése okán. Vö.: http://www.gnu.org/copyleft/

A Licensz minden változata egyedi verziószámmal van ellátva. Ha a Dokumentum jelen Licensz egy konkrét, számozott verziójára, „vagy bármely újabb verzióra” hivatkozik, úgy önnek a szóban forgó változat, vagy bármely újabb a Szabad Szoftver Alapítvány által (nem vázlatként) publikált verzió feltételeinek követésére lehetősége van. Ha a Dokumentum nem ad meg semmilyen verziószámot, úgy bármely a Szabad Szoftver Alapítvány által valaha (nem vázlatként) publikált változat megfelel.

FÜGGELÉK: A Licensz alkalmazása saját dokumentumaira

Ha e Licenszet egy ön által írt dokumentumban kívánja használni, akkor mellékelje hozzá a Licensz egy példányát, továbbá vezesse rá az alábbi szerzői jogi és licensz közleményeket, rögtön a címlapot követően:

> Copyright © ÉV AZ ÖN NEVE.
>
> E közlemény felhatalmazást ad önnek jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Szabad Szoftver Alapítvány által kiadott GNU Szabad Dokumentációs Licensz 1.1-es, vagy bármely azt követő verziójának feltételei alapján. A Nem Változtatható Szakaszok neve SOROLJA FEL A CÍMÜKET , a Címlap-szövegek neve LISTA, a Hátlap-szövegek neve pedig LISTA. E licensz egy példányát a „GNU Szabad Dokumentációs Licensz” elnevezésű szakasz alatt találja.

Ha a szövegben nincsenek Nem Változtatható Szakaszok, úgy írjon „nincs Nem Változtatható Szakasz”-t, ahelyett, hogy egyenként felsorolná azokat. Ha nincsenek Címlap-szövegek, akkor írjon „nincs Címlap-szöveg”-et, ahelyett, hogy „a Címlap-szövegek neve LISTA”, és hasonlóképp járjon el a Hátlap-szövegek esetében is.

Amennyiben a dokumentum haladó programkód-példákat is tartalmaz, úgy azt javasoljuk, hogy e példákat egy választása szerinti szabad szoftver licensz alatt közölje – mint például a GNU Általános Felhasználói Licensz –, hogy lehetővé tegye a kódok szabad szoftverekben való alkalmazását.

Hátlapszöveg