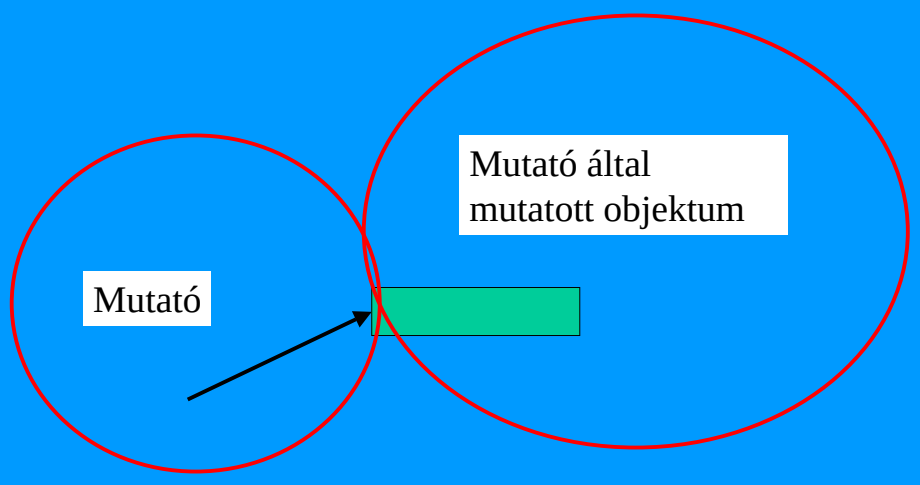
Mutatók

Készítette: Szabóné Nacsa Rozália

Mutatók



A deklaráció felépítése

```
char * szinek[] = { “piros”, “fehér”, “zöld”};
```

alaptípus

1

A deklaráció felépítése

```
char * szinek[] = { “piros”, “fehér”, “zöld”};
```

deklarátor

2

A deklaráció felépítése

```
char * szinek[] = { “piros”, “fehér”, “zöld”};
```

kezdőértéket adó kifejezés

3

A deklarátor felépítése

```
char * szinek [ ] = { “piros”, “fehér”, “zöld”};
```

előtag

utótag

*	mutató
*const	konstans mutató
&	referencia

[ ]	tömb
( )	függvény

Deklarátor operátorok

*** deklarátor**

(mutató)

```
char s[] = "Eper";
char* p=s;
```

1

& deklarátor

(referencia)

```
int i;
int& r=i;
```

2

*** operátor**

(dereferencia)

```
char* p1;
char* p2;
char c='A';
char d='D';
p2=&d;
p1=&c;
*p2=*p1;
```

3

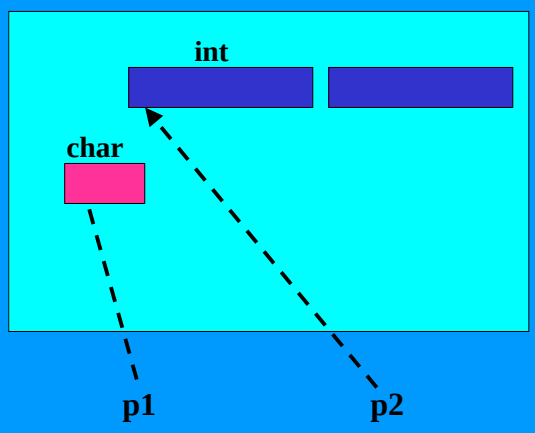
& operátor

(címe operátor)

```
char c='A';
char* p=&c;
```

4

Mutató deklarálása

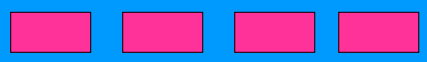



```
char* p1;
int* p2;
```

A mutatók deklarálásakor azt is meg kell mondani, hogy milyen típusú adat mutatója.

Mutató deklarálása.

char* p1;

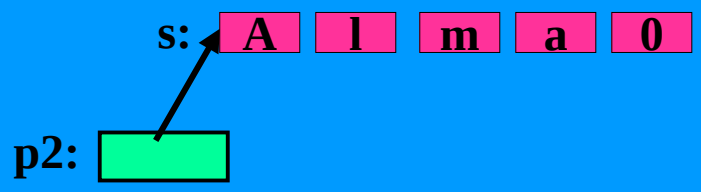


p1 →

p1 egy **mutató**
p1 egy **char típusra** mutató mutató.
p1 (egyenlőre) **nem mutat sehova**.

Mutató deklarálása

char s[]=„Alma”;
char* p2=s;



p2 egy **mutató**.
p2 egy **char típusra** mutató mutató.
p2 az s tömb elejére mutat

Több név bevezetése

A deklarátor operátorok felsorolásban mindig csak egyetlen tagra vonatkoznak.

```
int* p, y;          //int* p és int y
int x, *q;          //int x és int* q
int v[1],*pv;       //int v[1] és int* pv
```

Deklaráció és definíció

```
char* p1;
char* p2;
char c='A';
p1=&c;
*p2=*p1;
```

Futás idejű hiba!
P2 nem mutat sehova!

```
char* p1;
char* p2;
char c='A';
char d='D';
p2=&d;
p1=&c;
*p2=*p1;
```

Konstansra mutató mutató

```
char s[]=„Alma”;
const char* pc=s;
```

Konstans mutató

```
char s[]=„Alma”;
char *const cp=s;
```

Konstansra mutató konstans mutató

```
char s[]=„Alma”;
const char *const cpc=s;
```

```
//HIBA
const char c = 'A';
char* p=&c;
```

Konstans címet nem adhatunk értékül nem konstans mutatónak.

```
//JÓ
const char c = 'A';
const char *const p=&c;
```

Konstans címet értékül adhatunk konstans mutatónak.

```
//JÓ
char c = 'A';
const char *p=&c;
```

Változó címét értékül adhatjuk egy konstansra hivatkozó mutatónak

```
//HIBA
char* pc="Alma";
*pc='B';
```

Értékadás konstansnak. Az eredmény nem meghatározott.

Operátorok

* operátor (dereference operátor)

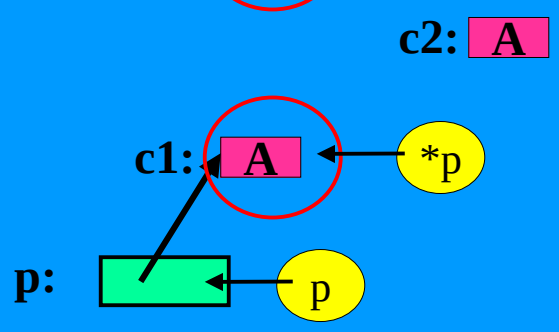
- **indirekciót jelez**
- **operandusa mutató**
- **eredménye: a mutató által megcímzett érték**

