Nem véletlenül használtuk már több ízben „az Univerzum mai állapota” kifejezést, ugyanis az Univerzum mai állapotát vizsgálva szinte elkerülhetetlenül arra a következtetésre jutunk, hogy korábbi idõszakban a Világegyetem állapota lényegesen különbözött a jelenlegitõl. Egyes elképzelések szerint elszigetelt „*zárványokban*” a mai napig fennmaradtak ilyen õsi viszonyok. Az ilyen hipotetikus tartományok (legismertebbek közülük az ún. *kozmikus szálak*) viszonylag kis térfogatúak, de igen nagy tömegûek lennének; belsejükben pedig az anyag a fentebb felsoroltaktól teljesen eltérõ formában létezhet. Nagyobb számban való elõfordulásuk az átlagos energiasûrûséget ugyan nem növelné jelentõsen, de komoly hatást gyakorolhat a Világegyetem fejlõdésére. Létezésükre azonban jelenleg bizonyíték nincsen.

Szenkovits Ferenc

Rekurzió egyszerûen és érdekesen

II. rész

Az elsõ részben megírtuk az n! értékét kiszámító f2 rekurzív függvényt. Emlékszel még rá?

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

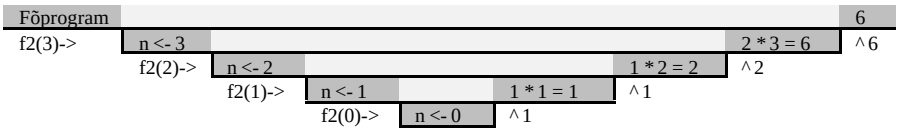
Pascal

```
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;
        }
}
```

Tegyük fel, hogy a *3!* értékét szeretnénk kiíratni a képernyõre az f_2 függvény segítségével. Hogyan bonyolódik le az $f_2(3)$ függvényhívás? Az egyszerûség kedvéért csak a Pascal változatot fogom nyomon követni, de az elv azonos C-ben is. Az alábbi ábra – az úgynevezett lépcsõ-módszert alkalmazva – grafikusan ábrázolja mindazt, ami egy rekurzív függvény hívásakor a háttérben történik. A „rekurzió lépésrõl-lépésre” bekeretezett rész pedig, mintegy kézen fogva vezet végig a „rekurzió útján”, és egyben magyarázattal is szolgál az ábra megértéséhez.



Az f_2 függvény „útja” az ábrán:

- Indul a „tetõrõl” (fõprogram).
- Lemegy a „lépcsõkön” a „földszintre”(banális eset), hogy megtalálja a *0!* értékét.
- Visszafelé jövet minden szinten kiszámítja az „emeletnek” megfelelõ faktoriális értéket (az alatta lévõ szintrõl hozott értéket megszorozza az „emelet szám ával”).
- Visszaérkezve a tetõre a kezében van az *n!* értéke.

Következtetések az ábra segítségével:

1. Az *n!* kiszámításához *f2*-t, *n+1*-szer hívjuk meg és hajtjuk végre.
2. A függvényhívások fordított sorrendben fejezõdnek be mint ahogy elkezdõdtek.
3. Ha *t*-vel jelöljük a függvény utasításainak végrehajtásához szükséges idõt *n > 0* esetén, és *t0*-val *n=0* estén, akkor az *f2(i)* függvényhívás *(0<=i<=n)* végrehajtása *i*t+t0* ideig tart, amibõl *i>0* esetén *(i-1)*t+t0* idõre fel van függesztve.
4. Mivel mindenik függvényhívásnak meg kell legyen a saját *n*-je és *talca*-ja, ezért *n* föltétlenül érték szerint átadott paraméter, *talca* pedig lokális változó kell legyen.

Nem nehéz belátni, hogy a *talca* változó használata nem föltétlenül szükséges, hiszen az *if* utasítás *else* ága nézhetne egyszerûen így ki:

Pascal	***C++***
else f2:=f2(n-1)*n;	else return f2(n-1)*n;

Mégis, azért vezettem be a *talca* változót, hogy az általános eset (*n>0*) kezelésénél – az *else* ágon – világosan különválasszam az *átruházott oroszlánrész* megoldását, amely a rekurzív hívás által történik, a *saját résztõl*, amikor is a tálcán kapott értékbõl felépítem az eredeti feladat megoldását.

Befejezésül állítsuk szembe a két függvény stratégiáját. f_1 úgy építette fel a megoldást, hogy az *egyszerûtõl haladt a bonyolult felé*. Vett egy *p* változót, amelybe kezdetben *1*-et tett, a *0!* értékét. Ezután pedig a *0!* értékébõl kiindulva a *p* változóban sorra elõállította *1*-tõl *n*-ig a természetes számok faktoriálisait: elõször *1!*-t azután *2!*-t és így tovább míg eljutott *n!*-ig.

f_2 pont fordítva látott hozzá a feladathoz: egybõl nekiszökött az *n!* kiszámításának. A rekurzió mechanizmusa által elõször *lebontotta a feladatot a bonyolulttól haladva az egyszerû felé*, majd pedig *felépítette a megoldást az egyszerûtõl haladva a bonyolult felé.* ... és beigazolódik a közmondás, miszerint a rest kétszer fárad.

