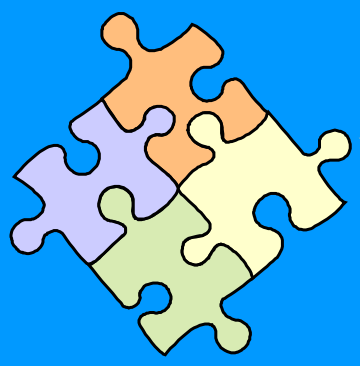
Elemi alkalmazások fejlesztése II.



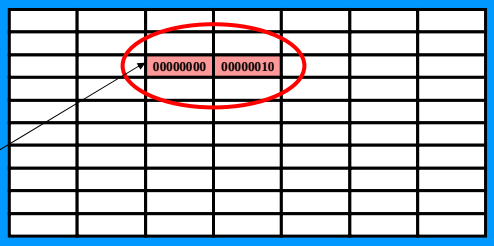
Mutatók

int i = 2;

típus **név** **érték**

A programban definiált változóink a memóriában helyezkednek el

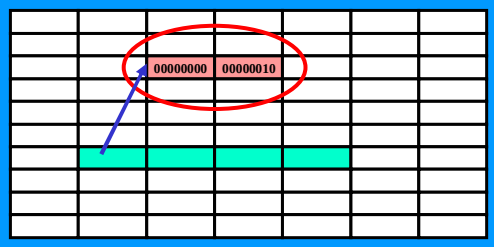
Memória



változó címe:
Az a hely, ahol a változónk a memóriában megtalálható.

Memória

int i = 2;



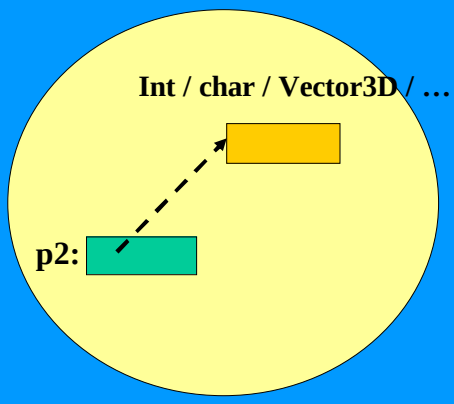
Azokat a változókat, amelyekben ilyen címeket tárolunk **mutatóknak** nevezzük.

Ezeket a változókat ugyanúgy, mint bármely más változót **deklarálni** és **definiálni** kell.

Mutató deklarálása - példa

```
char *p1;
int *p2;
Vector3D *p3;
```

A mutatók deklarálásakor meg kell mondani, hogy milyen típusú adatra mutatunk vele.



Mutatóval kapcsolatos operátorok

* operátor (dereference operátor)

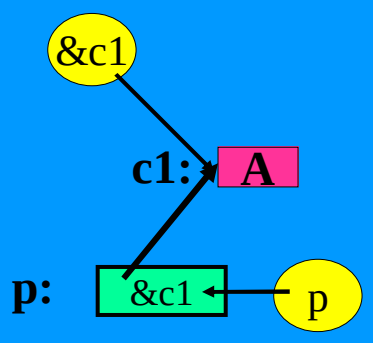
- **indirekciót jelez**
- **operandusa mutató**
- **eredménye: a mutató által megcímzett érték**

& operátor (címe operátor)

