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.

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

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 értékeit tárolja. (Egy programblokk változója a deklarálásakor foglal helyet, és a blokk elhagyásakor megszűnik.)

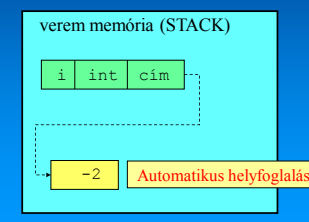
Dinamikus (HEAP) szegmens: a program futása során a programozó külön utasítására lefoglalt memória szeletek.

Megtanuljuk majd használni ezt is.

Változó

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

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

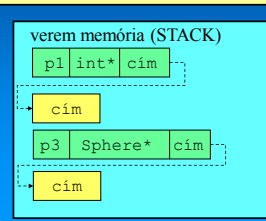


Pointer változó

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.

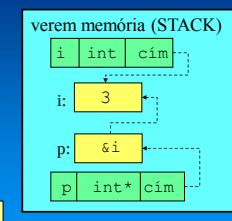
```
int *p1;
char *p2;
Pont *p3;
```



Pointer értéke, és a pointer által mutatott érték

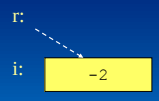
```
int i;
int *p;
p = &i;
*p = 3;
```

indirekt hivatkozás



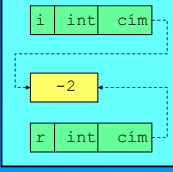
Referencia változó

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



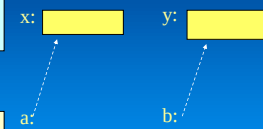
r az i „álneve”.
Az r-t nem elég csak deklarálni, azonnal értéket is kell kapnia.

verem memória (STACK)



Referencia szerinti paraméter átadás

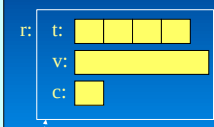
```
void main()
{
      int x=1;
      int y=2;
      csere(x,y);
      cout << x << "," << y;
}
```




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

Konstans referencia szerinti paraméter átadás

```
struct valami{
   int t[4];
   vector<int> v;
   char c;
};
void main()
{
      valami r;
      ...
      fv(r);
}
```




```
void fv(const valami &x)
{
      ...
}
```

x:

Automatikus helyfoglalású tömb és a pointer

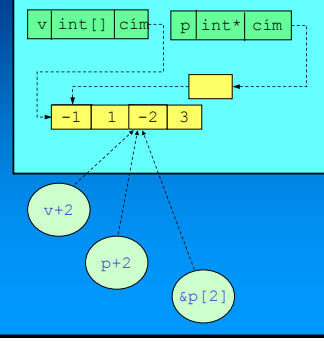
```
int  v[4]={0,1,2,3};
```

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

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

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

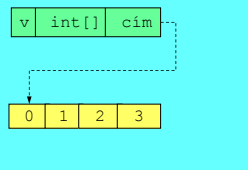
verem memória (STACK)



Pointer aritmetika

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

verem memória (STACK)




```
++p; --p; p++; p--;

p = p - 5; p -= 5;
```

sizeof(int)-tel változik a p-ben tárolt memóriacím

```
int *p1, *p2;
p1 = &v[0];
p2 = &v[4];
int d = p2 - p1;
```

4*sizeof(int)

Automatikus helyfoglalású tömb a paraméter átadásban

Ilyet szeretnénk:

```
int v[];
read(v);
```

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

- Automatikusan lefoglalt tömb memóriafoglalása felszabadul a foglalást végző blokkból való kilépéskor.
- Csak a tömb kezdőcímét adhatjuk át, a méretét nem.

Ilyet lehet:

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

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

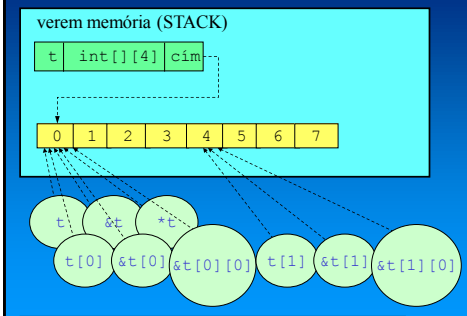
```
int n;
cin >> n;
int v[n];
read(v,n);
```

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

Csak kezdőcímet ad át, a méretet külön kell. A helyfoglalás és feltöltés szétválik

Automatikus helyfoglalású mátrix

```
int  t[2][4]={{0,1,2,3},{4,5,6,7}};
```




```
int *p[2] = {&t[0][0],&t[1][0]};
```

Automatikus helyfoglalású mátrix a paraméter átadásban

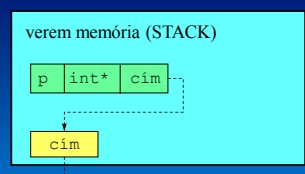
```
int max = 100;
int t[max][max];
int n, m;
read(t, n, m);
```

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

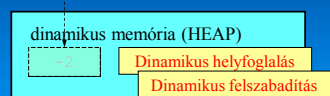
Csak az első dimenzió mérete hagyható el

Dinamikusan lefoglalt (név nélküli) változó

```
int *p;
```




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



Dinamikus helyfoglalás

Dinamikus felszabadítás

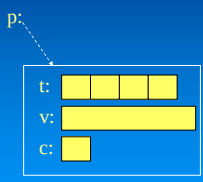
```
int *p;
*p = -5;
```

Szegmens hiba

- név nélküli változó:
  - nincs neve
  - van típusa
  - van értéke
  - van helyfoglalása és annak címe (ez most a p-ben van)

Dinamikusan lefoglalt struktúra

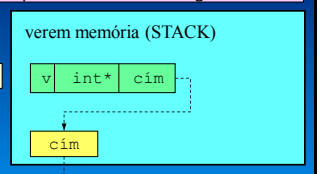
```
struct valami{
   int t[4];
   vector<int> v;
   char c;
};

void main()
{
      valami *p;
      ...
      p->t[3] = 3;      // *p.t[3]
      int i = p->v[0]
      p->c = 'a';
}
```



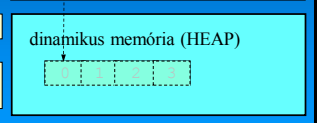
Dinamikus helyfoglalású egydimenziós tömb

A * helye (nem kötött) utal arra, hogy ez egy tömb azonosítására létrehozott pointer
`int*` ~ `int` típusú értékekből álló véges sorozat

```
int* v;
```




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




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

```
delete[] v;
```

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];
}
```

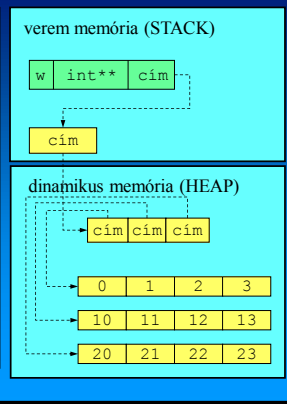
A méretet itt is külön kell hordozni, de a helyfoglalás és a feltöltés ugyanott történik

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

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;
```

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



Dinamikus helyfoglalású mátrix a paraméter átadásban

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

```
int** t, n, m;
create(t, n, m);
write(t, n, m);
free(t, n);
```

nincs **const**

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

```
void free(int** &a, const int n)
{
   for(int i = 0; i<n; ++i) delete[] a[i];
   delete[] a;
}
```