МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**“Национальный исследовательский Нижегородский государственный университет им. Н.И.Лобачевского”**

**(ННГУ)**

**Институт информационных технологий математики и механики**

Направление подготовки “Прикладная математика и информатика”

**ОТЧЕТ**

по учебной практике

**Арифметические выражения с полиномами**

**Выполнила:** студентка группы 831703-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шибаева И. В.

**Проверил:** ассистент кафедры

МОСТ ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Волокитин В.Д.

# Содержание

[**Содержание**](#_f7iuj18fq39k) **2**

[**Введение**](#_30j0zll) **3**

[**Постановка учебно-практической задачи**](#_1fob9te) **4**

[**2. Руководство пользователя**](#_chl01gca0x0i) **5**

[**3. Руководство программиста**](#_f3h51fhyknfo) **6**

[3.1 Описание алгоритмов арифметических действий и работы перегрузок](#_tyjcwt) 6

[3.2 Описание структур данных](#_oz8vw5f709ra) 8

[3.3 Общее описание структур программного комплекса](#_1m4l1dgu17ib) 8

[**Заключение**](#_macektxxkt3z) **10**

[**Список литературы**](#_wbcjq37wsnyu) **11**

# 

# **Введение**

Многочлены повсеместно используются различных отраслях математики, а также физики и биологии. Изучение полиномиальных уравнений и их решений составляло едва ли не главный объект «классической алгебры».

С использованием многочленов связан целый ряд преобразований в математике: введение в рассмотрение нуля, отрицательных, а затем и комплексных чисел, а также появление теории групп как раздела математики и выделение классов специальных функций в анализе.

# Постановка учебно-практической задачи

*Формулировка задачи:*

Разработать программу, выполняющую арифметические операции с полиномами трех переменных (x, y и z): сложение, вычитание, умножение на константу, умножение двух полиномов. Считается, что полином составлен из мономов от трех переменных с ограничением на степень каждой переменной от 0 до 9. Коэффициенты полинома - вещественные числа. Работоспособность программы необходимо проверить с помощью Google Test-ов. Кроме того, необходимо разработать пользовательское консольное приложение.

*Исходные данные:*

Комплект массивов для полинома: массив коэффициентов и массив степеней.

*Требуемый результат:*

1. Полином, полученный в результате сложения;
2. Полином, полученный путем вычитания;
3. Полином, полученный в результате умножения на константу;
4. Полином, полученный путем умножения двух полиномов

*Контрольный пример:*

Введем два коротких полинома следующего вида:

и

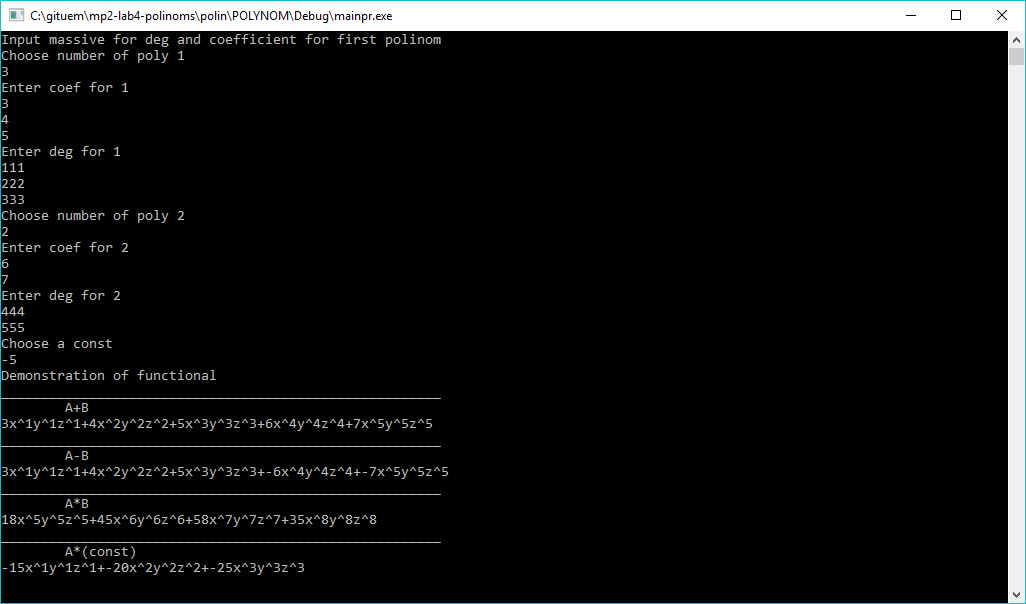


Рис. 1. Результат контрольного примера

# 2. Руководство пользователя

Для запуска программы необходимо открыть файл mainpr.exe. На экране консоли появится запрос ввода первого, а затем и второго полинома.

Программа выведет на экран результаты перечисленных выше операций.

# 

# 3. Руководство программиста

### **3.1 Описание алгоритмов арифметических действий и работы перегрузок**

Для решения поставленной задачи были реализованы алгоритмы и перегрузки арифметических операций, а также сравнение и присваивание.

*1) Функция вставки*

В программе обозначена как void Insert(double coef, int pow). Принимает на вход коэффициент и степень в формате XYZ. Функция находит нужное место для монома и вставляет его. Учитываются случаи, когда список пуст или существует такой моном(в этом случае складывает коэффициенты).

*2) Удаление*

В коде представлено как void Del(int). Последовательно ищет моном, соответствующий запросу, и исключает его из списка. Также выдает сообщение об ошибке, если такого не нашлось.

*3) Оператор сравнения*

bool operator==(const Poly&) const. Возвращает true, если последовательное сравнение степеней и коэффициентов полинома не выдало несостыковок. В противном случае вернет false.

*4) Оператор присваивания*

В коде - Poly& operator=(const Poly&). Сравнивает “головы” двух выбранных списков. Если не равны, левый операнд очищается, и в него поэлементно кладутся мономы правого операнда.

*5) Сложение полиномов*

Представлено оператором Poly operator+(Poly &). Выполняется как последовательное слияние двух списков в третий. За каждый шаг функция сравнивает элементы из первого и из второго полинома и вставляет элемент того, чья степень оказалась меньше. После окончания элементов одного из полинома, в полином суммы сохраняется “хвост” второго, если таковой имеется.

*6)Умножение на константу*

Выполняется через оператор Poly operator\*(double). Коэффициент каждого монома умножается на заданную константу.

*7)Вычитание полиномов*

Выполняется при помощи оператора Poly operator-(Poly &). Реализовано на основе сложения, но предварительно все коэффициенты умножены на -1 с помощью умножения на константу.

*8)Умножение полинома на полином.*

Обозначено в коде программы как Poly operator\*(Poly &). Последовательно умножает каждый моном первого на каждый моном второго полинома: коэффициенты перемножаются, числа степеней складываются(с учетом концепции, где число не может быть больше 999)

### 

### **3.2 Описание структур данных**

Для работы программы были созданы структура Mon(моном) и класс Poly, где реализован список и действия с его элементами

**struct Mon**

double coef; - хранение вещественных коэффициентов

int pow; - хранение степени в формате XYZ

Mon\* next; - хранение указателя на следующий элемент в списке

Mon\* prev; - хранение указателя на предыдущий элемент списка

**class Poly**

Mon\*head; - указатель на “голову” списка

public:

Poly(); - конструктор по умолчанию

Poly(const Poly&); - конструктор копирования

Poly(double\*, int\*, int); - конструктор инициализации

void Insert(double coef, int pow); - функция вставки

void Del(int); - функция удаления

bool operator==(const Poly&) const; - оператор сравнения

Poly& operator=(const Poly&); - оператор присваивания

Poly operator+(Poly &); - оператор сложения

Poly operator-(Poly &); - оператор вычитания

Poly operator\*(double); - оператор, осуществляющий умножение на константу

Poly operator\*(Poly &); - оператор умножения многочлен на многочлен

friend ostream& operator<<(ostream&, Poly &); - дружественная функция вывода

~Poly(); - деструктор

### **3.3 Общее описание структур программного комплекса**

Приложение представлено следующими файлами:

1. В файле List.h объявлены структура Mon и класс Poly и прототипы функций для осуществления операций;
2. В файле List.cpp реализованы методы класса Poly и оператор вывода;
3. В файле mainpr.cpp содержится интерфейс пользовательского приложения;
4. В файле t\_pol.cpp написаны тесты для проверки возможности создания полинома и различных действий над ним;
5. Файл t\_main.cpp запускает все тесты сразу.