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

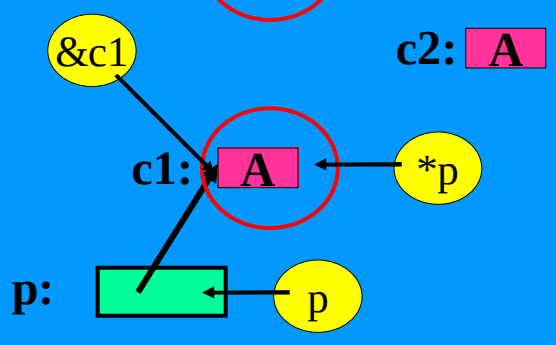
Objektum címe

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



Mutató által mutatott objektum

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



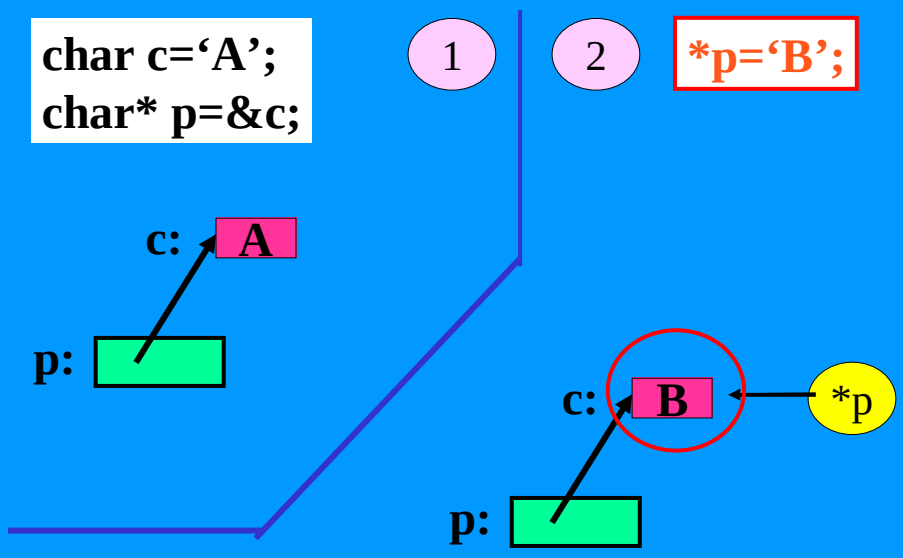
Mutató által mutatott objektum értékének megváltoztatása

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

1

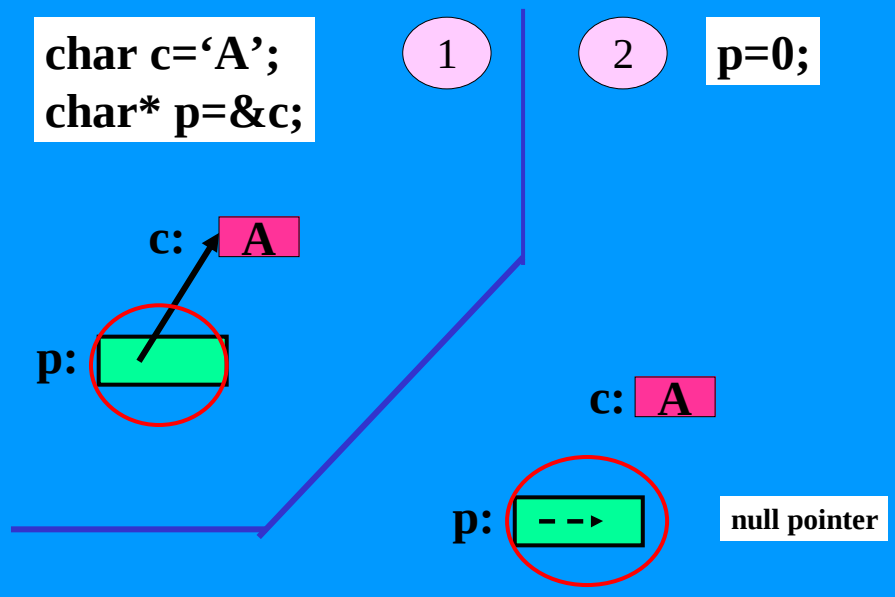
2

```
*p='B';
```



Nulla „értékadás”

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



Hol legyen a csillag?

A * jelet hol a típushoz, hol a változóhoz „ragasztjuk”. Ennek a fordító számára nincs jelentősége, de adott esetben támogathatja a program olvashatóságát.

