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 учебной практике

**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

**Содержание**

[Введение 3](#_Toc515330722)

[1 Постановка задачи 4](#_Toc515330723)

[2 Руководство пользователя 5](#_Toc515330724)

[3 Руководство программиста 7](#_Toc515330725)

[3.1 Структура программы 7](#_Toc515330726)

[3.2 Структуры данных 8](#_Toc515330727)

[3.3 Программная реализация 11](#_Toc515330728)

[Заключение 14](#_Toc515330729)

[Литература 15](#_Toc515330730)

[Приложение 16](#_Toc515330731)

[Приложение А. Программная реализация списка с головой 16](#_Toc515330732)

[Приложение Б. Программная реализация полиномов 18](#_Toc515330733)

[Приложение В. Программная реализация приложения 23](#_Toc515330734)

# 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.

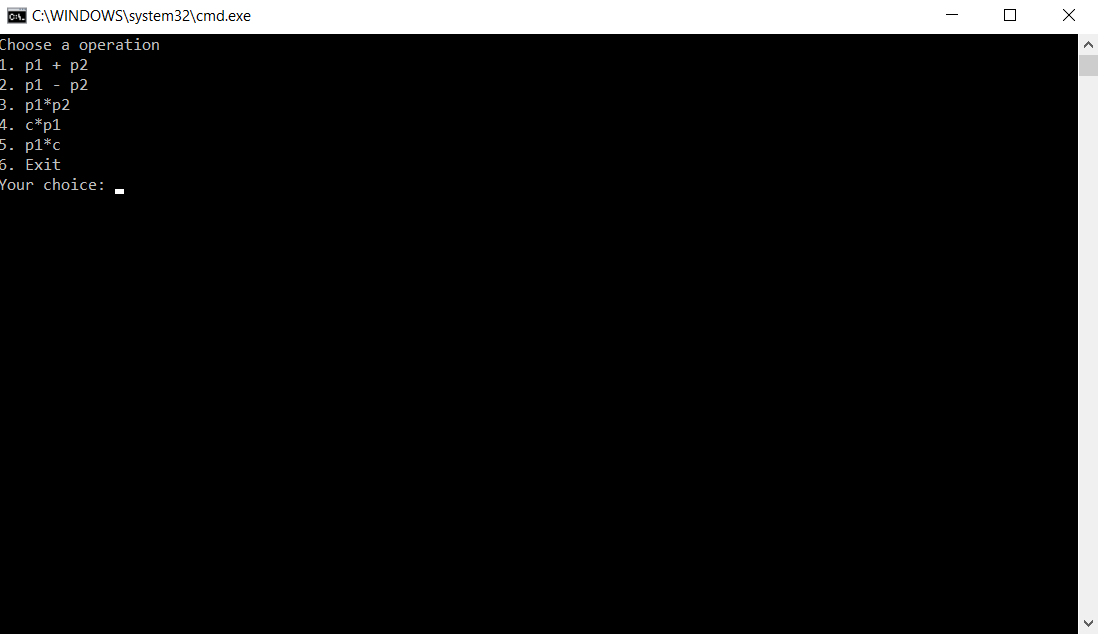
# Постановка задачи

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

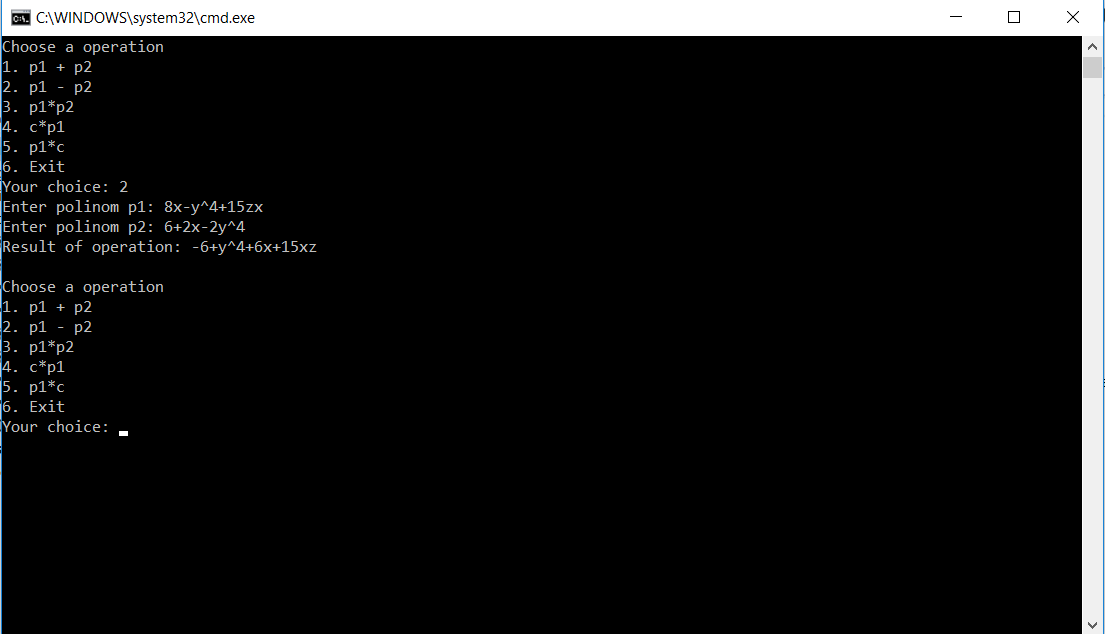
Запустите программу Polynom.exe.