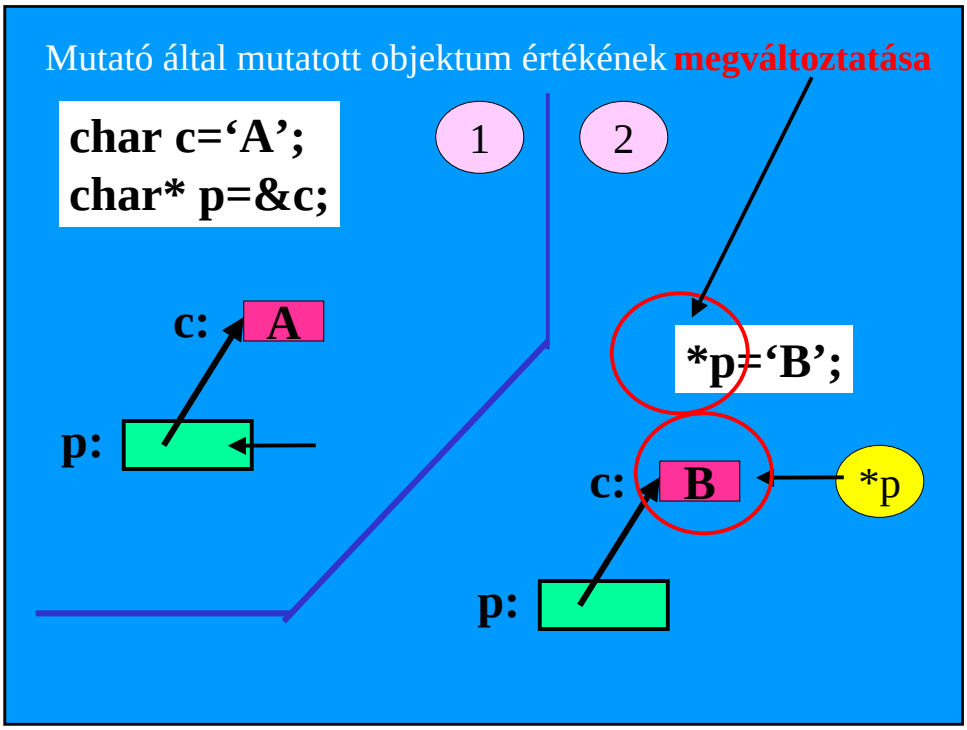
& operátor (címe operátor)

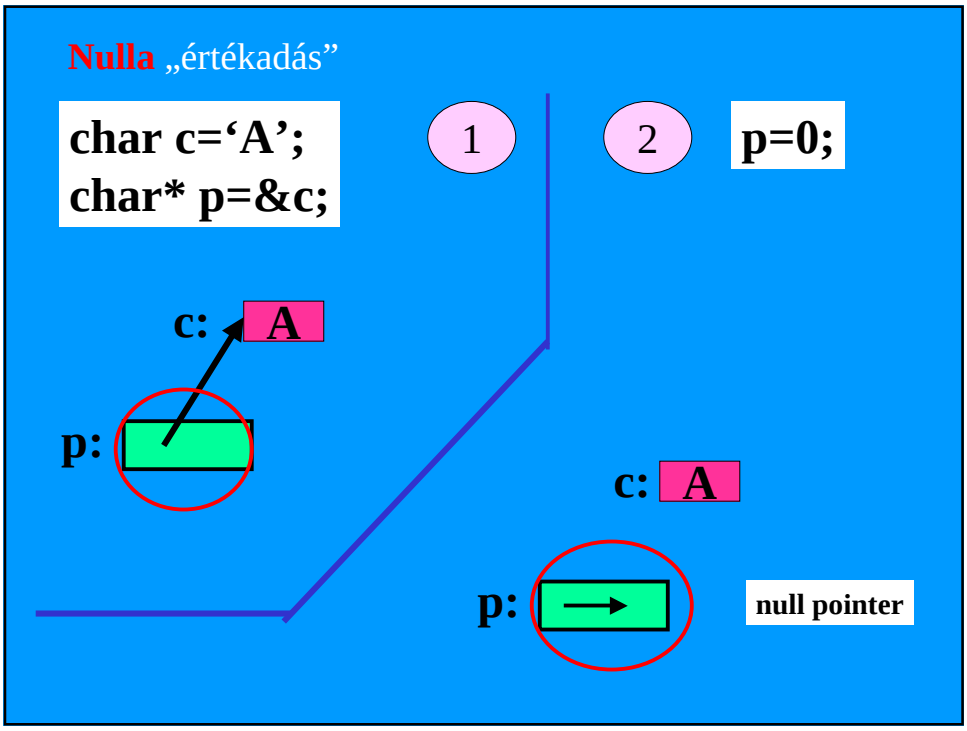
- **a * operátor inverze**
- **eredménye: az operandusa címe**

Hivatkozás a mutató által mutatott objektumra

```
char c1='A';
char* p=&c1;
char c2 = *p;
```

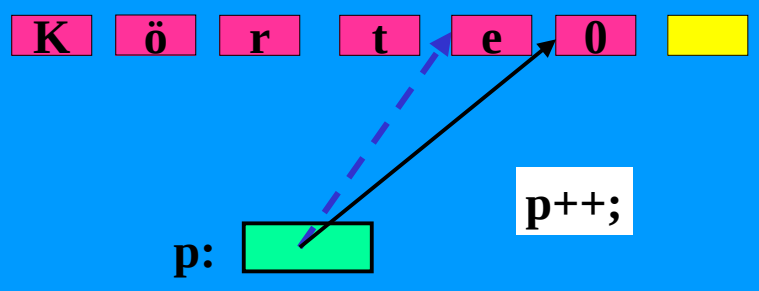



Mutató által mutatott objektum értékének megváltoztatása
char c='A';
char* p=&c;
1
2
c:
A
p:
*p='B';
c:
B
*p
p:


Nulla „értékadás"
char c='A';
char* p=&c;
1
2
p=0;
c:
A
p:
c:
A
p:
null pointer

Növelés

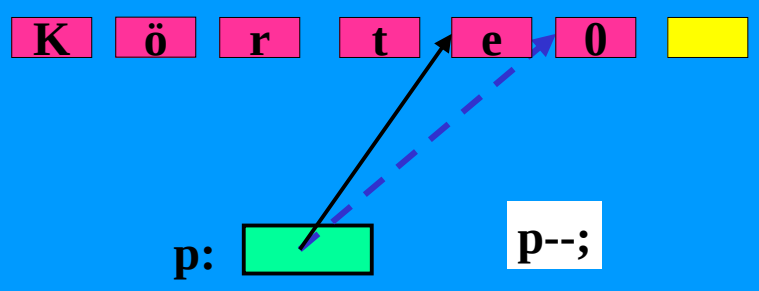
```
char v[6]=„Körte”;
char* p=&v[4];
. . .
```




```
p++;
```

Csökkentés

```
char v[6]=„Körte”;
char* p=&v[5];
. . .
```

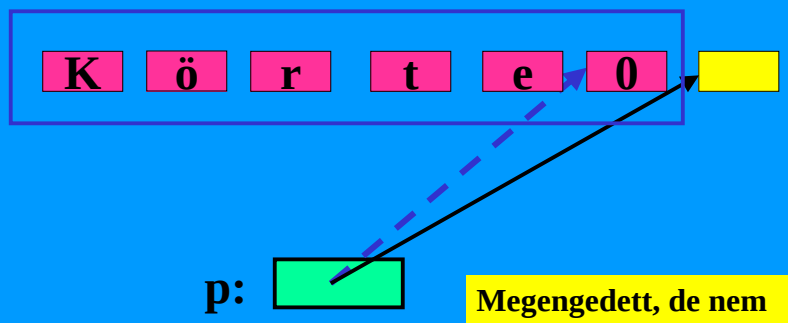



```
p--;
```

Növelés tömb végén

```
char v[6]=„Körte”;
char* p=&v[5];
. . .
```

```
p++;
```

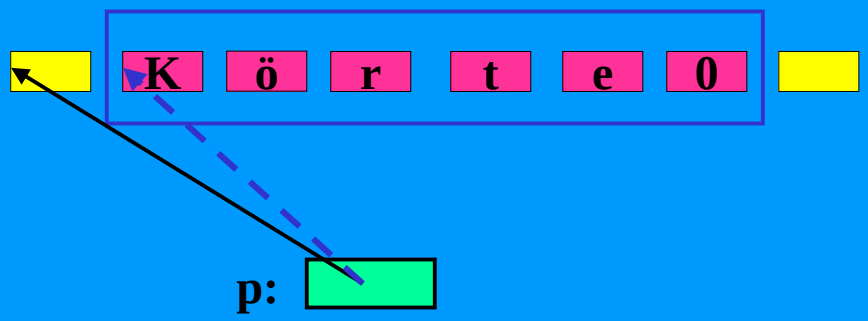


Megengedett, de nem hivatkozhatunk az általa mutatott objektumra.

