Міністерство освіти і науки України
Національний технічний університет України „КПІ”
Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки
інформації та управління

ЗВІТ

до лабораторної роботи № 4

з дисципліни ООП

Виконав студент *ІП-61 Кушка Михайло Олександрович*

(№ групи, прізвище, ім’я, по батькові)

Прийняв *Головченко М.М.*

(посада, прізвище, ім’я, по батькові)

Київ 2017

ЗМІСТ

1. МЕТА РОБОТИ

Мета роботи - вивчити особливості операційних і дружніх функцій. Освоїти принципи написання: функцій перетворення типів об'єктів, перевантаження операцій і дружніх функцій.

2. ПОСТАНОВКА ЗАДАЧІ

Спроектувати клас «Vector», який містить координати вектора в просторі. Для нього визначити: операцію складання «+», операцію віднімання «-», операцію векторного множення «*», операцію множення на константу «*», ці ж операції в скороченій формі, операцію унарний мінус «-», логічну операцію «==», яка перевіряє вектори на колінеарність. При необхідності дозволяється визначати інші операції (наприклад «=») і методи (наприклад, getter, setter та інше). Продемонструвати кожну операцію. Розглянути випадок коли вектор заданий на площині.

3. АНАЛІТИЧНІ ВИКЛАДКИ

Всі операції в С ++ можуть бути перевантажені, крім операцій точка (.), розіменування (*), дозвіл області дії (:), умовна (? :) і sizeof. Операції =, [], () і -> можуть бути перевантажені тільки як нестатичні функції-елементи, не можуть бути перевантажені для перелічуваних типів.

Всі інші операції можна перевантажувати, щоб застосовувати їх до якихось нових типів об'єктів, які вводяться користувачем. Операції перевантажуються шляхом складання опису функції, як це робиться для будь-яких функцій, за винятком того, що в цьому випадку ім'я функції складається з ключового слова operator, після якого записується перевантажувана операція.

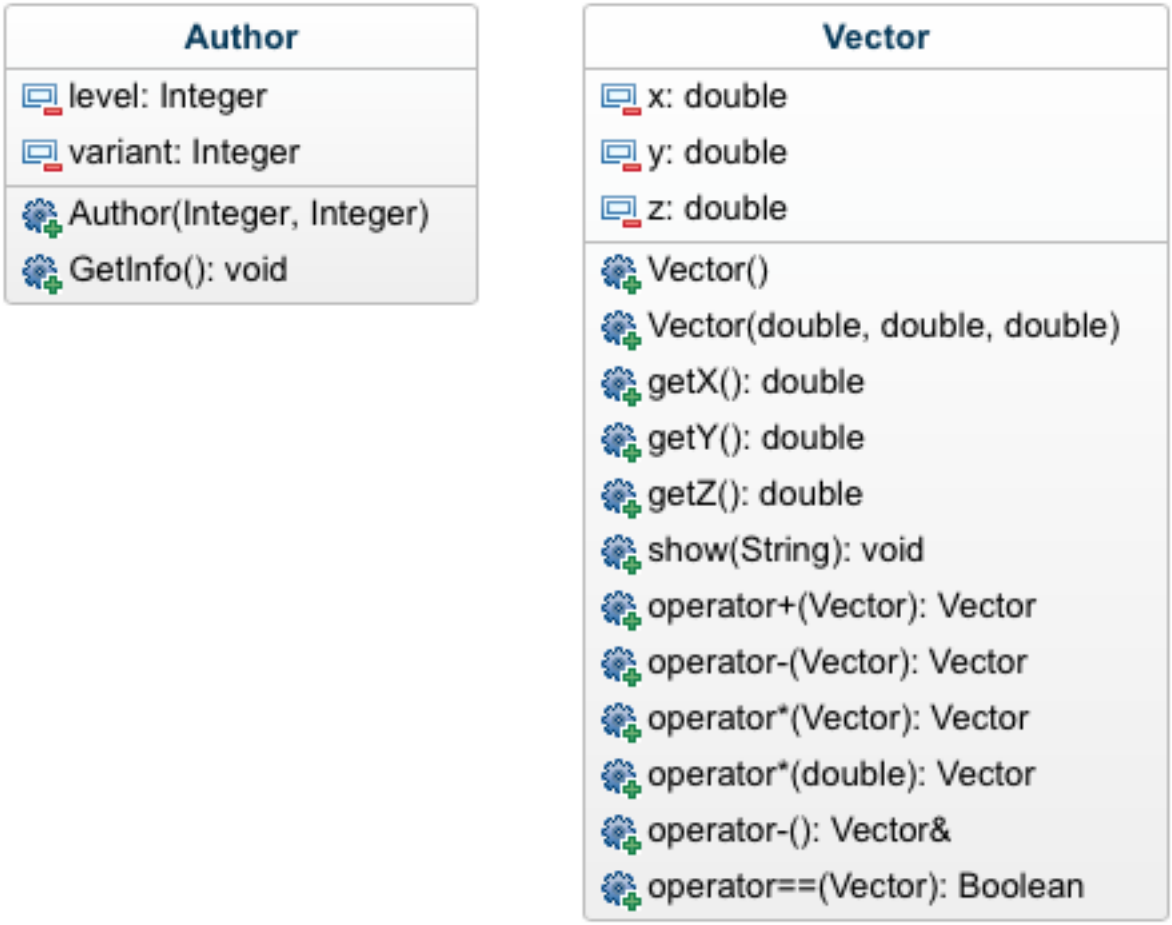
Щоб використовувати операцію над об'єктами класів, ця операція повинна бути перевантажена, проте є два винятки:

- операція = може бути використана будь-яким класом без явного перевантаження. За замовчуванням операція присвоювання зводиться до побітового копіювання елементів-даних класу. Таке копіювання небезпечно для класів з елементами, які вказують на динамічно виділені області пам'яті; для таких класів слід явно перевантажувати операцію присвоювання.

- операція адресації & також може бути використана з об'єктами будь-яких класів без перевантаження, вона просто повертає адресу об'єкта в пам'яті.

Перевантаження не може змінювати старшинство і асоціативність операцій, не може змінити число операндів операції. Перевантажені функції не успадковуються. Бінарні операції перевантажуються функцією з одним параметром, покажчиком на інший аргумент є this. Унарні операції перевантажуються функцією без параметрів.

4. UML-ДІАГРАМА КЛАСІВ



5. ВИХІДНИЙ КОД ПРОГРАМИ

prototypes.cpp

```
//
//  prototypes.cpp
//  Lab4
//
//  Created by Kushka Misha on 10/28/17.
//  Copyright © 2017 Kushka Misha. All rights reserved.
//

#include "prototypes.hpp"

// Sum
Vector Vector::operator+(Vector vec) {
    Vector temp;

    temp.x = x + vec.x;
    temp.y = y + vec.y;
    temp.z = z + vec.z;

    return temp;
}

// Substraction
Vector Vector::operator-(Vector vec) {
    Vector temp;

    temp.x = x - vec.x;
    temp.y = y - vec.y;
    temp.z = z - vec.z;

    return temp;
}

// Multiplication of a 2 vectors
Vector Vector::operator*(Vector vec) {
    Vector temp;

    temp.x = y * vec.z - z * vec.y;
    temp.y = z * vec.x - x * vec.z;
    temp.z = x * vec.y - y * vec.x;

    return temp;
}

// Multiplication by a constant
Vector Vector::operator*(double value) {
    Vector temp;

    temp.x = x * value;
    temp.y = y * value;
    temp.z = z * value;

    return temp;
}

// Unary minus
```

```
Vector& Vector::operator-() {
    Vector temp;

    temp.x = -x;
    temp.y = -y;
    temp.z = -z;

    return temp;
}

// Collinearity
bool Vector::operator==(Vector vec) {
    if (x == 0 && vec.x == 0) {
        if (y / vec.y == z / vec.z)
            return true;
        return false;
    } else if (y == 0 && vec.y == 0) {
        if (x / vec.x == z / vec.z)
            return true;
        return false;
    } else if (z == 0 && vec.z == 0) {
        if (x / vec.x == y / vec.y)
            return true;
        return false;
    } else {
        double alpha = 0;
        alpha = x / vec.x;
        if (y / vec.y == alpha && z / vec.z == alpha)
            return true;
    }
    return false;
}

void Author::GetInfo() {
    // Displays author info.
    cout << "\
    ----------------------\n\
    | Kushka Misha, IP-61 |\n\
    | Level: " << level << "             |\n\
    | Variant: " << variant << "         |\n\
    ----------------------\n\n";
}
```

