МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по лабораторной работе

**«Хранение и арифметические действия над полиномами»**

**Выполнила:** студентка группы 381603-1

Лакшина А.Р.

**Проверила:** кандидат тех. наук, старший преподаватель каф. МОСТ института ИТММ

Кустикова В.Д.

Нижний Новгород

2018

Оглавление

[Введение 3](#_Toc512113762)

[Постановка задачи 3](#_Toc512113763)

[Руководство пользователя 4](#_Toc512113764)

[Описание алгоритмов 5](#_Toc512113765)

[Описание структуры программы 6](#_Toc512113766)

[Описание структур данных 7](#_Toc512113767)

[Заключение 9](#_Toc512113768)

[Список литературы 9](#_Toc512113769)

[Приложение 10](#_Toc512113770)

## Введение

Полиномы как математический объект давно интересуют математиков. Уже в начале двадцатого века активно изучали оптимизацию операций над полиномами вычисление их значений.

Первооткрывателями в этой сфере были Карл Фридрих Гаусс, Маклорен, Эйлер и Декарт. Все они внесли непомерный вклад в развитие алгебры полиномов и алгебры в целом.

На сегодняшний день полиномы хорошо изучены в математике. Вычисление и привидение полиномов к каноническому виду очень важно для правильной работы многих алгоритмов, определяющих работу станков, военной и гражданской техники. Поэтому принципиальное значение имеют корректная работа операций над полиномами, скорость их выполнения и точность. Использование компьютера значительно упростит соответствие этим требованиям.

Важность обработки полиномов на ЭВМ неоспорима. Один из методов представлен в данной работе. Элементы представленной программы могут быть использованы при разработке других программных комплексов и приложений.

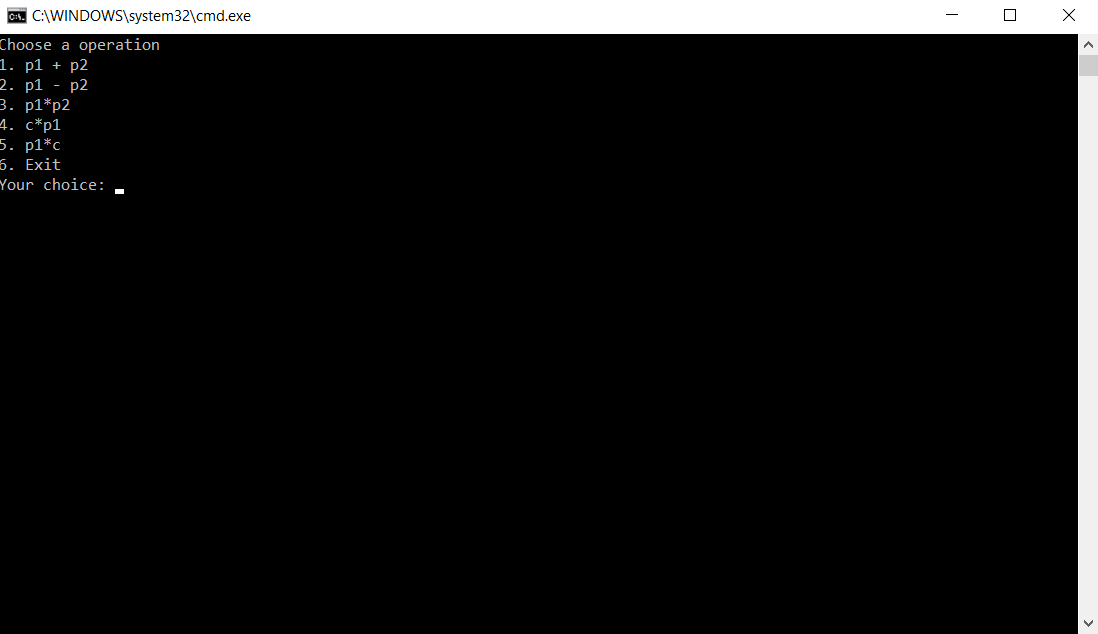
## Постановка задачи

Требуется разработать программу, выполняющую арифметические операции над полиномами от трех переменных (x,y,z). Полином – сумма мономов, степень каждого из которых от 0 до 9. Допустимые операции: сложение, вычитание, умножение на константу, перемножение двух полиномов. Коэффициенты перед мономами – вещественные числа.

