Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

"Национальный исследовательский Нижегородский государственный

университет им. Н. И. Лобачевского"

Институт информационных технологий, математики и механики

Отчет по лабораторной работе

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ПОЛИНОМАМИ

Выполнил: студент группы 381808-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шульман Е. А.

Подпись

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Панов А. А.

Подпись

Нижний Новгород

2019

Содержание

[**Введение** 3](#_Toc26474107)

[**Постановка задачи** 4](#_Toc26474108)

[**Руководство пользователя** 5](#_Toc26474109)

[**Руководство программиста** 8](#_Toc26474110)

[Описание структуры программы 8](#_Toc26474111)

[Описание структуры данных 9](#_Toc26474112)

[Описание алгоритмов 10](#_Toc26474113)

[**Заключение** 13](#_Toc26474114)

[**Литература** 14](#_Toc26474115)

[**Приложения** 15](#_Toc26474116)

[Приложение 1 15](#_Toc26474117)

[Приложение 2 15](#_Toc26474118)

[Фрагменты исходного кода программы 16](#_Toc26474119)

[**polynom.h** 16](#_Toc26474120)

[**polynom.cpp** 17](#_Toc26474121)

**Введение**

Полином - одна из базовых алгебраических структур, которая встречается в школьной и высшей математике. Изучение полинома - важнейшая тема в курсе алгебры, поскольку с одной стороны многочлены достаточно просты по сравнению с другими типами функций, с другой - широко применяются в решении задач математического анализа.

При решении различных прикладных задач нередко возникают такие объекты как полиномы. При достаточно большом количестве слагаемых выполнение «вручную» простых арифметических операций над ними становиться затруднительным. Поэтому возникает необходимость разработки программы, которая способна выполнять элементарные арифметические операции с полиномами и быстро вычислять результат той или иной операции.

В данной лабораторной работе будет рассматриваться один из возможных способов реализации полинома.

**Постановка задачи**

Требуется разработать программу, выполняющую арифметические операции с полиномами одной переменной (x): сложение, вычитание, умножение двух полиномов и нахождение полинома в заданной точке. Считается, что полином составлен из мономов от одной переменной. Коэффициенты полинома - вещественные числа. Степени полинома – целые числа. Кроме того, необходимо разработать пользовательское консольное приложение.

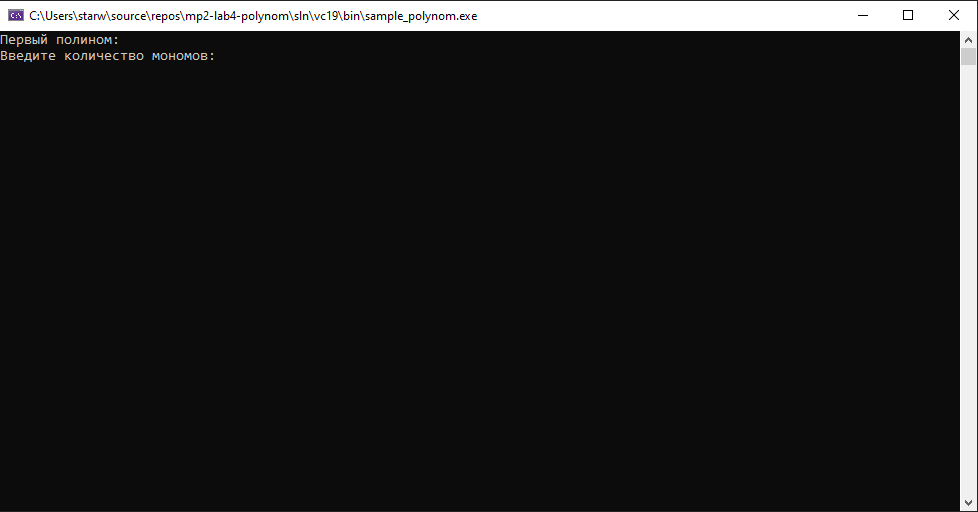
Особенности реализации:

* В качестве структуры хранения полинома использовать список мономов;
* Элементы списка хранить упорядоченными;
* При умножении и сложение (вычитание) необходимо следить, чтобы в итоговом полиноме были приведены подобные слагаемые и не хранилось мономов с нулевым коэффициентом;

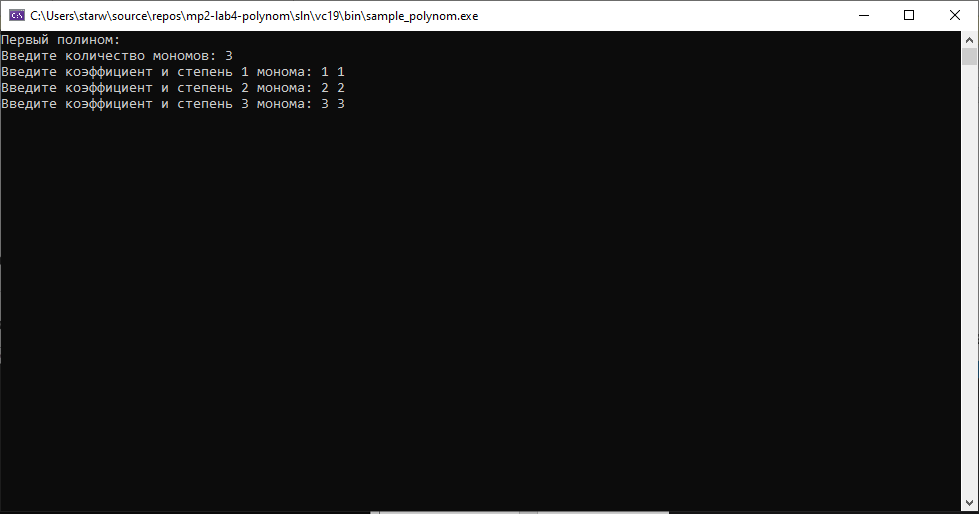
**Руководство пользователя**

Для начала работы программы необходимо запустить на выполнение файл sample\_polynom.exe.

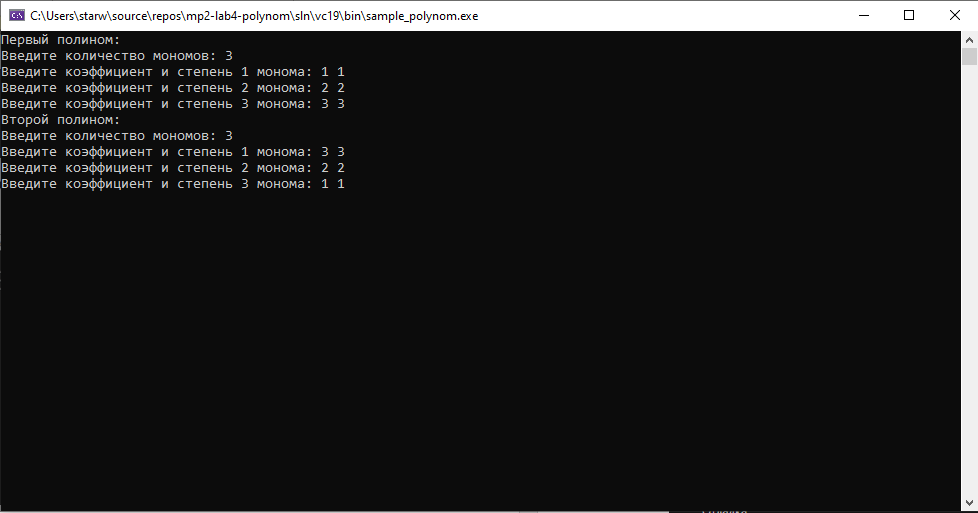
1. В появившемся окне необходимо ввести кол-во мономов первого полинома.



