



Eötvös Loránd Tudományegyetem  
Informatikai Kar

## **Alkalmazott modul: Programozás**

---

### **2. fejezet**

### **C++ alapismeretek**

---

**Giachetta Roberto**

A jegyzet az ELTE Informatikai Karának 2015. évi  
Jegyzetpályázatának támogatásával készült

# C++ alapismeretek

## Történet

---

- Wikipédia: a C++ általános célú, magas szintű programozási nyelv, mely támogatja az imperatív, az objektum-orientált, valamint a sablonprogramozást
- Első változata 1979-ben készült (Bjarne Stroustrup) a C programozási nyelvből, eredetileg *C with Objects* névvel
  - célja: objektumorientált programozási lehetőségekkel való kiegészítése a nyelvnek
  - jelenleg a 2014-es szabványt használjuk (C++14)
- Többek szerint közepes szintű nyelvnek tekinthető, mert alacsonyabb szinten használatos utasítások (bit szintű műveletek, direkt memóriaműveletek,...) is használhatóak

# C++ alapismeretek

## Történet

---

- Az alap nyelv csak konzol-alkalmazások készítésére szolgál
  - sok kiegészítő található hozzá, amelyek segítségével sok megoldást implementálni lehet (pl. grafikus környezet)
- A világ egyik leggyakrabban használt programozási nyelve:
  - nyílt forrású, bárki által felhasználható, bővíthető
  - önmagában minden lehetséges nyelvi eszközt megad, amelyre a programozás során szükségünk lehet
  - gyors, hatékony programokat készíthetünk benne
  - sok fordító, fejlesztőkörnyezet támogatja
  - több nyelv alapjául szolgált: JAVA, C#, PHP, ...

# C++ alapismeretek

## A „Hello World” program

---

*Feladat:* Írjuk ki a Hello World! feliratot a képernyőre.

*Megoldás:*

```
// fejrész:  
#include <iostream> // további fájlok beolvasása  
using namespace std; // használni kívánt névtér  
  
// törzsrész:  
int main() // főprogram deklaráció  
{  
    cout << "Hello, World!" << endl; // kiírás  
    return 0; // hibakód  
}  
// főprogram vége
```

- A program elején található a *fejrész*, ami tartalmazza:
  - a program működéséhez szükséges további fájlok neveit
  - a programban használt névtereket
  - a programban előre definiált elnevezések (makrók) értékeit:  
**#define <azonosító> <érték>**
    - olyan azonosító/érték (kifejezés) párok, ahol az azonosító összes előfordulása lecserélődik az értékre
    - pl.:  
**#define SIZE 100**  
...  
**int t[SIZE]; // int t[100]**  
**while (i < SIZE) { ... } // while (i < 100)**

# C++ alapismeretek

## Fejrész

- A C++ utasításai és típusai több fájlban helyezkednek el, amelyeket használatba kell vennünk a programunkban
  - a programozó is elhelyezheti a kódját több fájlban
  - a program fordításakor a hivatkozott fájlok teljes tartalma átmásolódik a mi programkódunkba
- A fájlok használatba vétele az `#include` utasítással történik
  - `#include <fájlnév>`: a C++ nyelvi könyvtárában keres
  - `#include "fájlnév"`: az aktuális könyvtárban keres
  - pl.:

```
#include <iostream> // konzol használat
#include <cmath> // matematikai függvények
```

# C++ alapismeretek

## Névterek

- Az egyes fájlokban található utasítások csoportosítva vannak úgynevezett *névterek*be, amelyek tükrözik az utasítások célját, felhasználási területét
  - egy névtéren belül nem lehet megegyező utasítás, vagy típus, de különböző névterekben igen
- A programban meg kell az utasítások névterét is, ennek módjai:
  - a fejrészben a `using namespace <név>;` utasítással, ekkor a teljes fájlban elérhető a névtér összes utasítása, pl.:  
`using namespace std;`
  - az utasítást a `<névtérnév>::<parancsnév>` formában írjuk le, ekkor megmondjuk, hogy a parancs a megadott névtérből való, pl.: `std::cout`

- A program törzsrészában található a főprogram:

```
int main()  
{  
    ...           // utasítások  
    return 0;     // program eredménye  
}
```