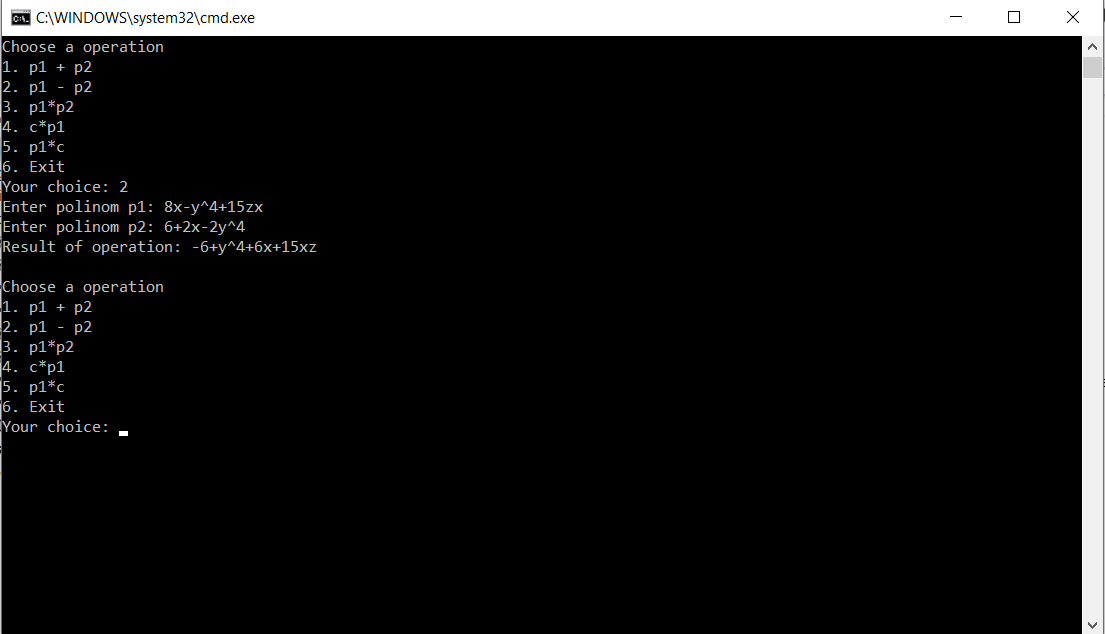
## Руководство пользователя

Для выполнения операций над полиномами необходимо открыть проект "Polinoms" и нажать Ctrl+F5.

На экране появится:



Далее нужно выбрать номер действия, которое будет производиться над полиномами, и ввести сами полиномы (или полином и константу). Степень обозначается через “^”:



На экран выведется результат операции. Далее можно повторить вычисления, выбрав любую операцию снова введя полиномы (или полином и константу).

Для выхода необходимо выбрать пункт 6 меню.

## Описание алгоритмов

1. Сложение полиномов

На входе имеем два полинома, которые представляют собой циклические списки с головой. Звено списка – моном.

1. Сдвигаем указатели обоих списков и списка-результата на звенья, следующие за головой.
2. Пока в обоих списках не дошли до конца:

Сравниваем звенья списков (обобщенные степени мономов, лежащих в звеньях);

Меньшее звено вставляем в конец списка-результата, сдвигаем указатели списка, из которого взяли элемент и списка-результата на одно звено вперёд;

Если звенья равны, создаём моном, коэффициент которого равен сумме коэффициентов мономов, лежащих в этих звеньях (обобщенная степень равна обобщенной степени монома, лежащего в одном из этих звеньев) и записываем его в конец списка-результата, сдвигаем указатели обоих списков и списка-результата на одно звено вперёд;

1. После того, как один из списков кончился, записываем остаток второго списка в результат;
2. Возвращаем полином-результат;
3. Нахождение разности полиномов

Реализуется посредством умножения «полинома-вычитаемого» на (-1) и сложения его с «полиномом-уменьшаемым»

1. Умножение полинома на константу

На входе имеем полином и константу.

Если константа не равна нулю:

Создаём полином-результат, равный исходному полиному;

В списке-результате сдвигаем указатель на звено, следующее за головой;

Пока не дошли до конца списка-результата:

Умножаем коэффициент монома, лежащего в текущем звене, на константу;

Сдвигаем указатель на следующее звено;

Возвращаем полином-результат;

1. Умножение полинома на полином

На входе имеем два полинома, которые представляют собой циклические списки с головой. Звено списка – моном.

1. Сдвигаем указатели обоих списков на звенья, следующие за головой.
2. Пока в первом списке не дошли до конца:

Создаём полином-копию второго полинома и сдвигаем в нём (рассматривая его как список) указатель на звено, следующее за головой;

Пока в нем не дошли до конца:

Если мономы в текущих звеньях первого списка и списка-копии второго удовлетворяют условию: сумма степеней “x” (“y”, “z”) одного и другого меньше 10, то складываем их обобщенные степени, перемножаем коэффициенты, результат записываем в текущее звено списка-копии второго полинома;

Иначе выводим сообщение об ошибке "large index";

Сдвигаем указатель в списке-копии второго полинома на следующее звено;

Полином-результат = полином-результат + полином-копия второго полинома;

Сдвигаем указатель в первом списке на следующее звено;

1. Возвращаем полином-результат;

## Описание структуры программы

Программа состоит из нескольких файлов:

1. **unit.h** содержит объявление шаблонного класса **unit** и его реализацию
2. **list.h**содержит объявление шаблонного класса **list** и его реализацию
3. **monom.h** содержит объявление класса **monom**
4. **monom.cpp** содержит реализацию класса **monom**
5. **polinom*.*h** содержит объявление класса **polynom**.
6. **polinom.cpp** содержит реализацию класса **polynom**.
7. **main.cpp** содержит реализацию пользовательского интерфейса.
8. **test\_list.cpp** содержит тесты на правильность реализации класса **list**.
9. **test\_ polinom.cpp** содержит тесты на правильность реализации класса **polynom** .