# 

# Заключение

По итогу проделанной работы достигнуты следующие результаты:

1. Разработана и написана программа, выполняющая арифметические операции над полиномами: сложение, вычитание, умножение на константу, умножение на другой полином.
2. В качестве структуры хранения полинома используется двухсвязный список мономов с ненулевыми коэффициентами.
3. Степени представлены как трехзначное число, где число сотен – это степень при переменной “x”, число десятков - степень при переменной “y”, число единиц - степень при переменной “z”.
4. Полином хранится в упорядоченном виде.
5. Сложение полиномов осуществлено алгоритмом слияния упорядоченных массивов.
6. Вычитание полиномов выполняется через сложение с умножением на константу (C = A - B = A + (-1)\*B)
7. Если при умножении полиномов полученные степени переменных больше 9, выводится сообщение об ошибке.
8. Все функции протестированы с помощью Google Tests.
9. Создано демонстрационное консольное приложение.

# Список литературы

1. Шилдт, Г. С++ для начинающих: самоучитель / Шилдт, Г. - Изд-во: Эком, 2013г.
2. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы: построение и анализ, М.:МЦНМО, 1999.-960 с.б 263 ил.

**Приложение**

**Файл List.h**

#ifndef \_POLINOM\_

#define \_POLINOM\_

#include <stdio.h>

#include <string>

#include <math.h>

#include <vector>

#include <iostream>

#define MAXC 99999.99999

using namespace std;

struct Mon

{

double coef;

int pow;

Mon\* next;

Mon\* prev;

};

class Poly

{

Mon\*head;

public:

Poly();

Poly(const Poly&);

Poly(double\*, int\*, int);

void Insert(double coef, int pow);

void Del(int);

bool operator==(const Poly&) const;

Poly& operator=(const Poly&);

Poly operator+(Poly &);

Poly operator-(Poly &);

Poly operator\*(double);

Poly operator\*(Poly &);

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

~Poly();

};

#endif

**Файл List.cpp**

#include "List.h"

Poly::Poly()

{

Mon\*n = new Mon;

n->coef = 0.0;

n->next = NULL;

n->prev = NULL;

n->pow = -1;

head = n;

}

Poly::Poly(const Poly &A):Poly()

{

Mon\*temp = A.head->next;

while (temp != NULL) {

Insert(temp->coef, temp->pow);

temp = temp->next;

}

}

Poly::~Poly()

{

while (head->next != NULL)

Del(head->next->pow);

delete head;

}

Poly::Poly(double\* cf, int\* pow, int size):Poly()

{

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

{

if ((pow[i] > 0) && (pow[i] <= 999))

Insert(cf[i], pow[i]);

else

throw("Uncorrect pow");

}

}

void Poly::Insert(double icoef, int ipow)

{

if ((ipow < 0) || (ipow>999))

throw ("Uncorrect pow");

else if (abs(icoef) > MAXC)

throw("Very big coef");

else

{

if (head->next == NULL)

{

Mon\*n = new Mon;

n->coef = icoef;

n->next = NULL;

n->prev = head;

n->pow = ipow;

head->next = n;

}

else

{

Mon\*temp = head;

while ((temp->next != NULL) && (temp->next->pow < ipow))

temp = temp->next;

if (temp->next != NULL)

{

if (temp->next->pow != ipow)

{

Mon\*n1 = new Mon;

n1->coef = icoef;

n1->next = temp->next;

n1->prev = temp;

n1->pow = ipow;

temp->next->prev = n1;

temp->next = n1;

}

else

{

if (abs(temp->next->coef + icoef) < MAXC)

temp->next->coef += icoef;

else throw ("Summ of coefs more than maximum");

