Objektum elvű alkalmazások fejlesztése

tárgyfelelősök: Gregorics Tibor
Sike Sándor

Tantárgy

- Cél: objektum elvű alkalmazások fejlesztéséhez szükséges kódolási készség kialakítása.
- Előfeltétel:
  - Programozás
- Eszköz: C++
- Kapcsolat:
  - Szoftvertechnológia
  - Algoritmusok és adatszerkezetek 1.
  - Programnyelvek C++
- Előadás (heti 1 óra)
  - Mintapéldák megoldása
  - Legszükségesebb háttérismeretek
- Konzultációs gyakorlat (heti 1 óra)
  - Házi feladatok bemutatása
  - Géptermi zárthelyi

Gyakorlati jegy

- Kötelező házi feladatok:
  - 4 darab feladat max. 5-5 pontért
  - határidőre (legfeljebb 4 hét késés, de szorgalmi időszakban)
- Géptermi zárthelyik:
  - félév közi és félév végi max 5-5 pontért
  - félév végi zárthelyi egyszer ismételhető
- Gyakorlati jegy: megszerzett pontok átlaga
  - zárthelyi duplán számítanak
  - ha az összes házi feladat legalább 1 pontos
  - ha a félév végi zárthelyi legalább 2 pontos

Háttér anyagok

- http://people.inf.elte.hu/gt/oaf
  - Előadások diái
  - Házi feladatok kitűzése
  - Egyéb segédanyagok
- Gregorics Tibor: Programozás (2.kötet) – Megvalósítás. ELTE-Eötvös Kiadó, 2013.

Objektum elvű alkalmazások fejlesztése

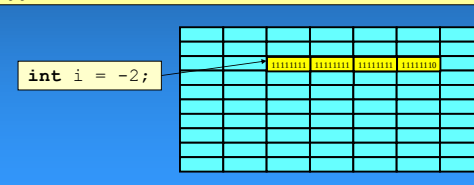
Memória kezelésről

Készítette: Gregorics Tibor

Memória

A memória bájtoknak (8 bites elemeknek) a sorozata.
A bájtokat megsorszámozzuk, ezek a sorszámok a memóriacímek.

Egy elemi adat értékének kódja néhány bájton helyezkedik el.
Az elemi adat címe, az értékét tartalmazó bájtsorozat első bájtjának sorszáma.



Program által használt memória szegmensek

Program szegmens: a futtatott program gépi kódját tartalmazza.

Adat szegmens: a program teljes futása alatt élő adatok (például statikus változók) értékeit tárolja.

Változóink eddig itt foglaltak helyett.

Verem (STACK) szegmens: a program futása során automatikusan létrejövő adatok (például egy programblokkba belépvén a blokk lokális változói) értékeit tárolja

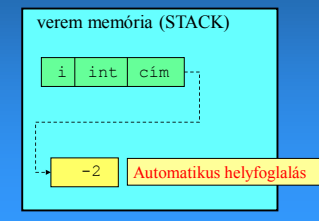
Dinamikus (HEAP) szegmens: a program futása során annak utasítására lefoglalt memória darabok

7

Változó

Egy változó az alábbi tulajdonságokkal rendelkezik:
- név
- típus
- érték
- memóriacím

```
int i;
i = -2;
```



8

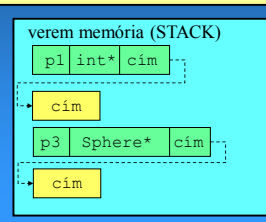
Pointer változó

A pointer egy adott típusú értéket tartalmazó memóriaterületnek a kezdőcímét tartalmazó változó.

Deklarálásakor meg kell adni, hogy ő egy pointer (*), és meg kell adni, hogy milyen típusú érték elhelyezésére szolgáló memóriaterület elejére mutat.

```
int *p1;
char *p2;
Sphere *p3;
```

```
class Sphere{
   Point c;
   double r;
public:
   ...
};
```

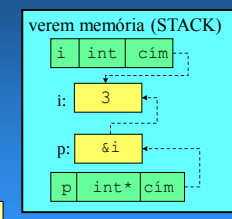


9

Pointer értéke, és a pointer által mutatott érték

```
int i;
int *p;
p = &i;
*p = 3;
```

indirekt hivatkozás



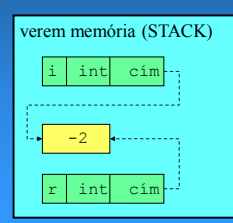
10

Referencia változó

```
int  i= -2;
int &r = i;
```

r:
i: -2

r az i „álneve”.
Az r-t nem elég csak deklarálni, azonnal értéket is kell kapnia.