- A főprogramban a **return** utasítással adjuk meg a programból az operációs rendszer számára visszaadott *hibakódot*
  - lehet 0 (ekkor nem történt hiba), illetve egyéb szám (amely valamilyen hibaeseményre utal)
  - a hibakód jelentése nincs előre szabályozva, programtól függhet (általában a dokumentáció tartalmazza)



- Szekvencia:
  - utasítások egymásutánja, az utasítások végére `;`-t kell tenni
  - nincs összefüggésben a sortöréssel
- Programblokk: `{ <utasítások> }`
  - utasítások csoportosítása, amelyet tetszőlegesen elhelyezhetünk a főprogramon belül
  - programblokkok tartalmazhatnak további blokkokat, pl.:

```
{  
    <utasítások>  
    { <utasítások> }  
}
```

- Elágazás (egy-, vagy kétágú):
  - egy feltételtől függően különböző utasítások végrehajtása:  
`if (<feltétel>)`  
    *<igaz ág>*  
`else`  
    *<hamis ág>*
  - a hamis ág elhagyható, amennyiben üres lenne:  
`if (<feltétel>)`  
    *<igaz ág utasításai>*
  - a feltétel logikai típusú kifejezés
  - egy ágba több utasítás is helyezhető programblokk segítségével

- Többágú elágazás:
  - egy változó aktuális értékének függvényében különböző utasítások futtatása:

```
switch (<változónév>) {  
    case <érték1>:  
        <utasítások>  
        break;  
    case <érték2>:  
        <utasítások>  
        break;  
    ...  
    default:  
        <utasítások>  
}
```

- az ágak végét a **break** utasítással jelöljük (nem szükséges programblokk megadása)
  - hiánya esetén a következő ágon folytatódik a végrehajtás
- lehet „egyébként” ágat készíteni a **default** utasítással
- Ciklus:
  - utasítások (*ciklusmag*) ismétlése a megadott feltétel (*ciklusfeltétel*, amely logikai típusú kifejezés) függvényében
  - *előltesztelő*, ahol az utasítások csak a feltétel teljesülése esetén hajtódnak végre:  
**while** (<*feltétel*>)  
    <*utasítások*>

# C++ alapismeretek

## Vezérlési szerkezetek

- *hátultesztelő*, ahol az utasítások egyszeri végrehajtása után ellenőrizzük a feltételt:

```
do {  
    <utasítások>  
} while <feltétel>;
```

- *számláló*, ahol a feltétel egy adott lépésszám elérése:

```
for (<számláló kezdőérték>;  
    <számláló feltétele>;  
    <számláló inkrementálás>)  
    <utasítások>
```

- a számláló ciklus kiváltható előtesztelővel

- pl.: 

```
for (in i = 0; i < 10; i++)  
    cout << i << endl;
```

# C++ alapismeretek

## Operátorok

---

- Matematikai műveletek: összeadás (+), kivonás (-), szorzás (\*), osztás (/), maradékvétel (%)
  - pl.: `a % 3 // a modulo 3`
- Logikai műveletek: tagadás (!), és (&&), vagy (||)
  - pl.: `!(a && b) // nem (a és b)`
- Összehasonlító műveletek:
  - egyenlőségvizsgálat (==), különbségvizsgálat (!=), kisebb (<), nagyobb (>), kisebb, vagy egyenlő (<=), nagyobb, vagy egyenlő (>=)
  - pl.: `b == 3 // a b értéke egyenlő-e 3-mal?`

- Értékadás (=):
  - a balértéket egyenlővé tesszük a jobbértékkel
  - a balérték csak változó, a jobbérték tetszőleges kifejezés lehet (amelynek eredménye átadható a változnak)
  - pl.: `b = 3 // a b értéke 3-ra változik`
- Művelettel egybekötött értékadások (értékmódosítások):
  - hozzáadás (`+=`), kivonás (`-=`), modulo vétel (`%=`), feltételes egyenlővé tétel (`>>=`, `<<=`), ...
  - pl.: `a += 2 // a = a + 2`

- Feltételes értékadás (`? :`)
  - egy feltétel függvényében más érték kerül a változóba
  - pl.: `c = (a < b) ? a : b;`  
`// a és b közül a kisebb érték kerül c-be`
- Értékmódosítások:
  - eggyel növeli (`++`), vagy csökkenti (`--`) a változót
  - két módon lehet megadni, a különbség a végrehajtási sorrendben van (értékadással való használat esetén)
  - pl.: `a++` `// a értékének növelése`  
`b = ++a;` `// előbb növelés, utána értékadás`  
`b = a++;` `// előbb értékadás, utána növelés`