}

}

else

{

Mon\*n1 = new Mon;

n1->coef = icoef;

n1->next = NULL;

n1->prev = temp;

n1->pow = ipow;

temp->next = n1;

}

}

}

}

void Poly::Del(int ipow)

{

if ((ipow > 0) && (ipow < 999) && (head->next!=NULL))

{

Mon\*temp = head->next;

while ((temp->next != NULL) && (temp->pow != ipow))

temp = temp->next;

if (temp->pow == ipow)

{

if (temp->next != NULL)

{

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

}

if (temp->next == NULL)

temp->prev->next = NULL;

delete temp;

}

}

else throw ("You choose wrong nuber or use empty list");

}

Poly& Poly::operator=(const Poly& A)

{

if (head != A.head)

{

while (head->next != NULL)

Del(head->next->pow);

Mon\*temp = A.head->next;

while (temp != NULL)

{

Insert(temp->coef, temp->pow);

temp = temp->next;

}

}

return \*this;

}

bool Poly::operator==(const Poly& A) const

{

Mon\* pt1 = head->next;

Mon\* pt2 = A.head->next;

while ((pt1 != NULL) || (pt2 != NULL))

{

if ((pt1->coef != pt2->coef) || (pt1->pow != pt2->pow))

return false;

pt1 = pt1->next;

pt2 = pt2->next;

}

if ((pt1 != NULL) || (pt2 != NULL))

return false;

return true;

}

Poly Poly::operator\*(double c)

{

Poly Con(\*this);

Mon\*temp = Con.head->next;

while (temp != NULL)

{

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

temp = temp->next;

}

return Con;

}

Poly Poly::operator+(Poly & A)

{

Poly Con;

Mon\*pt1 = head->next;

Mon\*pt2 = A.head->next;

while ((pt1 != NULL) && (pt2 != NULL))

{

if (pt1->pow < pt2->pow)

{

Con.Insert(pt1->coef, pt1->pow);

pt1 = pt1->next;

}

else

{

Con.Insert(pt2->coef, pt2->pow);

pt2 = pt2->next;

}

}

while (pt1 != NULL)

{

Con.Insert(pt1->coef, pt1->pow);

pt1 = pt1->next;

}

while (pt2 != NULL)

{

Con.Insert(pt2->coef, pt2->pow);

pt2 = pt2->next;

}

return Con;

}

Poly Poly::operator-(Poly & A)

{

Poly C;

C = (\*this) + A\*(-1);

return C;

}

Poly Poly::operator\*(Poly & A)

{

int d1, d2, d3, sum;

Poly Con;

Mon\*pt1 = head->next;

Mon\*pt2 = A.head->next;

while (pt1 != NULL)

{

d1=(int)pt1->pow/ 100 + (int)pt1->pow % 100 / 10 + (int)pt1->pow % 10;

Mon\*pt2 = A.head->next;

while (pt2 != NULL)

{

d2 = (int)pt2->pow / 100 + (int)pt2->pow % 100 / 10 + (int)pt2->pow % 10;

sum = (int)(pt1->pow+pt2->pow) / 100 + (int)(pt1->pow+pt2->pow) % 100 / 10 + (int)(pt1->pow+pt2->pow) % 10;

d3 = d1 + d2;

if (d3 == sum)

Con.Insert((pt1->coef)\*(pt2->coef), (pt1->pow) + (pt2->pow));

else throw ("wrong monom pow");

pt2 = pt2->next;

}

pt1 = pt1->next;

}

return Con;

}

ostream& operator<<(ostream& out, Poly &A)

{

bool fl = false;

int x, y, z, pow;

Mon\*temp = A.head->next;

while (temp != NULL)

{

pow = temp->pow;

z = pow % 10;

y = pow % 100/10;

x = pow/100;

out << (double)temp->coef;

if (x != 0)

out << "x^" << x;

if (y != 0)

out << "y^" << y;

if (z != 0)

out << "z^" << z;

if (temp->next != NULL)

out << "+";

temp = temp->next;

fl = true;

}

if (!fl)

out << "0";

return out;

}

**Файл t\_pol.cpp**

#include "List.h"

#include "gtest.h"

TEST(Poly, can\_polinom)