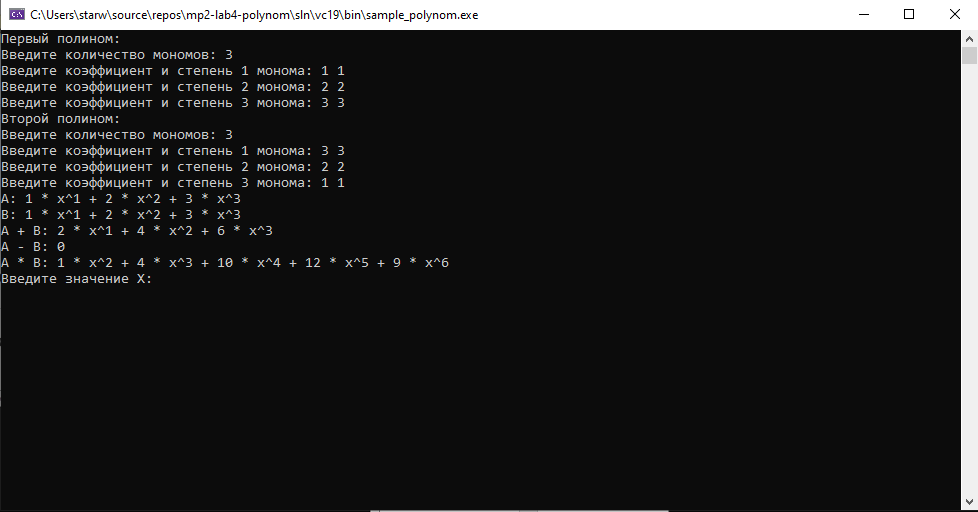
1. Далее для каждого монома вводим, через пробел, значение коэффициента и степени соответственно.



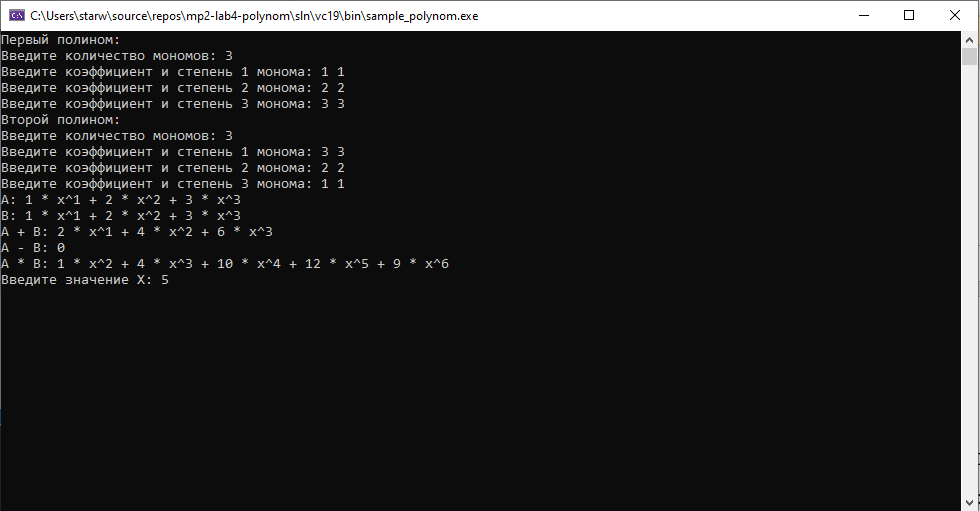
1. Повторяем шаги 1 и 2 для второго полинома.