## Описание структур данных

В качестве структуры хранения полинома используется циклический список с головой. Звенья списка – мономы, однозначно определяемые через коэффициент и обобщенную степень. Голова – фиктивное звено, не содержащее монома. Элементы списка хранятся в порядке возрастания обобщённых степеней.

1. Шаблонный класс **unit** – звено списка

template<class type>

class unit

{

public:

type data; //Данные в звене

unit\* next; //Указатель на следующее звено

unit() { next = nullptr; } //Конструктор по умолчанию

unit(type dz) { data = dz; next = nullptr; } //Конструктор с параметром

bool operator< (const unit& z) const { return (data < z.data); } //Оператор <

bool operator> (const unit& z) const { return (data > z.data); } //Оператор >

};

1. Шаблонный класс **list** – список с головой

template<class type>

class list

{

private:

unit<type>\* head; //Указатель на голову

unit<type>\* act; //Указатель на текущий элемент

public:

void Clean(); //Очистка списка

list(); //Конструктор по умолчанию

list(const list<type>& a); //Конструктор копирования

~list(); //Деструктор

list<type>& operator=(const list<type> &a); //Перегрузка оператора

присваивания

void Insert(type elem); //Вставка в упорядоченный список

void InsertToTail(type elem); //Вставка в конец списка

bool operator==(const list<type>& sp) const; //Оператор = bool operator!=(const list<type>& sp) const { return !(\*this == sp); } //Оператор !=

// Навигация

void Reset() {act = head -> next;} //Поставить указатель на элемент, следующий за головой

void Step() {act = act -> next;} //Поставить указатель на элемент, следующий за текущим

unit<type>\* GetAct() const {return act;} //Вернуть указатель на текущее звено

bool IsNotOver() const { return !(act == head); } //Проверить, что не дошли до конца списка

};

1. Класс **monom**

Обобщённая степень монома – число, число сотен которого – это степень “x”, число десятков - степень “y”, число единиц - степень “z”.

class monom

{

public:

double coeff; //Коэффициент перед мономом

unsigned int abc; //Обобщённая степень

monom(double c = 0, unsigned int a = 0); //Конструктор

monom& operator=(const monom& m); //Оператор присваивания

//Операторы сравнения. Сравниваются обобщенные степени мономов

bool operator< (const monom& m) const;

bool operator> (const monom& m) const;

bool operator==(const monom& m) const;

bool operator!=(const monom& m) const;

};

1. Класс **polynom**

class polynom

{

private:

list<monom> list\_pol; //Полином - список из мономов

public:

list<monom> similar\_terms(list <monom> sp); //Приведение подобных слагаемых

polynom(const string pol = "" ); //Разбор строки

polynom(const polynom& pol); //Конструктор копирования

polynom& operator=(const polynom &pol); //Присваивание

polynom operator+(const polynom& pol) const; //Сложение полиномов

polynom operator\*(const double a) const; //Умножение на константу слева friend polynom operator\* (const double a, const polynom& pol) { return pol\*a; } //Умножение на константу справа

polynom operator-(const polynom& pol) const { return \*this + pol\*(-1.0);}

//Разность полиномов

polynom operator\*(const polynom& pol) const; //Умножение полиномов

polynom operator-() const { return (-1.0)\*(\*this); } //Унарный минус

friend ostream& operator<<(ostream &ostr, const polynom &pol); //Оператор вывода

bool operator==(const polynom& pol) const { return list\_pol == pol.list\_pol; } //Оператор =

bool operator!=(const polynom& pol) const { return list\_pol != pol.list\_pol; } //Оператор !=

polynom(list<monom> &list) : list\_pol(list) {} //Конструктор по списку

};

## Заключение

В лабораторной работе был реализован программный комплекс, выполняющий арифметические операции над полиномами. Реализовано хранение полиномов на основе циклического односвязного списка с головой. Программа, как и требовалось, поддерживает операции над полиномами: сложение, вычитание, умножение на константу и перемножение двух полиномов.

## Список литературы

1. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К*. Алгоритмы: построение и анализ, Москва 2013.
2. Рабочие материалы к учебному курсу «Методы программирования». *Гергель В.П*. 2002 г.

