2. gyakorlat

For ciklus, függvények létrehozása, használata és parancsargumentumok beolvasása

A rejtélyes Binomusz völgyben talált kövek feliratait egy kutató megfejtette, azok egy-egy rejtekhelyre mutatnak a következő módon: A köveken látható a,b,n,k számokból ki kell számolni az (a+b)^n hatványra vonatkozó binomiális tétel tagjaiból a k-dikat és (k+1)-ediket (0-tól sorszámozva), és ezek lesznek a szélességi és hosszúsági koordináták.

(Segítségül a k-dik tag $\binom{n}{k} a^{n-k} b^k$.)

A legközpontibb helyen talált, feltehetőleg a legnagyobb kincsre mutató kő felirata ez:

1.345951 0.540425 9 4

Írjunk programot, ami beolvassa e számokat és kiírja a koordinátákat!

For ciklus

In [ ]:
```c
// Először a faktoriális számítást írjuk meg.
// Érdemes új projektet kezdeni pl. "binom" néven

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n=5;
    int f=1;
    int i;
    for (i = 2; i <= n; i++)
        f=f*i;               // Ez is rövidíthető így: f*=i;
    printf("%d\n",f);
    return 0;
}
```

Saját függvények definiálása, használata

In [ ]:
```c
/* Következő lépésként a binomiális együtthatót akarjuk kiszámolni.
Ehhez 3x kell a faktoriális, ne kelljen annyiszor leírni,
ezért függvénybe rakjuk ki a faktoriális kiszámolását: */

#include <stdio.h>
#include <stdlib.h>

int factorial(int n) {
    int f=1;
    int i;
    for (i = 2; i <= n; i++)
        f*=i;
    return f;
}
```

```c
int main()
{
    int n=9;
    int k=4;

    int binom=factorial(n)/factorial(k)/factorial(n-k);
    printf("%d\n",binom);

    return 0;
}

// Ha nem tévesztettünk el semmit, akkor 126-ot kapunk eredményül.
```

In [ ]:

```c
// Mivel a feladatban kétszer kell a binomiális együttható,
// annak kiszámolását is függvénybe tesszük.
// Másrészt meg így könyebb lesz másik programba áttenni.
// Ugyanakkor áttekinthetőbbé is teszi a programot.

// A megváltozott rész:

int binom(int n, int k) {
    return factorial(n)/factorial(k)/factorial(n-k);
}

int main()
{
    int n=9;
    int k=4;

    printf("%d\n",binom(n,k));

    return 0;
}
```

Újabb függvénykönyvtár használata, pl. a math könyvtáré

In [ ]:

```c
// Mostmár beírhatjuk a teljes képleteket.

/* A hatványozást a pow függvénnyel tudjuk elvégezni
de ehhez kell a math csomag,
ezért a program elején behívjuk annak headerjét
(ami a csomagban lévő függvények deklarációit, szignatúráit tartalmazza): */

#include <math.h>


// a main részt pedig értelemszerűen átírjuk:

int main()
{
    double a=1.345951;
    double b=0.540425;
    int n=9;
    int k=4;

    printf("%f N  %f E\n", binom(n,k)*pow(a,n-k)*pow(b,k), binom(n,k+1)*pow(a,n-k-1)

    return 0;
}
```

Ezek után a legtöbb rendszer esetén a codeblocksban már futtatható is a program. Egyes operációs rendszerekben még kaphatunk olyan hibaüzenetet, hogy a pow függvény nincs definiálva. Ez azért van, mert a függvény deklarációk behívása az include utasítással természetesen nem elég, hanem a tényleges függvénykönyvtárat még csatolni kell a programhoz a fordításkor. Ezt egyes rendszerekben a codeblocks automatikusan megteszi, míg másokban nem. Utóbbi esetben a következőképpen tudjuk ezt megtenni:

- Click on "Project"
- Click "Build options.."
- Click the "Linker settings" tab
- Click the "Add" button under the "Link libraries"
- Type in "libm.a" and click "OK"
- Click "OK" again (this time on the "Project build options" window)

Ezután, újabb fordítás után futtatható a program.

Parancssori futtatás a math könyvtárral

Hasonló okból a parancssori fordítás esetén is be kell hívni a math csomagot:

```
gcc -Wall binom.c -o binom -lm
```

ahol a -lm opciónak mindenképp az utolsónak kell lennie!

Talán érdemes megjegyezni, hogy a math és a libm (library math rövidítése) is a matematikai programcsomag jelölése (és a .a arra utal, hogy az archive tool-lal készült könyvtár). A -lm opcióból pedig az l a linkelés jele, m a matematikai csomagé.

Érdemes tudni, hogy a pow függvény nemegész kitevővel is működik, ezért kis egész kitevő esetén lassabb (esetleg pontatlanabb is), mintha szorzással vagy osztással számolnánk ki, pl. x*x*x, 1/x/x.

Az eredményül kapott koordináták sora egy az egyben bemásolható a google map-be, és az rámutat a rejtekhelyre. Ne lepődjünk meg, ha az eredmény egy egyetemre mutat, mert a tudás az egyik legnagyobb kincs!

Parancsargumentumok beolvasása

In [ ]:

```
/* Mivel az újabb kövek megfejtéséhez nem akarjuk mindig a programot átírni,
az a,b,c,d paramétereket jobb beolvasni.
Elsőként a parancsargumentumok beolvasásával próbáljuk megoldani: */

int main(int argc, char* argv[])
{
    /* double a=1.345951;
    double b=0.540425;
    int n=9;
    int k=4; */
    if (argc<5) {
        printf("4 arguments needed\n");
        exit(-1);
    }
    double a=atof(argv[1]);
    double b=atof(argv[2]);
    int n=atoi(argv[3]);
    int k=atoi(argv[4]);
...
```

Csak érvényes indexeket használhatunk

Az elméleti ismertetésben már szerepelt, hogy az argv-t úgy helyes használni, ha előbb leellenőriztük az argc értékét, mint a fenti példában. Egyébként a hibás indexelés a program leállásához vezethet. Ráadásul előfordul, hogy egy tömbnél egy nem létező elemre mutató index egyik gépen segmentation fault hibára vezet, másikon nem, illetve a program más részeinek módosításával is változhat a helyzet (mivel a memória kiosztása változik).

Ezért a beadandó feladatoknál ezt az ellenőrzést elvárjuk, ennek hiányáért pontlevonás jár.

Szorgalmi Házi Feladat

- Fibonacci számok előállítása
- pitagoraszi számhármasok keresése a gyakorlat weboldalon megjelenő útmutatás szerint,

ld. bővebben a weboldali "Gyakorló feladatok"-ban.