Informatika 3.
1. előadás: Bevezető

Tóth Dávid

Budapesti Műszaki és Gazdaságtudományi Egyetem

2026.02.17.

Történet

C

- Dennis Ritchie fejlesztette ki
- régi (1972), de a mai napig nagyon sokat használt
- fordított nyelv (nem interpretált, mint a python)
- hatékony, közel van a gépi kódhoz

C++

- Bjarne Stroustrup fejlesztette
- régi (1985), nagyon elterjedten használt
- tekinthető a C kiegészítésének
- a mai napig fejlesztett, legutóbbi szabvány C++23 (2024)

Miért C++

Miért C++?

- a mai napig széles körben használt
- gyors (hatékony) programokat lehet írni
- minden rendszeren elérhető ingyenesen valamilyen fordító/IDE
- rengeteg példa elérhető a web-en

C-vel kezdünk

```
#include <stdio.h>
int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Mielőtt le tudnánk futtatni, le kell fordítanunk.
Fordítók:

- Windows: mingw, visual studio
- Linux: gcc/g++
- Mac: Xcode -> gcc/g++, cc

Pythonról C-re

- meg kell adnunk a változók típusát, a függvények visszatérési értékének a típusát:

```
int i = 0;
double d = 0.0;
```

- a változókat használat előtt deklarálni kell, a deklarációnál meg kell adni a típust:

```
int i;
i = 1;
```

- deklarálásnál rögtön értéket is adhatunk a változónak az = operátorral:

```
int i = 1;
```

- írjuk ki az egész értékét:

```
int i = 1;
printf("Az␣i␣valtozo␣erteke␣%d!\n", i);
```

Pythonról C-re

- a program kiindulási pontja a *main* függvény:

```
int main() {
```

- a *for* ciklus nem listán iterál:

```
int i;
for(i = 0; i < 10; i++) {...}
```

- elágazás:

```
if(feltetel){
...
}
else if(feltetel){
...
}
else{
...
}
```

Pythonról C-re

- *while* (amíg) ciklus, minden alkalommal lefut addig, amíg a while után szereplő feltétel teljesül:

```
while(feltetel){
...
}
```

- *do-while* ciklus, addig fut újra, amíg a ciklus végén a while után lévő feltétel a futás után teljesül:

```
do{
...
}while(feltetel);
```

- nincsenek kényelmes beépített adatszerkezetek (lista, szótár), helyette használhatunk *tömböket*:

```
int t[10];
t[0] = 1;
```

Kiírás/bolvasás

- C-ben a `printf` ill. `scanf` parancsokkal tudunk a képernyőre szöveget írni ill. onnan olvasni, ezekről csak röviden és felületesen lesz szó (mert nem fogjuk használni őket).
- Az `stdio.h` fejállományban vannak deklarálva, ahhoz, hogy használni tudjuk őket, a program elejére a következő sort kell írni:

```
#include <stdio.h>
```

- A `printf` használatakor a kiírandó szöveget `" "` közé tesszük, ha változók értékeit akarjuk kiírni, azt a szövegben pl. `%d` (egész) vagy `%f` (tört) jelzi, majd vesszővel elválasztva az előfordulás sorrendjében adjuk meg a kiírandó változókat:

```
int a = 1;
double b = 0.2;
printf("A valtozok ertekei: %d, %f", a, b);
```

Kiírás/bolvasás

- A `scanf` esetében meg kell adni a változó típusát `" "` között, utána pedig megadni a változót, annak neve elé azonban egy `&` jelet kell írni (ennek jelentése később világossá válik majd).

