Bevezetés a programozásba

6. Előadás

C++ bevezető

Programozási nyelvek

- A PLanG interpretált nyelv
  - A programszöveget a keretrendszer értelmezi
    - Ennek eredménye a keretrendszer belső állapotának változása, a futtatáshoz szükséges adatok előkészítése, pl. mely sorokat kell megismételni a ciklus futtatásakor
    - A program futását a keretrendszer intézi
    - Az elkészített programból nem keletkezik *.exe* vagy más önállóan futtatható állomány

Programozási nyelvek

- Interpretált nyelvek **(Interpreted)**
  - PLanG
  - BASIC (egyes dialektusok), Eiffel
  - JavaScript, Mathematica, Matlab
  - Perl, Php, Python
- Lefordított nyelvek **(Compiled)**
  - C/C++, C#
  - Ada, BASIC (más dialektusok), COBOL
  - Pascal (a legtöbb implementáció), Java, Smalltalk
  - Python (compiled verziója is létezik)
- A GPU programozás tovább bonyolítja a képet

A C++ nyelv

- A C++ általános célú multiparadigmális programozási nyelv
- Első változata 1979-ben készült (Bjarne Stroustroup) a C programozási nyelvből.
- Eredeti célja: objektumorientált programozási lehetőségekkel való kiegészítése a nyelvnek
- Tanuláshoz azért jó mert szinte minden van benne ami programozási nyelvekben elő szokott fordulni

PLanG illetve C++ „Hello world!”

```
PROGRAM hello
   KI: "Hello world!", SV
PROGRAM_VÉGE
```

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Fejlécek

- Angolul header
- A C++-ban használhatunk dolgokat, amit mások már elkészítettek
- **`#include <`**fejlécnév**`>`**
- A félév során szükséges fejlécek
  - **`<iostream>`**: kimenet, bemenet (cout, cin)
  - **`<fstream>`**: fájl (ifstream, ofstream)
  - **`<string>`**: szöveg típus (string)
  - **`<cmath>`**: matematikai függvények (sin(x), cos(x))
  - **`<cstdlib>`**: véletlen szám, egyebek (rand(), abs(x), *conv.*)
- Ezek sokszor egymásra épülnek

Névterek

- **`namespace`**
- Arra való, hogy ugyanazt a nevet használhassuk más-más kontextusban, más-más értelemben
- Az **std** névteret fogjuk használni
  - Pl.: **`cout`** azonosító az std névtérben található, ezért kell így hívni: **`std::cout`**
- A **using namespace std;** azt jelenti, hogy ezeket az „std::” előtagokat nem kell kiírni

Vezérlési szerkezetek - szekvencia

- **Szekvencia:**
  - utasítások egymás után
  - az utasítások végére ; -t kell tenni
  - egy sorban lehet több utasítást is írni, illetve egy utasítást lehet több sorban is írni
- **Programblokk:**
  - utasítások csoportosításai, amelyek egy függvényen belül találhatóak
  - egy programblokk: { ***<utasítások>*** }
  - programblokkok tartalmazhatnak további blokkokat, így egymásba ágyazott szerkezetet készíthetünk

Kommentek

- Megjegyzést, avagy kommentet bárhol elhelyezhetünk a kódban
  - elhelyezhetjük a sor végén, akkor az utána a sorba írt utasításokat nem veszi figyelembe:

```
<utasítás>;    // megjegyzés
```

- elhelyezhetjük a sor közben, illetve bármely utasítás közben (ügyelve arra, hogy nem szó közben írjuk), ekkor a komment után lévő utasításokat figyelembe veszi

```
<utasítás>;   /*megjegyzés*/    <utasítás>;
```

```
/*
megjegyzés
…
*/
```

- ezzel a jelöléssel több soros megjegyzést is készíthetünk

Típusok

```
VÁLTOZÓK:
    i: EGÉSZ,
    v: VALÓS,
    c: KARAKTER,
    l: LOGIKAI,
    s: SZÖVEG,
    bf: BEFÁJL,
    kf: KIFÁJL,
    tomb: Típus[méret]
```

```
int i = 0;
double v = 0;
char c = ' ';
bool l;
string s = "szoveg";
ifstream bf;
ofstream kf;
Típus tomb[méret];
```

Értékadás, I/O műveletek

```
a := kif
```

```
a = kif ;
```

```
BE: Val1, Val2
BE bf: Val1, Val2
BE: SzovValt
BE bf: SzovValt
KI: Kif1, Kif2
KI: SV
KI kf: Kif1, Kif2
```

```
cin >> Val1 >> Val2;
bf >> Val1 >> Val2;
getline(cin, SzovValt);
getline(bf, SzovValt);
cout << Kif1 << Kif2;
cout << endl;
kf << Kif1 << Kif2;
```

Értékadás, I/O műveletek

- Az értékadás jele a C++ -ban: **=**

  Az egyenlőség vizsgálat jele más lesz: **==**
  - Könnyű összekeverni
  - Össze is fogod keverni…
- A **cin** bemenet maga a konzol, ahová gépelni lehet
  - A PLanG-gal ellentétben a **cin >> x;** beolvasás a C++ -ban blokkolja a program futását, tehát vár, amíg begépelünk valamit.
  - Igazi interakcióra lesz lehetőség!