Csökkentés tömb elején

```
char v[6]=„Körte”;
char* p=&v[0];
. . .
```

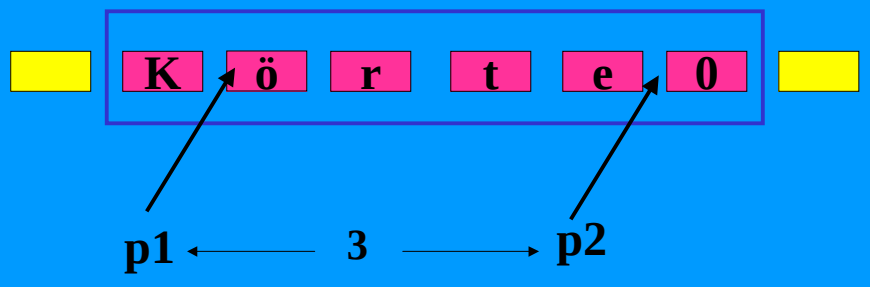
```
p--;
```



Nem megengedett, mert értéke nem definiált.

```
char v[6]=”Körte”;
char* p2=&v[5];
char* p1=&v[2];
int i = p2-p1;          //i=3
```

Kivonás



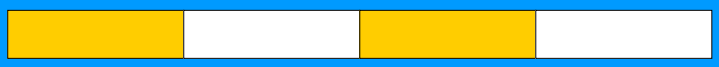
A mutatóknak ugyanarra a tömbre kell mutatni.

Kapcsolat mutatók és tömbök között

char v[4];

Deklarálunk egy négy elemű vektort, melynek elemei char típusúak.

v[0] v[1] v[2] v[3]

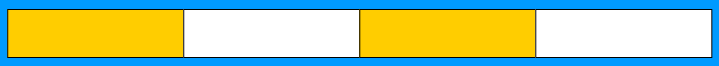


char v[4];

char *p;

A p egy olyan pointer, amellyel rámutathatunk valamilyen char típusú objektumra.

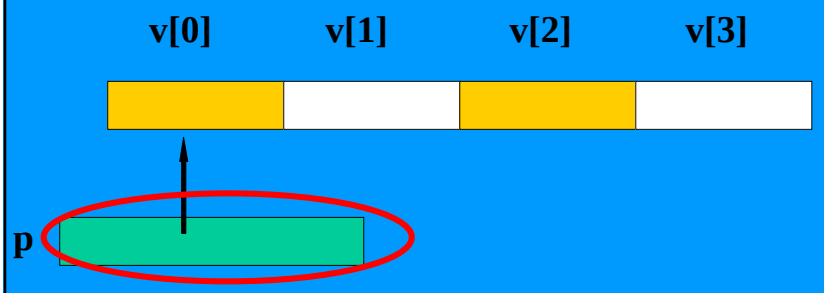
v[0] v[1] v[2] v[3]



p

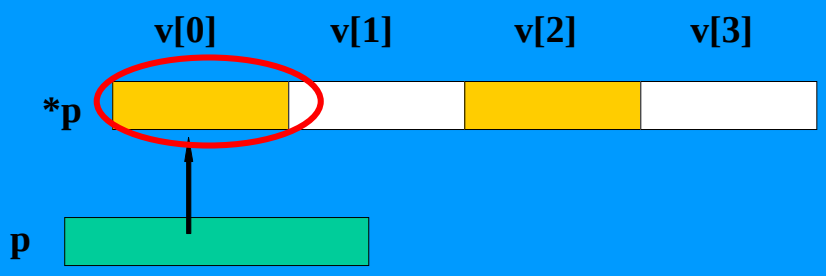
```
char v[4];
char *p;
p=&v[0];
```

Az értékadás elvégzése után p a vektor első elemére mutat.




```
char v[4];
char *p;
p=&v[0];
```

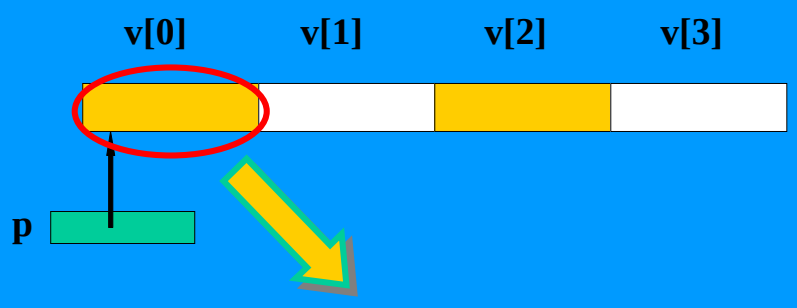
*p: a p pointerrel mutatott char típusú objektum értéke.



Mutató és vektor kapcsolata: p=&v[0];

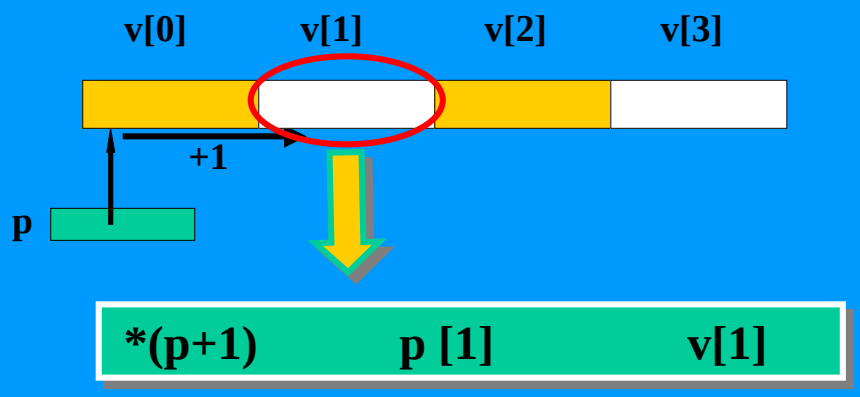
p + i	⟷	&v[i]
*(p + i)	⟷	v[i]
p	⟷	&v[0]

```
char v[4];
char *p;
p=&v[0];
```

