# 2. gyakorlat

## For ciklus, függvények létrehozása és használata

Ezen a kutatáson keresztül mutatjuk be a programozás következő elemeit:

A Binomusz völgyben rejtélyes feliratos köveket találtak. A feliratokat 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 $\binom{n}{k} a^{n-k} b^k$ . )

Előkerült egy minden eddiginél nagyobb kő, feltehetően ez mutatja a legnagyobb kincs helyét. Legyünk az elsők, akik dekódolják!

Ez a felirata:

1.345951 0.540425 9 4

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

## Ciklusok - emlékeztető

In [ ]:

```
// for ciklus
//   Pl. az 1--10 számok négyzeteinek kiírása
//   írjuk az eddigiek után a return elé, hogy azok mintaként megmaradjanak

#include <iostream>
using namespace std;

int main() {
    cout << endl;    // üres sort írunk ki, hogy áttekinthetőbb legyen
    int n=10;
    for (i = 1; i <= n; ++i)
    {
        cout << "i " << i << "  i^2"<< i*i << endl;;
    }

    return 0;
}

// A pythonnal szemben nem adhatunk meg tetszőleges listát,
// hogy a ciklusváltozó azon menjen végig. (Persze lehet az i-vel indexelni egy tömb
// Viszon a for ciklus sorában a 3 mező tartalmazhat bonyolultabb utasításokat.
// Pl. ha nem az i értékre akarunk feltételt szabni,
// hanem a négyzetszámokat kérjük 150-ig:

    int n=150;
    for (i = 1; i*i <= n; ++i)
    {
        cout << "i " << i << "  i^2"<< i*i << endl;
    }

// Tehát a leállási feltétel lehet bonyolultabb, mint a python esetében.
```

```
// Másrészt a léptetés is lehet bonyolultabb.
// Pl. ha egy szám hatványait akarjuk összegezni:

    double x,y,s;
    s=0;    // ENÉLKÜL NEM FIGYELMEZTET, HOGY NINCS INICIALIZÁLVA !!!!
    x=1/4.;
    for (y = 1; y > 1e-16; y*=x)    // másképp: y=y*x
    {
        s+=y;
        cout << "y " << y << endl;
    }
    cout << s << endl;
    cout << 1/(1-x) << endl;

// Formailag is a pythonhoz hasonlíthatjuk:
// A sorok betolása (indent) nem kötelező, de javasolt, mert ettől áttekinthető a pr
// Ehelyett {} fogja össze, ha több utasításos blokkot akarunk futtatni.
// Ez érvényes a függvényekre (ld. main()), if, for, while ciklus.
// (Egyetlen utasítást tartalmazó if és for utasítás esetén egyébként általában a {}
//    de áttekinthetőség kedvéért meggondolandó meghagyni)

// while ciklus:
//   pl. azt szeretnénk kiszámolni, hogy hány lépésben jutunk el
//   i=1-től 50-ig, ha mindig annyival növeljük i-t,
//   amennyi a négyzetgyökének az egész része

// matematikai függvények (mint most az sqrt) használata esetén
// be kell toldanunk egy újabb csomagbetöltést, pl. a 2. sorba:

#include <cmath>

// A főprogramrészt pedig így folytatjuk:

    cout << endl;
    i=1;            // kezdő pozíció
    k=0;            // megtett lépések száma
    while (i<50)
    {
        i+=sqrt(i);
        k++;
        cout << "i " << i << "  k"<< k << endl;
    }
```

In [ ]:

```
// A Binomusz-felirat megfejtéséhez először a faktoriális számítást írjuk meg.
// Érdemes új projektet kezdeni pl. "binom" néven

#include <iostream>
using namespace std;

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;
    cout << f << endl;

    return 0;
}

// Használat előtt gondoljuk át, jó lesz-e n=0 ill. 1 esetén is!
// n=5-re pedig 120-at kell kapnunk.
```

## Saját függvények definiálása, használata

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

#include <iostream>
using namespace std;

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

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

    int binom=factorial(n)/factorial(k)/factorial(n-k);
    cout << binom << endl;

    return 0;
}

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

In [ ]:
```
// Mivel a feladatban kétszer kell a binomiális együttható,
// annak kiszámolását is függvénybe tesszük.
// Így könyebb is 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;

    cout << binom(n,k) << endl;

    return 0;
}
```

## Újabb függvénykönyvtár használata, pl. a cmath könyvtáré

In [ ]:
```
// Mostmár beírhatjuk a teljes képleteket.

/* A hatványozást a pow függvénnyel tudjuk elvégezni
de ehhez kell a cmath csomag,
ezért a program elején behívjuk annak headerjét
```

```
(ami a csomagban lévő függvények deklarációit, szignatúráit tartalmazza): */

#include <cmath>


// a main részt pedig értelemszerűen átírjuk:

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

    double y = binom(n,k)*pow(a,n-k)*pow(b,k);
    double x = binom(n,k+1)*pow(a,n-k-1)*pow(b,k+1);
    cout << y << " N  " << x << " E  " << endl;

    return 0;
}
```

## Tudnivalók még

Parancssori fordítás, futtatás

```
gcc -Wall main.cpp -o main
./main
```

Érdemes tudni, hogy a pow függvény nemegész kitevővel is működik, ezért kis egész kitevő esetén lassabb (esetleg picit 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!

# Szorgalmi Házi Feladat

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

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