Héjprogramozás Linux alatt – az AWK nyelv (5. rész)

Héjprogramozásról szóló sorozatunk mostani részében az AWK sedégprogrammal és szövegfeldolgozó nyelvvel ismerkedünk meg.

Valószínűleg az olvasónak is feltűnt, hogy az előző részben bemutatott megoldás meglehetősen körülményes volt. A gondot alapvetően az okozta, hogy maga a héj nem rendelkezik olyan szolgáltatással, amivel egyszerűen hozzáférhetnénk egy táblázat elemeihez. Annak idején talán ez volt az egyik olyan felismerés, ami elvezetett az `awk` segédprogram és szövegfeldolgozó nyelv megalkotásához. (A furcsa név a szerzők neveinek kezdőbetűből állt össze: ***Aho, Weinberg, Kernighan***.)
Az `awk` – bár mérete általában nem haladja meg a néhány száz kilobájtot – a korszerű táblázatkezelők gyakorlatilag valamennyi szolgáltatásával rendelkezik. Az egyetlen – ám sokak számára igen lényeges – eltérés az, hogy ennek a programnak nincs grafikus felülete, vezérlésére ehelyett egy saját programozási nyelv szolgál.
Az `awk` nyelve számos ponton erősen hasonlít a C-re. Ennek megfelelően összes függvényét, vezérlési szerkezetét és egyéb szolgáltatását egyetlen cikkben lehetetlen bemutatni. Itt és most csak azokról az alapelvekről és lehetőségekről esik majd szó, amelyek a héjprogramokban a leggyakrabban felbukkannak.

Az awk működésének logikája

Az `awk` bemenete fájlból és a szabványos bemenetről egyaránt érkezhet. A feldolgozást vezérlő programot szintén meg lehet adni külön fájlban (ilyenkor a `-f` kapcsolót kell használnunk), de gyakoribb az a megoldás, amikor ezt is rögtön az `awk` kulcsszó után, egyszeres idézőjelek (aposztróf) közé zárva adjuk meg. Egy az `awk`-val kapcsolatos kódrészlet tehát sematikusan a következőképpen nézhet ki:

```
...bemenet... | awk 'BEGIN \
{ utasítások }
{ főprogram }
END { utasítások }'
```

Az első sor végén látható \ (visszaperjel) sorösszefűzést végez, vagyis azt jelzi a héj számára, hogy csupán az olvashatóság kedvéért írtuk a kódot több sorba. (Ügyeljünk rá, hogy a \ után már semmi nem állhat, még szóköz sem!)
A „főprogramban” található utasításokat az `awk` a bemenet valamennyi során önműködően végrehajtja, vagyis a program eleve tartalmaz egy rejtett (implicit) ciklust. (Emlékezzünk rá, hogy az előző részben a táblázat sorainak végigpásztázásához nekünk magunknak kellett ciklust írnunk.)
A `BEGIN` blokkban szereplő utasítások a sorok feldolgozásának megkezdése előtt hajtódnak végre, az `END` blokk tartalma pedig a feldolgozás után fut le.
És most következzék a lényeg: az `awk` feldolgozás előtt minden sort mezőkre bont. Alapértelmezésként mező az, amit mindkét oldalról legalább egy szóköz vagy tabulátor határol. A mezőkre a programban a `$1`, `$2` stb. szimbólumok segítségével egyenként lehet hivatkozni. A számozás egytől indul, de létezik a `$0` is, ami a teljes sort jelenti. (Figyelem, ezeknek a szimbólumoknak semmi közük a héjprogram parancssori kapcsolóihoz!)
Az `awk` programokban megadhatunk változókat, amelyekre itt a héj „szokásaitól” eltérően minden helyzetben egyszerűen a nevük leírásával lehet hivatkozni. Az `awk` az eddig használt `expr` paranccsal szemben lebegőpontos számokkal is képes műveleteket végezni, és minden olyan különleges függvényt is ismer, amit egy tudományos zsebszámológépen megtalálhatunk. Az eredmények kiíratására a `print` parancsot használhatjuk.