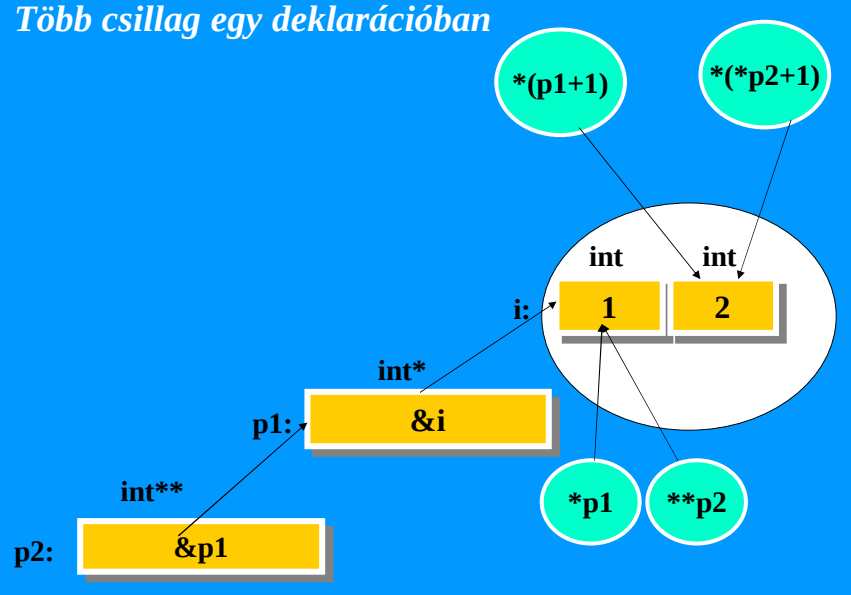
```
int *v;
```

Egészeket tartalmazó dinamikus tömb deklarációja.

```
int* p;
```

Egészekre mutató pointer.

Több csillag egy deklarációban



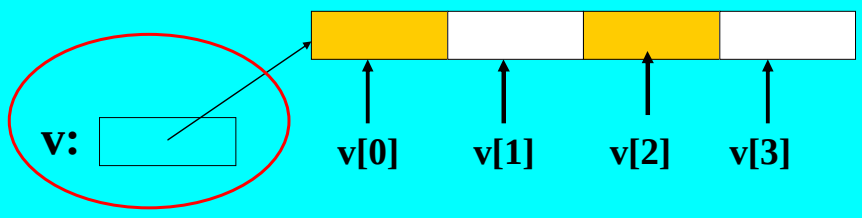
Mutatók és tömbök kapcsolata

Dinamikus helyfoglalás tömb számára - ISMÉTLÉS

1 int *v; — Deklaráció

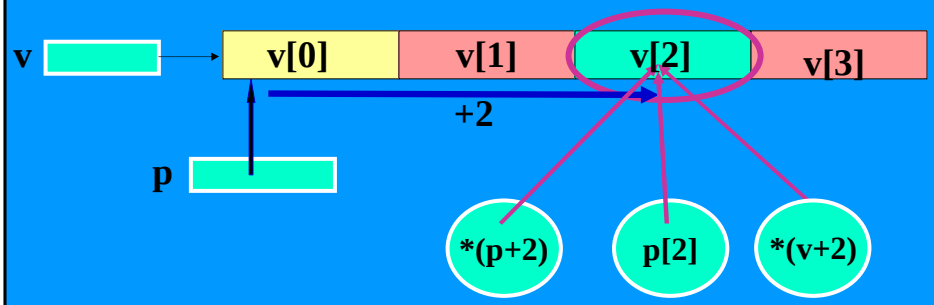
2 v =new int[4]; — Definició

3 delete[ ] v; — A tárterület felszabadítása.



