Héjprogramozás Linux alatt (4. rész)

Sorozatunk előző részeiben megismerkedhettünk a változókkal, a logikai kifejezésekkel, a döntési szerkezetekkel, a parancsbehelyettesítés módszerével és még néhány más, a héjprogramozás szempontjából alapvető lehetőséggel.

Ebben a részben szintén a héj alapszolgáltatásairól, a ciklusok szervezéséről és a függvényekről lesz szó. Minden programban, így a héjprogramokban is gyakran fordulnak elő ismetlődő részek. Ha ezekre a program különböző pontjain van szükség, akkor elkülönített blokkokban, függvényekben helyezhetjük el őket, ha viszont egy helyen kell ugyanazt a műveletsort sokszor végrehajtani, akkor ciklust szervezünk. Erről a két lehetőségről lesz szó ebben a részben.

Új feladat

Tegyük fel, hogy létezik egy több sorból és több oszlopból álló, kizárólag egész értékeket tartalmazó számtáblázatunk, amihez elő szeretnénk állítani a sorok, illetve oszlopok összegét. Írjunk olyan héjprogramot, ami elvégzi az összegzést, ha a táblázatot tartalmazó fájl nevét megadjuk, és a szabályos kimeneten megjeleníti az eredményt. Továbbá egy parancssori kapcsolóval azt is szabályozni lehessen, hogy a sorok vagy az oszlopok összegzését kell-e elvégeznie.
Azonnal érezzük, hogy itt tulajdonképpen két faladat megoldásáról van szó, vagyis a kész program kétféle teendőt lesz képes ellátni. Ennek megfelelően a kódot is célszerű lesz olyan módon kialakítani, hogy a két műveletet megvalósító részek jól elkülönüljenek egymástól. A magas szintű programnyelvekben ezt a moduláris felépítést eljárások vagy függvények segítségével valósítják meg. Természetesen héjprogramokban is lehetőségünk nyílik függvények megadására.

Függvények

A függvény a teljes program elkülönült, jól meghatározott feladatot ellátó része. Lehetnek paraméterei és van kimenete. Amolyan állam az államban. Héjprogramok esetében egy függvény általános alakja a következő:

```
függvénynév()
{
   ... a függvény törzse ...
}
```

Ha az így elkülönített programblokkot futtatni akarjuk, elegendő a függvény nevét beírni a főprogram (vagy egy másik függvény) megfelelő pontján. Lényeges, hogy a gömbölyű zárójeleket héjprogramok esetében ***nem*** kell kiírni. Ezek a függvény megadását és nem a hívását jelzik a parancsértelmező számára.

Ciklusok

Ahhoz, hogy egy táblázat sorain vagy oszlopain végig tudjunk haladni, nyilván ciklusokra lesz szükségünk. Héjprogramokban alapvetően kétféle ciklus létezik. Az egyiknél a más nyelvekben is szokásos módon egy logikai feltételt kell megadnunk, és a ciklus addig fut, ameddig ez a feltétel igaz. Ennek a szerkezetnek az alakja a következő:

```
while logikai_kifejezés
do
   ... ciklusmag ...
done
```

A logikai kifejezés megfogalmazására ugyanazok a szabályok érvényesek, mint amiket az előző részben a `test` parancs kapcsán már bemutattunk.
A másik ciklustípusnál meg kell adnunk azt a sorozatot, aminek az értékeit a ciklusváltozónak fel kell vennie. Általános alakja a következő:

```
for ciklusváltozó in lista
do
   ... ciklusmag ...
done
```

Ez természetesen nem feltétlenül jelenti azt, hogy a listát kézzel kell begépelnünk. Megadhatunk a helyén egy olyan héjváltozót is, ami a megfelelő sorozatot tartalmazza, magát a sorozatot pedig más utasításokkal is előállíthatjuk.

A sorok összegzése

Ennyi bevezető után írjuk meg a program lelkét képező két függvényt. Kezdjük a sorokon belül összegző algoritmus megírásával (lásd az *1. listát*).
A 3. sorban egyszerűen megszámoljuk, hogy hány sorból áll a táblázat.
Látható, hogy mindkét ciklustípusra szükségünk volt.

```
 1: sorosszegzes()
 2: {
 3:    sorszam=`cat $1 | wc -l`
 4:    i=1
 5:    while [ $i -le $sorszam ]
 6:    do
 7:       egysor=`head -$i $1 | tail -1`
 8:       osszeg=0
 9:       for elem in $egysor
10:       do
11:          osszeg=`expr $osszeg + $elem`
12:       done
13:       echo $osszeg
14:       i=`expr $i + 1`
15:    done
16: }
```

```
 1: oszloposszegzes()
 2: {
 3:   sorszam=`cat $1 | wc -l`
 4:   oszlopszam=`head -1 $1 | wc -w`
 5:   i=1
 6:   while [ $i -le $oszlopszam ]
 7:   do
 8:     osszeg=0
 9:     j=1
10:     while [ $j -le $sorszam ]
11:     do
12:       egysor=`head -$j $1 | tail -1`
13:       k=1
14:       for elem in $egysor
15:       do
16:         if [ $k -eq $i ]
17:         then
18:           osszeg=`expr $osszeg + $elem`
19:         fi
20:         k=`expr $k + 1`
21:       done
22:       j=`expr $j + 1`
23:     done
24:     echo $osszeg
25:     i=`expr $i + 1`
26:   done
27: }
```