{

ASSERT\_NO\_THROW(Poly A);

}

TEST(Poly, can\_insert\_ordinary\_mon)

{

Poly A;

ASSERT\_NO\_THROW(A.Insert(5,777));

}

TEST(Poly, can\_insert\_wrong\_mon1)

{

Poly A;

ASSERT\_ANY\_THROW(A.Insert(5, 1001));

}

TEST(Poly, can\_insert\_wrong\_mon2)

{

Poly A;

ASSERT\_ANY\_THROW(A.Insert(9999999999999, 111));

}

TEST(Poly, can\_del\_empty\_list)

{

Poly A;

ASSERT\_ANY\_THROW(A.Del(333));

}

TEST(Poly, can\_del)

{

double\*cf;

cf = new double[4];

int\*pw;

pw = new int[4];

for (int i = 0; i < 4; i++)

{

cf[i] = (i + 1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 4);

ASSERT\_NO\_THROW(A.Del(444));

}

TEST(Poly, can\_del\_wrong\_pow)

{

double\*cf;

cf = new double[4];

int\*pw;

pw = new int[4];

for (int i = 0; i < 4; i++)

{

cf[i] = (i + 1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 4);

Poly B = A;

A.Del(888);

EXPECT\_EQ(A, B);

}

TEST(Poly, can\_check\_regular)

{

Poly A;

A.Insert(4,444);

A.Insert(5,555);

Poly B;

B.Insert(5,555);

B.Insert(4,444);

EXPECT\_EQ(A,B);

}

TEST(Poly, can\_copy1)

{

double\*cf;

cf= new double[4];

int\*pw;

pw = new int[4];

for (int i = 0; i < 4; i++)

{

cf[i] = (i+1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf,pw,4);

Poly B(A);

EXPECT\_EQ(A, B);

}

TEST(Poly, can\_copy2)

{

double\*cf;

cf = new double[4];

int\*pw;

pw = new int[4];

for (int i = 0; i < 4; i++)

{

cf[i] = (i + 1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 4);

Poly B = A;

EXPECT\_EQ(A, B);

}

TEST(Poly, can\_insert\_in\_beg)

{

double\*cf;

cf = new double[3];

int\*pw = new int[3];

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

{

cf[i] = (i + 2) \* 10;

pw[i] = (i + 2) \* 100 + (i + 2) \* 10 + i + 2;

}

double\*cf1;

cf1 = new double[4];

int\*pw1;

pw1 = new int[4];

for (int i = 0; i < 4; i++)

{

cf1[i] = (i + 1) \* 10;

pw1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 3);

Poly S(cf1, pw1, 4);

A.Insert(10, 111);

EXPECT\_EQ(A, S);

}

TEST(Poly, can\_del\_begin)

{

double\*cf;

cf = new double[3];

int\*pw = new int[3];

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

{

cf[i] = (i + 2) \* 10;

pw[i] = (i + 2) \* 100 + (i + 2) \* 10 + i + 2;

}

double\*cf1;

cf1 = new double[4];

int\*pw1;

pw1 = new int[4];

for (int i = 0; i < 4; i++)

{

cf1[i] = (i + 1) \* 10;

pw1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly S(cf, pw, 3);

Poly A(cf1, pw1, 4);

A.Del(111);

EXPECT\_EQ(A, S);

}

TEST(Poly, can\_insert\_in\_end)

{

double\*cf;

cf = new double[3];

int\*pw = new int[3];

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

{

cf[i] = (i + 1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

double\*cf1;

cf1 = new double[4];

int\*pw1;

pw1 = new int[4];

for (int i = 0; i < 4; i++)

{

cf1[i] = (i + 1) \* 10;

pw1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 3);

Poly S(cf1, pw1, 4);

A.Insert(40, 444);

EXPECT\_EQ(A, S);

}

TEST(Poly, can\_plus\_eq\_poly)

{

double\*cf;

cf = new double[4];

int\*pw;

pw = new int[4];

for (int i = 0; i < 4; i++)

{

cf[i] = (i + 1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 4);

double\*cf1;

cf1 = new double[4];

int\*pw1;

pw1 = new int[4];

for (int i = 0; i < 4; i++)