prototypes.hpp

```
//
//  prototypes.hpp
//  Lab4
//
//  Created by Kushka Misha on 10/28/17.
//  Copyright © 2017 Kushka Misha. All rights reserved.
//

#ifndef prototypes_hpp
#define prototypes_hpp

#include "stdafx.hpp"

class Vector {
    double x, y, z;
```

```
public:
    Vector(double a, double b, double c) : x(a), y(b), z(c) {}; // Constructor
    Vector() { x = y = z = 0; }                                  // Blank constructor
    //    ~Vector() { cout << "Destructor..." << endl; }              // Destructor

    double getX() { return x; }
    double getY() { return y; }
    double getZ() { return z; }
    void show(string s) {
        cout << s << " = (" << x <<", " << y << ", " << z << ")" << endl;
    }

    Vector operator+(Vector);   // Sum of 2 vectors
    Vector operator-(Vector);   // Substraction of a 2 vectors
    Vector operator*(Vector);   // Multiplication of a 2 vectors
    Vector operator*(double);   // Multiplication by a constant

    Vector& operator-();        // Unary minus operation
    bool operator==(Vector);    // Check vectors collinearity
};

// Class to display some useful info about author of the program.
class Author {
    int level, variant;
public:
    Author(int level=3, int variant=15) : level(level), variant(variant) {}
    void GetInfo();
};

#endif /* prototypes_hpp */
```

functions.cpp

```
//
//  functions.cpp
//  Lab4
//
//  Created by Kushka Misha on 10/28/17.
//  Copyright © 2017 Kushka Misha. All rights reserved.
//

#include "functions.hpp"

// Show all class operator functions
void show_demo(Vector a, Vector b, Vector c) {
    // c = a + b
    cout << "c = a + b" << endl;
    c = a + b;
    c.show("C");
    cout << endl;

    // c = a - b
    cout << "c = a - b" << endl;
    c = a - b;
    c.show("C");
    cout << endl;

    // c = a * b
    cout << "c = a * b" << endl;
    c = a * b;
```

```
    c.show("C");
    cout << endl;

    // c = a * const
    cout << "c = a * 3" << endl;
    c = a * 3;
    c.show("C");
    cout << endl;

    // c = -a
    cout << "c = -a" << endl;
    c = -a;
    c.show("C");
    cout << endl;

    // isCollinear = a == b
    bool isCollinear = false;
    cout << "isCollinear = a == b" << endl;
    isCollinear = a == b;
    cout << "isCollinear == " << isCollinear << endl << endl;

    // isCollinear = a == a*2
    cout << "isCollinear = a == a*2" << endl;
    isCollinear = a == a*2;
    cout << "isCollinear == " << isCollinear << endl << endl;
}
```

functions.hpp

```
//
//  functions.hpp
//  Lab4
//
//  Created by Kushka Misha on 10/28/17.
//  Copyright © 2017 Kushka Misha. All rights reserved.
//

#ifndef functions_hpp
#define functions_hpp

#include "prototypes.hpp"

void show_demo(Vector, Vector, Vector);

#endif /* functions_hpp */
```

stdafx.hpp

```
//
//  stdafx.hpp
//  Lab4
//
//  Created by Kushka Misha on 10/28/17.
//  Copyright © 2017 Kushka Misha. All rights reserved.
//

#ifndef stdafx_hpp
#define stdafx_hpp
```

```cpp
#include <iostream>
#include <string>

using namespace std;

#endif /* stdafx_hpp */
```

main.cpp

```cpp
#include "prototypes.hpp"
#include "functions.hpp"

int main() {
    Author *auth = new Author();
    auth->GetInfo();

    Vector a(5, 3, -1), b(3, 3, 3), c;

    a.show("A");
    b.show("B");
    cout << endl;

    show_demo(a, b, c);

    // Vectors are given on the plane
    cout << endl << "=====Vectors are given on the plane=====" << endl;
    Vector a_plane(5, 3, 0), b_plane(3, 3, 0), c_plane;

    a_plane.show("A");
    b_plane.show("B");
    cout << endl;

    show_demo(a_plane, b_plane, c_plane);


    return 0;
}
```

6. ПРИКЛАДИ РОБОТИ ПРОГРАМИ

```
c = a + b
C = (8, 6, 2)

c = a - b
C = (2, 0, -4)

c = a * b
C = (12, -18, 6)

c = a * 3
C = (15, 9, -3)

c = -a
C = (-5, -3, 1)

isCollinear = a == b
isCollinear == 0

isCollinear = a == a*2
isCollinear == 1


=====Vectors are given on the plane=====
A = (5, 3, 0)
B = (3, 3, 0)

c = a + b
C = (8, 6, 0)

c = a - b
C = (2, 0, 0)

c = a * b
C = (0, 0, 6)

c = a * 3
C = (15, 9, 0)

c = -a
C = (-5, -3, -0)

isCollinear = a == b
isCollinear == 0

isCollinear = a == a*2
isCollinear == 1

Program ended with exit code: 0
```

7. ВИСНОВКИ

У даній лабораторній роботі я використав перевантаження операторів на прикладі роботи з класом для задання вектора у просторі, вивчив особливості використання та реалізації перевантаження.