Bevezetés a programozásba

6. Előadás: C++ bevezető

ISMÉTLÉS

PLanG features

- Utasítások
  - Értékadás, KI:, BE:
- Programkonstrukciók
  - Elágazás
  - Ciklus
- Típusok
  - Egész, valós, logikai, szöveg, karakter, fájl
- Típuskonstrukciók
  - Tömb

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 (interpret)
  - PLanG
  - Excel, Matlab, Lisp, JavaScript, Smalltalk
- Lefordított nyelvek (compile)
  - C/C++
  - Pascal, Ada, Eiffel
- Hibridek (fordítás interpretálandó kódra)
  - Java, Python, PHP
- A GPU programozás tovább bonyolítja a képet

C++

- A C++ egy általános célú programozási nyelv
  - Mindent meg lehet csinálni C++-ban*
- Fordítást igényel
  - Hibaellenőrzés
    - szintaktikai: lemaradt zárójelek, befejezetlen sorok
    - szemantikai: megfelelő típusok a műveletek körül
    - linkelési: megvannak-e a szükséges könyvtárak
  - Kódgenerálás
    - Elkészül a .exe vagy egyéb futtatható fájl

* Néhány kivételért keressétek Tihanyi Attilát, de PC-n tényleg mindenre jó a C++

C++

```
PROGRAM példa
  KI: "Hello"
PROGRAM_VÉGE
```

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

int main()
{
   cout << "Hello";
}
```

Ökölszabály PLanG program C++-osításához

```
PROGRAM [bármi]



  KI: "Hello"
PROGRAM_VÉGE
```

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

int main()
{
    cout << "Hello";
}
```

Megnézzük ennél részletesebben is hamarosan, mi mit jelent

Fejlécek

- C++-ban használhatunk dolgokat, amit mások már megcsináltak
- #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))
  - <cstdlib> : véletlen szám, egyebek (rand(), abs(x))
- Ezek sokszor egymásra épülnek

Névterek

- **`namespace`**
  - arra való, hogy ugyanazt a nevet használhassuk más-más kontextusban más-más értelemben
- Az std névteret fogjuk használni
- a cout teljes neve std::cout, aminek az olvasata „a cout azonosító, amelyik az std névtérben van”
- a **`using namespace std`** azt jelenti, hogy ezeket az „std::” előtagokat nem kell kiírni

Vagyis

```
PROGRAM [bármi]




   KI: "Hello"
PROGRAM_VÉGE
```

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

int main()
{
    cout << "Hello";
}
```

A cout azonosító az iostream könyvtárban van, az std névtérben

Típusok

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

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

Elágazás

```
HA FeltKif AKKOR
   Utasítások
HA_VÉGE
```

```
if (FeltKif)
{
   Utasítások
}
```

```
HA FeltKif AKKOR
  Utasítások
KÜLÖNBEN
  Utasítások-Különben
HA_VÉGE
```

```
if (FeltKif)
{
   Utasítások
}
else
{
   Utasítások-Különben
}
```

Ciklus

```
CIKLUS AMÍG FeltKif
  Utasítások
CIKLUS_VÉGE
```

```
while (FeltKif)
{
   Utasítások
}
```

```
CIKLUS
   Utasítások
AMÍG FeltKif
```

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

Ciklus: a for ciklus

```
i := 0
CIKLUS AMÍG i<10
   ciklusmag
   i:=i+1
CIKLUS_VÉGE
```

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

Az **változó++** jelentése: a változó értéke nőjön egy értéknyit. Innen kapta a nevét a C++, miszerint a C nyelv után következő nyelvről van szó

Blokkok

- 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
- ugyanaz a **`while (felt) ut;`**
  mint **`while (felt) { ut };`**
- NEM ugyanaz a **`while (felt) ut1;ut2;`**
  mint **`while (felt) { ut1; ut2 };`**
  - blokk nélkül az ut1 a ciklusmag, és ut2 már nem az

Értékadás, I/O

```
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;
```

Tanulság: a **`cin`** és a **`cout`** a „sima” bemenet és kimenet nevei. Maga az író és az olvasó művelet a << és a >> .

Értékadás, I/O

- Az értékadás jele a „=” ! Az egyenlőség vizsgálat jele más lesz. („==”)
  - Könnyű összekeverni
  - Össze is fogod keverni
- A cin bemenet a konzol, ahová gépelni lehet
  - A PLanG-gal ellentétben a **`cin >> x`** beolvasás C++ -ban blokkol, tehát vár, amíg begépelünk valamit, a program addig nem fut tovább
  - Interakció!

Értékadás, I/O

- a „Forrás >> változó” típusos olvasás, tehát a változó típusától függően működik, pontosan mint a PLanG „BE: változó”
- a „getline(Forrás, szövegesváltozó)” típus nélküli olvasás, kizárólag szöveges változóba lehet így olvasni
- a „Forrás >> szövegesváltozó” egy szót olvas be. Szónak számít az, ami szóközzel, tab-bal, vagy sorvégével van elválasztva.

Példa: összegzés C++-ban, végjeles sorozatra

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

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

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**
  - **A || B**
  - **! A**

Egész eredményű műveletek

- PLanG
  - **-a**
  - **|a|**
  - **a + b**
  - **a - b**
  - **a * b**
  - **a DIV b**
  - **a MOD b**
  - **VÉL 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`**

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 cout << int(a) pedig az ASCII kódját, mert az átalakítással számként kezelést kértünk

Függvények

- Láthattunk több olyan C++ elemet, ami azonosító + zárójelpár, amin belül paraméterek vannak
- Ezek függvényhívások
- Önálló programrészleteket tartalmaznak,
  - amiknek a bemenete a paraméterlista
  - kimenete pedig a kifejezés eredménye
- 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 VÉL X
- 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 deklarálsz új változót, ahol jólesik, nem kell kigyűjteni a program elejére
- 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

- 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 <fejléc neve>, és a dokumentáció szerint használja
  - grafika, fájlformátumok, 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)
  - Érdemes bekapcsolni általában is
  - Ha nem értesz egy figyelmeztetést, az intő jel.

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 környezetek

- GCC (windowson MINGW)
  - ingyenes, szabványos
  - fapados
- CodeBlocks, Dev-C++, …
  - GCC-hez feltupírozott szövegszerkesztők
- Visual Studio, CodeGear (Borland) C++ Builder
  - Színes-szagos
  - Nem szabványos
  - Platformfüggő