Értékadás, I/O műveletek

- A **„bementicsatorna >> változó”** típusos olvasás, tehát a változó típusától függően működik, pontosan mint a PLanG **„BE: változó”**
- Viszont a PLanG-gal ellentétben a **„bementicsatorna >> szövegesváltozó”** csak egy szót olvas be. Szónak számít az, ami szóközzel, tab-bal, vagy sorvégével (**WHITE SPACE** karakterrel) van elválasztva.
- A **„getline(bemeneticsatorna, szövegesváltozó)”** típus nélküli olvasás, kizárólag szöveges változóba lehet így olvasni, **ENTER**-ig olvas

Elágazás

```
HA FeltKif AKKOR

   igaz-ág: utasítások
HA_VÉGE
```

```
if( FeltKif )
{
 igaz-ág: utasítások
}
```

```
HA FeltKif AKKOR

   igaz-ág: utasítások

KÜLÖNBEN

   hamis-ág: utasítások
HA_VÉGE
```

```
if( FeltKif )
{
 igaz-ág: utasítások
}
else
{
 hamis-ág: utasítások
}
```

Ciklus

```
CIKLUS AMÍG FeltKif

   utasítások
CIKLUS_VÉGE
```

```
while( FeltKif )
{
   utasítások
}
```

```
CIKLUS

   Utasítások
AMÍG FeltKif
```

```
do
{
   utasítások
} while( FeltKif );
```

Számláló ciklus

```
i := 0
CIKLUS AMÍG i<10

    ciklusmag
    i := i+1
CIKLUS_VÉGE
```

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

}
```

Számláló ciklus C++

- A számláló szerkezete itt teljesen elkülönül:

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

- De természetesen előtesztelő ciklussal is meg lehet fogalmazni a számlálót, az eredmény ugyanaz lesz:

```
<számláló kezdőérték>;
while( <számláló feltétele> )
{
   utasítások
   <számláló inkrementálás>;
}
```

Blokkok II.

- Észrevehető, hogy a PLanG-féle **XY_VÉGE** jelek C++ nyelvben egyformák: '**}**'
- A blokkos szerkezet alapvető, blokkot bármikor nyithatunk, ez jelenti azt, hogy egy utasításként gondolunk egy programrészletre

```
while( FeltKif ) {
   <utasítás1>;
}
```

=

```
while( FeltKif )
   <utasítás1>;
```

```
while( FeltKif ) {
   <utasítás1>;
   <utasítás2>;
}
```

!=

```
while( FeltKif )
   <utasítás1>;
<utasítás2>;
```

Példa: összegzés PLanG / C++-ban, végjeles sorozatra

```
PROGRAM összeg
  VÁLTOZÓK:
    a, sum: EGÉSZ

  sum := 0
  BE: a
  CIKLUS AMÍG a >= 0
    sum := sum + a
    BE: a
  CIKLUS_VÉGE
  KI: "Összeg: ", sum, SV

PROGRAM_VÉGE
```

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

int main()
{
    int a;
    int sum = 0;
    cin >> a;
    while( a >= 0 ) {
        sum += a;
        cin >> a;
    }
    cout << "Összeg: " << sum << endl;
    return 0;
}
```

Logikai eredményű műveletek

PLanG

- a = b
- a /= b
- a < b
- a >= b
- A ÉS B
- A VAGY B
- NEM A

C++

- a == b
- a != b
- a < b
- a >= b
- A && B *vagy* A and B
- A || B *vagy* A or B
- ! A

Egész eredményű műveletek

PLanG

- `-a`
- `|a|`
- `a + b`
- `a - b`
- `a * b`
- `a DIV b`
- `a MOD b`
- `RND a`

C++

- `-a`
- `abs(a)`
- `a + b`
- `a - b`
- `a * b`
- `a / b`
- `a % b`
- `rand() % a`

Az osztásról

- Van két osztás művelet
  - egész / egész (PLanG „DIV”)
  - valós / valós (PLanG „/”)
- Ezek egyformán néznek ki, de másképp működnek
  - **1 / 2 == 0**
  - **double(1) / double (2) == 0.5**
  - **1.0 / 2.0 == 0.5**
- Valós osztáshoz elég ha az egyik operandus nem egész