После запуска программы на экране появится главное меню (рис. 1).



1. Меню программы

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



1. Результат работы программы

На экран выведется результат операции (рис. 2). Далее можно повторить вычисления, выбрав любую операцию и снова введя полиномы (или полином и константу).

Для выхода необходимо выбрать пункт 6 меню.

# Руководство программиста

## Структура программы

Проекты и файлы, из которых состоит программа:

1. gtest — проект, содержащий Google Tests Framework:

gtest.h — заголовочный файл для Google Tests Framework

gtest-all.cc — файл с кодом Google Tests Framework

1. list — проект, содержащий реализацию списка с «головой»:

unit.h — заголовочный файл, в котором находится объявление и реализация шаблонного класса unit (узел списка)

list.h — заголовочный файл, в котором содержится объявление и реализация шаблонного класса list (список «с головой»)

1. polinom — проект, содержащий реализацию полиномов:

monom.h — заголовочный файл, содержащий объявление класса monom

polinom.h — заголовочный файл, содержащий объявление класса polynom

monom.cpp — файл, содержащий реализацию класса monom

polynom.cpp — файл, содержащий реализацию класса polynom

1. sample — проект, содержащий реализацию пользовательского интерфейса:

main.cpp — файл исходного кода пользовательского интерфейса

1. tests — проект, содержащий тесты на правильность реализации списков и полиномов:

test\_main.cpp — файл запуска всех тестов

test\_list.cpp — файл, содержащий тесты на правильность реализации класса list

test\_polinom.cpp — файл, содержащий тесты на правильность реализации класса polynom

## Структуры данных

### Класс список

Поля класса:

* указатель на «голову» списка
* указатель на текущий элемент

Описание алгоритмов:

*Упорядоченная вставка в циклический список с «головой»*

На вход подаются данные для хранения в новом узле списка.

Алгоритм:

1. Создать указатель actual, установить на голову списка;
2. Создать указатель el, выделяя память под новый узел;
3. До тех пор пока список не кончился и данные следующего после actual узла меньше входных, смещаем actual;
4. Вставить el после actual.

### Класс полином

Поля класса:

* список из мономов

Описание алгоритмов:

*Разбиение строки на мономы*

На входе строка, которую будем преобразовывать в полином

Алгоритм:

1. Найти в данной строке “+”, “-” или конец. Если фрагмент до этого ненулевой, то выделить его как подстроку part;
2. Найти в part “x”, “y” , “z” и выделить символы до переменной как коэффициент. Если символов нет или этот символ  — “+”, то коэффициент равен единице, если этот символ  — “-”, то коэффициент равен минус единице;
3. Если коэффициент не равен нулю, то нужно найти “x”, “y”, и “z” и соответствующие степени после знака “^”, если этого знака нет, то степень равна единице;
4. Сформированный моном вставить в список мономов res;
5. Если строка не кончилась, то нужно перейти к пункту 1.

*Сложение полиномов*

На входе два полинома

1. Сдвигаем указатели обоих списков и списка-результата на звенья, следующие за головой;
2. Пока в обоих списках не дошли до конца:
   1. Сравниваем звенья списков (обобщенные степени мономов, лежащих в звеньях);
   2. Меньшее звено вставляем в конец списка-результата, сдвигаем указатели списка, из которого взяли элемент и списка-результата на одно звено вперёд;
   3. 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;
3. После того, как один из списков кончился, записываем остаток второго списка в результат;
4. Возвращаем полином-результат.

*Разность полиномов*

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

*Умножение полинома на константу*

На входе полином и константа

1. Если константа не равна нулю:
   1. Создаём полином-результат, равный исходному полиному;
   2. В списке-результате сдвигаем указатель на звено, следующее за головой;
   3. Пока не дошли до конца списка-результата:
      1. Умножаем коэффициент монома, лежащего в текущем звене, на константу;
      2. Сдвигаем указатель на следующее звено;
2. Возвращаем полином-результат.