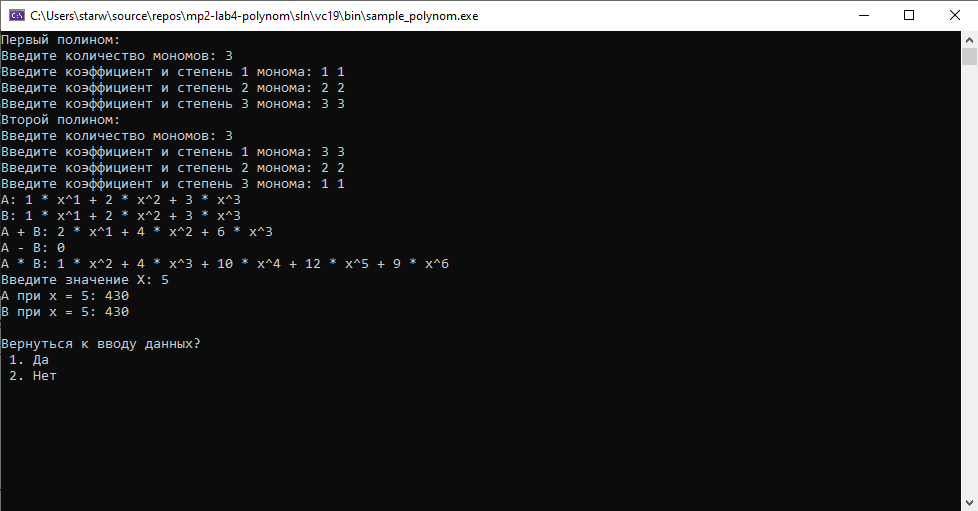
1. Программа выводит: первый полином, второй полином, сумму полиномов, разность полиномов и результат умножения первого полинома на второй.



1. Программа просит ввести значение переменной Х, вводим его.



1. Программа выводит нам результат вычисления полинома 1 и полинома 2 в заданной точке Х.



**Руководство программиста**

Описание структуры программы

Программа состоит из 2 проектов:

* sample\_polynom – пользовательское консольное приложение, выполняющее арифметические операции над полиномами;
* polynom – модуль со структурой Monom и с классом Polynom, реализующим арифметические операции над полиномами.

Описание структуры данных

Структура Monom.

Поля структуры:

* int pow – степень монома;
* double coef – коэффициент монома;
* Monom\* next – указатель на следующий моном в полиноме;

Класс Polynom.

Поля класса:

* Monom\* head – указатель на первый моном в полиноме;
* int size – размер полинома;

Описание алгоритмов

Структура Monom.

Наиболее сложные методы стурктуры

* Monom();

Конструктор по умолчанию. Создает моном с коэффициентом 0 и степенью 0, указывает так же на 0, т. е. не имеет следующего монома.

* Monom(double \_coef, int \_pow);

Конструктор инициализатор. Создает моном с коэффициентом coef и степенью pow, указывает так же на 0, т. е. не имеет следующего монома.

* Monom(double \_coef, int \_pow, Monom\* \_next);

Конструктор инициализатор. Создает моном с коэффициентом coef и степенью pow, указывает на next.

* Monom(const Monom& monom);

Конструктор копирования. Создает моном с коэффициентом и степенью монома monom, указывает на тот же моном, на который указывает моном monom.

* bool operator==(const Monom& right);

Сравнивает два монома, если степени равны, а коэффициенты не различаются на бесконечно малое число, то возвращает 1, иначе 0.

Класс Polynom.

Наиболее сложные методы класса

* Polynom();

Конструктор по умолчанию. Создает полином размером 0.

* Polynom(const Polynom& polynom);

Конструктор копирования. Создает полином с таким же размером, как и у polynom, и в цикле заполняет его мономами, из polynom.

* ~Polynom();

Деструктор. Вызывает метод clear();

* int getSize();

Возвращает размер полинома.

* void clear();

Пока размер полинома не равен 0 и указатель head не указывает на 0, вызывает метод pop\_front();

* void insert(Monom monom);

Вставляет моном monom, в правильное место списка, т. е. по показателю pow.

* void insert(double coef, int pow);

Вставляет моном полями coef и pow, в правильное место списка,

т. е. по показателю pow.

* void merge(const Polynom& polynom);

Сливает два полинома в один, с правильным расположением мономов, т.е. по показателям pow.

* void pop\_front();

Удаляет первый моном в полиноме.

* void push\_front(Monom monom);

Добавляет моном monom в начало.

* void push\_back(Monom monom);

Добавляет моном monom в конец.

* void del(Monom monom);

Удаляет из полинома моном со степенью monom->pow

* bool find(int pow);

Проверяет наличие степени pow в полиноме.

* int Calculate(int x);

Вычисляет значение полинома при заданном x.

* Polynom operator+(const Polynom& right) const;

Вычисляет сумму полиномов.