Akkor hát melyik a jobb stratégia, az f_1-é vagy az f_2-é, az iteratív vagy a rekurzív? Tény, hogy az iteratív módszer gyorsabb és kevésbé memória igényes. De hát akkor szól-e valami is a rekurzió oldalán? Kétségtelenül! A rekurzív megközelítés egyszerû, elegáns és ezért könnyen programozható. Amint magad is tapasztalni fogod, vannak feladatok amelyeknek iteratív megoldása rendkívül bonyolult, rekurzívan viszont egy néhány soros programmal megoldhatók.

Ezek után, ha majd legközelebb belépsz valamely hivatal ajtaján, mit fogsz mondani? Azt, hogy kezdõdik a kálvária, vagy, hogy kezdõdik a rekurzió?

Rekurzió lépésrõl-lépésre

Megszakad a fõprogram végrehajtása és válaszként az *f2*(3) hívásra elkezdõdik a *f2* függvény végrehajtása:

1/1 megszületik az *f2* függvény *n* nevû formális paramétere a Stack-en;

– az *n* formális paraméter megkapja a függvény végrehajtását kiváltó hívás aktuális paraméterét, a 3-ast;

1/2 megszületik a *talca* nevû lokális változó a Stack-en;

az utasításrész végrehajtása:

1/3 mivel 3<>0, az if else ágán folytatódik a függvény végrehajtása;

1/4 a *talca:=f2*(2) utasítást megint nem tudjuk végrehajtani – ez alkalommal az *f2*(2) értek hiányában. Ezért *felfüggesztõdik az f2 függvény végrehajtása* is – ebben a pontban – és az *f2*(2) hívásra válaszolva *elkezdõdik az f2 függvény egy újbóli végrehajtása, átjárása, a második:*

2/1 megszületik a második átjárás saját *n* nevû formális paramétere a Stack-en;

– ez az *n* megkapja a jelen végrehajtást kiváltó hívás aktuális paraméterének az értékét, a 2-est;

2/2 megszületik a második végrehajtás saját *talca* nevû változója;

a második átjárás utasításrészének végrehajtása:

2/3 mivel 2<>0 az if else ágán folytatódik a függvény második átjárása is;

2/4 a *talca:*=f2(1) utasítást megint nem tudjuk végrehajtani – ez alkalommal az *f2*(1) érték hiányában. Ezért *felfüggesztõdik az f2 függvény második átjárása* is – ebben a pontban – és válaszként az *f2*(1) hívásra, *elkezdõdik az f2 függvény egy további átjárása, immár a harmadik:*

3/1 megszületik a harmadik átjárás saját *n* nevû formális paramétere a Stack-en;

- ez az *n* megkapja a jelen végrehajtást kiváltó hívás aktuális paraméterének az értékét, a 1-est;

3/2 megszületik a harmadik végrehajtás saját *talca* nevû változója;

a harmadik átjárás utasításrészének végrehajtása:

3/3 mivel 1<>0 az if else ágán folytatódik a függvény harmadik átjárása is;

3/4 a *talca:*=f2(0) utasítást megint csak nem tudjuk végrehajtani – ez alkalommal az *f2*(0) értek hiányában. Ezért *felfüggesztõdik az f2 függvény harmadik átjárása* is – ebben a pontban – és válaszként az *f2*(0) hívásra, *elkezdõdik az f2 függvény egy további átjárása, immár a negyedik:*

4/1 megszületik a negyedik átjárás saját *n* nevû formális paramétere a Stack-en;

- ez az *n* megkapja a jelen végrehajtást kiváltó hívás aktuális paraméterének az értékét, a 0-st;

4/2 megszületik a negyedik végrehajtás saját *talca* nevû változója;

4/3 mivel ez esetben 0 = 0 az if then ágán fog folytatódni a függvény ezen negyedik végrehajtása, a banális feladat megoldásával;

4/4 a függvény neve megkapja a visszatérítendõ értéket, a banális feladat eredményét, az 1-est. *Bingo, megvan az 0! értéke!;*

befejezõdik a függvény negyedik átjárása:

- eltûnik a Stack-rõl a negyedik átjárás *talca* változója;
- eltûnik a Stack-rõl a negyedik átjárás *n* nevû formális paramétere;

3/5 *folytatódik az f2 függvény harmadik átjárása* abban a pontban, ahol annak idején felfüggesztõdött. A *talca* nevû változóba bekerül a most már rendelkezésre álló *f2*(0) érték, ami nem más, mint a negyedik átjárás által visszatérített eredmény, a 0! értéke, vagyis az 1-es;

