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**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по учебной практике

**Арифметические операции с полиномами**

**Выполнил:** студент группы 381603-1

Хорькин А.С.

**Проверила:** к.т.н.,доцент каф. МОСТ института ИТММ

Кустикова В.Д.

Нижний Новгород  
2018

**Содержание**

[Введение 3](#_Toc515581536)

[1 Постановка задачи 4](#_Toc515581537)

[2 Руководство пользователя 5](#_Toc515581538)

[3 Руководство программиста 6](#_Toc515581539)

[3.1 Описание структуры программы 6](#_Toc515581540)

[3.2 Описание структуры данных 7](#_Toc515581541)

[3.3 Описание программной реализация 8](#_Toc515581542)

[Заключение 10](#_Toc515581543)

[Список литературы 11](#_Toc515581544)

[Приложение 12](#_Toc515581545)

[Приложение A. Node.h 12](#_Toc515581546)

[Приложение B. List.h 12](#_Toc515581547)

[Приложение C. Monom.h 12](#_Toc515581548)

[Приложение D. Polinom.h 13](#_Toc515581549)

[Приложение E. Polinom.cpp 13](#_Toc515581550)

[Приложение F. main.cpp 20](#_Toc515581551)

[Приложение G. test\_polinom.cpp. Тесты программы 21](#_Toc515581552)

# Введение

Необходимо было реализовать программу, которая умеет делать некоторые арифметические операции над полиномами, имеющими три переменные и степень от 0 до 9. Язык программирования С++. Данная программа может помочь как при обыкновенном подсчёте нужных пользователю полиномов, так и при разработке других программ.

Реализованы полиномы с помощью такой структуры данных как циклический список с головой. При реализации различных алгоритмов данной программы приходит хорошее понимание как эта структура данных устроена.

# Постановка задачи

Формулировка задачи:

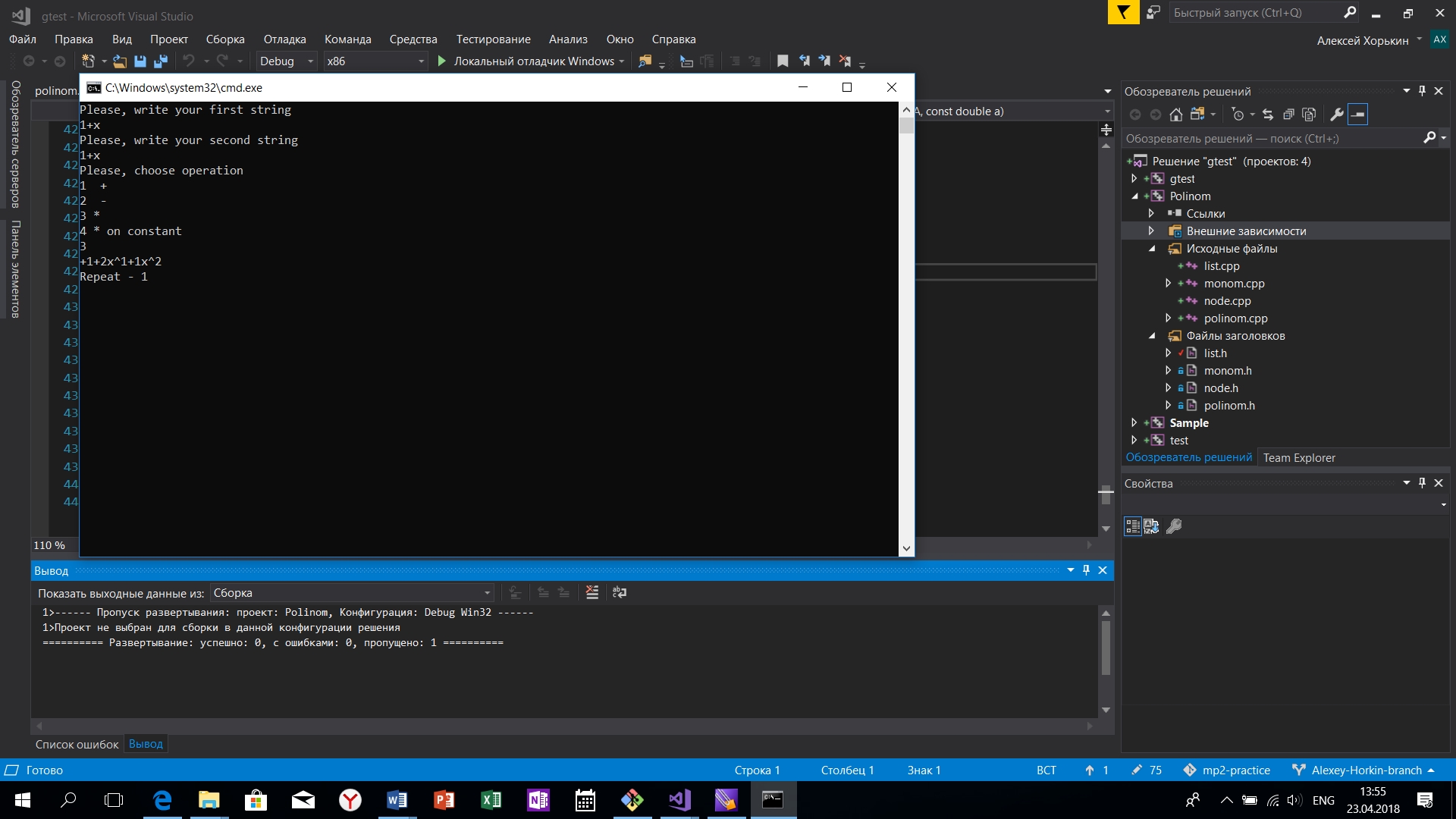
Разработать программу, выполняющую арифметические операции над полиномами от трех переменных. Полином должен быть составлен из мономов трёх переменных, а именно: x, y, z со степенями от 0 до 9. Коэффициенты перед мономами – вещественные числа. Допустимые операции: умножение двух полиномов, сумма, вычитание, умножение полинома на константу.

