Bevezetés a programozásba

9. Előadás: Rekordok

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

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

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)…
```

ISMÉTLÉS

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

ISMÉTLÉS

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

ISMÉTLÉS

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~~ most nézünk majd egy másik megközelítést

Típuskonstrukció

- Meglevő típusokból új típus létrehozása
- Eddigi példák:
  - **`T v[10]`** : 10 darab T típusú változó alkot egy primitív vektort
  - **`T m[10][10]`** : 10x10 méretű (primitív) mátrix
  - **`vector<T> v(10,t)`** : 10 darab T típusú változó alkot egy STL vektort t kezdeti értékekkel
  - **`vector<vector<T> > m(10, vector<T>(10, t))`** : 10x10 méretű mátrix t kezdeti értékekkel

Típuskonstrukció

- A rekord: vegyes típusokból álló új típus
- **`struct`**
- Az elv: az összetartozó adatok összetartozása jelenjen meg, mint nyelvi elem
- Példa: kétdimenziós koordinátákat eddig két tömbben tároltunk: double x[100] és double y[100]
- Mostantól írhatjuk majd, hogy koord k[100]

struct

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

struct koord {
    double x,y;
};

int main()
{
  koord k;
  k.x=1.0; k.y=1.0;
  cout << "[" << k.x << "," << k.y
    << "]" << endl;
  return 0;
}
```

struct

- A struct kulcsszó jelentése: most egy új típust fogok leírni
- struct név {
  T1 mező1, mező2..;
  T2 mezőX, ..;
  ...
  };
- A típus neve bármi lehet, ami nem foglalt még
- T1, T2, .. típusoknak már ismert típusúak legyenek
- A mezőnevek különbözőek legyenek

A mezők

- A rekord mezőkből áll („tag”, angolul „member”)
- Minden mező egy már ismert típusba sorolható, és változóként használható: kaphat értéket, olvasható, referálható, stb.
- A mezők használatakor a struct típusú változók után '.' karakterrel elválasztva a mezőnevet kell írni
- Tehát az előző példában a 'k.x' jelentése: a k változó koord típusú, annak az x mezője, tehát egy double típusú érték

Mező vs változó

- Változónak szokás nevezni mindent, amit külön deklaráltunk
  - Mezőt a többi mező nélkül nem lehet deklarálni
- Egy struct egy mezőjét külön nem deklaráltuk, de mégis úgy használjuk, mint egy változót: adunk neki értéket, kiolvassuk, stb
- A szóhasználat tehát nem a képességeket, hanem a szerepet fedi: önállóan használandó, vagy egy nagyobb adatcsoport része?

Mi legyen mező, és mi ne?

- A mezők összessége mindig a rekord szerepét teljesen töltse ki, és ne legyen benne felesleg
- „Reprezentáció”: egy absztrakt fogalom ábrázolása meglevő típusokkal
  - pl. 2D koordináta két valós számmal, racionális szám két egész számmal, mint számláló és nevező
- A ciklusváltozó például nincs a mezők között: nem tartozik a fontos adatokhoz
- Az viszont nem nagy baj, ha a teljes értékkészletek nincsenek kihasználva, pl. tanuló jegye int mező. Ez csak kis baj.

struct és a típusok

- Az adott struct leírása után a megadott név már egy kész típus, használható deklarációkban, paraméterlistákban
  - Akár egy következő struct mezőjénél is, vektorban, primitív tömbben, stb..
- A rekord mezője is lehet vektor, vagy primitív tömb
- Fontos, hogy a forráskód fentről lefelé olvasva ezt a sorrendet betartsa

struct structban

```
struct ember {
    string nev;
    string lakcim;
    int szuletesi_ev;
};

struct diak {
    ember e;
    vector<int> jegyek;
};
```

struct structban

```
struct pixel {
    char r,g,b;
};

struct kep {
    int x, y;
    vector<vector<pixel> > p;
};
```

struct és függvények

- Az új típusaink használhatóak függvény paraméterekként
- Itt esetleg számítani kezd a hatékonyság, egy double[1000] mezővel is rendelkező struct érték szerint átadva lemásolódik, ami lassú
- Visszatérési értékként is használható, vagyis így lehet ismét több eredményt egyszerre visszaadni: több, összetartozó, és ezért egy típusba összefogható adatot

struct és függvények: előtte

```
struct pont {
    double x,y;
    string nev;
};

void kiir(double x, double y, string nev) {
    cout << "[" << x << "," << y << "," <<
nev << "]" << endl;
}

int main(){
   pont p;
   ...
   kiir(p.x, p.y, p.nev);
   return 0;
}
```

struct és függvények: utána

```
struct pont {
    double x,y;
    string nev;
};

void kiir(pont p) {
    cout << "[" << p.x << "," << p.y << "," <<
p.nev << "]" << endl;
}

int main(){
   pont p;
   ...
   kiir(p);
   return 0;
}
```

struct visszatérési típusként

```
struct koord {
    double x,y;
};

koord olvas(istream &be) {
    koord a;
    be >> a.x >> a.y;
    return a;
}

int main() {
    koord a;
    a = olvas(cin);
    cout << a.x << "," << a.y <<endl;
    return 0;
}
```

Intermezzo pesszimistáknak

Az elégséges jegyhez szükséges anyag immár teljes mértékben elhangzott*.

*a hirdetés nem minősül ajánlatnak

Inicializálás

- Figyelem: az értékadás és az inicializálás két különböző dolog, csak mindkettő jele a „=”
- int a=0; **inicializálás**: deklarálással együtt adunk kezdeti értéket
- int a; a=0; értékadás, a deklaráció után, bármikor máskor adunk új értéket.
- A struct esetében értéket adni mezőnként lehet, vagy másik struct-tal
- Inicializálni viszont lehet egyben is:
  koord k={0.0, 0.0}; // mezők sorrendjében!

struct inicializálás

```
struct pont {
    double x,y;
    string nev;
};

void kiir(pont p) {
    cout << "[" << p.x << "," << p.y << "," <<
p.nev << "]" << endl;
}

int main(){
   pont p = {1.0,1.0, "a"};
   kiir(p);
   return 0;
}
```

struct a structban inicializálás

```
struct ember {
    string nev;
    string lakcim;
    int szuletesi_ev;
};
struct diak {
    ember e;
    vector<int> jegyek;
};
int main()
{
   diak d = {{"Jani","Bp",1989},
           vector<int>(10,5)};
   cout << d.e.nev << " " << d.e.lakcim;
   return 0;
}
```

struct inicializálása

- Ez egy extra lehetőség, érdemes a használatát minimálisra csökkenteni
- Hátrányok:
  - ha változik a struct összetétele, mert bekerül egy új mező, akkor az összes inicializálás sérül, ki kell javítani
  - ha változik a mezők sorrendje, akár csendes hiba is keletkezhet, pl. összekeverhető a magasság a születési évvel, mert mindkettő int
- Ugyanakkor egyszerű esetekben hatékony