```
int a;
printf("Adjon meg egy egesz szamot: ");
scanf("%d", &a);
```

- Ez a kód MS Visual Studio-ban le sem fordul, mert a `scanf` függvény használata "nem biztonságos". A fordító a `scanf_s` függvényt javasolja, ami ugyanígy használható.

iostream

- C++-os kiírás/beolvasás: a két kulcsszó `cin` és `cout`.
- `cin`-el tudunk beolvasni értékeket, ez helyettesíti a `scanf`-et:

```
int a;
float f;
cin >> a;
cin >> f;
```

- Nem kell jelezni, hogy milyen típust olvasunk be, így sokkal kényelmesebb mint a `scanf`.
- A kiíráshoz használt `cout`-hoz hasonlóan nem kell típus, és lehet "fűzni":

```
int a = 5;
float f = 6.4;
cout << "a: " << a << endl << "f: " << f << endl;
```

- Az `endl` az új sor (end line), a `"\n"` eredménye ugyanez.

iostream

- A `cin` és `cout` az `iostream` fejállományban vannak deklarálva. Mostantól ezt használjuk az `stdio.h` helyett.
- A C++-os fejállományok nevét kiterjesztés nélkül használjuk, így lettek megkülönböztetve a C-s fejállományoktól (ahol a .h kiterjesztés szokás).
- Egy teljes program `cin` és `cout`-al:

```
#include<iostream>
using namespace std;
int main() {
  float x;
  cin >> x;
  cout << "x erteke: " << x << endl;
  return 0;
}
```

- A `using namespace std` sorról később lesz szó, addig csak másoljuk be mindig az `iostream` alá.

Egész típusok

- a C++-ban a sztenderd egész típus az `int`
- az, hogy mekkora adatokat tárolhatunk egy `int` típusú változóban, számítógépfüggő
- az `int` típus méretét a `sizeof(int)` utasítással kérhetjük le (sokszor 4 byte)
- ha b biten tárolunk egy `int` típust, akkor az ilyen típusú változók lehetséges értékeiket a $[-2^{b-1}; 2^{b-1} - 1]$ intervallumon veszik fel
- további egész típusok: `short`, `long`, `long long`
- `sizeof(short)` $\leq$ `sizeof(int)` $\leq$ `sizeof(long)` $\leq$ `sizeof(long long)`
- a min és max értékeket a `climits` fejállomány tatralmazza
- az előjel nélküli típusok az `unsigned` előjelet kapják (pl. `unsigned int`, `unsigned long long` stb.)

Lebegőpontos típusok

- valós számok ábrázolására C++-ban a `float`, `double` és `long double` ún. lebegőpontos típusokat használjuk
- `sizeof(float)` $\leq$ `sizeof(double)` $\leq$ `sizeof(long double)`
- ha különös érvünk nincs más mellett, érdemes a `double`-t használni (pontosabb, mint a `float`, több értéket tud ábrázolni)
- literálok: 0.5f (`float`), .543, 1.2e10 (`double`), 10e-3L (`long double`)

Egyéb típusok

- logikai típus: `bool`, értéke `true` (1) vagy `false` (0)
- ezek tipikusan összehasonlító operátorok visszatérési típusai: pl. `x==y`, `a<=b` stb.
- a következő programrészlet 0-t vagy 1-et ír ki a képernyőre

```
cout << (x==y) << endl;
```

- karaktereket `char` típusú változókban tárolhatunk
- az értéküket megadhatjuk karakterliterálok segítésével, vagy pedig annak értéke segítségével:

```
char c = 'A';
char d = 65;
cout << c << d << endl;
```

Operátorok

- szokásos aritmetikai operátorok: `+`, `-`, `*`, `/`, egész típusú változókon `%` (maradékos osztás),
- ezek értékadással együtt is használhatók, pl. `x %= 2` jelentése `x = x % 2`
- (ha nem muszáj, akkor) ne végezzünk különböző típusú változókon műveleteket,
- `++`, `--` operátorok egyoperandusúak, eggyel növelik/csökkentik a változó értékét
- összehasonlító operátorok: `==` (egyenlőség), `!=` (nem egyenlő), `<` (kisebb), `<=` (kisebb vagy egyenlő) stb.
- logikai operátorok: `!` (nem), `&&` (logikai és), `||` (logikai vagy)

Függvények

- Függvény deklarálása:

```
visszatérési típus név(paraméterek);
```

- A `teglalap` függvénynek két `float` típusú paramétere van, ezek nevei `a` és `b`, és egy `float` értéket ad vissza:

```
float teglalap(float a, float b);
```

- Függvény definiálás

```
float teglalap(float a, float b) {
    return a * b;
}
```

- A `return` parancs után a függvény futása véget ér, a `return` után szereplő értéket adja vissza a függvény.

Függvények

- Ha egy függvénynek nincs visszatérési értéke, akkor `void` visszatérési típussal deklaráljuk:

```
void f(int m, int n){
    for(int i = 0; i < n-1; ++i){
        cout << m+i << ", ";
    }
    cout << m+n-1 << endl;
}
```

- A `return;` parancs ekkor is alkalmazható a függvény futásának befejezésére.

Függvények

- Egy függvényt akkor lehet használni, ha korábban legalább deklaráltuk:

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

float teglalap(float a, float b);

int main() {
    cout <<  teglalap(5, 7) << endl;
    return 0;
}
float teglalap(float a, float b) {
    return a * b;
}
```

Függvények paraméterei

- Függvények argumentumai másolódnak:

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

void hibas(float x, float y, float sum) {
    sum = x + y;
}

int main() {
    float a = 0.0;
    hibas(5.0, 2.0, a);
    cout << a << endl;
    return 0;
}
...
0.0
```

Függvények paraméterei

- Ha át akarjuk írni a függvény argumentumában szereplő változó értékét, referenciát adunk át:

```
void helyes(float x, float y, float& sum) {
    sum = x + y;
}
```

- A referencia egy konkrét változó egy "álneve", jele `X&` (referencia `X` típusra).
- Mindig kell hozzá tartoznia objektumnak:

```
int x = 2;
int& r = x;
```

(itt r és x ugyanarra a változóra vonatkoznak)

Függvények paraméterei

- A referencia jó eszköz, ha egy függvénynek több értéket kell visszaadnia:

```
void division(long a, long b, long& q, long& r) {
    q = a / b;
    r = a % b;
}
```

- Mi a "hiba" a fenti függvényben?

A ternáris feltételes operátor

- A ? : ternáris operátor három operandusú operátor.
- Az első operandus (a ?) előtt egy feltétel, azaz logikai kifejezés áll, ami igaz vagy hamis értéket ad vissza.
- Ha ez igaz, akkor a ? utáni első, különben pedig a második kifejezés értékelődik ki.
- Tömörebb kód írható a segítségével, pl. a következő két kód ugyanazt csinálja:

```
if( i > 0 ) {
    i *= 2;
}
else{
    i *= -2;
}
```

illetve

```
i *= i > 0 ? 2 : -2;
```

A ternáris feltételes operátor

- A második és harmadik operandusban állhatnak műveletek is, sőt függvényhívás is.
- Ha az egyik operandusban olyan függvényt hívunk, ami nem ad vissza értéket, akkor a másiknak is olyannak kell lennie.
- Kerüljük az olyan eseteket, amikor a két kifejezés különböző típusú értékkel értékelődik ki, vagy a két meghívott függvény különböző típusú értékkel tér vissza (ha lehetséges konverzió a két érték közt, akkor ilyen esetben is lefordul a program, egyébként nem).
- A következő kód pl. nem fordul le (az első művelet visszatérési típusa `ostream&`, a másodiké `int`):

```
int i = 0;
i > 4 ? cout << "i>4" : i++;
```

Operátorok visszatérési értékei

- Az operátorok speciális függvények, melyeknek van visszatérési értékük.
- Pl. az `a+=1` művelet visszaadja az `a` változó értékét (valójában egy referenciát a változóra), így az alábbi kódban a `b` változó értéke 1:

```
int a = 0;
int b = a += 1;
```

- Ezekről később részletesen fogunk tanulni.
- Különbség a `++a` ill. `a++` műveletek között:
  - A `++a` megnöveli az `a` változó értékét, majd visszaad egy referenciát az `a` változóra.
  - Az `a++` megnöveli az `a` változó értékét, de még a régi értéket adja vissza.
- Kerüljük az olyan kódot, ami a fenti különbségeket kihasználja! (Hibalehetőség + nehezen olvasható kód.)