Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Уфимский государственный авиационный технический университет»

Кафедра ВВТиС

Отчет по лабораторной работе №7  
по дисциплине «Программирование»

«Перегрузка операций»

Группа ПМИ-150

Выполнил \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ Поташин А.В.

(дата) (подпись) (ФИО)

Проверил \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

(дата) (подпись) (ФИО)

Уфа 2019

# 1. Цели и задачи

Целью лабораторной работы является изучить на практике создание операторов/операций, специфичных для объектов одного класса.

1. Разработать полный класс Многочлен, содержащий необходимые функции конструктора и деструктора, а также функции get и set. Класс содержит поле, хранящее степень многочлена, и массив коэффициентов при членах. Например, многочлен 2 4 x + 2x имеет степень 4 и массив коэффициентов {0.,0.,1.,0.,2.}.

Определить для класса конструктор копирования, перегрузить стандартные операции:

- присваивания,

- вывода в поток и ввода из потока,

- сложения с вещественным числом (слева и справа).

Создать объекты данного класса в функции main() и продемонстрировать работу всех перегруженных операций.

2. На основе существующих классов или объектов библиотеки SFML создать собственный класс, описывающий графический объект следующей формы:   
 <<Тут снеговик>>

Определить конструкторы, деструктор и методы, необходимые для работы с объектом.

Перегрузить для созданного класса операцию сложения двух объектов, результатом которой будет объект, координаты и размеры которого есть среднее геометрическое значение соответствующих полей двух операндов.

Создать программу, демонстрирующую работу перегруженной операции.

# 2. Теоретическая часть

## Что такое конструктор копирования и для чего он нужен?

## Что является входным параметром конструктора копирования? Почему входной параметр должен описываться со служебным словом const

## Какие функции называются дружетсвенными? Как описываются?

## Для чего необходимы дружественные функции?

## В чем заключается перегрузка операторов?

## Какие операторы нельзя перегружать?

## Можно ли создать свой собственный оператор?

## В каких случаях перегрузка оператора может осуществляться только при помощи дружественной функции? Приведите примеры.

## Почему при перегрузке операторов ввода из потока и вывода в поток необходимо возвращать ссылку на поток?

## 2.1. Ответы на контрольные вопросы

## Конструктор, который вызывается для копирования(создания нового аналогичного объекта с выделенной под него памятью) объекта при передаче его куда либо и т.д.

1. Ссылка на объект. Для защиты передаваемого объекта от случайных изменений.
2. Это функции имеющие доступ к полям с модификатором доступа private и protected, но не являющиеся членами этого класса непосредственно. Friend….
3. Для получения привилегий доступа к нескольким классам.
4. В переопределении операции для объектов класса.
5. тернарный оператор, доступ к вложенным именам, доступ к полям, доступ к полям по указателю, sizeof, typeid, операторы каста.
6. Нет
7. Когда левый операнд не принадлежит классу перегрузки и когда идет работа с объектами разных классов.
8. Данные передаются в поток этими операторами.

# 3. ПРАКТИЧЕСКАЯ ЧАСТЬ

**3.1**

**Main.cpp**

#include <iostream>

#include <math.h>

#include <random>

#include <ctime>

#include "polynomial.h"

int main() {

srand((unsigned int)time(NULL));

polynomial\* a = new polynomial();

polynomial\* b = new polynomial(3);

cout << \*b << endl;

cout << \*a << endl;

cout << \*a + 1 << endl;

cin >> \*a;

cout << \*a << endl;

cout << \*b \* \*a << endl;

a = b;

cout << \*a;

return 0;

}

**Polynomial.h**

#pragma once

#include <iostream>

using namespace std;

class polynomial

{

public:

polynomial();

polynomial(int);

polynomial(const polynomial&);

~polynomial();

void set\_degree(int);

void set\_coefficients(double\*);

int get\_degree();

void get\_coefficients();

int random(int, int);

double\* create\_coef(int);

polynomial& operator=(const polynomial&);

polynomial& operator+(double);

polynomial& operator\*(polynomial&);

friend ostream& operator<<(ostream&, polynomial&);

friend istream& operator>>(istream&, polynomial&);

private:

int degree;

double\* x;

};