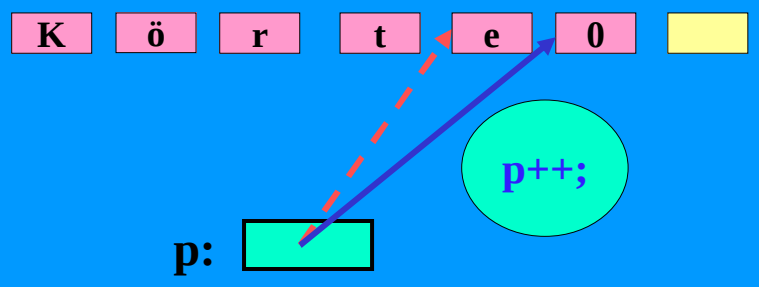

```
T     v[4];
T*    p;
p=v;
```

Mutatók és tömbök kapcsolata



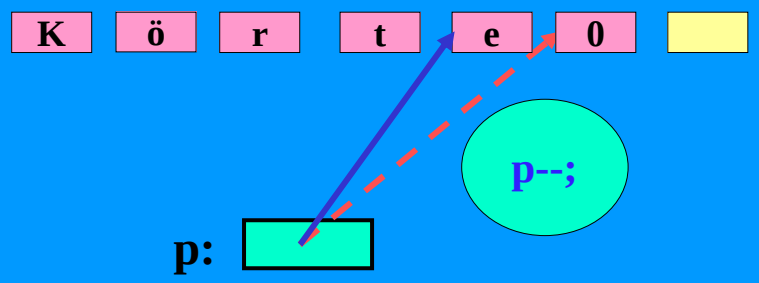
Növelés

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



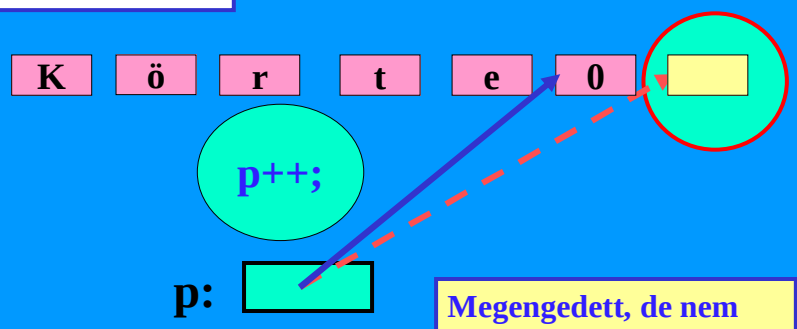
Csökkentés

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



Növelés tömb végén

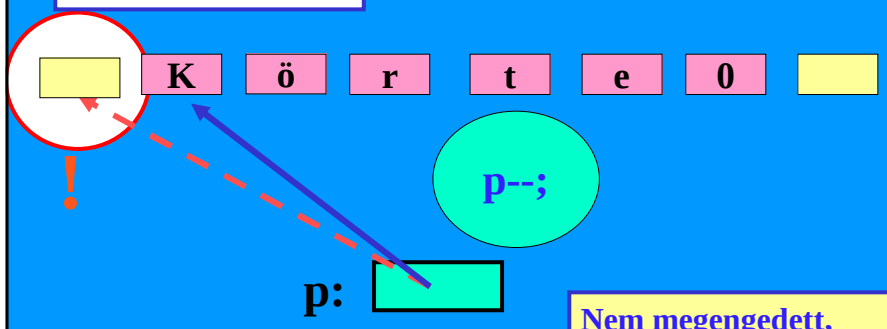
```
char v[6]=„Körte”;
char* p=&v[4];
. . .
```



Megengedett, de nem hivatkozhatunk az általa mutatott objektumra.

Csökkentés tömb elején

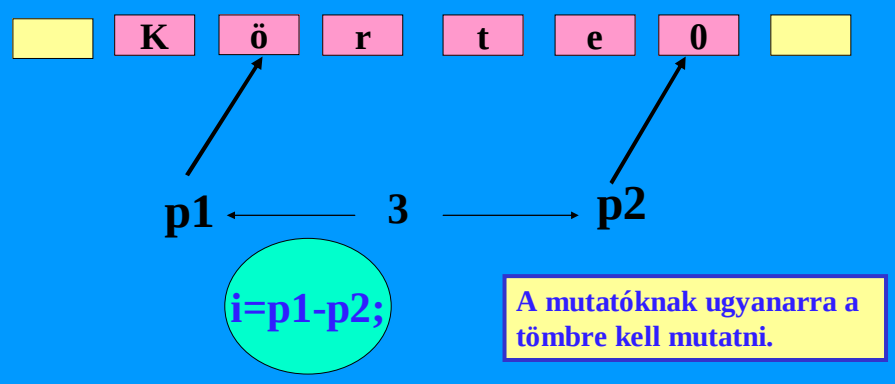
```
char v[6]=„Körte”;
char* p=&v[4];
. . .
```



Nem megengedett, mert értéke nem definiált.

Kivonás

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



A mutatóknak ugyanarra a tömbre kell mutatni.

Hivatkozás tagra, metódusra

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

A Vector3D osztály

A Vector3D osztály egy példánya

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

Hivatkozás tagra, metódusra - 2

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

p egy olyan mutató, amely a Vector3D(1,2,3) példányra mutat.

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

Hivatkozás tagra, metódusra - 3

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

(*p).abs()
vagy
p->abs()

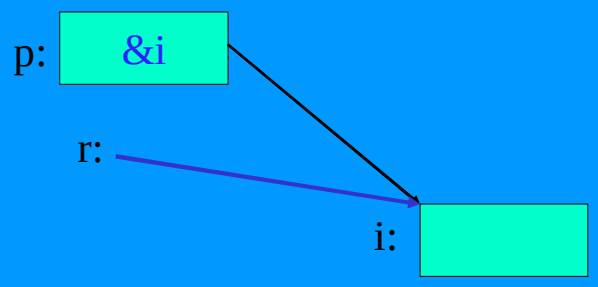
Ha meg szeretnénk tudni a vektor hosszát erre a konkrét példányra, akkor erre a példányra meg kell hívni az abs metódust.