Исходные данные:

2 строки, содержащие полином A и полином B соответственно.

Требуемый результат:

Программа правильно производит арифметические операции, которые нужны пользователю



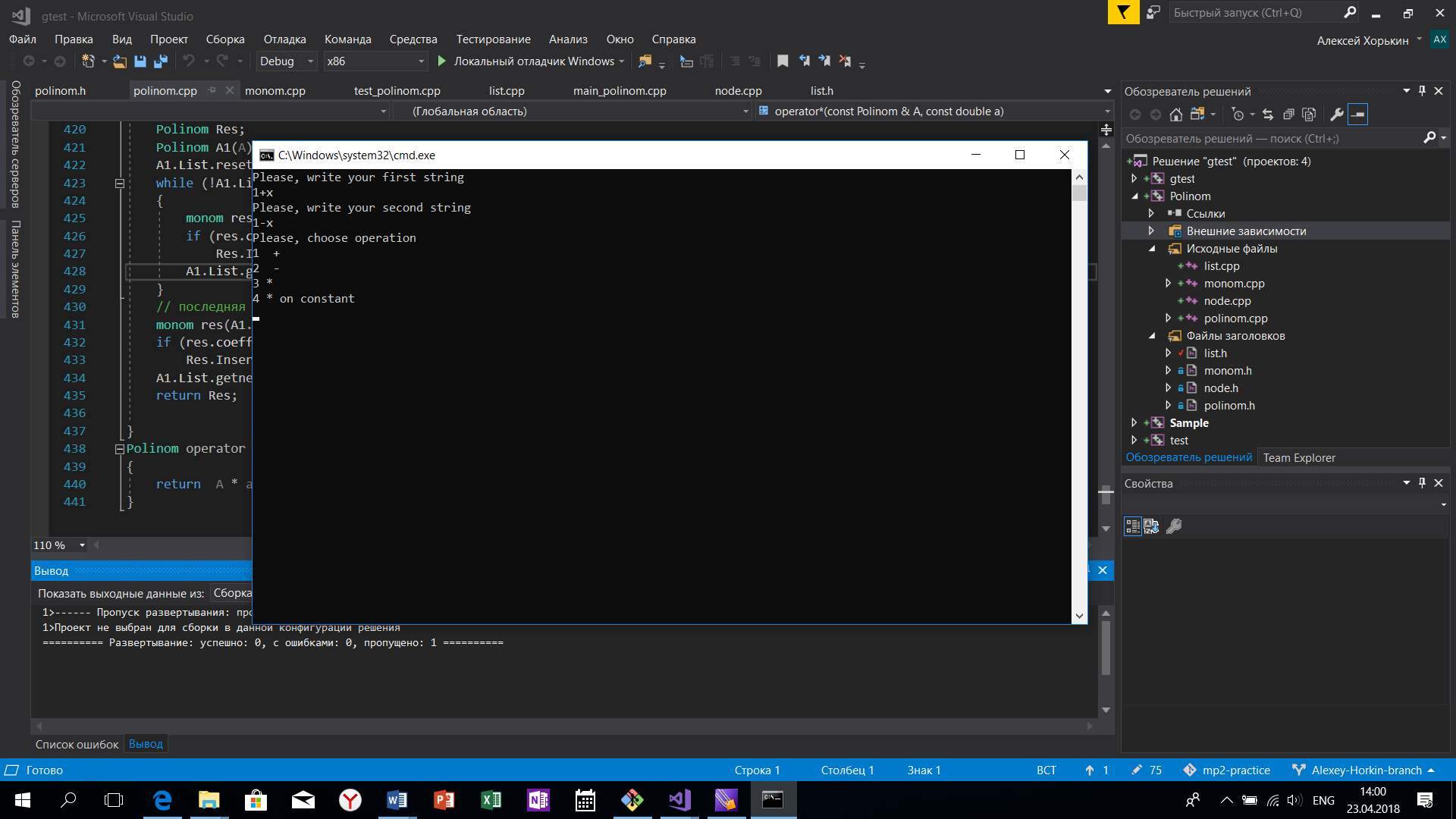
1. Пример работы программы.