11

Referencia szerinti paraméter átadás

```
void main()
{
      int x=1;
      int y=2;
      csere(x,y);
      cout << x << ”,” << y;
}
```

```
void csere(int &a, int &b)
{
      int s;
      s=a;
      a=b;
      b=s;
}
```

x: y:

a: b:

12

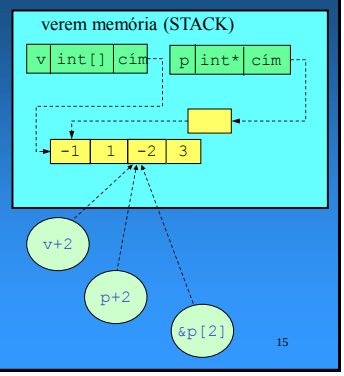
Automatikus helyfoglalású tömb és a pointer

```
int  v[4]={0,1,2,3};
```

```
v == &v[0]
*v == v[0]
*v = -1;
*(v+2) = -2;
```

```
int *p;
p = v;
```

```
*(p+2) == -2;
p[2]   == -2;
```



15

Automatikus helyfoglalású tömb a paraméter átadásban

```
const int max = 100;
int v[max];
int n;
read(v,n);
```

Csak kezdőcímet ad át

```
void read(int a[], int &n){
    cin >> n;
    for(int i = 0; i<n; ++i)
        cin >> a[i];
}
```

```
int n; cin >> n;
int v[n];
read(v,n);
```

```
void read(int a[], const int n){
    for(int i = 0; i<n; ++i)
        cin >> a[i];
}
```

Nem lehet olyan beolvasó függvényt készíteni, amely a tömb hosszának lefoglalását is elvégzi!

```
int v[4] = {0,1,2,3};
write(v,4);
```

```
void write(const int a[], const int n){
    for(int i = 0; i<n; ++i)
        cout << a[i];
}
```

Dinamikusan lefoglalt (név nélküli) változó

```
int *p;
```

verem memória (STACK)

p | int* | cím

cím

- név nélküli változó:
  - nincs neve
  - van típusa
  - van értéke
  - van helyfoglalása és annak címe

```
p = new int;
*p = -2;
delete p;
```

dinamikus memória (HEAP)

-2

Dinamikus helyfoglalás

Dinamikus felszabadítás

```
int *p;
*p = -5;
```

Szegmens hiba

20

Dinamikus helyfoglalású egydimenziós tömb

A * helye (nem kötött) utal arra, hogy ez egy tömb azonosítására létrehozott pointer

`int*` ~ `int` típusú értékekből álló véges sorozat

```
int* v;
```

verem memória (STACK)

v | int* | cím

cím

```
v = new int[4];
```

dinamikus memória (HEAP)

0 | 1 | 2 | 3

```
for (int i=0; i<4; ++i)
    v[i] = i;
```

```
delete[] v;
```

21

Dinamikus helyfoglalású tömb a paraméter átadásban

```
int* t, n;
create(t, n);
write(t, n);
delete[] t;
```

`int* t; int n;`

```
void create(int* &a, int &n)
{
    cin >> n;
    a = new int[n];
    for(int i=0; i<n; ++i)
        cin >> a[i];
}
```

```
void write(const int* a, const int n)
{
    for(int i = 0; i<n; ++i)
        cout << a[i] << "\t";
}
```

22

Dinamikus helyfoglalású mátrix

```
//Helyfoglalás
int** w;
w= new int*[3];
for(int i=0; i<3; ++i)
    w[i]= new int[4];

//Kezdeti értékadás
for(int i=0; i<3; ++i) {
    for(int j=0; j<4; ++j)
        w[i][j] = i*10 + j;
}

//Felszabadítása
for(int i=0; i<3; ++i)
    delete[] w[i];
delete[] w;
```

verem memória (STACK)

w | int** | cím

cím

dinamikus memória (HEAP)

cím | cím | cím

0 | 1 | 2 | 3

10 | 11 | 12 | 13

20 | 21 | 22 | 23

`int**` ~ `int` típusú értékekből álló véges kétdimenziós alakzat

23


VÉGE
25