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

+1 : Referencia típus: T&

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

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

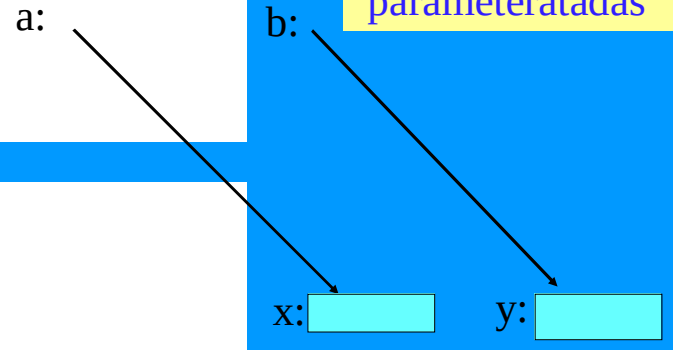


Referencia, mint függvényparaméter

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

„Cím szerinti paraméterátadás”

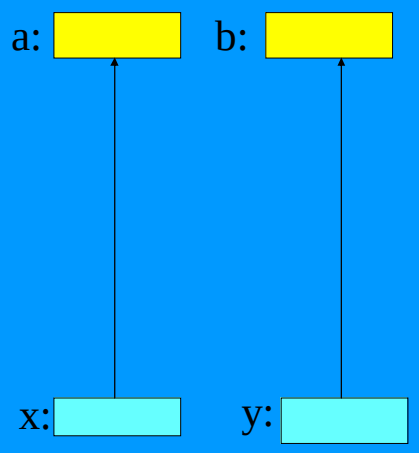
```
void main (){
int x=1;
int y=2;
csere(x,y);
cout << x << “,” << y;
}
```



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;
}
```



* és & lehetséges előfordulási helyei - összefoglalás

Deklarátorok

*** (mutató)**

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

& (referencia)

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

Operátorok

*** (dereferencia)**

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

& (címe operátor)

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

Címszerinti paraméterátadás

```
void csere (int& a, int& b){
. . .
}
```

Feladat

Egy szöveges állományban megadunk **térvektorokat.**
Keressük meg a leghosszabbat.

Maximumkeresés

Modulszerkezet

vmax01.cpp

Vector3D.h

Vector3D.cpp

Vector3D osztály: ld. Elemi alkalmazások I.

Kiegészítés

1. **>> művelet**

   A Vector3D típusú elemek beolvasásához

2. **default konstruktor**

   A Vector3D típusú elemeket tartalmazó tömbhöz.

A Vector3D.h kiegészítése

Vector3D.h

```
class Vector3D {
   . . . .
   friend Vector3D 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;
};
```

>> művelet → **1**

default konstruktor → **2**

Implementációs döntés

1 Tömb elemei térvektorok

tomb:

A tömb minden egyes elemében egy-egy térvektor van.

max:

2 A tömb elemei térvektorokra mutató pointerek

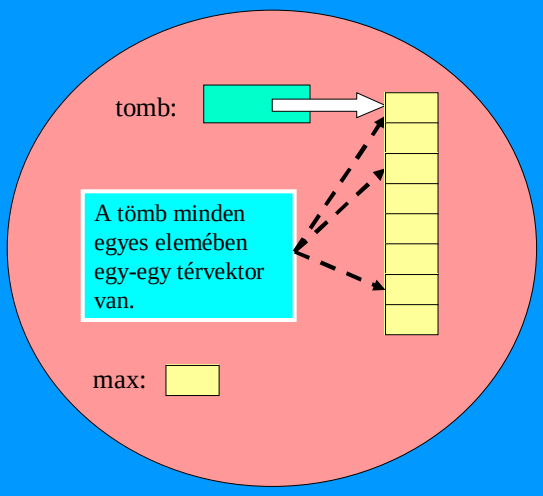
tomb:

A tömb minden egyes elemében egy-egy térvektorra mutató pointer van.

max:

Megoldás – első változat

A tömb elemei térvektorok.