**Polynomial.cpp**

#define \_USE\_MATH\_DEFINES

#include "polynomial.h"

#include <iostream>

#include <math.h>

#include <random>

#include <ctime>

int polynomial::random(int a, int b) {

int minimum = min(a, b);

int maximum = max(a, b);

return rand() % (maximum - minimum) + minimum;

};

void polynomial::set\_degree(int degree) {

this->degree = degree;

}

int polynomial::get\_degree() {

return this->degree;

}

void polynomial::get\_coefficients() {

for (int i = 0; i < this->degree + 1 ; i++)

{

cout << this->x[i] << " ";

}

}

double\* polynomial::create\_coef(int degree) {

double\* coef = new double[degree + 1];

for (int i = 0; i < degree + 1; i++)

{

coef[i] = random(-4, 4);

}

if (coef[degree] == 0) {

coef[degree]++;

}

return coef;

}

void polynomial::set\_coefficients(double \*coef) {

for (int i = 0; i < this->degree + 1; i++)

{

this->x[i] = coef[i];

}

}

polynomial::polynomial(int degree)

{

set\_degree(degree);

x = new double[degree + 1];

set\_coefficients(create\_coef(degree));

}

polynomial::polynomial() {

set\_degree(0);

x = new double[1];

set\_coefficients(create\_coef(0));

}

polynomial::polynomial(const polynomial& copy) {

this->degree = copy.degree;

this->x = new double[this->degree + 1];

for (int i = 0; i < this->degree + 1; i++)

{

this->x[i] = copy.x[i];

}

}

polynomial::~polynomial()

{

delete[]x;

}

istream& operator>>(istream& stream, polynomial& obj)

{

cout << "polynomial enter" << endl;

cout << "degree=" << endl;

stream >> obj.degree;

delete[]obj.x;

obj.x = new double[obj.degree + 1];

for (int i = 0; i <= obj.degree;)

{

cout << "x^(" << i << ")\*";

stream >> obj.x[i++];

}

cout << endl;

return stream;

}

ostream& operator<<(ostream& stream, polynomial& obj)

{

cout << "polynomianl out" << endl;

for (int i = 0; i <= obj.degree; i++) {

stream << "x^(" << i << ")\*" << int(obj.x[i]) << endl;

}

return stream;

}

polynomial& polynomial::operator=(const polynomial& Obj)

{

degree = Obj.degree;

delete[]x;

x = new double[degree + 1];

for (int i = 0; i < degree + 1; i++) {

x[i] = Obj.x[i];

}

return \*this;

}

polynomial& polynomial::operator+(double plus) {

cout << "polynomial plus" << endl;

polynomial \*temp = new polynomial();

for (int i = 0; i < this->degree + 1; i++)

{

temp->x[i] = this->x[i];

}

temp->x[0] += plus;

return \*temp;

}

polynomial& polynomial::operator\*(polynomial& obj) {

cout << "polynomial \*" << endl;

int newDegree = obj.degree + this->degree;

cout << newDegree << endl;

double\* newX = new double[newDegree + 1];

for (int i = 0; i < newDegree + 1; i++) {

newX[i] = 0;

}

polynomial\* result = new polynomial();

result->set\_degree(newDegree);

result->set\_coefficients(newX);

for (int i = 0; i < obj.degree + 1; i++)

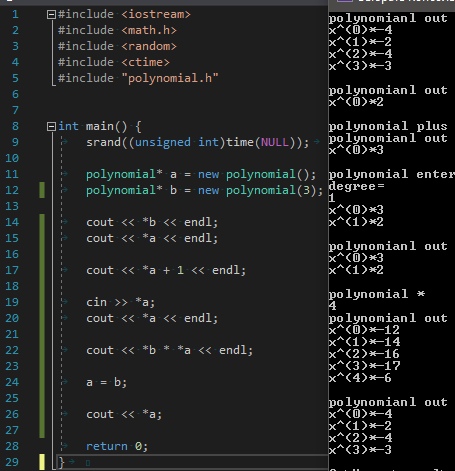
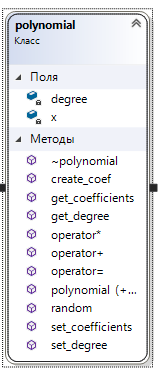
for (int j = 0; j < this->degree + 1; j++) {

result->x[i + j] += obj.x[i] \* x[j];