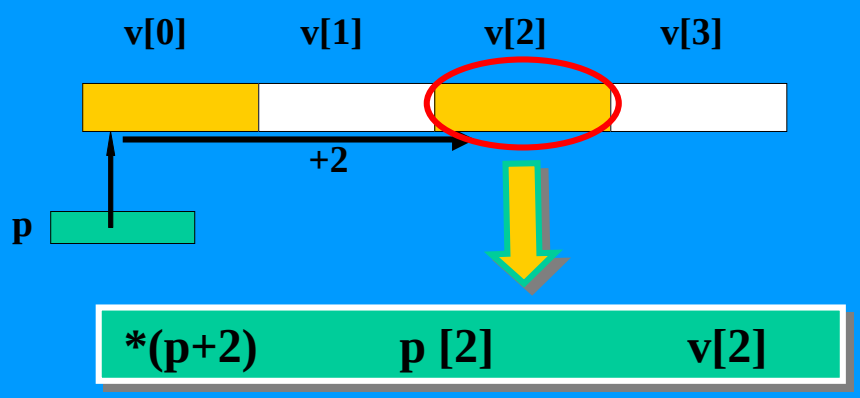


*(p+0) p [0] v[0]

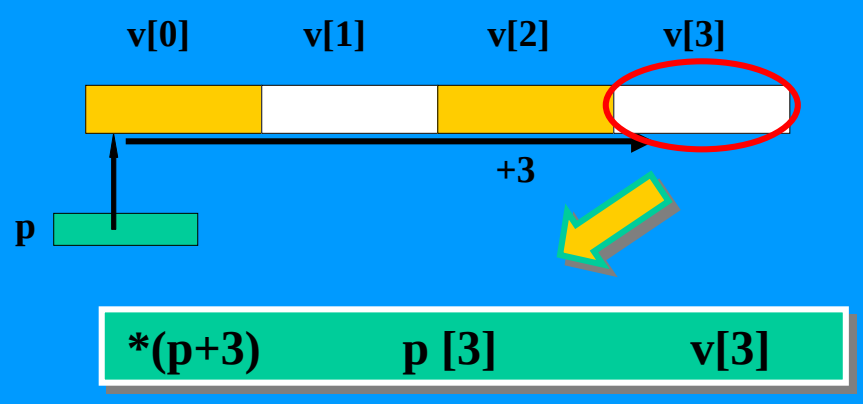
```
char v[4], *p=&v[0];
```




```
char v[4];
char *p=v;
```

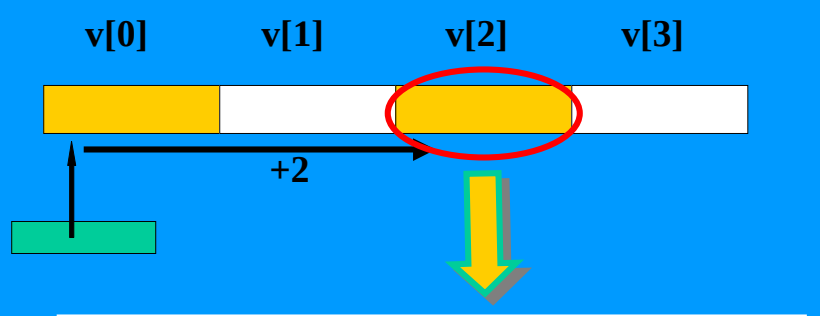


```
char v[4], *p=v;
```



*(p+3) p [3] v[3]

```
Vector3D v[4];
Vector3D* p=v;
```



*(p+2) p [2] v[2]

Tagra hivatkozó mutató: ->

```
class Vector3D . . .
public:
. . .
double abs(void) const;
. . .
```

```
. . .
Vector3D* p=&Vector3D(1,2,3);
double d1,d2;
d1=(*p).abs();
d2=p->abs();
cout << "d1: " << d1 << " d2:" << d2 << endl;
. . .
```

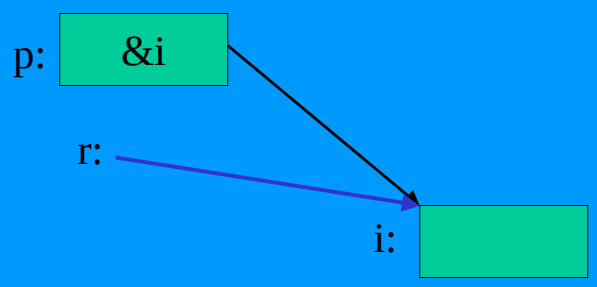
Referencia típus

X&

referencia X-re

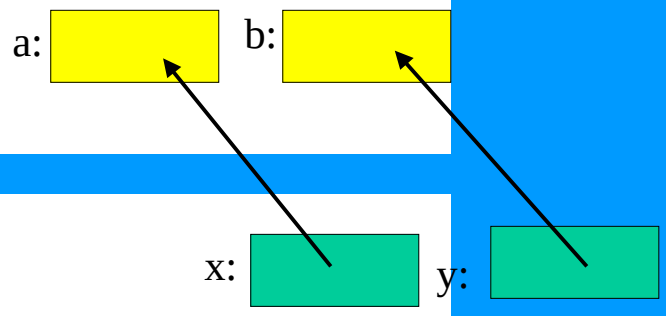
```
int i=0;
int& r = i;
r++;
int*p=&r;
```

r az i „álneve”.
r-et inicializálni kell.



Érték szerinti paraméterátadás

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

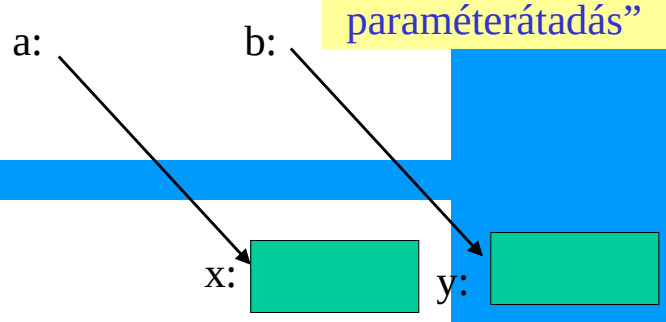



```
void main (){
int x=1;
int y=2;
csere(x,y);
cout << x << "," << y;
}
```


Referencia, mint függvényparaméter

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

„Cím szerinti paraméterátadás”




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

Feladat:

Egy szöveges állományban megadunk térvektorokat.
Keressük meg a leghosszabbat.

Maximumkeresés

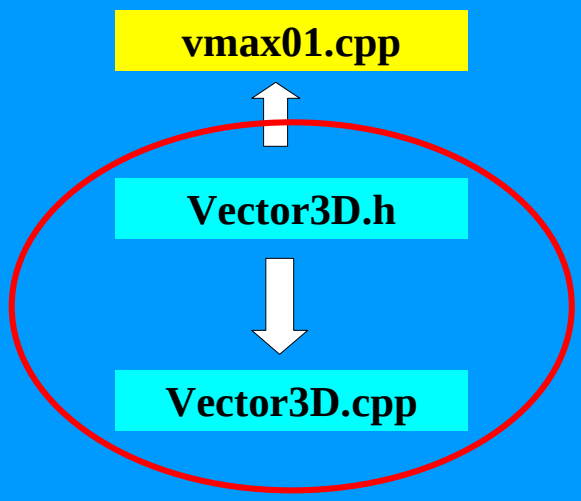
Modulszerkezet

vmax01.cpp

Vector3D.h

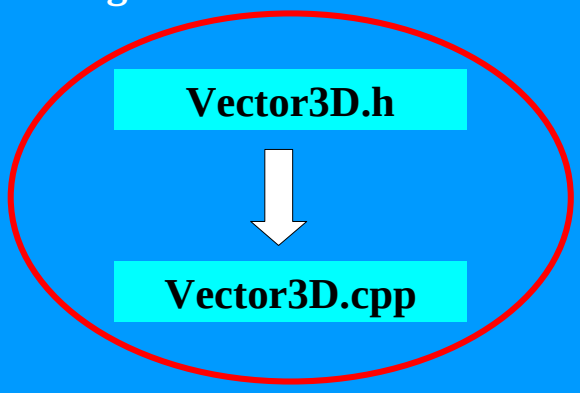
Vector3D.cpp

Modulszerkezet



Ezt a részt már elkészítettük az előző félévben.