* Polynom operator-(Polynom& right);

Вычисляет разность полиномов.

* Polynom operator\*(const Polynom& right);

Вычисляет умножение полиномов.

* Polynom operator\*(const double& val) const;

Вычисляет умножение полинома на число val.

* const Polynom& operator = (const Polynom& right);

Отчищает текущий полином и копирует в него размер и мономы полинома right.

* friend ostream& operator << (ostream& os, const Polynom& object);

Выводит полином.

**Заключение**

В данной лабораторной работе реализован программный комплекс, способный выполнять элементарные арифметические операции над полиномами одной переменной, причём степень каждой переменной является целым числом.

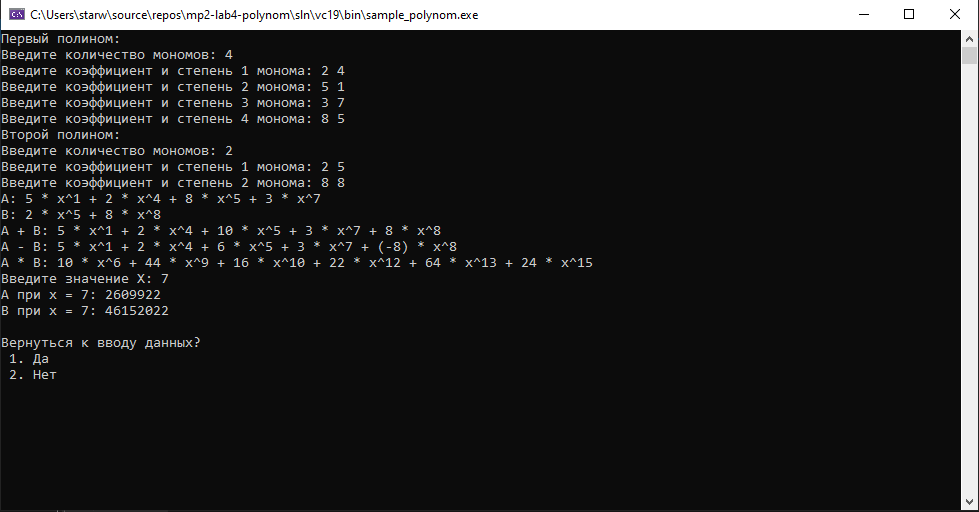
Достигнутые результаты:

1. Реализация требуемых арифметических операций (сложение, вычитание, умножение полиномов, вычисление полинома в заданной точке);
2. Корректное выполнение арифметических операций;
3. Диалоговое пользовательское приложение.

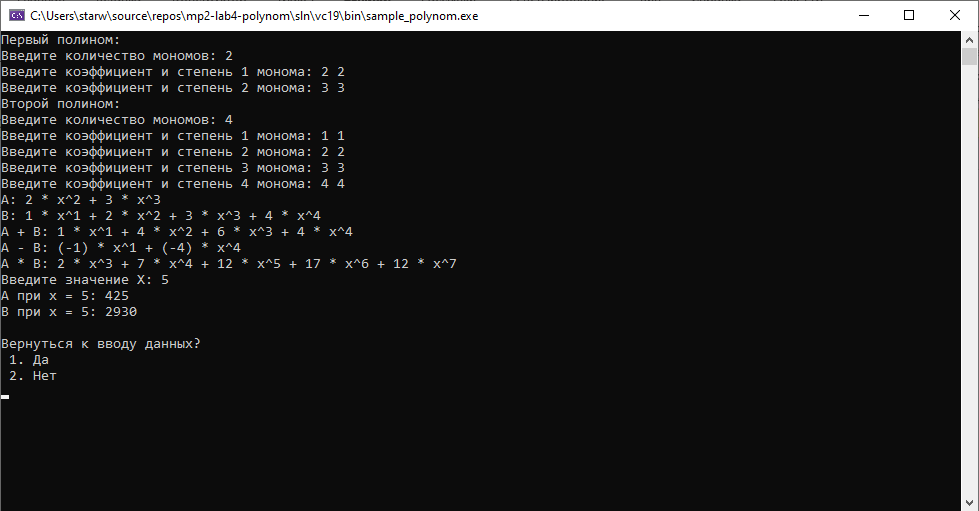
**Литература**

**Приложения**

Приложение 1



Приложение 2



## Фрагменты исходного кода программы

### **polynom.h**

#pragma once

#include <iostream>

using namespace std;

struct Monom {

double coef;

int pow;