}

return \*result;

}

**3.2**

**lab2\_3.2.cpp**

#include <iostream>

#include "Object.h"

#include <math.h>

#include <SFML\Graphics.hpp>

#include <random>

#include <ctime>

using namespace std;

using namespace sf;

int main()

{

srand((unsigned int)time(NULL));

RenderWindow window(VideoMode(720, 480), "SFML", Style::Close);

Object obj = Object();

Object objT = Object(540, 360, 40);

Object objC = obj + objT;

obj.render(&window);

objT.render(&window);

objC.render(&window);

window.display();

while (window.isOpen())

{

Event event;

while (window.pollEvent(event))

{

if (event.type == Event::Closed)

window.close();

}

}

return 0;

}

**Object.h**

#pragma once

#include <math.h>

#include <SFML\Graphics.hpp>

#include <iostream>

#include <random>

#include <ctime>

using namespace std;

using namespace sf;

class Object

{

public:

struct coord

{

double x;

double y;

};

Object();

Object(double, double, double);

~Object();

void setNewSize(double);

double getSize();

Vector2f getCoord();

void render(RenderWindow\*);

void figureToSmall();

void figureToNormal();

void figureToLarge();

Vector2f left\_finish();

Vector2f right\_finish();

void line(RenderWindow\*, Vector2f, Vector2f);

Object operator+(const Object&);

private:

double radius;

Vector2f coord;

CircleShape circle;

};

**Object.cpp**

#include "Object.h"

#include <math.h>

#include <SFML\Graphics.hpp>

#include <iostream>

#include <random>

#include <ctime>

using namespace std;

using namespace sf;

Object::Object()

{

radius = 30;

circle.setOrigin(radius, radius);

coord = Vector2f(180, 120);

circle.setPosition(coord);

circle.setFillColor(Color(255, 255, 255));

}

Object::Object(double x, double y, double rad)

{

radius = rad;

circle.setOrigin(radius, radius);

coord = Vector2f(x, y);

circle.setPosition(coord);

circle.setFillColor(Color(255, 255, 255));

}

Object::~Object()

{

}

void Object::setNewSize(double rad) {

this->radius = rad;

}

double Object::getSize() {

return this->radius;

}

Vector2f Object::getCoord() {

return this->coord;

}

void Object::figureToSmall() {

this->circle.setPosition(this->coord.x + this->radius \* 0.3, this->coord.y - this->radius \* 1);

this->circle.setRadius(this->radius \* 0.7);

}

void Object::figureToNormal() {

this->circle.setPosition(this->coord);

this->circle.setRadius(this->radius);

}

void Object::figureToLarge() {

this->circle.setPosition(this->coord.x - this->radius \* 0.3, this->coord.y + this->radius \* 1.5);

this->circle.setRadius(this->radius \* 1.3);

}

Vector2f Object::left\_finish() {

return this->coord - Vector2f(this->radius \* 1.5, -this->radius);

}

Vector2f Object::right\_finish() {

return this->coord + Vector2f(this->radius \* 1.5, this->radius);

}

void Object::line(RenderWindow\* window, Vector2f start, Vector2f finish) {

sf::Vertex line[] = {

sf::Vertex(start),

sf::Vertex(finish)

};

line[0].color = Color(255, 255, 255);

line[1].color = Color(255, 255, 255);

window->draw(line, 2, sf::Lines);

}

void Object::render(RenderWindow\* window) {

this->figureToNormal();

window->draw(this->circle);

this->figureToSmall();

window->draw(this->circle);

this->figureToNormal();

this->figureToLarge();

window->draw(this->circle);

this->figureToNormal();

line(window, this->coord - Vector2f(this->radius, 0), this->left\_finish());

line(window, this->coord + Vector2f(this->radius, 0), this->right\_finish());

}

Object Object::operator+(const Object& obj) {

Object object;

object.radius = int(sqrt(this->radius \* obj.radius));

object.coord.x = int(sqrt(this->coord.x \* obj.coord.x));

object.coord.y = int(sqrt(this->coord.y \* obj.coord.y));

return object;

}