{

cf1[i] = (i + 1) \* 20;

pw1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly B(cf1, pw1, 4);

Poly C = A + A;

EXPECT\_EQ(C, B);

}

TEST(Poly, can\_plus\_not\_eq)

{

double\*cf;

cf = new double[3];

int\*pw;

pw = new int[3];

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

{

cf[i] = (i + 1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 3);

Poly K;

K.Insert(40, 444);

double\*cf1;

cf1 = new double[4];

int\*pw1;

pw1 = new int[4];

for (int i = 0; i < 4; i++)

{

cf1[i] = (i + 1) \* 10;

pw1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly B(cf1, pw1, 4);

Poly C = A + K;

EXPECT\_EQ(C, B);

}

TEST(Poly, can\_mult\_const)

{

double\*cf;

cf = new double[4];

int\*pw;

pw = new int[4];

for (int i = 0; i < 4; i++)

{

cf[i] = (i + 1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 4);

double\*cf1;

cf1 = new double[4];

int\*pw1;

pw1 = new int[4];

for (int i = 0; i < 4; i++)

{

cf1[i] = (i + 1) \* 30;

pw1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly B(cf1, pw1, 4);

A = A\*3;

EXPECT\_EQ(A, B);

}

TEST(Poly, can\_mult\_two\_eq\_poly)

{

double\*cf;

cf = new double[2];

int\*pw;

pw = new int[2];

for (int i = 0; i < 2; i++)

{

cf[i] = (i + 1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 2);

Poly B;

B.Insert(100, 222);

B.Insert(400, 333);

B.Insert(400, 444);

Poly C = A\*A;

EXPECT\_EQ(C, B);

}

TEST(Poly, can\_mult\_two\_noteq)

{

double\*cf;

cf = new double[2];

int\*pw;

pw = new int[2];

for (int i = 0; i < 2; i++)

{

cf[i] = (i + 1) \* 10;

pw[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(cf, pw, 2);

Poly C;

C.Insert(2, 111);

Poly B;

B.Insert(20, 222);

B.Insert(40, 333);

Poly D = A\*C;

EXPECT\_EQ(D, B);

}

**Файл mainpr.cpp**

// mainpr.cpp : Defines the entry point for the console application.

//

#include <iostream>

#include "List.h"

#include "vector"

using namespace std;

void main()

{

int kol = 0, rezd;

double rezc;

cout << "Demonstration of arithmetic operations with polynomials" << endl<< "Input massive for deg and coefficient for first polinom" << endl;

cout << "Choose number of poly 1"<<endl;

cin >> kol;

int\* deg1 = new int[kol];

double\* coef1 = new double[kol];

cout << "Enter coef for 1" << endl;

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

{

cin >> rezc;

coef1[i]=rezc;

}

cout << "Enter deg for 1" << endl;

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

{

cin >> rezd;

if (rezd < 100 || rezd>999)

{

cout << "Please, enter deg(x)deg(y)deg(z)";

i--;

}

else

deg1[i]=rezd;

}

Poly A(coef1 ,deg1, kol);

delete[]coef1;

delete[]deg1;

cout << "Choose number of poly 2" << endl;

cin >> kol;

deg1 = new int[kol];

coef1 = new double[kol];

cout << "Enter coef for 2" << endl;

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

{

cin >> rezc;

coef1[i] = rezc;

}

cout << "Enter deg for 2" << endl;

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

{

cin >> rezd;

if (rezd < 100 || rezd>999)

{

cout << "Please, enter deg(x)deg(y)deg(z)";

i--;

}

else

deg1[i] = rezd;

}

Poly B(coef1, deg1, kol);

Poly C;

cout << "Choose a const" << endl;

double cons\_t;

cin >> cons\_t;

cout << "Demonstration of functional" << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "\tA+B" << endl;

C = A + B;

cout << C<< endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "\tA-B" << endl;

C = A - B;

cout << C << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "\tA\*B" << endl;

C = A \* B;

cout << C<< endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "\tA\*(const)" << endl;

C = A \* cons\_t;

cout << C<<endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}