Monom\* next;

Monom() { coef = 0; pow = 0; next = nullptr; }

Monom(double \_coef, int \_pow);

Monom(double \_coef, int \_pow, Monom\* \_next);

Monom(const Monom& monom);

bool operator==(const Monom& right);

};

class Polynom {

private:

Monom\* head;

int size;

public:

Polynom();

Polynom(const Polynom& polynom);

~Polynom();

int getSize() { return size; }

void clear();

void insert(Monom monom);

void insert(double coef, int pow);

void merge(const Polynom& polynom);

void pop\_front();

void push\_front(Monom monom);

void push\_back(Monom monom);

void del(Monom monom);

bool find(int pow);

int Calculate(int x);

Polynom operator+(const Polynom& right) const;

Polynom operator-(Polynom& right);

Polynom operator\*(const Polynom& right);

Polynom operator\*(const double& val) const;

const Polynom& operator = (const Polynom& right);

friend ostream& operator << (ostream& os, const Polynom& object);

};

### **polynom.cpp**

#include "polynom.h"

#include <cmath>

#define eps 0.00000001

Monom::Monom(double \_coef, int \_pow)

{

if (\_pow > 999 || \_pow < 0) {

throw (string)"Ошибка степени!";

}

coef = \_coef;

pow = \_pow;

next = 0;

}

Monom::Monom(double \_coef, int \_pow, Monom\* \_next)

{

coef = \_coef;

pow = \_pow;

next = \_next;

}

Monom::Monom(const Monom& monom)

{

coef = monom.coef;

pow = monom.pow;

next = monom.next;

}

bool Monom::operator==(const Monom& right)

{

if (pow == right.pow && abs(coef - right.coef) < eps) return true;

else return false;

}

Polynom::Polynom()

{

size = 0;

head = 0;

}

Polynom::Polynom(const Polynom& polynom) : Polynom()

{

Monom\* tmp = polynom.head;

while (tmp != 0)

{

this->insert(tmp->coef, tmp->pow);

tmp = tmp->next;

}

size = polynom.size;

}

Polynom::~Polynom()

{

clear();

}

inline void Polynom::clear()

{

while (size && head != 0) {

pop\_front();

}

}

void Polynom::merge(const Polynom& pol)

{

Polynom tmp;

Monom\* curr1 = (this)->head;

Monom\* curr2 = pol.head;

while (curr1 != 0 && curr2 != 0) {

if (curr1->pow <= curr2->pow) {

Monom newMonom(\*curr1);

newMonom.next = 0;

tmp.push\_back(newMonom);

curr1 = curr1->next;

}

else {

Monom newMonom(\*curr2);

newMonom.next = 0;

tmp.push\_back(newMonom);

curr2 = curr2->next;

}

}

while (curr1 != 0) {

Monom newMonom(\*curr1);

newMonom.next = 0;

tmp.push\_back(newMonom);

curr1 = curr1->next;

}

while (curr2 != 0) {

Monom newMonom(\*curr2);

newMonom.next = 0;

tmp.push\_back(newMonom);

curr2 = curr2->next;

}

\*this = tmp;

tmp.head = 0;

tmp.size = 0;

}

inline void Polynom::pop\_front()

{

Monom\* tmp = head;

head = head->next;

delete tmp;

size--;

}

inline void Polynom::push\_front(Monom monom)

{

head = new Monom(monom.coef, monom.pow, head);

size++;

}

void Polynom::push\_back(Monom monom)

{

if (head == 0) {

head = new Monom(monom);

}

else {

Monom\* curr = head;

while (curr->next != 0) {

curr = curr->next;

