Bevezetés a programozásba

8. Előadás: Függvények 2.

ISMÉTLÉS

Helló

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

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

ISMÉTLÉS

Függvények írásmódja

- Ha egy C++ kódban ezt látjuk:
  típusnév bármilyennév(tetszőleges deklarációk)
  {
  ....
  return *valami*;
  }
- Akkor az egy függvény

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

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

ISMÉTLÉS

Függvényhívás

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

double terulet(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha,hb,hc;
    cin >> ha >> hb >> hc;
    double t = terulet(ha,hb,hc);
    cout << "terulet: " << t << endl;
    return 0;
}
```

ISMÉTLÉS

Függvények

```
double terulet(double a, double b, double c)
```

- Szignatúra:
  - Visszatérési típus (a függvény típusának is nevezik)
  - Függvény neve
  - Zárójelben
    - Paraméter 1, ha van
    - Paraméter 2, ha van
    - ..
- A szignatúrából minden kiderül arról, hogyan kell használni a függvényt

ISMÉTLÉS

Szimbolikus és aktuális paraméter

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

double terulet(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha,hb,hc;
    cin >> ha >> hb >> hc;
    double t = terulet(ha,hb,hc);
    cout << "terulet: " << t << endl;
    return 0;
}
```

szimbolikus

aktuális

ISMÉTLÉS

Változó láthatósága

- Egy változó látható a programkód azon soraiban, ahol a nevének leírásával használható
  - C++ nyelvben a lokális változók nem láthatóak más függvényekben
  - A mindenhol látható változót függvényen kívül kell deklarálni: globális változó
    - A globális változókat ahol lehet, kerüljük. Néhány értelmes használata van, ezekre általában kitérünk majd, minden másnál a jó stratégia az, hogy paraméterben adjuk át a szükséges adatokat
    - Ha mégis használjuk, azokban a függvényekben lesz látható, amelyek a deklaráció alatt vannak

ISMÉTLÉS

Változó érvényessége

- A változó érvényes a program futásának azon szakaszain, ahol az értékét megtartja
- A lokális változó a függvényben levő deklarálásától számítva addig él, amíg a deklarálásának blokkja be nem fejeződik.
- A paraméterek a függvény végéig élnek
- Ha egy A függvényből egy B függvényt hívunk, az A lokális változói ugyan nem láthatóak B-ben, de érvényesek maradnak, mert mikor B befejeződik, A-ban továbbra is használhatóak, és értéküket megtartották ezalatt.

ISMÉTLÉS

Strukturált programozás

- Programtervezési elv: a feladatot osszuk részfeladatokra, és függvényekben csoportosítsuk azokat
  - „dekompozíció”, „redukció”
- int nagyonnehézfeladat(P1 P2 P3 ..) {
  egyikkicsitkönnyebb(P2 P4 ..);
  másikkicsitkönnyebb(P1 P2 ..);
  ...
  }
- Top-down vagy Bottom-up felépítés
- 80-as évekig nagy szoftvereknél egyeduralkodó

ISMÉTLÉS

Rekurzív függvény

```
int faktor(int p)
{
    if (p>1) {
        return p*faktor(p-1);
    } else {
        return 1;
    }
}

int main()
{
    cout << faktor(5) << endl;
    return 0;
}
```

Nyitott kérdések

- Miért nem tudtam tömböt átadni paraméterként?
  - Néztem fórumokat, és ott valami int* volt int[10] helyett..
- Azt írta a PLanG-C++ táblázat, hogy sort tudok beolvasni úgy, hogy **`getline(cin, s)`**. Ez miért nem úgy van, hogy **`s=getline(cin)`**?
- Akkor hogy is lehet több eredményt visszaadni?
- Nem tudtam fájlt sem paraméterként átadni!

Tömb

- Nagy vonalakban kétféle megoldás:
- C nyelv, mutatók, memóriában egymást követő elemek
  - Nincs „mérete” művelet
  - Mutatót kell átadni (int * : egészre mutató mutató) és méretet
- C++ szabványos könyvtár
  - deklaráció: vector<int> tombom(10,0)
  - formális paraméter: vector<int> tomb
  - méret lekérdezése: tomb.size()

„Primitív” tömb

- PLanG:
- **VÁLTOZÓK : t:EGÉSZ[10]**
- **t[0] := 1**
- *[ilyet PLanG-ban nem lehet csinálni]*

- C
- **int t[10];**
- **t[0]=1**
-
```
int osszeg(int *t, int m){
  int sum=0;
  for (int i=0;i<m;i++){
    sum+=t[i];
  }
  return sum;
}
…osszeg(t,10)…
```

STL vector

- PLanG:
- *[semmi]*
- **VÁLTOZÓK : t:EGÉSZ[10]**
- **t[0] := 1**
- *[ilyet PLanG-ban nem lehet csinálni]*

- C++
- **#include <vector>**
- **vector<int> t(10); vagy vector<int> t(10,0);**
- **t[0]=1**
-
```
int osszeg(vector<int> t){
  int sum=0;
  for (int i=0;i<t.size()
  ;i++){
    sum+=t[i];
  }
  return sum;
}
…osszeg(t)…
```

Nyitott kérdések

- Miért nem tudtam tömböt átadni paraméterként?
  - Néztem fórumokat, és ott valami int* volt int[10] helyett..
- Azt írta a PLanG-C++ táblázat, hogy sort tudok beolvasni úgy, hogy **`getline(cin, s)`**. Ez miért nem úgy van, hogy **`s=getline(cin)`**?
- Akkor hogy is lehet több eredményt visszaadni?
- Nem tudtam fájlt sem paraméterként átadni!