Kiegészítés



1. >> művelet
2. default konstruktor

Vector3D.h

(kiegészítés)

1

>> művelet

default konstruktor

2

```
class Vector3D {
   friend double smul(const Vector3D& larg,
                      const Vector3D& rarg);
   friend Vector3D operator+(const Vector3D& larg,
                             const Vector3D& rarg);
   friend Vector3D operator-(const Vector3D& larg,
                             const Vector3D& rarg);
   friend Vector3D operator*(const Vector3D& larg,
                             const Vector3D& rarg);
   friend bool operator==(const Vector3D& larg,
                          const Vector3D& rarg);
   friend bool operator!=(const Vector3D& larg,
                          const Vector3D& rarg);
   friend ostream& operator<<(ostream& s, const Vector3D& v);
   friend istream& operator>>(istream& s, Vector3D& v);
public:
   Vector3D(double x, double y, double z);
   Vector3D();
   double abs(void) const;
   static const Vector3D NULLVECT;
private:
   double _x;
   double _y;
   double _z;
};
```

Vector3D.cpp

(kiegészítés)

>> művelet

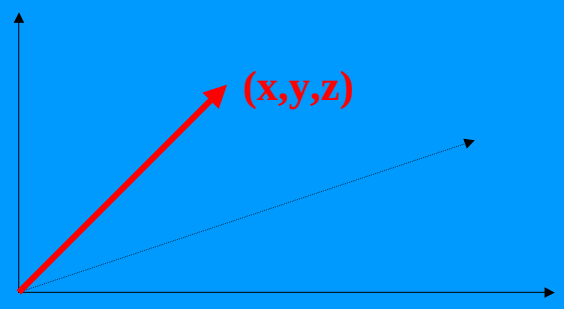
```
istream& operator >>(istream& s, Vector3D& v)
{
   char separator;
   s >> separator  >> v._x;
   s >> separator >> v._y;
   s >> separator >> v._z  >> separator;
   return s;
}
```

1

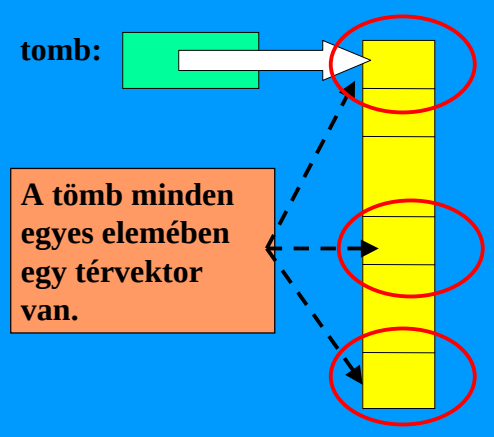
default konstruktor

```
Vector3D::Vector3D()
{
Vector3D(0,0,0);
}
```

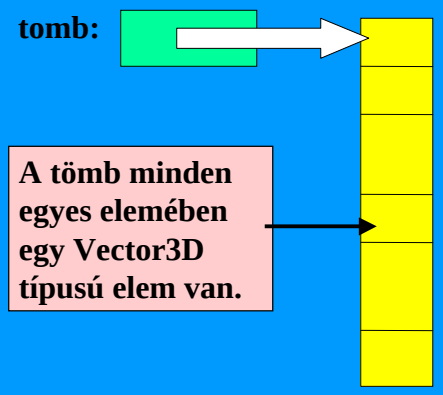
2



Első változat:
A tömb elemei térvektorok.

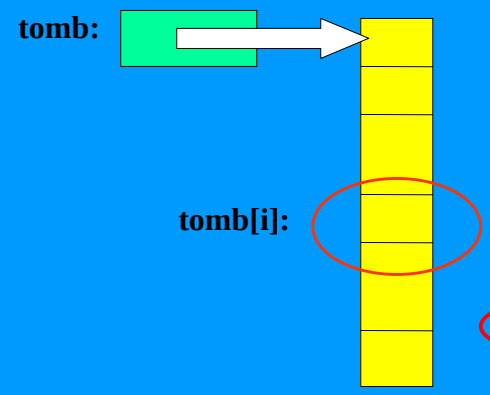


Tárterület lefoglalása



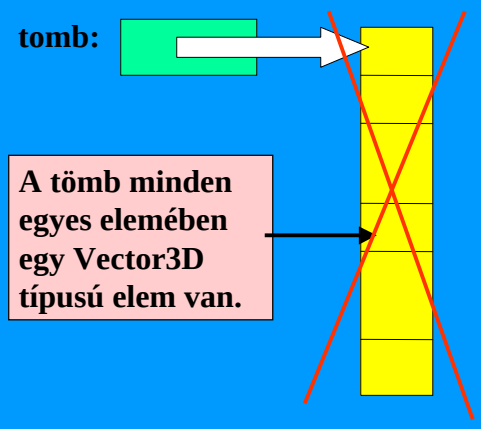

```
//Foglalás
Vector3D  *tomb;
tomb= new Vector3D[n];
. . .
```

Hivatkozás egy elemre




```
//Foglalás
Vector3D  *tomb;
tomb = new Vector3D[n];
. . .
//Használat
. . . tomb[i] . . .
```

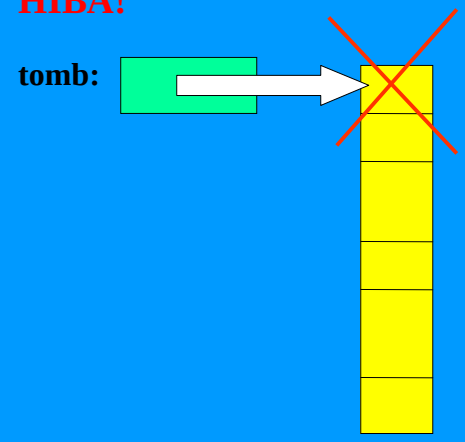
Tárterület felszabadítása



A tömb minden egyes elemében egy Vector3D típusú elem van.

```
//Foglalás
Vector3D  *tomb;
tomb = new Vector3D[n];
. . .
//Használat
. . . tomb[i] . . .
. . .
//Felszabadítás
delete[] tomb;
```

HIBA!




```
//Foglalás
Vector3D  *tomb;
tomb = new Vector3D[n];
. . .
//Használat
. . . tomb[i] . . .
. . .
//Felszabadítás
delete tomb;
```

!