A sorok összegzése

Ennyi bevezető után lássuk, miként valósítható meg az `awk` segítségével egy táblázat sorait összegző függvény:

```
1: sorosszeg()
2: {
3:   cat $1 | awk \
4:   '{ osszeg=0
5:      for(i=1;i<=NF;i++) osszeg+=$i
6:      print osszeg}'
7: }
```

A `cat` parancs utáni `$1` szimbólum most is a függvény (és nem a program) első parancssori értékét jelenti, a 3. sor végén pedig az előbb említett, az olvashatóságot megkönnyítő sorösszefűző karakter látható. A pillanatnyi sor mezőin végig haladó `for` ciklus akár a C-program része is lehetne. A C nyelvet nem ismerők kedvéért talán érdemes megemlíteni, hogy az `i++` szimbólum az `i=i+1` művelet „rövidített” megfelelője, a `+=` műveleti jel (operator) pedig az `osszeg=osszeg+$i` művelet leírását teszi egyszerűbbé. Mindkét megoldás a C nyelvből ered.
A ciklus leállási feltételében szereplő `NF` az `awk` egyik önműködően frissülő belső változója, ami a feldolgozás alatt álló sor mezőinek számát tartalmazza. Gyakran szükséges ezenkívül még az `NR` belső változó, ez a pillanatnyi sor sorszáma (a számozás egytől indul).

Az oszlopok összeadása

A sorok összegzésénél nem volt szükség az `awk` programon kívüli ciklusra, mivel a végigpásztázásuk önműködő. Az oszlopok esetében ilyen automatizmus már nincs, így itt két ciklusra lesz szükségünk. A megoldás azonban még így is rövidebb és áttekinthetőbb lesz, mint az előző részben bemutatott.

```
1: oszloposszeg()
2: {
3:   oszlopszam=`head -1 $1 | wc -w`
4:   i=1
5:   while [ $i -le $oszlopszam ]
6:   do
```

```
 7:      cat $1 | awk -v oszlop=$i \
 8:      'BEGIN { osszeg=0 }
 9:             { osszeg+=$oszlop }
10:       END { print osszeg }'
11:      i=`expr $i + 1`
12:    done
13: }
```

A 3. sorban kiderítjük, hogy hány oszlopból áll a táblázat. Ehhez a `head` segítségével kiemelt első sorban megszámoljuk a szavakat. A következő érdekes momentum a 7. sorban látható. Itt az `awk -v` kapcsolója segítségével kívülről állítjuk be az oszlop nevű `awk` változó értékét. Ez fogja megmutatni, hogy éppen hányadik oszlop összegzésénél tartunk. A szabványos ki- és bemeneten kívül a `-v` kapcsoló szolgáltatja az egyetlen kapcsolattartási lehetőséget a héjprogram és az `awk` program között. Ezzel a módszerrel természetesen több változó tartalmát is beállíthatjuk, a `-v` kapcsolót azonban minden értékadásnál meg kell adni.
Magának az `osszeg` változónak a nullázása most a `BEGIN` blokkba került, hiszen ezt a műveletet csak egyszer akarjuk elvégezni. Hasonló okok miatt került az eredmény kiíratása az `END` blokkba. Maga a főprogram csupán egy összegzést tartalmaz.
A kész héjprogram most is a fenti részek egyszerű összemásolásával állítható elő, a parancssori kapcsolók kezeléséhez pedig ugyanazt az egyszerű `case` szerkezetet használhatjuk, mint a sorozat előző részében.

Szépészet awk-val

Befejezésül bemutatom az `awk` használatának egy másik lehetőségét is. Biztosan az olvasónak is feltűnt már, hogy milyen „rendszerető módon” vannak sorszámozva a sorozatban megjelent kódszövegek. Egy igazi Linux-felhasználónak természetesen eszébe sem jut, hogy ilyesmit kézzel tegyen meg – helyette megírja a következő nyúlfarknyi programot:

```
1: #!/bin/sh
2: cat $1 | awk \
3: '{ if (NR<10)
4:        { print " "NR": " $0 }
5:    else
6:        {print NR": " $0}
7: }'
```

Mint korábban említettem, az `NR` belső változó, amelynek a tartalma az éppen feldolgozott sor sorszáma, vagyis éppen az, amire nekünk szükségünk van.
Ha azt akarjuk, hogy az egy- és kétjegyű sorszámok utáni kettőspontok szépen egymás alatt sorakozzanak, az egyjegyűek elé egy-egy szóközt is be kell illesztenünk. Éppen ezt teszi a fenti döntési szerkezet igaz ága. (Figyeljük meg, hogy `awk`-ban a karakterláncok összefűzéséhez elegendő egyszerűen megszakítás nélkül egymás mellé írni őket.)
Van ebben a megoldásban egy kis tökéletlenség: akkor is beljebb tolja az egyjegyű sorszámokat, ha nincsenek kétjegyűek. Ennek megoldását az olvasóra bízom.

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.