Feladat:

Valósítsuk meg a térvektorok típusát. A műveletek legyenek az összeadás, kivonás, vektoriális és skaláris szorzás, egyenlőségvizsgálat és abszolút érték meghatározása. Készítsünk a típushoz output operátort is.

Megoldás:

Reprezentáljuk a térvektorokat a koordinátáikkal, azaz három (lebegőpontos) számmal. A műveletek implementálásakor felhasználjuk a térvektorokról (pl. lineáris algebrából) tanultakat. Tekitsük az alábbi két térvektort, és a hozzájuk tartozó reprezentációt:

$$\vec{a} \quad \vec{b}$$
$$\uparrow \quad \uparrow$$
$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \quad \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Ekkor az egyes műveletek jelölése és számítási módja:

Összeadás	***Kivonás***	***Vektoriális szorzás***
$\overrightarrow{a+b}$	$\overrightarrow{a-b}$	$\overrightarrow{a*b}$
$\uparrow$	$\uparrow$	$\uparrow$
$\begin{pmatrix} a_1+b_1 \\ a_2+b_2 \\ a_3+b_3 \end{pmatrix}$	$\begin{pmatrix} a_1-b_1 \\ a_2-b_2 \\ a_3-b_3 \end{pmatrix}$	$\begin{pmatrix} a_2b_3-a_3b_2 \\ a_3b_1-a_1b_3 \\ a_1b_2-a_2b_1 \end{pmatrix}$

A skaláris szorzat eredménye egy valós szám, jelölése és számítási módja:

$$smul(a,b)=a_1b_1+a_2b_2+a_3b_3$$

Az abszolútérték számítás eredménye szintén egy valós szám, jelölése és számítási módja:

$$a.abs()=\sqrt{a_1^2+a_2^2+a_3^2}$$

Az egyenlőségvizsgálat logikai értéket ad vissza, két vektor akkor egyenlő, ha a koordinátáik megegyeznek:

$$a=b \Leftrightarrow (a_1=b_1 \wedge a_2=b_2 \wedge a_3=b_3)$$

Az $s \ll a$ output művelet írja ki az a vektort az s streambe az alábbi formában:

$$(a_1, a_2, a_3)$$

Megoldás C++-ban:

A C++ nyelvben a típusmegvalósítás eszköze az osztály. Az osztály definícióját egy header file-ban helyezzük el. A file neve legyen `vector3d.h`.

Osztályon belül megvalósított típusműveletek

Először készítsük el a típus egy leegyszerűsített változatát: kódoljuk le a reprezentációt és az abszolútérték meghatározását

```
#ifndef VECTOR3D_H
#define VECTOR3D_H

class Vector3D {
public:
   Vector3D(double x, double y, double z);
   double abs(void) const;
private:
   double _x;
   double _y;
   double _z;
};
#endif
```

Az osztály műveleteinek megvalósítását egy source file-ban helyezzük el (`vector3d.cpp`).

```
#include "vector3d.h"
#include <cmath>
using namespace std;
Vector3D::Vector3D(double x, double y, double z):
_x(x), _y(y), _z(z)
{}

double Vector3D::abs(void) const
{
   return sqrt(_x*_x+_y*_y+_z*_z);
}
```

Osztályon kívül megvalósított típusműveletek

Terjesszük ki most az osztályt a skaláris szorzás műveletével. A skaláris szorzást két vektoron lehet elvégezni, és a két vektor közül egyiknek sincs kiemelt szerepe. Mivel a művelet nem köthető egyik argumentumához sem, az osztályon kívül, globális függvényként valósítjuk meg. Az új header file:

```
#ifndef VECTOR3D_H
#define VECTOR3D_H

Vector3D smul(const Vector3D& larg,
              const Vector3D& rarg);


class Vector3D {
   friend Vector3D smul(const Vector3D& larg,
                        const Vector3D& rarg);
public:
   Vector3D(double x, double y, double z);
   double abs(void) const;
private:
   double _x;
   double _y;
   double _z;
};
#endif
```

A művelet megvalósítását -- mivel ehhez az osztályhoz kötődik -- az osztályműveletek megvalósítását tartalmazó file-ba (`Vector3D.cpp`) tesszük:

```
double smul(const Vector3D& larg, const Vector3D& rarg)
{
   return larg._x*rarg._x+larg._y*rarg._y+larg._z*rarg._z;
}
```

Külső operátorként megvalósított típusművetek

Az összeadás, kivonás és szorzás műveletek megvalósítása hasonló módon történhet. Az egyetlen eltérés, hogy ezeket a műveletek infix operátorként szeretnénk használni, ezért őket speciális operátorfüggvényként valósítjuk meg. Ugyanilyen megfontolásból az összehasonlítás is operátorfüggvény lesz.

```
#ifndef VECTOR3D_H
#define VECTOR3D_H

Vector3D smul(const Vector3D& larg,
              const Vector3D& rarg);
Vector3D operator+(const Vector3D& larg,
                   const Vector3D& rarg);
Vector3D operator-(const Vector3D& larg,
                   const Vector3D& rarg);
Vector3D operator*(const Vector3D& larg,
                   const Vector3D& rarg);
bool     operator==(const Vector3D& larg,
                    const Vector3D& rarg);
bool     operator!=(const Vector3D& larg,
                    const Vector3D& rarg)
{
   return !(larg==rarg);
}

class Vector3D {
   friend Vector3D 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);
```

```
public:
   Vector3D(double x, double y, double z);
   double abs(void) const;
private:
   double _x;
   double _y;
   double _z;
};
#endif
```

