




* operátor – példák		 :
char c1='A'; char* p=&c; char c2 = *p;	char c='A'; char* p=&c; *p='B';
char c='A'; char* p=&c; p=0;	char t[6]; char* p2=&t[5]; char* p1=&t[2]; int I = p2-p1;
T v[6]=„Körte”; T* p=&v[4]; ... p++;	T v[6]=„Körte”; T* p=&v[5]; ... p--;

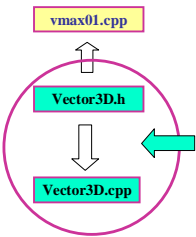


* deklarátor	& deklarátor
char s[] = "Eper"; char* p=s;	int i; int& r=i;
* operátor	& operátor
char* p1; char* p2; char c='A'; p1=&c; *p2=*p1;	char c='A'; char* p=&c;

Mutatók és tömbök kapcsolata	
<pre>char v[4]; char *p; p=v;</pre>	<p>v[2] = *(p+2) = p[2] = *(v+2)</p>

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();	 :
Referencia		
int i=0; int& r = i; r++; int*p=&r;		 :
Referencia, mint paraméter		
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; }

Feladat:

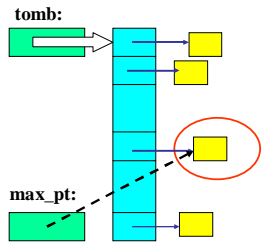
Egy szöveges állományban megadunk térvektorokat. Keressük meg a leghosszabbat.

<p>Modulszerkezet</p> 	<p> A <code>Vector3D</code> osztályt az előző félévben már elkészítettük. Nekiünk csak a <code>vmax01.cpp</code> programot kell megírni.</p>
<p>Vector3D kiegészítése</p> <p>Egészítsük ki a <code>Vector3D</code> osztályunkat a <code>>></code> operátorral és a default konstruktorral.</p> <p>Vector3D.h</p> <pre>friend istream& operator>>(istream& s, Vector3D& v); Vector3D();</pre> <p>Vector3D.cpp</p> <pre>istream& operator >>(istream& s, Vector3D& v) { char separator; s >> separator >> v._x; s >> separator >> v._y; s >> separator >> v._z >> separator; return s; } Vector3D::Vector3D() { Vector3D(0,0,0); }</pre>	<p> A <code>>></code> operátort a térvektorok adatainak beolvasására használjuk. A <code>Vector3D</code> default konstruktorára azért van szükség, mert ilyen elemeket tartalmazó vektorral szeretnénk dolgozni.</p>



Saját jegyzeteim

1.változat:A tömb elemei térvektorok	
<pre> ... //Foglalás Vector3D *tomb; tomb = new Vector3D[n]; ... //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; } ... //Felszabadítás delete[] tomb; ... </pre>	
Maximumkeresés - A	Maximumkeresés - B
<pre> //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; } </pre>	<pre> //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; } } </pre>
Maximumkeresés - C	Maximumkeresés - D
<pre> 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; } } </pre>	<pre> 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; } } </pre>

<p>2.változat</p> <p>A vektor minden egyes elemében egy olyan mutató van, amely térvektorra mutat.</p> <pre> ... //Foglalás Vector3D **tomb; tomb= new Vector3D*[n]; for (int i=0; i!=n; i++) { tomb[i]= new Vector3D(); } ... //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; } ... //Felszabadítás for (int i=0; i!=n; i++) { delete tomb[i]; } delete[] tomb; ... </pre>	
<p>Maximumkeresés - A</p> <pre> //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++; } </pre>	<p>Maximumkeresés - B</p> <pre> //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); cout << k << endl; } i++; p++; } </pre>

A teljes megoldóprogram - 1. változat

```
/******  
/*Feladat: Egy szöveges állományban megadott térvektorok közül          */  
/*          keressük meg a leghosszabbat.                                */  
/******  
  
//Fordítási direktívák  
#include <iostream> using namespace std;  
#include <fstream>  
#include <string>  
#include "vector3d.h"  
int main()  
//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;  
}  
int n; inp >> n;  
cout << endl << "Elemek szama: " << n << endl;  
//Előfeltétel ellenőrzése  
if(n<1){  
    cout << "Nincs elem" << endl;  
    cin >> barmi;  
    return 2;  
}  
//Dinamikus tárterület lefoglalása és a tömb feltöltése  
Vector3D *tomb;  
tomb=new Vector3D[n];  
for (int j=0; j!=n; j++){  
    inp >> tomb[j];  
}  
inp.close();  
//Maximumkeresés  
Vector3D max;  
Vector3D* p;  
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;  
    }  
}  
//Eredmény megjelenítése  
cout << endl << "A leghosszabb térvektor: " << max << "." << endl;  
cout << "A térvektor hossza: " << max.abs() << endl;  
cout << "Ez pedig a tomb " << (k+1) << ". eleme. " << endl;  
cin >> barmi;  
//Dinamikusan lefoglalt tárterület felszabadítása  
delete[] tomb;  
return 0;  
}
```

A teljes megoldóprogram - 2. változat

```
//Fordítási direktívák
#include <iostream> using namespace std;
#include <fstream>
#include <string>
#include "vector3d.h"
int main(){
//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;
}
int n; inp >> n;
cout << endl << "Elemek szama: " << n << endl;
//Előfeltétel ellenőrzése
if(n<1){
    cout << "Nincs elem" << endl;
    cin >> barmi; return 2;
}
//Dinamikus helyfoglalás a tomb számára
Vector3D **tomb;
tomb=new Vector3D*[n];
for (int i=0; i!=n; i++) {
    tomb[i]= new Vector3D();
    inp >> *tomb[i];
}
inp.close();
//Maximumkeresés
Vector3D* max_pt;
Vector3D** p;
int 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++;
    p++;
}
//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
for (int i=0; i!=n; i++){
    delete tomb[i];
}
delete[] tomb;
return 0;
}
```