# Руководство пользователя

Для начала работы программы необходимо открыть файл **Sample.exe**

После чего программа попросит пользователя ввести первый полином, а затем и второй (допустимые символы: x, y, z,^,0,1,2,3,4,5,6,7,8,9,+.-).

Далее пользователь вправе выбрать одну из предложенных операций.



1. Выбор операции

После пользователю выводится результат. В конце работы программы вам будет предложено либо продолжить вычисления, либо завершить ее. Для продолжения необходимо нажать 1, для завершения – любую.

# Руководство программиста

## Описание структуры программы

Структура решения:

* gtest (статическая библиотека Гугл тестов)
* Polinom (статическая библиотека для полиномов)
* Sample (файл main.cpp)
* Test (консольное приложение, запускающее тесты)

**Библиотека gtest** состоит из:

gtest.h заголовочный файл для Гугл тестов

gtest – all.cpp source файл для Гугл тестов

**Библиотека Polinom** состоит из:

monom.h – заголовочный файл, который содержит специальную структуру данных, предназначенную для хранения слагаемый многочлена.

node.h – шаблонный класс, стандартная структура данных, необходимая для листа

list.h – шаблонный класс, циклический список с головой

polinom.h – содержит list<monom> List

Файлы .cpp содержат реализацию, за исключением шаблонных классов.

**Проект Sample** состоит из:

main\_polinom.cpp – файл, содержащий main() функцию

**Проект test** состоит из:

Test\_main.cpp – содержит функцию main(), запускающую Гугл тесты.

Test\_polinom.cpp – содержит разнообразные тесты для всех структур данных, реализованных в программе.

## Описание структуры данных

В программе используется несколько структур данных или классов: класс мономов, класс ноды, класс кольцевой лист с головой, класс полиномов. Моном состоит из двух полей, а именно double coeff – коэффициент каждого звена и int abc – хранит в себе степень звена. Нода – шаблонный класс, стандартно состоит из двух полей: полей данных и указателя не следующую ноду. Класс кольцевой лист состоит нод, последняя нода указывает на следующую после головы. Полином же состоит из одного поля – Лист, ноды которого хранят структуру данных моном.

Описание алгоритмов, применяющихся в программе.

Ввод полинома осуществляется с клавиатуры, необходимо преобразовать строку полинома в структуру данных - Polinom. Для такого преобразования используется алгоритм разбиения строки на мономы:

На вход подается строка

Строка проходится слева-направо до тех пор, пока не встретится символ '+', '-' или не случится конец строки.

Как только мы встретили символ, мы выделяем подстроку до этого символа и начинается перевод этой подстроки в структуру монома.

В подстроке происходит поиск коэффициента и степеней x, y, z. Найденные значения записываются в структуру Monom, таким образом, что степень x это сотый разряд, степень y десятый разряд, а степень z – первый разряд. Таким образом int abc хранит степень монома. В сoeff – записывается коэффициент.

Полученный моном записывается в список. Если нужно приводятся подобные.

Формируется упорядоченный список из мономов.

Сложение полиномов осуществляется алгоритмом слияния упорядоченных массивов. В данном случае он применяется к упорядоченным спискам:

На вход подаются два упорядоченных списка.

Сравниваем элементы списков:

Больший элемент вставляем в результат. Двигаем указатель списка, из которого взяли элемент, на одно звено вперед.

В случае равенства, складываем коэффициенты и записываем в результат. Двигаем указатели обоих списков на одно звено вперед.

После того, как один из списков кончился, записываем остаток второго списка в результат.

Вычитание полиномов выполняется через сложение с умножением на константу:

C = A - B = A + (-1) \*B

Если при умножении полиномов полученные степени переменных больше 9, выводится сообщение об ошибке.

## Описание программной реализация

Node:

node – шаблонная структура, является узлом списка, с параметром T – тип данных, хранящийся в узле.