```
//Foglalás
Vector3D  *tomb;          1
tomb = new Vector3D[n];
. . .
//Használat
. . . tomb[i] . . .       2
. . .
//Felszabadítás
delete[] tomb;            3
```

Adatok előkészítése és megjelenítése

```
//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

```
//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

```
//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, a tömb elemeinek kiírása

```
//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

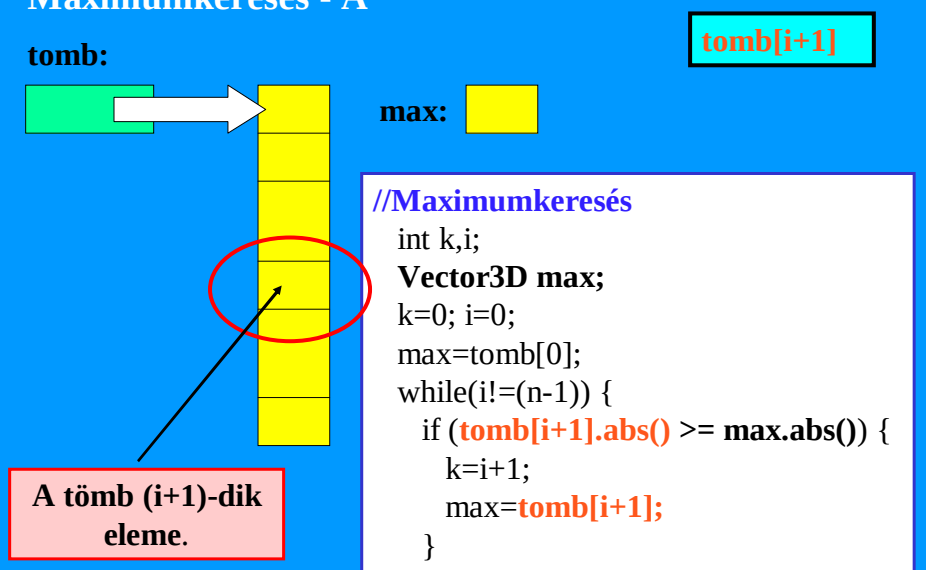
```
//Maximumkeresés
 int k,i;
 Vector3D max;
 k=0; i=0;
 max=tomb[0];
 while(i!=(n-1)) {
   if (tomb[i+1].abs() >= max.abs()) {
     k=i+1;
     max=tomb[i+1];
   }
   i=i+1;
 }
```

Eredmény megjelenítése, dinamikusan lefoglalt tárterület felszabadítása

```
//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




```
//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)

tomb:

max:

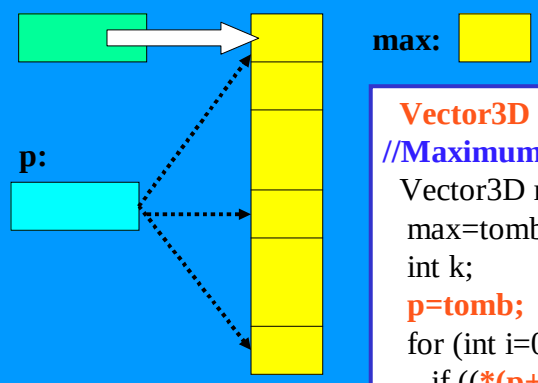
p:

A tömb (i+1)-dik eleme.

```
  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)



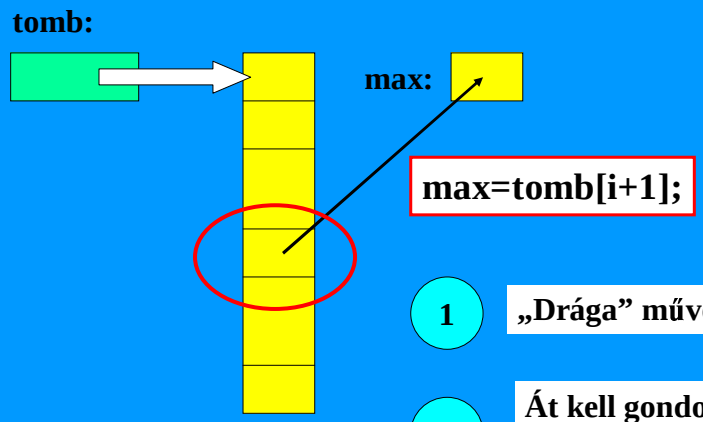

```
 Vector3D *p;
//Maximumkeresés
 Vector3D max;
  max=tomb[0];
  int k;
  p=tomb;
  for (int i=0; i!=n-1; i++, p++){
    if ((*(p+1)).abs() >= max.abs()) {
        max=*(p+1);
        k=i+1;
   }
  }
```