Paraméterátadások

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

double terulet(double a, double b, double c)
{
    double s = (a+b+c)/2.0;
    return sqrt((s-a)*(s-b)*(s-c)*s);
}

int main()
{
    double ha,hb,hc;
    cin >> ha >> hb >> hc;
    double t = terulet(ha,hb,hc);
    cout << "terulet: " << t << endl;
    return 0;
}
```

Paraméterátadások

- Azt beszéltük meg, hogy a függvény paraméterei
  - a paraméterlistában vannak deklarálva
  - az értéküket az aktuális paraméterekből kapják
  - érvényességük és láthatóságuk a függvényre szorítkozik
  - tehát az aktuális paraméter másolatáról van szó
- Következésképp a **`getline(cin, s)`** nem változtathatja meg **`s`** értékét, tehát sort sem olvashatunk bele?

Paraméterátadások

- A különbség a paraméter átadásában és átvételében van
- Lehet úgy átvenni paramétert, hogy csak másolat készül, és így az átadott aktuális paraméter „írásvédett”
- Lehet viszont úgy is, hogy magát a változót adjuk át, és minden, amit a függvény csinál vele, az „maradandó károsodást okoz”
- Ezeket érték szerinti és referencia szerinti paraméterátadásnak hívjuk

Érték szerinti paraméterátadás

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

void fv(double a)
{
    a=0;
}

int main()
{
    double d;
    d=1;
    fv(d);
    cout << "eredmeny: " << d << endl;
    return 0;
}
```

Az eredmény:
1

Referencia szerinti paraméterátadás

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

void fv(double & a)
{
    a=0;
}

int main()
{
    double d;
    d=1;
    fv(d);
    cout << "eredmeny: " << d << endl;
    return 0;
}
```

Az eredmény:
0

Referencia szerinti paraméterátadás

- **`void fv(double & a)`**
  Érdemes úgy gondolni rá, hogy „a függvényen belül a paraméterként kapott változót így fogjuk hívni”
- Maga a referencia egy önálló fogalom C++-ban, tetszőleges változóra lehet referenciát állítani, és minden amit az egyikkel csinálunk, az a másikra is hatással van
- **`int a; int &r=a;`**
  `a=0` → **`r==0`** és viszont.
- Paraméter átvétele is így történik

Referencia szerinti paraméterátadás

- Következmények:
  - ha szeretnénk, hogy egy függvény változtassa a változónkat, akkor lehetséges referencia szerint átvenni a paramétert
  - csak változó lehet a paraméter, kifejezés eredménye vagy konstans nem
  - gyors, mert nem kell másolatot készíteni, ami nagyobb memóriaigényű változóknál (pl string egymillió karakterrel) lassíthat
  - veszélyes lehet, a legtöbb függvényhívásnál nem számítunk arra, hogy megváltozhat a paraméterül átadott változó (pl. matematikai függvények)

Nyitott kérdések

- Miért nem tudtam tömböt átadni paraméterként?
  - Néztem fórumokat, és ott valami int* volt int[10] helyett..
- Azt írta a PLanG-C++ táblázat, hogy sort tudok beolvasni úgy, hogy **`getline(cin, s)`**. Ez miért nem úgy van, hogy **`s=getline(cin)`**?
- Akkor hogy is lehet több eredményt visszaadni?
- Nem tudtam fájlt sem paraméterként átadni!

Több eredmény

- Lesz még egy megoldás, de ez most könnyű:
- Egyszerűen több paramétert átveszünk referencia szerint
  - ezek értékével egyáltalán nem foglalkozunk, csak felülírjuk azokat
  - így a paraméter jelentése az lesz, hogy „ide meg ide kérem az eredményt”
- A jövő héten nézünk majd egy másik megközelítést

Nyitott kérdések

- Miért nem tudtam tömböt átadni paraméterként?
  - Néztem fórumokat, és ott valami int* volt int[10] helyett..
- Azt írta a PLanG-C++ táblázat, hogy sort tudok beolvasni úgy, hogy **`getline(cin, s)`**. Ez miért nem úgy van, hogy **`s=getline(cin)`**?
- Akkor hogy is lehet több eredményt visszaadni?
- Nem tudtam fájlt sem paraméterként átadni!

Fájlok paraméterben

- A fájlokat (ifstream, ofstream) mindig referencia szerint kell átvenni
- Miért? Mert nem készíthető róluk másolat, aminek az az oka, hogy a „hol tartunk” bennevan a fájl típusban:
  - képzeljük el, hogy egy függvény lemásol (érték szerint átvesz) egy ifstream-et, és olvas belőle
  - visszatérés után a következő olvasásnál a fájl előző olvasásainak kéne újra megtörténni, hisz csak a másolatból olvastunk, az átadottból nem, az tehát nem is mehetett odébb
  - ez viszont követhetetlen, és technikailag rémálom

Fájlok paraméterben

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

void olvas(ifstream & f, double & a)
{
    f >> a;
}

int main()
{
    double d;
    ifstream befile("a.txt");
    olvas(befile, d);
    cout << "eredmeny: " << d << endl;
    return 0;
}
```

Összefoglalás

- Paraméterként a fájlok és a tömbök speciálisak
- Paramétert kétféleképpen is át lehet adni
  - Érték szerint: másolat készül lokális változóra
  - Referencia szerint: ugyanaz a változó több néven
- Mivel veszélyes mindent referenciaként átvenni, ezért csak akkor tegyük, ha a specifikációnk szerint eredményt adunk vissza benne