- Bitenkénti műveletek:
  - bitenkénti eltolás balra (<<), bitenkénti eltolás jobbra (>>), komplement képzés (~), bitenkénti és (&), bitenkénti vagy (|), kizáró vagy (^)
  - pl.: `a ^ b // a XOR b`
- Tömbelem indexelése ([ ]):
  - tömb (vektor), illetve szöveg bármely elemét lekérdezhetjük, módosíthatjuk
  - pl.: `a[3] // az a tömb 3-as indexű eleme`

- Változókat bárhol deklarálhatunk a kódunkban, nincs külön deklarációs rész
  - a minden programblokkon kívül deklarált változók a *globális változók* (a programban bárhol elérhetőek)
  - programblokkon belül deklarált változók a *lokális változók* (csak az adott programblokk végéig érhetőek el)
- Minden változónak létrehozásakor meg kell adnunk a típusát, és azt a fordító nyomon követi a program lefordításakor
  - változó deklaráció: **<típus> <változónév>;**
  - változó deklaráció kezdeti értékadással:  
**<típus> <változónév> = <kezdőérték>;**

# C++ alapismeretek

## Változók, konstansok

- amennyiben nem adunk egy változónak kezdőértéket, akkor bármilyen érték szerepelhet benne
- egyszerre több változót (ugyanazon típussal) vesszővel elválasztva deklarálhatunk:

*<típus> <változó 1>, <változó 2>;*

- Elnevezett konstansokat a **const** kulcsszóval hozhatunk létre, ekkor értéket is kell adnunk:

*const <típus> <változónév> = <kezdőérték>;*

- Pl.:

*int a = 0;*

*int first, second;*

*const string helloWorld = "Hello World!";*

- Logikai típus (`bool`):
  - felvehető értékek: `true`, `false`
  - meg lehet adni egész számokat is, ekkor a 0 értéke a hamis, minden más igaz
  - ha kiíratunk egy logikai változót, akkor 0-t, vagy 1-t ír ki
  - logikai, illetve bitenkénti műveletek végezhetőek rajta, a bitenkénti művelet megfeleltethető a logikai műveletnek (pl. tagadás és komplementer)
  - pl.:

```
bool a, b, c;  
a = true; b = false;  
c = a || !(~a && b) && true;
```

- Egész típusok:
  - különböző kulcsszavakat használhatunk (a változó memóiafoglalásának függvényében, a pontos méretek implementációfüggőek):
    - **short**:  $-32\,768 \dots +32\,767$
    - **int**:  $-2\,147\,483\,648 \dots +2\,147\,483\,647$
    - **long**:  $-9\,223\,372\,036\,854\,775\,808 \dots$
  - kompatibilisek egymással, illetve a logikai és valós típussal
  - pl.: 

```
int a = 2, b = 4;  
    short c = a / b;  
    long d = (a * (c + 6) - 4) % b;  
    int e = d + true;
```

- Valós, lebegőpontos típusok:
  - különböző kulcsszavakat használhatunk (a változó memórafoglalásának függvényében, a pontos méretek implementációfüggőek):
    - `float`:  $3.4 \cdot 10^{-38} \dots 3.4 \cdot 10^{38}$
    - `double`:  $1.7 \cdot 10^{-308} \dots 1.7 \cdot 10^{308}$
  - a megvalósítás a tartomány mellett a pontosságban is eltér
  - a típusok kompatibilisek egymással, az egész típusokkal, illetve a logikai típusal
  - pl.: `double a = 2.3, b = 1.0;`  
`double c = b / 3; // c == 0.333333333...`

- Karakter típus (**char**):
  - a karaktereket szimpla idézőjelben kell megadnunk
  - a karakter ASCII kódját tárolja, ezért kompatibilis az egész típussal
  - pl.: `char ch = 'y';`  
`int chInt = ch; // chInt == 97`
  - manipulációs műveletek (a `cctype` fájlból):
    - kisbetűvé alakítás: `tolower(<karakter>)`
    - nagybetűvé alakítás: `toupper(<karakter>)`
    - betű-e a karakter: `isalpha(<karakter>)`
    - szám-e a karakter: `isdigit(<karakter>)`