}

curr->next = new Monom(monom);

}

size++;

}

void Polynom::insert(Monom monom)

{

if (head == 0 || head->pow > monom.pow) {

push\_front(monom);

return;

}

Monom\* curr = head;

while (curr->next != 0 && curr->next->pow <= monom.pow && curr->pow <= monom.pow) {

curr = curr->next;

}

if (curr->pow == monom.pow) {

curr->coef += monom.coef;

if (abs(curr->coef) < eps) {

del(\*curr);

}

return;

}

Monom\* newObject = new Monom(monom);

newObject->next = curr->next;

curr->next = newObject;

size++;

}

void Polynom::insert(double coef, int pow)

{

Monom monom(coef, pow);

if (head == 0 || head->pow > monom.pow) {

push\_front(monom);

return;

}

Monom\* curr = head;

while (curr->next != 0 && curr->next->pow <= monom.pow && curr->pow <= monom.pow) {

curr = curr->next;

}

if (curr->pow == monom.pow) {

curr->coef += monom.coef;

if (abs(curr->coef) < eps) {

del(\*curr);

}

return;

}

Monom\* newObject = new Monom(monom);

newObject->next = curr->next;

curr->next = newObject;

size++;

}

void Polynom::del(Monom monom)

{

Monom\* curr = head;

if (curr->pow == monom.pow && abs(curr->coef - monom.coef) < eps) {

pop\_front();

return;

}

while (curr->next != 0 && curr->next->pow != monom.pow) {

curr = curr->next;

}

Monom\* tmp = curr->next->next;

delete curr->next;

curr->next = tmp;

size--;

}

bool Polynom::find(int pow)

{

Monom\* curr = head;

while (curr != 0) {

if (curr->pow == pow) {

return true;

}

curr = curr->next;

}

return false;

}

Polynom Polynom::operator\*(const double& val) const

{

Polynom tmp;

Monom\* a = this->head;

while (a != 0) {

tmp.insert(a->coef, a->pow);

a = a->next;

}

Monom\* temp = tmp.head;

while (temp != 0) {

temp->coef = temp->coef \* val;

temp = temp->next;

}

return tmp;

}

const Polynom& Polynom::operator = (const Polynom& right)

{

if (this->head != right.head) {

clear();

Monom\* tmp = right.head;

while (tmp != 0) {

insert(tmp->coef, tmp->pow);

tmp = tmp->next;

}

}

return \*this;

}

Polynom Polynom::operator + (const Polynom& right) const

{

Polynom curr;

Monom\* thisHead = this->head;

while (thisHead != 0) {

curr.insert(thisHead->coef, thisHead->pow);

thisHead = thisHead->next;

}

Monom\* rightHead = right.head;

while (rightHead != 0) {

curr.insert(rightHead->coef, rightHead->pow);

rightHead = rightHead->next;

}

return curr;

}

Polynom Polynom::operator \* (const Polynom& right)

{

Polynom curr;

Monom\* f = head;

Monom\* s = right.head;

while (f != 0) {

s = right.head;

while (s != 0)

{

double coef = s->coef \* f->coef;

int pow = s->pow + f->pow;

curr.insert(coef, pow);

s = s->next;

}

f = f->next;

}

return curr;

}

Polynom Polynom::operator-(Polynom& right)

{

Polynom c = right \* (-1.);

return \*this + c;

}

int Polynom::Calculate(int x)

{

Monom\* tmp = head;

int res = 0;

while (tmp != 0)

{

res += tmp->coef \* pow(x,tmp->pow);

tmp = tmp->next;

}

return res;

}

ostream& operator << (ostream& os, const Polynom& object)

{

if (object.head == 0) {

os << 0;

return os;

}

Monom\* temp = object.head;

while (temp != 0)

{

if (temp->coef > 0) {

os << temp->coef << " ";

}

else {

os << "(" << temp->coef << ") ";

}

if (temp->pow != 0) {

os << "\* x^" << temp->pow << " ";

}

if (temp->next != 0) {

os << "+ ";

}

temp = temp->next;

}

return os;

}