Szöveg eredményű műveletek

PLanG

- "a"
- **|s|**
- **s[i]**
- **s[k : v]**
- **s1 @ s2**
- **s @ kar**
- **s1 + s2**

C++

- "a"
- **s.length()**
- **s[i]**
- **s.substr(k, v-k)**
- **s1.find(s2)**
- **s.find(kar)**
- **s1 + s2**

A karakterekről

- `char a;`
- A char típus egyszerre jelent számot (8 bites előjeles, -128 .. 127) és karaktert
- ASCII táblázat (‘A’:65, ‘1’:49, ‘a’:97, …)
- a `cout << a;` egy karaktert fog kiírni a konzolra
- a `cout << int(a);` pedig az ASCII kódját, mert az átalakítással *(explicit típuskonverzió)* számként kezelést kértünk

Függvények

- Láthattunk több olyan C++ elemet, amely azonosító + zárójelpár, amelyen belül paraméterek vannak
- Ezek a függvényhívások
- Önálló programrészleteket tartalmaznak,
  - amelyeknek a bemenete a paraméterlista
  - kimenete pedig egy visszatérési érték
- Pl. `v1 = sin( v2 );`
  jelentése: indítsd el a **sin** nevű függvényt **v2** bemenettel és az eredményt írd a **v1**-be

Függvények

- PLanG programban is használtunk függvényhívásokat, pl. **RND x**, **NAGY c** …
- C++ -ban könnyen felismerhető a függvény: mindig kell mögé zárójelet tenni, akkor is, ha nem kell neki paraméter (pl. **`rand()`**)
- Néhány függvényhívás olyan, hogy egy kiemelt paramétert nem a zárójelbe kell írni, hanem elé és ponttal kell elválasztani, pl. a string műveletei közül több is, **s.length()**
  - Ezt egyelőre így fogadjátok el, ez az objektumorientált szintaxis, lesz még róla szó

Markáns különbségek

- C++ -ban ott deklarálunk új változót, ahol jólesik, nem kell kigyűjteni a program elejére
- Viszont bejön a **változó élettartama**
- Ha egy blokkon belül van egy deklaráció, akkor az a változó csak arra a (legszűkebb) blokkra érvényes, annak végével megszűnik
  - újrafelhasználhatóak a nevek, pl. minden ciklusnak lehet saját ***i*** változója (amíg nem egymásba ágyazottak)
  - mégsem keverhetőek össze véletlenül

Markáns különbségek

- C++ -ban ott deklarálunk új változót, ahol jólesik, nem kell kigyűjteni a program elejére
- Viszont bejön a változó élettartama
- Ha egy blokkon belül van egy deklaráció, akkor az a változó csak arra a (legszűkebb) blokkra érvényes, annak végével megszűnik
  - újrafelhasználhatóak a nevek, pl. minden ciklusnak lehet saját *i* változója (amíg nem egymásba ágyazottak)
  - mégsem keverhetőek össze véletlenül

```
{
    int szam;
    {
    string szoveg;
    szam = 12;
    }
    cout << szam << endl;
    szoveg = "Ez egy mondat!";
}
```

Markáns különbségek

- A PLanG véges abban az értelemben, hogy nem bővíthetőek a műveletek
- C++-ban lehetőség van új függvények létrehozására, ezek megosztására
- A legtöbb gyakorlati probléma megoldható úgy, hogy az ember megkeresi a megfelelő könyvtárat, `#include <fejlécnév>` és a dokumentáció szerint használja
  - fájlformátumok, grafika, adatbázis, hálózat, stb.
- Második félévben …

Markáns különbségek

- C++-ban nincs „nem kapott kezdeti értéket” hibaüzenet
- Memóriaszemét: kapunk valahol egy kis memóriát, ki tudja mi maradt ott az előző programok után
- Hibaüzenet helyett fura működés
- Esetleg kaphatunk figyelmeztetést (warning)
  - Ha nem értesz egy figyelmeztetést, az intő jel
- C++ -ban mindent szabad, amit nem tilos
  - Érdemes lesz beállítani, hogy a fordító minden gyanús dologért szóljon!

Markáns különbségek

- C++-ban akkor sincs „nem kapott kezdeti értéket” hibaüzenet, ha fájl vége után olvasunk tovább
- Figyelmeztetés sincs
- Általában végtelen sokáig ugyanazt az értéket ismétli, de bármi más viselkedés előfordulhat

C++ fejlesztői eszközök

- Fordítóprogram (forráskód → gépi kód)
  - GCC (windows-on MINGW)
  - Microsoft Visual C++
  - CLang
- IDE (programozáshoz való szövegszerkesztő extrákkal)
  - CodeBlocks, CLion, QtCreator, Visual Studio
- Dokumentáció, pl. cppreference.com