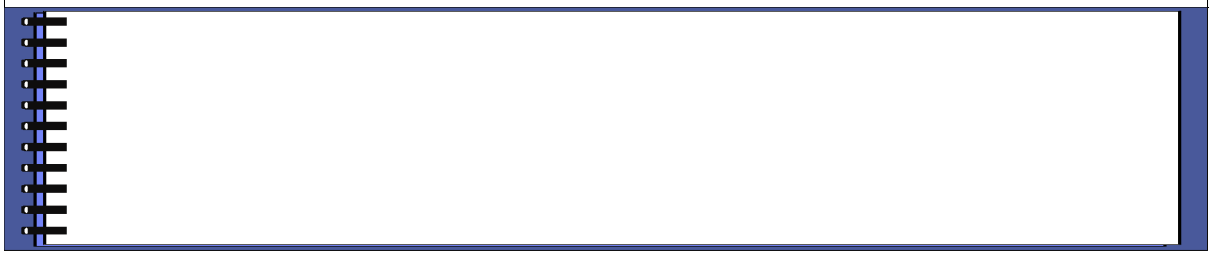
A bevezetett műveletek megvalósítását szintén az osztály implementációs file-jához fűzzük.

```
Vector3D operator+(const Vector3D& larg,
                   const Vector3D& rarg)
{
   return Vector3D(larg._x+rarg._x,
                   larg._y+rarg._y,
                   larg._z+rarg._z);
}

Vector3D operator-(const Vector3D& larg,
                   const Vector3D& rarg)
{
   return Vector3D(larg._x-rarg._x,
                   larg._y-rarg._y,
                   larg._z-rarg._z);
}

Vector3D operator*(const Vector3D& larg,
                   const Vector3D& rarg)
{
   return Vector3D(larg._y*rarg._z-larg._z*rarg._y,
                   larg._z*rarg._x-larg._x*rarg._z,
                   larg._x*rarg._y-larg._y*rarg._x);
}

bool operator==(const Vector3D& larg, const Vector3D& rarg)
{
   return larg._x == rarg._x &&
          larg._y == rarg._y &&
          larg._z == rarg._z;
}
```



Az output operátor egyformán kötődik az output streamhez és a kiírandó objektumhoz. Ezenkívül szigorúan kötött a prototípusa, ezért ezt is friend globális függvényként valósítjuk meg.

```
#ifndef VECTOR3D_H
#define VECTOR3D_H

#include <iostream>
using namespace std;

Vector3D smul(const Vector3D& larg,
              const Vector3D& rarg);
Vector3D operator+(const Vector3D& larg,
                   const Vector3D& rarg);
Vector3D operator-(const Vector3D& larg,
                   const Vector3D& rarg);
Vector3D operator*(const Vector3D& larg,
                   const Vector3D& rarg);
bool     operator==(const Vector3D& larg,
                    const Vector3D& rarg);
bool     operator!=(const Vector3D& larg,
                    const Vector3D& rarg)
{
   return !(larg==rarg);
}

ostream& operator<<(ostream& s, const Vector3D& v);

class Vector3D {
   friend Vector3D 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);
public:
   Vector3D(double x, double y, double z);
   double abs(void) const;
private:
   double _x;
   double _y;
   double _z;
};
#endif
```

```
ostream& operator<<(ostream& s, const Vector3D& v)
{
   s << '[ ' << v._x << ',' << v._y << ',' << v._z << ']';
   return s;
}
```

Vezessünk be egy bárhol használható konstanst a nullvektor jelölésére (deklaráció) és hozzunk létre egy térvektor objektumot, amely e nullvektort ábrázolja!

```
...
class Vector3D {
...
public:
   Vector3D(double x, double y, double z);
   double abs(void) const;
   static const Vector3D NULLVECT;
private:
   double _x;
   double _y;
   double _z;
};
```

A konstans definícióját helyezzük el az osztály implementációs file-jában:

```
static const Vector3D::Vector3D NULLVECT(0,0,0);
```

A főprogramban néhány egyszerű műveleten keresztül ellenőrizzük a típusműveletek működését (`main.cpp`).

```
#include <iostream>
using namespace std;

#include "vector3d.h"

int main(int argc, char *argv[])
{
  Vector3D a(1,2,3), b(4,5,6);
  cout << "a = " << a << endl;
  cout << "b = " << b << endl;
  cout << "NULVECT = " << Vector3D::NULVECT << endl;
  cout << "a+b = " << a+b << endl;
  cout << "a-b = " << a-b << endl;
  cout << "a*b = " << a*b << endl;
  cout << "a*b = " << smul(a,b) << endl;
  cout << "(a == b) = " << ((a == b)? "true": "false")
       << endl;

  return 0;
}
```