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

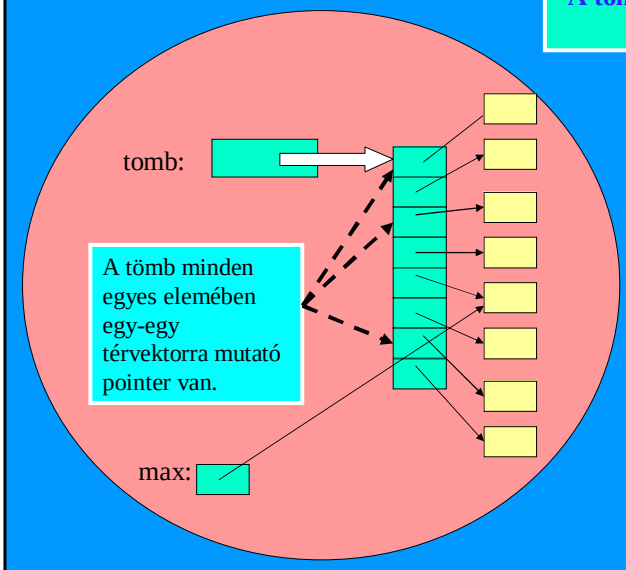


max=tomb[i+1];

1. **„Drága” művelet**
2. **Át kell gondolni, jól működik-e az értékadás operátor.**

Megoldás – második változat

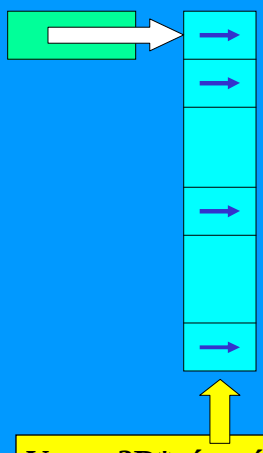
A tömb elemei térvektorokra mutató pointerek.



A tömb minden egyes elemében egy-egy térvektorra mutató pointer van.

Tárterület lefoglalása - 1

tomb:



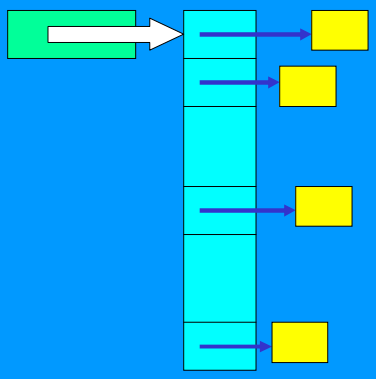
```
//Foglalás
Vector3D  **tomb;
tomb= new Vector3D*[n];
. . .
```

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

Vector3D* típusú elemek.
Ide még nem rakhatjuk be a térvektorokat!

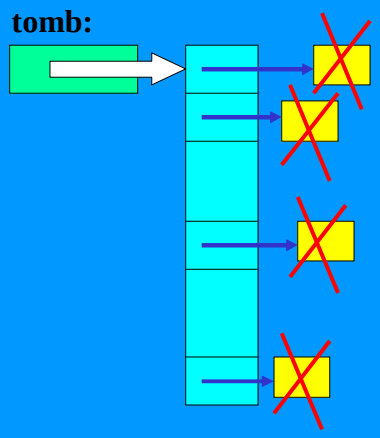
Tárterület lefoglalása - 2

tomb:



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

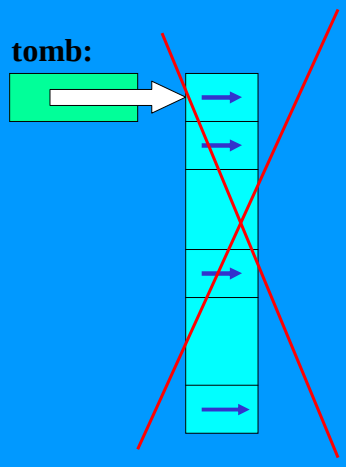
Tárterület felszabadítása




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

Felszabadítjuk a térvektoroknak lefoglalt helyet.