## Приложение

**unit.h**

#pragma once

//Звено списка

template<class type>

class unit

{

public:

type data; //Данные в элементе

unit\* next; //Указатель на следующий элемент

unit() { next = nullptr; } //Конструктор по умолчанию

unit(type dz) { data = dz; next = nullptr; } //Конструктор с параметром

bool operator< (const unit& z) const { return (data < z.data); }

bool operator> (const unit& z) const { return (data > z.data); }

};

**list.h**

#pragma once

#include "unit.h"

//Циклический список с головой

template<class type>

class list

{

private:

unit<type>\* head; //Указатель на голову

unit<type>\* act; //Указатель на текущий

public:

void Clean();

list(); //Конструктор по умолчанию

list(const list<type>& a); //Конструктор копирования

~list(); //Деструктор

list<type>& operator=(const list<type> &a);//Перегрузка оператора присваивания

void Insert(type elem); //Вставка в упорядоченный список

void InsertToTail(type elem); //Вставка в конец

bool operator==(const list<type>& sp) const;

bool operator!=(const list<type>& sp) const { return !(\*this == sp); }

// Навигация

void Reset() {act = head -> next;}

void Step() {act = act -> next;}

unit<type>\* GetAct() const {return act;}

bool IsNotOver() const { return !(act == head); }

};

//Очистка списка

template <class type>

void list<type>::Clean()

{

unit<type>\* actual = head->next;

while (actual != head)

{

unit<type>\* temp = actual->next;

delete actual;

actual = temp;

}

head->next = head;

}

//Конструктор по умолчанию

template <class type>

list<type>::list()

{

head = new unit<type>;

head->next = head;

act = head;

}

//Конструктор копирования

template <class type>

list<type>::list(const list<type>& a)

{

head = new unit<type>;

unit<type>\* A = a.head;

unit<type>\* B = head;

if ( A->next == a.head)

{

head->next = head;

return;

}

while (A->next != a.head)

{

A = A->next;

B->next = new unit<type>(A->data);

B = B->next;

}

B->next = head;

act = head -> next;

}

//Деструктор

template <class type>

list<type>::~list()

{

Clean();

delete head;

}

//Оператор присваивания

template <class type>

list<type>& list<type>::operator=(const list<type>& a)

{

Clean();

unit<type>\* A = a.head;

unit<type>\* B = head;

while (A->next != a.head)

{

A = A->next;

B->next = new unit<type>(A->data);

B = B->next;

}

B->next = head;

act = head;

return \*this;

}

//Вставка в упорядоченный список

template <class type>

void list<type>::Insert(type elem)

{

unit<type>\* actual = head;

unit<type>\* el = new unit<type>(elem);

while ((actual->next != head) && (\*(actual->next) < \*el))

actual = actual->next;

unit<type>\* actual\_2 = actual->next;

actual->next = el;

actual->next->next = actual\_2;

}

// Оператор равно

template<class type>

bool list<type>::operator==(const list<type>& sp) const

{

bool res = true;

if (this != &sp)

{

unit<type>\* a = head->next;

unit<type>\* b = sp.head->next;

while (a->data == b->data && a != head && b != sp.head)

{

a = a->next;

b = b->next;

}

if (a != head || b != sp.head)

res = false;

}

return res;

}

// Вставка в конец

template<class type>

void list<type> :: InsertToTail(type elem)

{

Reset();

while (act ->next != head)

Step();

unit<type>\* temp = act->next;

act->next = new unit<type>(elem);

act->next->next = temp;

}

**monom.h**

#pragma once

class monom

{

public:

double coeff; //Коэффициент перед мономом

unsigned int abc; //Обобщённая степень

monom(double c = 0, unsigned int a = 0); //Конструктор

monom& operator=(const monom& m); //Оператор присваивания

bool operator< (const monom& m) const;

bool operator> (const monom& m) const;

bool operator==(const monom& m) const;

bool operator!=(const monom& m) const;

};

**monom.cpp**

#include "monom.h"

monom:: monom(double c, unsigned int a)

{

coeff = c;

abc = a;

}

monom& monom:: operator=(const monom& m)

{

coeff = m.coeff;

abc = m.abc;

return \*this;

}

bool monom:: operator< (const monom& m) const

{

bool res = true;

if (abc >= m.abc)

res=false;

return res;

}

bool monom:: operator> (const monom& m) const

{

bool res = true;

if (abc <= m.abc)

res=false;

return res;

}

bool monom:: operator==(const monom& m) const

{

bool res = true;

if (abc != m.abc || coeff != m.coeff )

res=false;

return res;

}

bool monom:: operator!=(const monom& m) const

{

return !(\*this == m);

}

**polinom.h**

#pragma once

#include "monom.h"

#include "list.h"