*Умножение полинома на полином*

На входе два полинома

1. Сдвигаем указатели обоих списков на звенья, следующие за головой;
2. Пока в первом списке не дошли до конца:
   1. Создаём полином-копию второго полинома и сдвигаем в нём (рассматривая его как список) указатель на звено, следующее за головой;
   2. Пока в нём не дошли до конца:
      1. 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;
      2. Иначе выводим сообщение об ошибке "large index";
      3. Сдвигаем указатель в списке-копии второго полинома на следующее звено;
   3. Полином-результат = полином-результат + полином-копия второго полинома;
   4. Сдвигаем указатель в первом списке на следующее звено;
3. Возвращаем полином-результат.

## Программная реализация

### Класс unit

unit — шаблонный класс, является узлом списка. Параметр type — тип данных, хранящихся в узле.

*Поля класса:*

type data — данные, хранящиеся в звене

unit\* next – указатель на следующий элемент

*Методы класса:*

unit() — конструктор по умолчанию

unit(type dz) — конструктор с параметром

bool operator< (const unit& z) const  — оператор <

bool operator> (const unit& z) const — оператор >

### Класс list

list – шаблонный класс, циклический список с “головой”. Параметр type — тип данных, хранящихся в списке

*Поля:*

unit<type>\* head — указатель на “голову” списка

unit<type>\* act — указатель на текущий элемент

*Методы:*

void Clean() — очистка списка

list() — конструктор по умолчанию

list(const list<type>& a) — конструктор копирования

~list() — деструктор

list<type>& operator=(const list<type> &a) — оператор 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 — оператор !=

void Reset() — поставить указатель на элемент, следующий за головой

void Step() — поставить указатель на элемент, следующий за текущим

unit<type>\* GetAct() const — вернуть указатель на текущее звено

bool IsNotOver() const — проверить, что не дошли до конца списка

### Класс monom

monom –класс, представляет собой моном — слагаемое полинома.

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

*Поля:*

double coeff — коэффициент перед мономом

unsigned int abc — обобщённая степень

*Методы:*

monom(double c = 0, unsigned int a = 0)— конструктор с параметрами

monom& operator=(const monom& m) — оператор присваивания

bool operator< (const monom& m) const — оператор <

bool operator> (const monom& m) const — оператор >

bool operator==(const monom& m) const — оператор =

bool operator!=(const monom& m) const — оператор !=

### Класс polynom

polynom –класс, представляет собой полином

*Поля:*

list<monom> list\_pol — список из мономов

*Методы:*

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) — оператор присваивания

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) —умножение на константу справа

polynom operator-(const polynom& pol) const — разность полиномов

polynom operator\*(const polynom& pol) const — умножение полиномов

polynom operator-() const — унарный минус

friend ostream& operator<<(ostream &ostr, const polynom &pol — оператор вывода

bool operator==(const polynom& pol) const — оператор =

bool operator!=(const polynom& pol) const — оператор !=

polynom(list<monom> &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.

# Литература

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

## Приложение А. Программная реализация списка с головой

**unit.h**

#pragma once

//Звено списка

template<class type>

class unit

{

public:

type data;

unit\* next;

unit() { next = nullptr; }

unit(type dz) { data = dz; next = nullptr; }

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;

unit<type>\* act;

public:

void Clean();

list();

list(const list<type>& a);

~list();

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

void Insert(type elem);

void InsertToTail(type elem);

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

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

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;

}

## Приложение Б. Программная реализация полиномов

**monom.h**

#pragma once

class monom

{

public:

double coeff;

unsigned int abc;

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

monom& operator=(const monom& m

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;

public:

list<monom> similar\_terms(list <monom> sp);

polynom(const string pol = "" );

polynom(const polynom& pol);

polynom& operator=(const polynom &pol);

polynom operator+(const polynom& pol) const;

polynom operator\*(const double a) const;

friend polynom operator\*(const double a,const polynom& pol) { return pol\*a;}

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

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

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

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

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) {}

**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);

pol.erase(0, pos);

pos = 0;

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

pos++;

string c = part.substr(0,pos);

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

temp.coeff = 1;

else

if (c == "-")

temp.coeff = -1;

else

temp.coeff = stod(c);

part.erase(0, pos);

part += ' ';

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

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

{

pos = part.find((char)(120 + i));

if (pos > -1)

{

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

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

temp.abc += a[i] \* stoi(part.substr(pos + 2, 1));

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;

}

## Приложение В. Программная реализация приложения

**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;

}