Az 5. sorban induló `while` ciklus a bemeneti fájl sorain halad végig, és a 7. sorban mindig a megadott sorszámút vágja ki. A `head` parancs egy fájl elejéről jeleníti meg a megadott számú sort, a `tail` pedig ugyanezt teszi, de a fájl végéről visszafelé haladva. A 7. sorban előbb kivágjuk a bemenet első i sorát, majd ezekből csak az utolsót vesszük. Az eredmény egyetlen sor, mégpedig pontosan az i-edik.
A 9. sorban induló `for` ciklus ezen az egyetlen soron (`egysor`) mint listán halad végig, és az `osszeg` nevű változóban gyűjti a részeredményeket. Az összegzést az `expr` parancs segítségével végezzük, ami egy kizárólag egész számokkal működő „számológép”. Azt is érdemes megjegyezni, hogy a műveleti jelet a tulajdonságoktól (argumentum) mindig szóköznek kell elválasztania, ha pedig szorozni akarunk vele, akkor a szorzásjelet `\*` alakban kell megadni, a csillag karakter különleges jelentése miatt.

Az oszlopok összegzése

Az oszlopok összegzése ennél egy kissé bonyolultabb, bár a megvalósítás hasonló az előzőhöz. Itt három ciklusra van szükségünk. Az első (6. sor) a feldolgozott oszlopokat számolja, a második (10. sor) a bemenet sorain halad végig, a harmadik (14. sor) pedig minden sorból csak a pillanatnyi oszlopindexnek ($i) megfelelő sorszámú tagot adja hozzá a részösszeghez (lásd a 2. *listát*).
Valószínűleg az olvasó is érzi, hogy ez nem túl elegáns megoldás, sőt talán már arra is rájött, hogy a feladat lényege az oszlopok címzése. Nincs olyan egyszerű módszerünk, amivel ki tudnánk emelni egy sorozatnak mondjuk a harmadik elemét. (Valójában létezik ilyen módszer, de ez már egy másik történet lesz.)
Figyeljük meg, hogy a ciklusváltozók növelését szintén az `expr` segítségével oldottuk meg.

```
 1: if [ $# -ne 2 ]
 2: then
 3:   echo "A program használata:"
 4:   echo $0 "[-s | -o] fájlnév"
 5:   exit 1
 6: fi
 7:
 8: if [ ! -f $2 ]
 9: then
10:   echo "A megadott fájl nem létezik!"
11:   exit 2
12: fi
13:
14: case $1 in
15: "-s") sorosszegzes $2;;
16: "-o") oszloposszegzes $2;;
17: *) echo "Ismeretlen kapcsoló!" ; exit 3
;;
18: esac*
```

A főprogram

Ezzel a feladatot lényegében meg is oldottuk. Az utolsó lépés a vezérlést ellátó főprogram megírása, és a teljes kód összeállítása. A feladat kiírása szerint a sorok és oszlopok összegzése között egy kapcsolóval lehet váltani. Ez azt jelenti, hogy a programunknak két parancssori kapcsolója lesz. Az első a kapcsoló, a második pedig a feldolgozandó fájl neve (3. *lista*).
A főprogram elején megvizsgáljuk, hogy valóban két kapcsolót kaptunk-e a parancssorban (1. sor), majd a biztonság kedvéért azt is leellenőrizzük, hogy a megadott fájl létezik-e. Hibás kapcsolószám esetén a unixos szokásoknak megfelelően tájékoztató szöveget íratunk ki. Figyeljük meg, hogy a különböző hibákat eltérő visszatérési értékkel jelezzük.
A 14. sorban egy az első parancssori kapcsolóban (`$1`) megadott érték alapján működő többszörös elágazás kezdődik. Esetünkben itt dől el, hogy a két függvény közül melyik fog lefutni. A lehetőségeket karakterláncok formájában kell megadnunk (15. és 16. sor), a hozzájuk tartozó műveletsorok végét pedig kettős pontosvessző jelzi. A * (csillag) karakter a „bármi más” jelölésére szolgál. Esetünkben ez csak ismeretlen kapcsoló lehet, így hibaüzenetet küldünk rá. Magát a `case` szerkezetet az `esac` (case visszafelé) kulcsszó zárja le.
Figyeljük meg, hogy a függvények meghívásánál azok parancssori kapcsolójaként `$2`-t (a feldolgozandó fájl nevét) adjuk meg, maguk a függvények azonban `$1`-ként hivatkoznak erre. Ez azért van így, mert a függvényeknek önálló értékkészletük van, aminek így a számozása sem azonos a főprograméval.
A változókkal kicsit bonyolultabb a helyzet. A függvény belelát a főprogram változóiba, és ugyanez fordítva is igaz (röviden tehát minden változó globális). Ugyanakkor ha a függvény hoz létre egy változót, akkor a főprogram csak az első függvényhívás után tud rá hivatkozni.
A teljes program a fenti három részlet egyszerű összemásolásával előállítható.

Büki András (buki@vuk.chem.elte.hu)
Körülbelül kilenc éve dolgozik Linux operációs rendszerrel. Állandó szerzőtársa Prof. Dr. H. V. Kuksinak, akivel a Duna vagy a Tisza partján szoktak az élet és a tudomány viselt dolgairól töprengeni.