#include <iostream>

#include <string>

using namespace std;

class polynom

{

private:

list<monom> list\_pol; //Полином - список из мономов

public:

list<monom> similar\_terms(list <monom> sp); //Подобные слагаемые

polynom(const string pol = "" ); //Разбор строки

polynom(const polynom& pol); //Конструктор копирования

polynom& operator=(const polynom &pol); //Присваивание

polynom operator+(const polynom& pol) const; //Сложение полиномов

polynom operator\*(const double a) const; //Умножение на константу слева friend polynom operator\*(const double a,const polynom& pol) { return pol\*a; } //Умножение на константу справа

polynom operator-(const polynom& pol) const { return \*this + pol\*(-1.0); } //Разность полиномов

polynom operator\*(const polynom& pol) const; //Умножение полиномов

polynom operator-() const { return (-1.0)\*(\*this); } //Унарный минус

friend ostream& operator<<(ostream &ostr, const polynom &pol); //Оператор вывода

bool operator==(const polynom& pol) const { return list\_pol == pol.list\_pol; }

bool operator!=(const polynom& pol) const { return list\_pol != pol.list\_pol; }

polynom(list<monom> &list) : list\_pol(list) {} //Конструктор по списку

};

**polinom.cpp**

#include "polinom.h"

using namespace std;

list<monom> polynom::similar\_terms(list <monom> sp)

{

list<monom> res;

res.Reset();

sp.Reset();

unit<monom> mon(sp.GetAct()->data.coeff);

while (sp.IsNotOver())

{

mon.data.abc = sp.GetAct()->data.abc;

if (sp.GetAct()->data.abc == sp.GetAct()->next->data.abc && (sp.GetAct()->next->data.coeff || sp.GetAct()->next->data.abc))

mon.data.coeff += sp.GetAct()->next->data.coeff;

else

{

if (mon.data.coeff)

{

res.InsertToTail(mon.data);

res.Step();

}

mon.data.coeff = sp.GetAct()->next->data.coeff;

}

sp.Step();

}

return res;

}

//Разбор строки

polynom::polynom(string pol)

{

list<monom> res;

while (pol.length())

{

string part;

monom temp;

int pos = 1;

while (pos < pol.length() && pol[pos] != '+' && pol[pos] != '-')

pos++;

part = pol.substr(0, pos); // substr возвращает строку, являющуюся подстрокой исходной строки (начиная с какого, сколько)

pol.erase(0, pos); // erase удаляет из строки последовательность символ заданной длины, начиная с указанной позиции (начиная с какого, сколько)

pos = 0;

while (part[pos] != 'x' && part[pos] != 'y' && part[pos] != 'z' && pos < part.length())

pos++;

string c = part.substr(0,pos); // с - коэф part

if (c == "+" || c.length() == 0)

temp.coeff = 1;

else

if (c == "-")

temp.coeff = -1;

else

temp.coeff = stod(c); // stod извлекает число с плавающей точкой из строки

part.erase(0, pos); // удаляем коэф

part += ' ';

int a[3] = { 100,10,1 };

for (int i = 0; i < 3; i++)

{

pos = part.find((char)(120 + i)); // 120 - код символа x, 121 - y, 122 - z

if (pos > -1)

{

if (part[pos + 1] != '^')

part.insert(pos + 1, "^1");

temp.abc += a[i] \* stoi(part.substr(pos + 2, 1)); // stoi преобразует последовательность символов в целое число.

part.erase(pos, 3);

}

}

list\_pol.Insert(temp);

}

list\_pol = similar\_terms(list\_pol);

}

//Конструктор копирования

polynom:: polynom (const polynom& pol)

{

list\_pol = pol.list\_pol;

}

//Присваивание

polynom& polynom:: operator=(const polynom &pol)

{

list\_pol = pol.list\_pol;

return \*this;

}

//Сложение полиномов

polynom polynom::operator+(const polynom& pol) const

{

polynom res;

polynom pthis = \*this;

polynom p = pol;

pthis.list\_pol.Reset();

p.list\_pol.Reset();

res.list\_pol.Reset();

while (pthis.list\_pol.IsNotOver() && p.list\_pol.IsNotOver())

{

if (pthis.list\_pol.GetAct()->data > p.list\_pol.GetAct()->data)

{

res.list\_pol.InsertToTail(p.list\_pol.GetAct()->data);

p.list\_pol.Step();

res.list\_pol.Step();

}

else

if (pthis.list\_pol.GetAct()->data < p.list\_pol.GetAct()->data)

{

res.list\_pol.InsertToTail(pthis.list\_pol.GetAct()->data);

pthis.list\_pol.Step();

res.list\_pol.Step();

}

else

{

double new\_coeff = pthis.list\_pol.GetAct()->data.coeff + p.list\_pol.GetAct()->data.coeff;

if (new\_coeff)

{

monom temp(new\_coeff, pthis.list\_pol.GetAct()->data.abc);

res.list\_pol.InsertToTail(temp);

res.list\_pol.Step();