3/6 a függvény neve megkapja a tálca*n szorzat értéket. Mivel ezen harmadik átjárásnak az n-je 1, így az általa visszatérített érték 1*1=1 lesz. *Ez nem más, mint az 1! értéke.*

befejezõdik a függvény harmadik átjárása:

eltûnik a Stack-rõl a harmadik átjárás *talca* változója;

eltûnik a Stack-rõl a harmadik átjárás *n* nevû formális paramétere;

2/5 *folytatódik a f2 függvény második átjárása* abban a pontban, ahol annak idején felfüggesztõdött. A *talca* nevû változóba belekerül a most már rendelkezésre álló *f2*(1) érték, ami nem más, mint a harmadik átjárás által visszatérített eredmény, az 1! értéke, vagyis az 1-es;

2/6 a függvény neve megkapja a tálca*n szorzat értéket. Mivel ezen második átjárásnak az n-je 2, így az általa visszatérített érték 1*2=2 lesz. *Ez nem más, mint a 2! értéke.*

befejezõdik a függvény második átjárása:

- eltûnik a Stack-rõl a második átjárás *talca* változója;
- eltûnik a Stack-rõl a második átjárás *n* nevû formális paramétere,

1/5 *folytatódik az f2 függvény elsõ átjárása* abban a pontban, ahol annak idején felfüggesztõdött. A *talca* nevû változóba belekerül a most már rendelkezésre álló *f2*(2) érték, ami nem más, mint a második átjárás által visszatérített eredmény, a 2! értéke, vagyis a 2-es;

1/6 a függvény neve megkapja a tálca*n szorzat értéket. Mivel ezen elsõ átjárásnak az n-je 3, így az általa visszatérített érték 2*3=6 lesz. *Ez nem más, mint a 3! értéke.*

befejezõdik a függvény elsõ átjárása:

- eltûnik a Stack-rõl a elsõ átjárás *talca* változója;
- eltûnik a Stack-rõl a elsõ átjárás *n* nevû formális paramétere;

Folytatódik a fõprogram abban a pontban, ahol annak idején felfüggesztõdött. ugyanis most már rendelkezésre áll az *f2*(3) érték, ami nem más, mint a függvény elsõ átjárása által visszatérített eredmény, a 3! értéke, vagyis az 6-os.

Kátai Zoltán

Optikai anyagvizsgálati módszerek

I. rész: Történeti bevezetõ

Az emberi civilizáció fejlõdését a tudattal rendelkezõ embernek a környezetében ható jelenségek megfigyelése, magyarázata, s hasznára való alkalmazása biztosította. Az ember számára legelsõ tapasztalható kölcsönhatások a környezeti hõ és fényhatások voltak. Az égitestek fényének követése alapozta meg a csillagászattan fejlõdését, melynek kezdetei több ezer évre vezethetõk vissza.

Az ismeretek fejlõdését attól az idõtõl követhetjük, amikor az ember megtanult kommunikálni a jövõ számára. Jeleket használt maradandó anyagokon (sziklafal, kõlap, agyaglap, bõr, papirusz). Ezért tudjuk, hogy a görög gondolkodók már csillagászati megfigyeléseik alapján Naprendszer-modellt állítottak fel, bizonyították a Föld gömbalakú voltát, s a valós értéknek jó megközelítésével kiszámították a kerületét. Ismerték a tükröt, amivel össze lehetett gyûjteni a fénysugarakat. Így sikerült Arkhimédésznek felgyújtani tükrökkel irányított napsugarakkal az ellenséges hajókat. A római birodalom kiterjedése nem kedvezett a természettudományok fejlõdésének, de mivel a görög eszmék már az egész világon elterjedtek, nem szenvedtek végzetes törést, csak viszonylagos lassulást. Az iszlám világ terjeszkedése, az arabok „szent háborúja“ a Keleti-római és Perzsa birodalom ellen új lendületet adott a természettudományok fejlõdésének. Egyetemeket, tudományos társaságokat alapítottak, csillagvizsgálókat építettek. Nagy haladást tettek a fénytan terén is. Lencséket kezdtek használni, kezdetben látás javítására, majd nagyításra. A FIRKA ez évi (12.évf.) 1. számában már röviden írtunk arról, hogy milyen fejlõdésen mentek át a természettudományok az emberi érdeklõdés hajtóereje eredményeként. Itt említettük meg egyik leghíresebb arab természettudóst, Al. Hasent, aki a mai geometriai fénytani ismereteinknek megfelelõ kísérleti tényekkel már tisztában volt. Azok tudományos magyarázatát, matematikai megfogalmazását viszont csak félezred év múlva sikerült megadni, miután Roger Becom (1561–1626) távollátóját elkészítette és Tyco Brahe (1546–1601), J. Kepler (1571–1630), Galilei (1564–1642) megalkották az új, heliocentrikus világképet az addig uralkodó geocentrikus nézetekkel szemben és míg P. Fermat (16011–1665), G. W. Leibniz (1646–1716), Isaac Newton (1642–1727) ki dolgozta az új természettudományokra alkalmas matematikát, a differenciál és integrálszámítást.