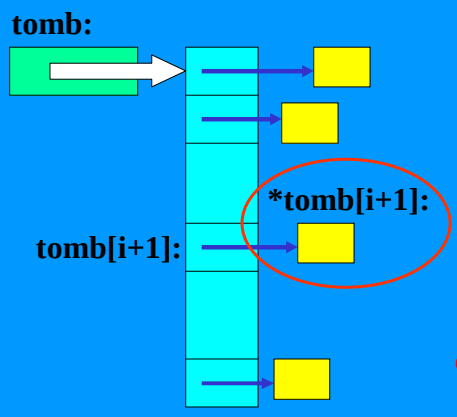
Tárterület felszabadítása -2




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

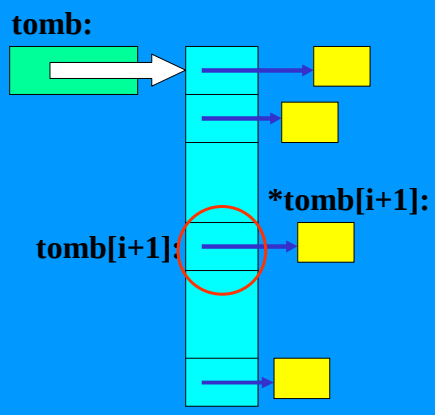
Majd felszabadítjuk a térvektorokra mutató pointereket tartalmazó tömböt.

Hivatkozás az (i+1)-dik térvektorra



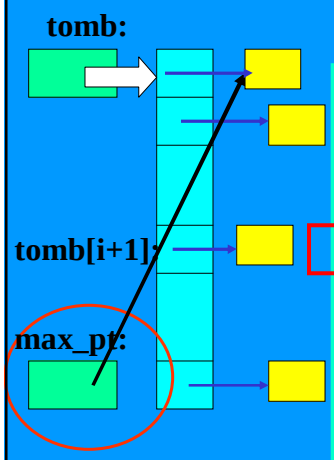

```
//Foglalás

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



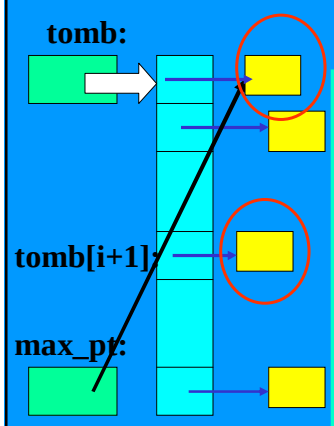

```
//Foglalás

Vector3D*  *tomb;

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

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

Maximumkeresés - 1



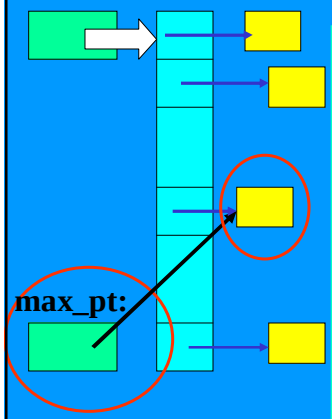

```
//Maximumkeresés
int k,i;
Vector3D *max_pt;
k=0; i=0;
max_pt=tomb[0];
while(i!=(n-1)) {
    if ((*tomb[i+1]).abs() >= (*max_pt).abs())
    {
        k=i+1;
        max_pt=tomb[i+1];
    }
    i=i+1;
}
```

Maximumkeresés -2



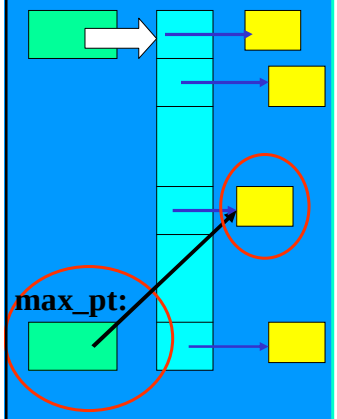

```
//Maximumkeresés
int k,i;
Vector3D *max_pt;
k=0; i=0;
max_pt=tomb[0];
while(i!=(n-1)) {
    if ((*tomb[i+1]).abs() >= (*max_pt).abs())
    {
        k=i+1;
        max_pt=tomb[i+1];
    }
    i=i+1;
}
```

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;
```

Vége