}

pthis.list\_pol.Step();

p.list\_pol.Step();

}

}

while (pthis.list\_pol.IsNotOver())

{

res.list\_pol.InsertToTail(pthis.list\_pol.GetAct()->data);

pthis.list\_pol.Step();

res.list\_pol.Step();

}

while (p.list\_pol.IsNotOver())

{

res.list\_pol.InsertToTail(p.list\_pol.GetAct()->data);

p.list\_pol.Step();

res.list\_pol.Step();

}

return res;

}

//Умножение на константу слева

polynom polynom::operator\*(const double a) const

{

polynom res;

if (a)

{

res = \*this;

res.list\_pol.Reset();

while (res.list\_pol.IsNotOver())

{

res.list\_pol.GetAct()->data.coeff \*= a;

res.list\_pol.Step();

}

}

return res;

}

//Умножение полиномов

polynom polynom::operator\*(const polynom& pol) const

{

polynom res;

polynom pthis = \*this;

polynom p = pol;

pthis.list\_pol.Reset();

p.list\_pol.Reset();

while (pthis.list\_pol.IsNotOver())

{

double pthis\_coeff = pthis.list\_pol.GetAct()->data.coeff;

int pthis\_abc = pthis.list\_pol.GetAct()->data.abc;

polynom temp(pol);

temp.list\_pol.Reset();

while (temp.list\_pol.IsNotOver())

{

int temp\_abc = temp.list\_pol.GetAct()->data.abc;

if ((temp\_abc % 10 + pthis\_abc % 10) < 10 && (temp\_abc/10 % 10 + pthis\_abc/10 % 10) < 10 && (temp\_abc/100 + pthis\_abc/100) < 10)

{

temp.list\_pol.GetAct()->data.abc += pthis\_abc;

temp.list\_pol.GetAct()->data.coeff \*= pthis\_coeff;

}

else

throw "large index";

temp.list\_pol.Step();

}

res = res + temp;

pthis.list\_pol.Step();

}

return res;

}

//Оператор вывода

ostream& operator<<(ostream &ostr,const polynom& pol)

{

polynom p = pol;

p.list\_pol.Reset();

while (p.list\_pol.IsNotOver())

{

monom temp = p.list\_pol.GetAct()->data;

if (temp.coeff > 0)

{

ostr << "+";

if (temp.coeff == 1 && temp.abc ==0)

ostr << "1";

else

if (temp.coeff != 1)

ostr << temp.coeff;

}

else

ostr << temp.coeff;

int a = temp.abc / 100;

if (a>1)

ostr << "x^" << a;

else

if (a == 1)

ostr << "x";

a = temp.abc / 10 % 10;

if (a>1)

ostr << "y^" << a;

else

if (a == 1)

ostr << "y";

a = temp.abc % 10;

if (a>1)

ostr<< "z^" << a;

else

if (a == 1)

ostr << "z";

p.list\_pol.Step();

}

return ostr;

}

**main.cpp**

#include "polinom.h"

#include <iostream>

#include <string>

using namespace std;

char menu()

{

char choice;

cout<<"Choose a operation"<<endl;

cout<<"1. p1 + p2"<<endl;

cout<<"2. p1 - p2"<<endl;

cout<<"3. p1\*p2"<<endl;

cout<<"4. c\*p1"<<endl;

cout<<"5. p1\*c"<<endl;

cout<<"6. Exit"<<endl;

cout << "Your choice: ";

cin >> choice;

return choice;

}

int main()

{

string s1,s2;

polynom p1, p2;

double c;

char number;

do

{

number = menu();

switch(number)

{

case '1':

{

cout << "Enter polinom p1: ";

cin >> s1;

p1 = s1;

cout << "Enter polinom p2: ";

cin >> s2;

p2 = s2;

cout << "Result of operation: " << p1 + p2 << endl << endl;

break;

}

case '2':

{

cout << "Enter polinom p1: ";

cin >> s1;

p1 = s1;

cout << "Enter polinom p2: ";

cin >> s2;

p2 = s2;

cout << "Result of operation: " << p1 - p2 << endl << endl;

break;

}

case '3':

{

cout << "Enter polinom p1: ";

cin >> s1;

p1 = s1;

cout << "Enter polinom p2: ";

cin >> s2;

p2 = s2;

cout << "Result of operation: " << p1 \* p2 << endl << endl;

break;

}

case '4':

{

cout << "Enter polinom p1: ";

cin >> s1;

p1 = s1;

cout << "Enter const c: ";

cin >> c;

cout << "Result of operation: " << c\*p1 << endl << endl;

break;

}

case '5':

{

cout << "Enter polinom p1: ";

cin >> s1;

p1 = s1;

cout << "Enter const c: ";

cin >> c;

cout << "Result of operation: " << p1\*c << endl << endl;

break;

