

Додаток 1

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут
імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 3 з дисципліни
«Основи програмування-2.
Модульне програмування»
«Перевантаження операторів»

Варіант 20

Виконала студентка ПІ-13, Лисенко Анастасія Олегівна
(шифр, прізвище, ім'я, по батькові)

Перевірив Всечерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Лабораторна робота 3 Перевантаження операторів

Мета — вивчити механізми створення класів з використанням перевантажених операторів (операцій).

Варіант 20

Постановка задачі.

Визначити клас "Трикутник", членами якого є сторони трикутника в просторі. Реалізувати для нього декілька конструкторів, геттери, методи обчислення площі трикутника. Перевантажити оператори: "++" / "--" - для інкрементування/ декрементування довжин сторін трикутника відповідно, "+=" / "-=" - для збільшення / зменшення довжин сторін трикутника на вказану величину. Створити три трикутника (T1, T2, T3), використовуючи різні конструктори. Інкрементувати довжини сторін трикутника T1, а довжини сторін трикутника T2 декрементувати. Збільшити довжини сторін трикутника T3 на вказану величину. Серед трикутників T1, T2, T3 визначити трикутник, що має найбільшу площу.

Код на мові C++:

lib.cpp:

```
#include "lib.h"

Triangle::Triangle(double vertice_1, double vertice_2, double vertice_3) {
    this->vertice_1 = vertice_1;
    this->vertice_2 = vertice_2;
    this->vertice_3 = vertice_3;
}

Triangle::Triangle(string line){

    vector <string> vertices = split(line, ' ');
    vertice_1 = stod(vertices[0]);
    vertice_2 = stod(vertices[1]);
    vertice_3 = stod(vertices[2]);
}

Triangle::Triangle(Triangle& tr) {
    this->vertice_1 = tr.vertice_1;
    this->vertice_2 = tr.vertice_2;
    this->vertice_3 = tr.vertice_3;
}

double Triangle::GetVertice1() {
    return this->vertice_1;
}

double Triangle::GetVertice2() {
    return this->vertice_2;
}

double Triangle::GetVertice3() {
    return this->vertice_3;
}

double Triangle::findArea() {
    double p = (vertice_1 + vertice_2 + vertice_3) / 2;
    double area = pow(p * (p - vertice_1) * (p - vertice_2) * (p - vertice_3), 0.5);
    return area;
}
```

```
}
Triangle Triangle::operator++ () {
    this->vertice_1++;
    this->vertice_2++;
    this->vertice_3++;
    return *this;
}
Triangle Triangle::operator-- () {
    this->vertice_1--;
    this->vertice_2--;
    this->vertice_3--;
    return *this;
}
Triangle Triangle::operator+= (double n) {
    this->vertice_1 += n;
    this->vertice_2 += n;
    this->vertice_3 += n;
    return *this;
}
Triangle Triangle::operator-= (double n) {
    this->vertice_1 -= n;
    this->vertice_2 -= n;
    this->vertice_3 -= n;
    return *this;
}
bool isValid(double ver_1, double ver_2, double ver_3) {
    if (((ver_1 + ver_2) <= ver_3) || ((ver_2 + ver_3) <= ver_1) || ((ver_1 + ver_3) <= ver_2)
    || (ver_1<=0) || (ver_2 <= 0)|| (ver_3 <= 0))
        return 0;
    else
        return 1;
}
double max_area(double area_1, double area_2, double area_3, vector <string>& pos) {
    double max;
    if ((area_1 >= area_2) && (area_1 >= area_3)) {
        max = area_1;
        pos.push_back("T1");
    }

    if ((area_2 >= area_1) && (area_2 >= area_3)) {
        max = area_2;
        pos.push_back("T2");
    }
    if ((area_3 >= area_1) && (area_3 >= area_2)) {
        max = area_3;
        pos.push_back("T3");
    }
    return max;
}
vector <string> split(string line, char del) {
    string str = "";
    vector <string> vertices;
    line += del;
    for (int i = 0; i < line.length(); i++) {
        if (line[i] == del) {
            if (str.length() > 0) {
                vertices.push_back(str);
            }
            str = "";
        }
        else {
            str += line[i];
        }
    }
    return vertices;
}
```

lib.h:

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
using namespace std;

class Triangle{
private:

    double vertice_1;
    double vertice_2;
    double vertice_3;

public:

    Triangle(double vertice_1, double vertice_2, double vertice_3);
    Triangle(string line);
    Triangle(Triangle& tr);

    double GetVertice1();
    double GetVertice2();
    double GetVertice3();
    double findArea();
    Triangle operator++ ();
    Triangle operator-- ();
    Triangle operator+= (double n);
    Triangle operator-= (double n);
};
vector <string> split(string, char );
bool isValid(double , double , double );
double max_area(double, double , double, vector <string>&);
```

lab4.cpp:

```
#include "lib.h"

int main()
{
    double ver_1 = 0, ver_2 = 0, ver_3 = 0;
    do{
        cout << "Enter the correct sizes of the vertices you are looking for:\nvertex 1: ";
cin >> ver_1;
        cout << "vertex 2: "; cin >> ver_2;
        cout << "vertex 3: "; cin >> ver_3; cout << endl;
    } while (isValid(ver_1, ver_2, ver_3) == 0);
    Triangle T1(ver_1, ver_2, ver_3);
    printf("vertices of T1: 1. %.3f; 2. %.3f; 3. %.3f;\n", T1.GetVertice1(), T1.GetVertice2(),
T1.GetVertice3());
    cout << "_____ " << endl;

    string line;
    cout << "T2:\nEnter the sizes of the vertices in one line(like this: v1 v2 v3): ";
    cin.ignore();
    getline(cin, line, '\n');
    vector <string> temp = split(line, ' ');
    while ((isValid(stod(temp[0])-1, stod(temp[1])-1, stod(temp[2])-1) == 0)) {
        cout << "Enter the correct sizes of the vertices in one line(like this: v1 v2 v3): ";
        cin.clear();
        getline(cin, line, '\n');
        temp = split(line, ' ');
    }
    Triangle T2(line);
    printf("\nvertices of T2: 1. %.3f; 2. %.3f; 3. %.3f;\n\n", T2.GetVertice1(),
T2.GetVertice2(), T2.GetVertice3());
    cout << "_____ " << endl;
```

```
cout << "T3:" << endl;
Triangle T3(T1);
printf("vertices of T3: 1. %.3f; 2. %.3f; 3. %.3f;\n\n", T3.GetVertice1(), T3.GetVertice2(),
T3.GetVertice3());
cout << "_____ " << endl;

cout << "Incremintation of T1:\n";
++T1;
printf("vertices of T1: 1. %.3f; 2. %.3f; 3. %.3f;\n", T1.GetVertice1(), T1.GetVertice2(),
T1.GetVertice3());

cout<< "Decreminatation of T2:\n";
--T2;
printf("vertices of T2: 1. %.3f; 2. %.3f; 3. %.3f;\n\n", T2.GetVertice1(), T2.GetVertice2(),
T2.GetVertice3());

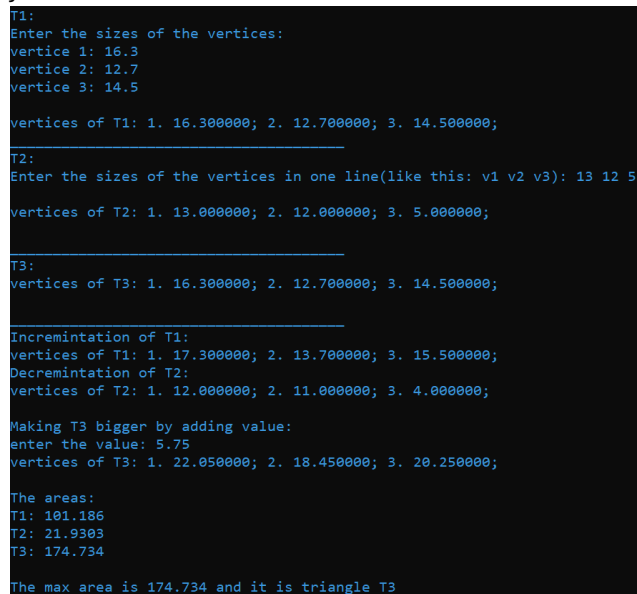
cout << "Making T3 bigger by adding value:\n";
double n = 0;
do {
    cout << "enter the value: "; cin >> n;
} while (ifValid(T3.GetVertice1() + n, T3.GetVertice2() + n, T3.GetVertice3() + n) == 0);

T3 += n;
printf("vertices of T3: 1. %.3f; 2. %.3f; 3. %.3f;\n\n", T3.GetVertice1(), T3.GetVertice2(),
T3.GetVertice3());

cout << "The areas:"<<endl;
cout << "T1: " << T1.findArea()<<endl;
cout << "T2: " << T2.findArea()<<endl;
cout << "T3: " << T3.findArea()<<endl<<endl;
vector <string> pos;
cout << "The max area is " << max_area(T1.findArea(), T2.findArea(), T3.findArea(), pos)<< "
and it is triangle/s ";
for (size_t i = 0; i < pos.size(); i++)
{
    if (i == pos.size() - 1)
        cout << pos[i] << endl;
    else
        cout << pos[i] << ", ";
}

return 0;
```

}Консоль:



```
T1:
Enter the sizes of the vertices:
vertex 1: 16.3
vertex 2: 12.7
vertex 3: 14.5

vertices of T1: 1. 16.300000; 2. 12.700000; 3. 14.500000;
_____
T2:
Enter the sizes of the vertices in one line(like this: v1 v2 v3): 13 12 5
vertices of T2: 1. 13.000000; 2. 12.000000; 3. 5.000000;
_____
T3:
vertices of T3: 1. 16.300000; 2. 12.700000; 3. 14.500000;
_____
Incremintation of T1:
vertices of T1: 1. 17.300000; 2. 13.700000; 3. 15.500000;
Decreminatation of T2:
vertices of T2: 1. 12.000000; 2. 11.000000; 3. 4.000000;
_____
Making T3 bigger by adding value:
enter the value: 5.75
vertices of T3: 1. 22.050000; 2. 18.450000; 3. 20.250000;
_____
The areas:
T1: 101.186
T2: 21.9303
T3: 174.734

The max area is 174.734 and it is triangle T3
```

1. Висновки

Ми вивчили механізми створення класів з використанням перевантажених операторів (операцій). Та навчилися перевантажувати оператори (операції) та використовувати їх при написанні програмного коду.