- Tömb típusok:
  - bármely típusból előállható tömb, ehhez létrehozáskor egy méretet kell megadnunk az indexelő (`[]`) operátor segítségével (méretként csak konstans egész adható meg)
  - elem lekérdezése és beállítása szintén az indexelő operátorral történik, az indexelés 0-tól indul
  - pl.: `int array[10]; // egészek 10 elemű tömbje`  
`array[0] = 1; // első elem beállítása`  
`cout << array[9]; // utolsó elem kiírása`
  - lehet többdimenziós tömböket (mátrixokat) is készteni azáltal, hogy egymásba ágyazzuk a tömböket



- Szöveg típus (**string**):
  - a szövegeket dupla idézőjelben adjuk meg
  - igazából karakterek tömbjét valósítja meg, nem beépített típus, használatához kell a **string** fájlt
  - számos művelete adott, pl.:
    - adott karakter lekérdezése: `<szöveg>[<index>]`
    - szöveg összefűzése: `<szöveg1> + <szöveg2>`
    - szöveg hosszának lekérdezése: `<szöveg>.length()`
    - üres-e a szöveg: `<szöveg>.empty()`
    - szöveg törlése: `<szöveg>.erase()`
    - karaktertömbbé konvertálás: `<szöveg>.c_str()`

- részszöveg lekérdezése:  
`<szöveg>.substr(<kezdőindex>, <hossz>)`
- karakter első előfordulásának helye (indexe, ha nem találja `string::npos` a visszatérési érték):  
`<szöveg>.find(<karakter>)`
- szövegrész lecserélése:  
`<szöveg>.replace(<kezdőindex>, <hossz>, <új szövegrész>)`
- pl.:  

```
string s; // üres szöveg
string s1 = "árvíztűrő", s2 = "tükörfúrógép";
s = s1 + " " + s2;
// s = "árvíztűrő tükörfúrógép" lesz
```

# C++ alapismeretek

## Típusok

```
s.append("!");  
    // s = "árvíztűrő tükörfúrógép!" lesz  
s.replace(0,1,"Á");  
    // s = "Árvíztűrő tükörfúrógép!" lesz  
int length = s.length(); // length = 23 lesz  
char ch = s[2];           // ch = 'v' lesz  
string sub1 = s.substr(0,5);  
    // sub1 = "Árvíz" lesz  
int index1 = s.find('ő'); // index1 = 8 lesz  
int index2 = s.find('r');  
    // index2 = 1 lesz, az elsőt találja meg  
string sub2 = s.substr(0, s.find(' '));  
    // sub2 = "Árvíztűrő" lesz  
s.erase(); // s = "" lesz
```

# C++ alapismeretek

## Véletlen generálás

- Véletlen számok előállítására a `cstdlib` fájlban lévő utasításokat használhatjuk:
  - `srand(<kezdőérték>)`: inicializálja a generátort
  - `rand()`: megad egy véletlen egész értékű pozitív számot
- A generálás mindig a kezdőértékhez viszonyítva történik, amely lehet konstans és változó is
  - általában az aktuális időpillanatot szokás megadni, amit lekérdezhetünk a `time(0)` utasítással (a `ctime` fájlból)
  - pl.: 

```
srand(time(0)); // inicializálás  
int r = rand(), // tetszőleges szám  
q = rand() % 10 + 1; // 1 és 10 közötti szám
```

# C++ alapismeretek

## Konzol használat

- A C++ adatbeolvasásra és megjelenítésre alapvetően konzol felületet használ, amelyhez az `iostream` fájl szükséges
  - beolvasás a `cin` utasítással és a `>>` operátorral, kiírás a `cout` utasítással és a `<<` operátorral történik
  - kiírásnál az operátorok között lehetnek konstansok, változók és kifejezések tetszőleges típusból, illetve speciális karakterek, pl. sorvége jel (`endl`)
  - pl.:

```
int val;  
cin >> val;    // val bekérése  
cout << "A értéke: " << val;    // kiíratása  
cout << "Egy sor" << endl << "Másik sor";  
                // sortörés beiktatása
```