}

case '6':

{

break;

}

}

}

while (number != '6');

return 0;

}

**test\_list.cpp**

#include "gtest.h"

#include "list.h"

class Test\_List : public ::testing::Test

{

protected:

list<int> A;

list<int> B;

public:

Test\_List() { B.Insert(5); B.Insert(1);}

~Test\_List() {}

};

// Пустой список

TEST(lists, can\_create\_empty\_list)

{

ASSERT\_NO\_THROW(list<int> l);

}

TEST\_F(Test\_List, created\_empty\_list\_is\_empty)

{

EXPECT\_EQ(true, !A.IsNotOver());

}

TEST\_F(Test\_List, can\_copy\_empty\_list)

{

ASSERT\_NO\_THROW(list<int> l(A));

}

TEST\_F(Test\_List, empty\_list\_copy\_is\_correct\_1)

{

list<int> l(A);

EXPECT\_NE(A.GetAct(), l.GetAct());

}

TEST\_F(Test\_List, empty\_list\_copy\_is\_correct\_2)

{

list<int> l(A);

EXPECT\_EQ(true, !A.IsNotOver());

}

TEST\_F(Test\_List, can\_assign\_empty\_list)

{

ASSERT\_NO\_THROW(list<int> l = A);

}

TEST\_F(Test\_List, assigned\_empty\_list\_is\_correct)

{

list<int> l = A;

EXPECT\_NE(A.GetAct(), l.GetAct());

}

TEST\_F(Test\_List, can\_compare\_list\_with\_itself)

{

EXPECT\_EQ(true, A == A);

}

TEST\_F(Test\_List, empty\_lists\_are\_equal)

{

list<int> l(A);

EXPECT\_EQ(true, l == A);

}

TEST\_F(Test\_List, can\_insert\_in\_empty\_list)

{

ASSERT\_NO\_THROW(A.Insert(7));

}

TEST\_F(Test\_List, insert\_in\_empty\_list\_is\_correct\_1)

{

A.Insert(7);

A.Reset();

EXPECT\_EQ(A.GetAct(), A.GetAct() -> next -> next );

}

TEST\_F(Test\_List, insert\_in\_empty\_list\_is\_correct\_2)

{

A.Insert(7);

A.Reset();

EXPECT\_EQ(7, A.GetAct() -> data);

}

// Непустой список

TEST\_F(Test\_List, can\_copy\_not\_empty\_list)

{

ASSERT\_NO\_THROW(list<int> l(B));

}

TEST\_F(Test\_List, not\_empty\_list\_is\_copied\_correctly\_1)

{

list<int> l(B);

EXPECT\_NE(B.GetAct(), l.GetAct());

}

TEST\_F(Test\_List, not\_empty\_list\_is\_copied\_correctly\_2)

{

list<int> l(B);

EXPECT\_EQ(l.GetAct(), l.GetAct()->next->next->next);

}

TEST\_F(Test\_List, not\_empty\_list\_is\_copied\_correctly\_3)

{

list<int> l(B);

l.Reset();

EXPECT\_EQ(1, l.GetAct()->data);

EXPECT\_EQ(5, l.GetAct()->next->data);

}

TEST\_F(Test\_List, can\_assign\_not\_empty\_list)

{

ASSERT\_NO\_THROW(list<int> l = B);

}

TEST\_F(Test\_List, assigned\_not\_empty\_list\_is\_correct\_1)

{

list<int> l = B;

EXPECT\_NE(l.GetAct(), B.GetAct());

}

TEST\_F(Test\_List, assigned\_not\_empty\_list\_is\_correct\_2)

{

list<int> l = B;

EXPECT\_EQ(l.GetAct(), l.GetAct()->next->next->next);

}

TEST\_F(Test\_List, assigned\_not\_empty\_list\_is\_correct\_3)

{

list<int> l = B;

l.Reset();

EXPECT\_EQ(1, l.GetAct()->data);

EXPECT\_EQ(5, l.GetAct()->next->data);

}

TEST\_F(Test\_List, not\_empty\_lists\_are\_equal)

{

list<int> l(B);

EXPECT\_EQ(true, l == B);

}

TEST\_F(Test\_List, not\_empty\_and\_empty\_lists\_are\_not\_equal)

{

EXPECT\_EQ(false, B == A);

}

TEST\_F(Test\_List, insert\_to\_the\_beginning\_1)

{

B.Insert(0);

EXPECT\_EQ(B.GetAct()->next->next->next->next, B.GetAct());

}

TEST\_F(Test\_List, insert\_to\_the\_beginning\_2)

{

B.Insert(0);

B.Reset();

EXPECT\_EQ(0, B.GetAct()->data);

EXPECT\_EQ(1, B.GetAct()->next->data);

EXPECT\_EQ(5, B.GetAct()->next->next->data);

