Irodalom

1] *Blundo, J.* – Digital Scanner, http://web.mit.edu/2.972/www/reports/scanner/scanner.html
2] *Miklóssy D.* – Prezentációs oktatási segédanyag kidolgozása a PC perifériák és muködésük bemutatására; Magyar Elektronikus Könyvtár (http://www.mek.iif.hu), PTE-Pollack Mihály Muszaki Foiskolai Kar (http://vili.pmmf.hu/diplom/2001/miklossy/szakdolgozat.htm)
3] *Tyson, J.* – How Scanners Work, Marshall Brain's HowStuffWorks, http://www.howstuffworks.com/scanner.htm
4] *** – Digitális képrögzítés elmélete; MACSBK – Magyar Amatorcsillagászok Baráti Köre, Cikkarchívum, http://macsbk.csillagaszat.hu/cikkek/digicam.htm
5] *** – The PC Technology Guide – Scanners, http://www.pctechguide.com
6] *** – A szkennerek, http://eotvos.isk.tvnet.hu/intranet/computer/hardware/scan.htm

Kaucsár Márton

Rekurzió egyszeruen és érdekesen

"A tanulás legyen teljesen gyakorlatias, teljesen szórakoztató, ..., olyan, hogy általa az iskola valóban a játék helyévé, vagyis az egész élet elojátékává váljon."
(Comenius)

I. rész

Tegyük fel, hogy egy bizonyos engedélyt szeretnél kiváltani a polgármesteri hivataltól. Az elso irodában közlik veled, hogy az engedély megszerzése feltételezi egy másik engedély birtoklását, amelyet egy másik irodában állítanak ki. Amikor belépsz ide ugyanazt a választ kapod, mint az elozo irodában. És ez így folytatódik addig, míg egy olyan engedélyhez nem jutsz, amelyik megszerzése már nem feltételezi egy további engedély birtoklását. Minekutána ezt kiváltottad, folytathatod a félbehagyott kísérleteidet – fordított sorrendben – míg minden szükséges engedélyt meg nem szerzel. Végül az elso irodában fogják a kezedbe adni azt az engedélyt, amiért beléptél a hivatal ajtaján.

Rekurzió a matematikában

Bár a fenti kálváriához hasonlót tapasztalhattál már, mégis a rekurzió fogalmával valószínu matek órán találkoztál eloször, a rekurzív képletek kapcsán. Klasszikus példa erre a faktoriális rekurzív képlete. A matematikusok az elso n ($n>0$) természetes szám szorzatát *n faktoriálisnak* nevezik és $n!$-el jelölik. A $0!$ értéke megegyezés szerint 1.

$$n! = \begin{cases} 1 & \text{ha } n = 0 \\ 1.2. \ldots . (n-1).n & \text{ha } n > 0 \end{cases} \quad (1)$$

Ha a fenti képletben az $1.2. \ldots .(n-1)$ szorzatot $(n-1)!$-al helyettesítjük, akkor eljutunk a faktoriális rekurzív képletéhez.

$$n! = \begin{cases} 1 & \text{ha } n = 0 \\ (n-1)!n & \text{ha } n > 0 \end{cases} \quad (2)$$

Ezt a képletet azért nevezik rekurzívnak, mert az $n!$ kiszámítását ($n>0$ estén) visszavezeti $(n-1)!$ kiszámítására, egy *hasonló, de egyszerubb* (eggyel kevesebb szorzást feltételez) feladatra. Természetesen $(n-1)!$ is hasonló módon visszavezetheto $(n-2)!$ -ra és így tovább, míg eljutunk $0!$ –ig.

$n! = (n-1)!n = (n-2)!(n-1).n = \ldots = 0!.1.2. \ldots (n-1).n = 1.1.2. \ldots (n-1).n = 1.2. \ldots .(n-1).n$

Rekurzió az informatikában

Tegyük fel, hogy két függvény, f_1 és f_2 célul tuzi ki, hogy kiszámítja a paraméterként kapott *n* faktoriálisának értékét. f_1 az (1) képlet alapján lát hozzá a feladathoz és a következoképpen oldja meg:

Pascal

```
function f1(n: integer): integer;
var i, p: integer;
begin
        if n = 0 then f1 := 1
                else
                begin
                        p := 1;
                        for i := 1 to n do
                                p := p * i;
                        f1 := p;
                end;
end;
```

C/C++

```
int f1 (int n)
{
        if (n == 0) return 1;
        else
        {
                int p = 1;
                for(int i = 1; i <= n; i++)
                        p *= i;
                return p;
        }
}
```

Megfigyelhetted, hogy f_1 felvállalja az egész feladatot, ami azt jelenti, hogy elvégzi az *n!* kiszámításához szükséges mind az *n* szorzást. Az *n>0* estben ezt egy for ciklussal oldja meg. Ezt a megközelítést – amikor bizonyos utasítások ismétlését ciklus segítségével valósítjuk meg – *iteratív* módszernek nevezzük.

f_2 viszont ennél sokkal kényelmesebb. A faktoriális rekurzív képletébol véve az ötletet, a következoképpen gondolkodik:

1. Ha *n = 0*, akkor én is készségesen megoldom a feladatot – elvégre ez csak annyit jelent, hogy bemondom a *0!* értékét, az *1*-et.

2. Ha viszont *n > 0*, nem vállalom fel a teljes feladatot, túl fárasztó lenne nekem *n* szorzást elvégezni. A következoképpen fogok eljárni:

2.1. Átruházom „valaki”-re az elso *n-1* szorzás elvégzésének feladatát – ami valójában az *(n-1)!* értékét jelenti – és nyújtok neki egy „tálcát” amire rá tegye az eredményt.

2.2. A tálcán kapott eredményt még megszorzom *n*-el, és máris büszkélkedhetek az *n!* értékével.

Ez mind jó és szép, de ki lesz ez a „valaki”, és mi lesz a „tálca”?

És most jön a hideg zuhany f_2 úrfinak. Mivel ez a „valaki” az *(n-1)!* kiszámításánál hasonlóképpen fog kelleni eljárjon mint o, ezért a „valaki” o maga lesz.

? Hogy-hogy? Azt jelentse ez, hogy a feladatom oroszlánrészét átruházom magamra, és annak oroszlánrészét megintcsak magamra, és így tovább míg eljutok a *0!*-ig?

? Igen, pontosan errol van szó!

? De hát ez lehetetlen, hiszen azt feltételezné, hogy klónozzak még *n* példányt magamból.

? Pedig amint látni fogod, valami hasonlóról van szó.

? És a tálca?

? Mivel „mindenik f_2” a *saját* tálcáját kell nyújtsa a következonek, ezért a tálca szerepét egy – a függvény típusával azonos típusú – *lokális (saját)* változó fogja betölteni.

Íme az f_2 függvény Pascal és C/C++ változatban:

Pascal

```
function f2(n: integer): integer;
var talca: integer;
begin
        if n = 0 then f2 := 1
                else
                begin
                        talca := f2(n-1);
                        f2 := talca*n;
                end;
end;
```

C++

```
int f2 (int n)
{
        int talca;
        if (n == 0) return 1;
        else
        {
                talca = f2(n-1);
                return talca*n;
        }
}
```

Ez tényleg elegáns, de hogyan muködik? Errol szól majd a következo rész!

Kátai Zoltán, Marosvásárhely