# C++ alapismeretek

## Forrásfájlok és fordítás

- C++ programok `cpp` (`cc`, `c++`) és `hpp` (`h`, `h++`) kiterjesztésű fájlokban helyezkednek el
- A fájlok összefogására a fejlesztőkörnyezetek (pl.: Eclipse, Code::Blocks, Visual Studio, ...) projekteket használnak, amelyben lévő fájlokat egyszerre fordítjuk le
- Az egyik alapvető fordítóprogram a `g++`, használata:  
`g++ <kapcsolók> <fájlnev1> ... <fájlnevn>`
  - pl.: `g++ main.cpp`
  - alapértelmezetten egy `a.out` nevű programot készít, de ezt a `-o <fájlnev>` kapcsolóval megváltoztathatjuk
  - a `-pedantic` csak a szabvány kódot fogadja el

*Feladat:* Írjuk ki egy egész szám rákövetkezőjét.

- egy egész (`int`) típusú változóban (`val`) bekérjük konzol felületen az értéket
- a változót inkrementálással (`++`) megnöveljük, az eredményt azonnal kiíratjuk
- mindezt a főprogramban, amely 0-val tér vissza minden esetben
- a konzol használatához szükségünk van az `iostream` fájlra és az `std` névtérre

# C++ alapismeretek

## Példák

*Megoldás:*

```
#include <iostream>
using namespace std;

int main() {    // főprogram
    int val;    // val nevű, egész típusú változó

    cin >> val; // val értékének bekérése
    cout << ++val;
        // val érték megnövelése, kiíratás

    return 0;    // visszatérési érték
}                // főprogram vége
```



*Feladat:* Olvassunk be egy egész és egy valós számot, és írjuk ki a hányadosukat.

- egész és valós szám hányadosa valós lesz, az eredményt a kiírás előtt szintén eltároljuk
- létrehozunk egy egész (**long**) típusú változót (**integer**), valamint két valós (**float**) változót (**real**, **result**)
- beolvassuk a két számot a konzol képernyőről, a beolvasást a felhasználó számára kiírással könnyítjük meg
- a hányadost eltároljuk az eredmény változóba, majd annak értékét kiírjuk

# C++ alapismeretek

## Példák

*Megoldás:*

```
#include <iostream>
using namespace std;

int main() {
    long integer; // egész szám
    float real, result; // valós számok

    cout << "Első szám: "; cin >> integer;
    cout << "Második szám: "; cin >> real;
    result = integer / real;
    cout << "Hányados: " << result << endl;
    return 0;
}
```

# C++ alapismeretek

## Példák

*Feladat:* Döntsük el egy egész számról, hogy páros-e.

- egy egész típusú változót használunk (**nr**), amelynek értékét bekérjük a felhasználótól
- egy kétágú elágazással kiírjuk a megfelelő választ, ahol a feltétel a változó párossága
- a párosság eldöntése maradékvétellel történik, amennyiben 2-vel osztva a maradék nulla, a szám páros

*Megoldás:*

```
#include <iostream>  
using namespace std;
```

# C++ alapismeretek

## Példák

```
int main(){
    long nr;
    cout << "A szám: ";
    cin >> nr;

    if (nr % 2 == 0) // ha páros
        cout << "A szám páros.";
    else // ha nem páros
        cout << "A szám páratlan.";

    return 0;
}
```

# C++ alapismeretek

## Példák

*Feladat:* Írjunk ki N darab csillagot a képernyőre.

- N értékét bekérjük a felhasználótól egy egész (**short**) típusú változóba (**n**)
- egy számláló ciklust használunk, amellyel minden lépésben kiírunk egy csillagot
- a számláló egy egész változó lesz (**i**), amelyet egyenként inkrementálunk

*Megoldás:*

```
#include <iostream>
using namespace std;
```

# C++ alapismeretek

## Példák

```
int main()  
{  
    short n;  
    cout << "A csillagok száma: ";  
    cin >> n;  
  
    for (short i = 0; i < n; i++)  
    {  
        // számláló ciklus az i ciklusváltozóval  
        cout << "*";  
    }  
  
    return 0;  
}
```

*Feladat:* Adjuk meg egy természetes szám valódi osztóinak számát.