Ha nem tesszük ki az indexelő operátort, akkor csak a tömb első elemét szabadítjuk fel. A többi elem tovább foglalja a helyet a tárban.

```
//Adatok előkészítése és megjelenítése
  char barmi;
  ifstream inp;
  string InpFileName;
  cout << "Kerem adja meg a fajl nevet: ";
  cin >> InpFileName;
  inp.open(InpFileName.c_str());
  if(inp.fail()){
     cerr << "A megadott fajlt nem talalom! \n";
     cin >> barmi;
     return 1;
  }
```

Folytatás

Ez a rész ugyanaz, mint a max02b-ben.

```
//Elemszám beolvasása és ellenőrzése
   int n; inp >> n;
  cout << endl <<"Elemek szama: " << n << endl;
  if(n<1)
 {
     cout << "Nincs elem" << endl;
     cin >> barmi;
     return 2;
  }
```

Folytatás

Ez a rész ugyanaz, mint a max02b-ben.

```
//Dinamikus tárterület lefoglalása és a tömb feltöltése
  Vector3D *tomb;
  tomb=new Vector3D[n];
  for (int j=0; j!=n; j++)
 {
    inp >> tomb[j];
  }
```

```
//Fájl bezárása
  inp.close();
```

```
//A tömb elemeinek kiírása
  cout << endl << "Az elemek: " << endl << endl;
  for (int j=0; j!=n; j++){
    cout << tomb[j] << endl;
  }
```

Folytatás

Ez a rész ugyanaz, mint a max02b-ben.

```
//Maximumkeresés
 int k,i;
 Vector3D max;
 k=0; i=0;
 max=tomb[0];
 while(i!=(n-1)) {
   if (tomb[i+1].abs() >= max.abs()) {
     k=i+1;
     max=tomb[i+1];
   }
   i=i+1;
 }
```

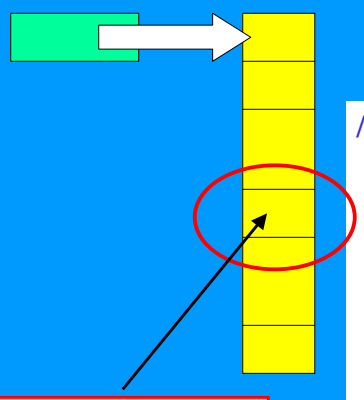
```
//Eredmény megjelenítése
  cout << endl << "A leghosszabb tervektor: " << max << "."
<< endl;
  cout << "A tervektor hossza: " << max.abs() << endl;
  cout << "Ez pedig a tomb " << (k+1) << ". eleme. " << endl;
  cin >> barmi;
```

```
//Dinamikusan lefoglalt tárterület felszabadítása
  delete[] tomb;
  return 0;
}
```

Maximumkeresés - A

tomb[i+1]

tomb:



max:

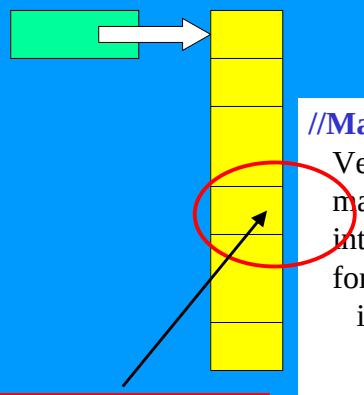
A tömb (i+1)-dik eleme.

```
//Maximumkeresés
  int k,i;
  Vector3D max;
  k=0; i=0;
  max=tomb[0];
  while(i!=(n-1)) {
    if (tomb[i+1].abs() >= max.abs()) {
      k=i+1;
      max=tomb[i+1];
    }
    i=i+1;
  }
```

Maximumkeresés - B

***(tomb+i+1)**

tomb:

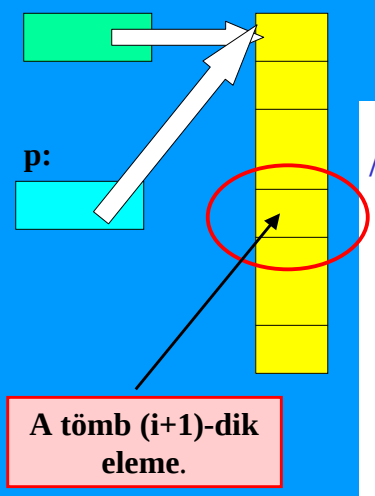


max:

A tömb (i+1)-dik eleme.

```
//Maximumkeresés
  Vector3D max;
  max=tomb[0];
  int k;
  for (int i=0; i!=n-1; i++){
    if ((*(tomb+i+1)).abs() >= max.abs()) {
        max=*(tomb+i+1);
        k=i+1;
    }
  }
```

Maximumkeresés - C

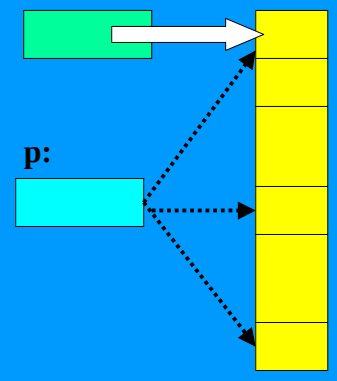


***(p+i+1)**

max:

```
 Vector3D *p;
//Maximumkeresés
  Vector3D max;
  max=tomb[0];
  p=tomb; // vagy: p=&tomb[0]
  int k;
  for (int i=0; i!=n-1; i++){
    if ((*(p+i+1)).abs() >= max.abs()) {
          max=*(p+i+1);
          k=i+1;
    }
  }
```

Maximumkeresés - D



***(p+1)**

max:

```
 Vector3D *p;
//Maximumkeresés
  Vector3D max;
  max=tomb[0];
  int k;
  p=tomb;
  for (int i=0; i!=n-1; i++, p++){
    if ((*(p+1)).abs() >= max.abs()) {
          max=*(p+1);
          k=i+1;
    }
  }
```

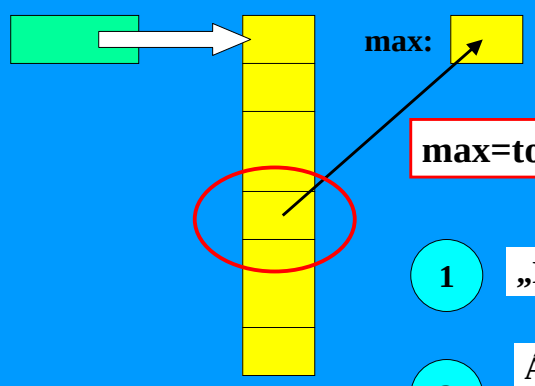
Tagra hivatkozó mutatók

```
 Vector3D *p;
//Maximumkeresés
 Vector3D max;
  max=tomb[0];
  int k;
  p=tomb;
  for (int i=0; i!=n-1; i++, p++){
    if ((p+1)->abs() >= max->abs()) {
        max=*(p+1);
        k=i+1;
    }
  }
```

```
 Vector3D *p;
//Maximumkeresés
 Vector3D max;
  max=tomb[0];
  int k;
  p=tomb;
  for (int i=0; i!=n-1; i++, p++){
    if ((*(p+1)).abs() >= max.abs()) {
        max=*(p+1);
        k=i+1;
    }
  }
```

Az első változat gyengéi

tomb:

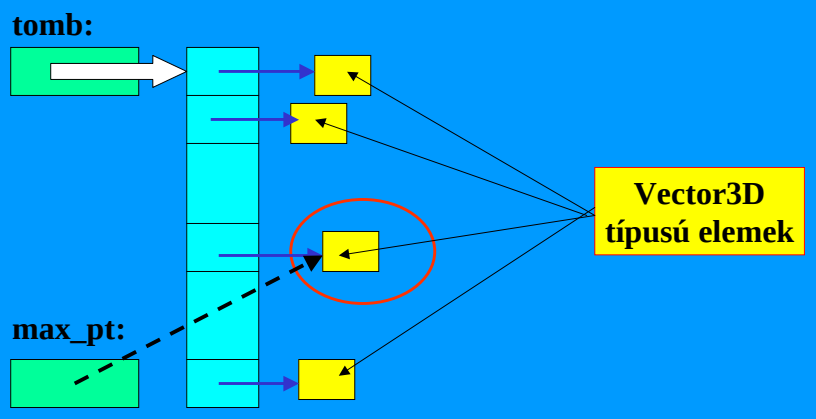


max=tomb[i+1];

1 **„Drága” művelet**

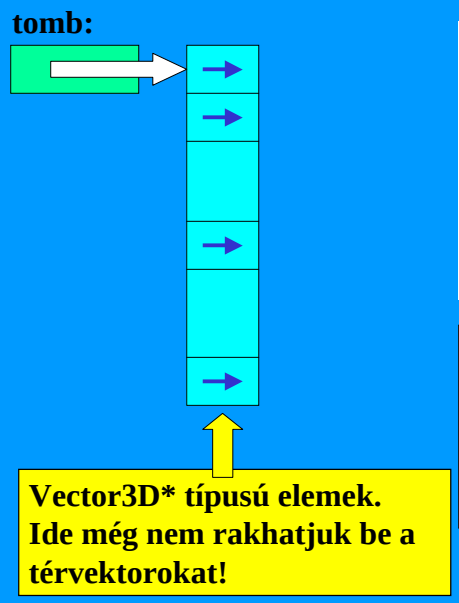
2 **Át kell gondolni, jól működik-e az értékadás operátor.**

Második változat:
A tömb elemei térvektorokra mutató pointerek



A vektor minden egyes elemében egy olyan mutató van, amely Vector3D típusú elemre mutat.

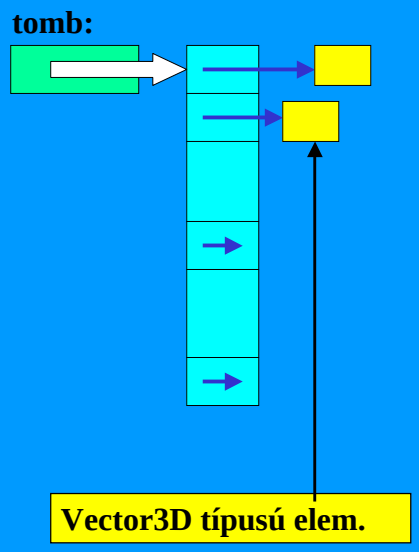
Tárterület lefoglalása - 1




```
//Foglalás
Vector3D  **tomb;
tomb= new Vector3D*[n];
. . .
```

Lefoglalunk egy n elemű, mutatókat tartalmazó tömböt.
(A mutatók térvektorokra mutathatnak)

Tárterület lefoglalása - 2/a

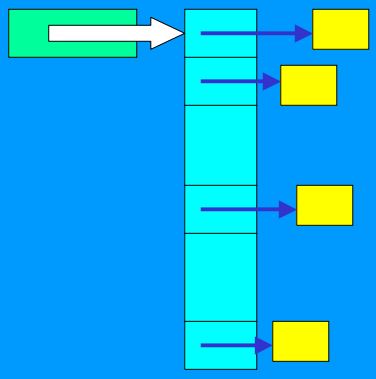



```
//Foglalás
Vector3D **tomb;
tomb= new Vector3D*[n];
tomb[1]=new Vector3D();
tomb[2]=new Vector3D();
. . .
```

A szabad tárból helyet foglalunk aVector3D típusú elemek számára.
A tömb elemei mutassanak ide.

Tárterület lefoglalása - 2/b

tomb:

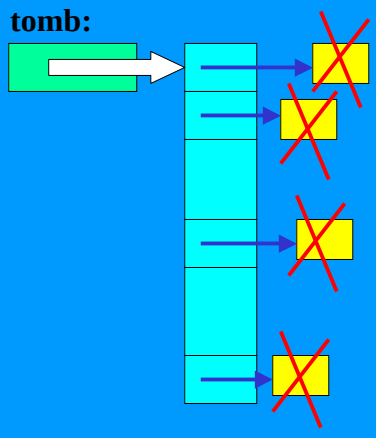


```
//Foglalás
Vector3D **tomb;
tomb= new Vector3D*[n];
for (int i=0; i!=n; i++) {
    tomb[i]= new Vector3D();
}
. . .
```

Ciklussal

Tárterület felszabadítása -1

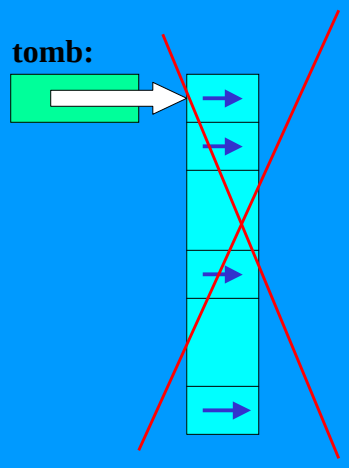
tomb:



```
//Felszabadítás
for (int i=0; i!=n; i++) {
    delete tomb[i];
}
. . .
```

Felszabadítjuk a térvektoroknak lefoglalt helyet.

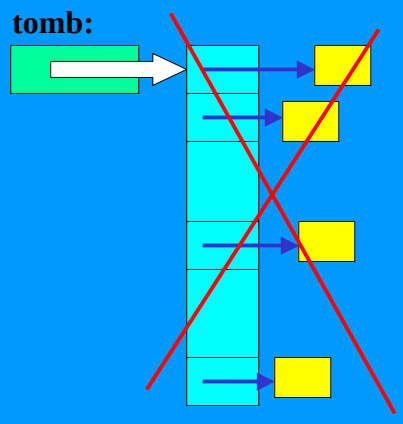
Tárterület felszabadítása -2




```
//Felszabadítás
for (int i=0; i!=n; i++) {
    delete tomb[i];
}
delete[] tomb;
. . .
```

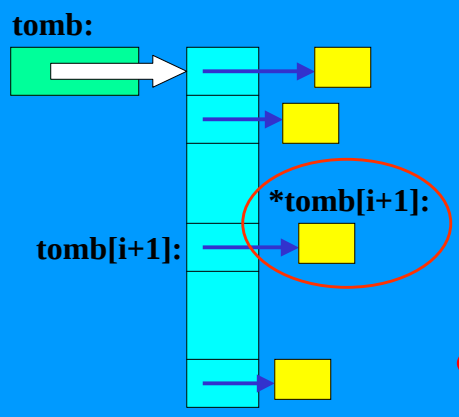
Majd felszabadítjuk a térvektorokra mutató pointereket tartalmazó tömböt.