}

TEST\_F(Test\_List, insert\_in\_the\_middle\_1)

{

B.Insert(3);

EXPECT\_EQ(A.GetAct()->next->next->next->next, A.GetAct());

}

TEST\_F(Test\_List, insert\_in\_the\_middle\_2)

{

B.Insert(3);

B.Reset();

EXPECT\_EQ(1, B.GetAct()->data);

EXPECT\_EQ(3, B.GetAct()->next->data);

EXPECT\_EQ(5, B.GetAct()->next->next->data);

}

TEST\_F(Test\_List, insert\_in\_the\_end\_1)

{

B.Insert(8);

EXPECT\_EQ(B.GetAct()->next->next->next->next, B.GetAct());

}

TEST\_F(Test\_List, insert\_in\_the\_end\_2)

{

B.Insert(8);

B.Reset();

EXPECT\_EQ(1, B.GetAct()->data);

EXPECT\_EQ(5, B.GetAct()->next->data);

EXPECT\_EQ(8, B.GetAct()->next->next->data);

}

**test\_polinom.cpp**

#include "gtest.h"

#include "polinom.h"

#include <vector>

using namespace std;

TEST(polynom, degree\_not\_more\_than\_9\_1 )

{

polynom a("x^4"), b("x^5");

ASSERT\_NO\_THROW(a\*b);

}

TEST(polynom, degree\_not\_more\_than\_9\_2 )

{

polynom a("x^5"), b("x^5");

ASSERT\_ANY\_THROW(a\*b);

}

struct calc\_st

{

string a,b,res;

double c;

calc\_st(string s, string ia, string ib = "", double ic = 0)

{

a = ia;

b = ib;

c = ic;

res = s;

}

};

// Сумма полиномов

class sum\_cl : public ::testing::TestWithParam<calc\_st>

{

protected:

polynom pa, pb, pres;

public:

sum\_cl() : pa(GetParam().a), pb(GetParam().b), pres(GetParam().res) {}

~sum\_cl() {}

};

TEST\_P(sum\_cl, sum\_of\_polynoms)

{

EXPECT\_EQ(pres, pa+pb);

}

INSTANTIATE\_TEST\_CASE\_P(Inst2, sum\_cl,::testing::Values(

calc\_st("", "0.9", "-0.9"),

calc\_st("", "-6y-2", "6y+2"),

calc\_st("", "2.1zy", "-2.1zy"),

calc\_st("2x","2x", ""),

calc\_st("x+z","x","z"),

calc\_st("7x", "10x", "-3x"),

calc\_st("8zy^3+2x", "5zy^3", "3zy^3+2x"),

calc\_st("2xy+z+1", "9+xy+5z", "xy-8-4z"),

calc\_st("0.5x+z", "8.5x+y", "-8x-y+z")

));

// Умножение полинома на полином

class mult\_pol\_cl : public ::testing::TestWithParam<calc\_st>

{

protected:

polynom pa, pb, pres;

public:

mult\_pol\_cl() : pa(GetParam().a), pb(GetParam().b), pres(GetParam().res) {}

~mult\_pol\_cl() {}

};

TEST\_P(mult\_pol\_cl, polynom\_mult\_polynom)

{

EXPECT\_EQ(pres, pa \* pb);

}

INSTANTIATE\_TEST\_CASE\_P(Inst3, mult\_pol\_cl,::testing::Values(

calc\_st("48","2", "24"),

calc\_st("x","x", "1"),

calc\_st("xz","x", "z"),

calc\_st("15x^3", "5x^2", "3x"),

calc\_st("34.02xz", "8.1x", "4.2z"),

calc\_st("25x^2z^5", "-5x", "-5xz^5"),

calc\_st("-2z^2-9z+4yz+20y+5", "1+4y-2z", "z+5"),

calc\_st("xz^2+2xyz+xy^2z","xz", "z+2y+y^2"),

calc\_st("10+5z+2xz+xz^2","xz+5", "z+2")

));

// Умножение полинома на константу

class mult\_const\_cl : public ::testing::TestWithParam<calc\_st>

{

protected:

polynom p, pres;

double cn;

public:

mult\_const\_cl() : p(GetParam().a), pres(GetParam().res) { cn = GetParam().c; }

~mult\_const\_cl() {}

};

TEST\_P(mult\_const\_cl, polynom\_mult\_const)

{

EXPECT\_EQ(pres, p \* cn);

}

INSTANTIATE\_TEST\_CASE\_P(Inst4, mult\_const\_cl,::testing::Values(

calc\_st("", "x", "", 0),

calc\_st("-4z", "z", "", -4),

calc\_st("12x-18", "-6x+9", "", -2),

calc\_st("16xy+104x-8", "8xy+52x-4", "", 2),

calc\_st("4.5yz+18x+1.5y", "3yz+12x+y", "", 1.5)

));