- természetes számot nem tudunk bekérni, csak egész számot (**nr**)
- ezért ellenőrizzük, hogy a megadott egész szám természetes-e, különben kilépünk (1-es hibakóddal)
- a számlálás programozási tételét használjuk, minden nála kisebb, de 1-nél nagyobb számról ellenőrizzük, hogy osztója-e (az osztás maradéka 0)

# C++ alapismeretek

## Példák

*Megoldás:*

```
#include <iostream>
using namespace std;

int main(){
    int nr, c = 0;    // nr a szám, c a számláló

    cin >> nr;
    if (nr < 0) // ellenőrzés
        return 1; // visszatérés 1-es hibakóddal

    for (int i = 2; i < nr; i++) // 2-től nr-1-ig
        if (nr % i == 0) // ha valódi osztó
            c++; // növeljük c-t
```



# C++ alapismeretek

## Példák

```
/* megvalósítás előltesztelő ciklussal:

int i = 2; // ciklusszámláló kezdőérték
while (i < nr){
    if (nr % i == 0)
        c++;
    i++; // ciklusszámláló növelés
}
*/

cout << nr << " valódi osztók száma: " << c
    << endl; // kiírjuk az eredményt
return 0;
}
```

*Feladat:* Olvassunk be 5 egész számot a képernyőről, és adjuk meg, van-e köztük negatív érték.

- először olvassuk be a számokat, majd egy második ciklusban keressük meg, hogy van-e negatív érték (lineáris keresés)
- az értékeket el kell tárolnunk, ezért fel kell vennünk egy 5 hosszú egész tömböt
- használjunk számláló ciklusokat, a második feltételét ki kell egészítenünk a logikai értékkel

*Megoldás:*

```
#include <iostream>
using namespace std;

int main(){
    int t[5];    // 5 egész számból álló tömb

    for (int i = 0; i < 5; i++)
        cin >> t[i]; // tömb elemeinek beolvasása

    bool l = false; // logikai érték
    // megvalósítás számláló ciklussal:
    for (int i = 0; i < 5 && !l; i++) {
        l = (t[i] < 0);
    }
```

# C++ alapismeretek

## Példák

```
/* megvalósítás előltesztelő ciklussal
   (nagyon tömören):
int i = 0;
while (i < 5 && 1 = (t[i++] < 0)) { }
*/

if (1)
    cout << "Van negatív szám!";
else
    cout << "Nincs negatív szám!";

return 0;
}
```

*Feladat:* Adjuk meg, hogy egy előre definiált méretű számsorozatban hány eleme kisebb az átlagnál.

- az adatokat egy tömbben tároljuk (**array**), kiszámítjuk az átlagot (**avg**) és a kisebb elemek számát (**smaller**)
- a változókat létrehozzuk a program elején, és megadjuk nekik a kezdő értéket
- bekérjük az adatokat, majd kiszámítjuk az átlagot (összegzés segítségével), végül az átlagnál kisebb elemek számát (számlálás segítségével)
- a tömb méretét előre rögzítjük a program fejrészában (**size**)

# C++ alapismeretek

## Példák

*Megoldás:*

```
#include <iostream>
using namespace std;
#define SIZE 10 // definiáljuk a méretet 10-nek

int main(){
    double array[SIZE], avg = 0, smaller = 0;
    // változók létrehozása kezdeti értékkel

    cout << "Bemenő számok: " << endl;
    for (int i = 0; i < SIZE; i++) {
        cout << i+1 << ". szám: ";
        cin >> array[i]; // beolvasás
    }
```

# C++ alapismeretek

## Példák

```
// összegzés
for (int i = 0; i < SIZE; i++) {
    avg += array[i];
}
avg = avg / SIZE;

// számlálás
for (int i = 0; i < SIZE; i++) {
    if (array[i] < avg)
        smaller++;
}
cout << "Az átlagnál " << smaller
      << " kisebb szám van.";
return 0;
}
```

# C++ alapismeretek

## Véletlen generálás

*Feladat:* Generáljunk 5 véletlen számot 0 és 100 között, és írjuk ki őket a képernyőre.

*Megoldás:*

```
#include <iostream>
#include <cstdlib> // kell a véletlen generátorhoz
#include <ctime> // kell az aktuális időhöz
using namespace std;

int main(){
    srand(time(0)); // inicializálás
    for (int i = 0; i < 5; i++)
        cout << (rand() % 101) << endl; // generálás
    return 0;
}
```