Tárterület felszabadítása -összefoglalás



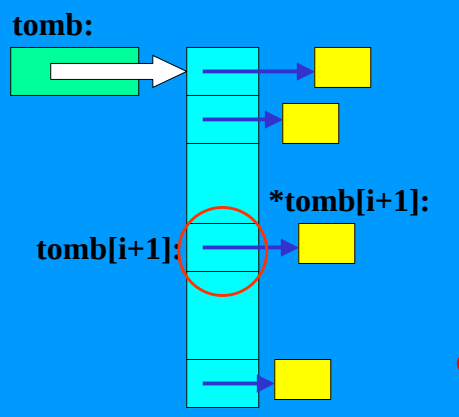

```
//Felszabadítás
for (int i=0; i!=n; i++) {
    delete tomb[i];
}
delete[] tomb;

. . .
```

Hivatkozás az (i+1)-dik térvektorra



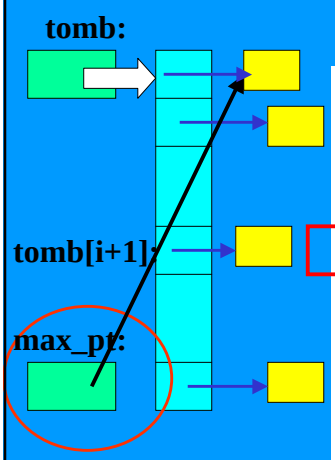

```
//Foglalás

Vector3D*  *tomb;

tomb= new Vector3D*[n];
for (int i=0; i!=n; i++) {
     tomb[i]= new Vector3D();
}

. . .*tomb[i+1] . . .
```

Hivatkozás az (i+1)- dik tömbelemre



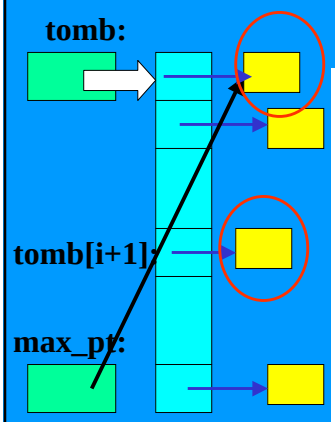

```
//Foglalás

Vector3D*  *tomb;

tomb= new Vector3D*[n];
for (int i=0; i!=n; i++) {
     tomb[i]= new Vector3D();
}

. . .tomb[i+1] . . .
```

Maximumkeresés - 1



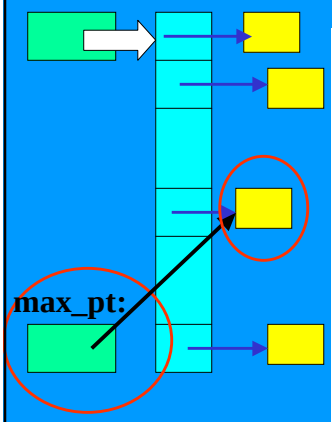

```
//Maximumkeresés
int k,i;
Vector3D *max_pt;
k=0; i=0;
max_pt=tomb[0];
while(i!=(n-1)) {
   if ((*tomb[i+1]).abs() >= (*max_pt).abs())
   {
       k=i+1;
       max_pt=tomb[i+1];
   }
    i=i+1;
}
```

Maximumkeresés -2



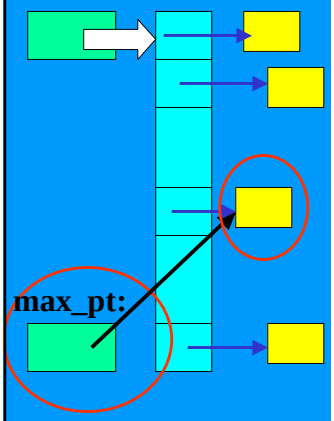

```
//Maximumkeresés
int k,i;
Vector3D *max_pt;
k=0; i=0;
max_pt=tomb[0];
while(i!=(n-1)) {
   if ((*tomb[i+1]).abs() >= (*max_pt).abs())
   {
       k=i+1;
       max_pt=tomb[i+1];
   }
    i=i+1;
}
```

Maximumkeresés -3




```
//Maximumkeresés
int k,i;
Vector3D *max_pt;
k=0; i=0;
max_pt=tomb[0];
while(i!=(n-1)) {
   if ((*tomb[i+1]).abs() >= (*max_pt).abs())
   {
      k=i+1;
      max_pt=tomb[i+1];
   }
    i=i+1;
}
```

Maximumkeresés pointerrel




```
//Maximumkeresés
  int k,i;
  Vector3D* max_pt;
  Vector3D** p;
  k=0; i=0;
  max_pt=tomb[0];
  p=tomb;
  while(i!=(n-1)) {
    if ((*(*p+1)).abs() >= (*max_pt).abs()) {
      k=i+1;
      max_pt=*(p+1);
    }
    i=i++;
    p++;
  }
```

A két változat összehasonlítása

```
//Dinamikus tárterület lefoglalása és a tömb feltöltése
  Vector3D *tomb;
  tomb=new Vector3D[n];
  for (int j=0; j!=n; j++)
 {
    inp >> tomb[j];
  }
```

1

```
//Dinamikus tárterület lefoglalása és a tömb feltöltése
  Vector3D **tomb;
  tomb=new Vector3D*[n];
  for (int i=0; i!=n; i++)
 {
    tomb[i]= new Vector3D();
   inp  >> *tomb[i];
 }
```

2

```
//A tömb elemeinek kiirása
  cout << endl << "Az elemek: " << endl << endl;
  for (int j=0; j!=n; j++){
    cout << tomb[j] << endl;
  }
```

1

```
//A tömb elemeinek kiirása
   cout << endl << "A tomb elemei: " << endl;
  for (int i=0; i!=n; i++){
    cout << *tomb[i] << ", ";
  }
  cout << endl;
```

2

```
//Maximumkeresés
 int k,i;
 Vector3D max;
 k=0; i=0;
 max=tomb[0];
 while(i!=(n-1)) {
   if (tomb[i+1].abs() >= max.abs()) {
     k=i+1;
     max=tomb[i+1];
   }
   i=i+1;
 }
```

1

```
//Maximumkeresés
 int k,i;
 Vector3D* max_pt;
 k=0; i=0;
 max_pt=tomb[0];
 while(i!=(n-1)) {
   if ((*tomb[i+1]).abs() >= (*max_pt).abs()) {
     k=i+1;
     max_pt=tomb[i+1];
   }
   i=i+1;
 }
```

2

1

```
//Eredmény megjelenítése
  cout << endl << "A leghosszabb tervektor: " << max << "." << endl;
  cout << "A tervektor hossza: " << max.abs() << endl;
  cout << "Ez pedig a tomb " << (k+1) << ". eleme. " << endl;
  cin >> barmi;
```

2

```
//Eredmény megjelenítése
  cout << endl << "A leghosszabb tervektor: " << *max_pt << "." << endl;
  cout << "A tervektor hossza: " << (*max_pt).abs() << endl;
  cout << "Ez pedig a tomb " << (k+1) << ". eleme. " << endl;
  cin >> barmi;
```

```
//Dinamikusan lefoglalt tárterület felszabadítása
  delete[] tomb;
```

1

```
//Dinamikusan lefoglalt tárterület felszabadítása
  for (int i=0; i!=n; i++)
  {
    delete tomb[i];
  }
  delete[] tomb;}
```

2

Vége