*Поля:*

T Data – данные в узле

Node\* Next – указатель на следующий узел

*Методы:*

Node() – конструктор по умолчанию

Node(T Data=0, Node\* ptr = NULL) – конструктор с параметрами

Node(const Node<T> &NodeCopy)- конструктор копирования

Node(const Node\* con) – конструктор копирования по указателю

Лист:

list – шаблонный класс, циклический список с “головой”, с параметром T – тип данных, хранящийся в списке

*Поля:*

Node<T>\* current; – указатель на текущий

Node<T>\* Head; – указатель на голову

Методы:

list(); - конструктор

list(const list<T> &list2); - конструктор с параметрами

~list(); -деструктор

list<T>& operator = (const list<T> &list2); – оператор присваивания

bool IsEmpty() const; – проверка на пустоту стиска

void Insert\_to\_tail(const T con = 0); – вставка в конец

void clear();– очистка списка

bool operator == (const list<T> &list2) const; – оператор сравнения

bool operator != (const list<T> &list2) const; – оператор сравнения

void reset(); - установка на начало списка

void Dellcurrent(); - удаление текущей ноды

void getnext();– сдвиг указателя на следующий узел

T getcurrentdate(); - получить значение текущих данных

T getcurrentnextdata(); - получить значение следующих данных

void InsertAfterCurrent(const T con = 0); - вставить после текущего

void ChangeCurrent(const T con = 0); - вставить в текущий

bool IsEnd();проверка на окончания списка

Monom:

monom –класс, представляет собой моном, из которых составлен полином

*Поля:*

int abc; – степень монома

double coeff; – коэффициент монома

Методы:

monom(int abc\_a = 0, double coef\_f = 0); – конструктор с параметрами

monom(const string &A); – конструктор от строки

bool operator ==(const monom &A) const; – оператор сравнения

monom& operator = (const monom&A); – оператор присваивания

bool operator !=(const monom &A) const; – оператор сравнения

bool operator <(const monom &A) const; – оператор сравнения

bool operator >(const monom &A) const; – оператор сравнения

Класс полином:

Polynom –класс, представляет собой полином

*Поля:*

list<monom> List;

*Методы:*

Polinom(const string &A =""); – конструктор

Polinom(const Polinom &A); – конструктор копирования

~Polinom(); – деструктор

bool operator==(const Polinom& Polinom2) const; – оператор сравнения

Polinom& operator=(const Polinom& Polinom2**) –** оператор присваивания

Polinom operator+(const Polinom& Polinom2) **–** оператор сложения

Polinom operator-(const Polinom& Polinom2) **–** оператор вычитания

Polinom operator \* (const Polinom& Polinom2) **–** оператор умножения

Friend ostream& operator<<(ostream& os, const Polinom& P) **–** оператор вывода

void Insert(const monom A); **–**  вставка в нужное место и приведение подобных

# Заключение

В лабораторной работе была реализована программа, выполняющая арифметические операции с полиномами. Был разработан класс полиномов, на основе циклического односвязного списка с головой. Таким образом программа может выполнять все необходимые операции с полиномами.

# 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

## Приложение A. Node.h

#pragma once

#include <string>

#include <sstream>

using namespace std;

template <typename T>

class Node

{

public:

T Data;

Node\* Next;

Node(T Data= 0, Node\* NextPtr= NULL);

Node(const Node<T> &NodaCopy);

Node(const Node\* con);

};

## Приложение B. List.h

#pragma once

#define NULL 0

#include "node.h"

/\*СПИСОК С ГОЛОВОЙ\*/

template <typename T>

class list {

private:

Node<T>\* current; // это просто напросто ходилка

Node<T>\* Head; // это указательн а фиктивную штуку ... след за ней данные

public:

list();

list(const list<T> &list2);

~list();

list<T>& operator = (const list<T> &list2);

bool IsEmpty() const;

void Insert\_to\_tail(const T con = 0);

void clear();

bool operator == (const list<T> &list2) const;

bool operator != (const list<T> &list2) const;

void reset();//reset - переход в начало

void Dellcurrent();

void getnext();//Getnext - переход в листе на следующий за ним

T getcurrentdate();//GetCurrent - возвращать данные data

T getcurrentnextdata(); // возвращает данные следущего за текущим

void InsertAfterCurrent(const T con = 0); // вставка после текущего

void ChangeCurrent(const T con = 0); // вставка в текущий

bool IsEnd();//дошли ли до конца

};

## Приложение C. Monom.h

struct monom

{

public:

int abc;

double coeff;

monom(int abc\_a = 0, double coef\_f = 0);

monom(const string &A);

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

monom& operator = (const monom&A);

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

bool operator <(const monom &A) const;

bool operator >(const monom &A) const;

private:

//функции для работы с char

bool determinatechar(char s); //private

bool determinatecharonly(char s); //private

};

## Приложение D. Polinom.h

#include <string>

#include "list.h"

#include "monom.h"

using namespace std;

class Polinom

{

public:

Polinom(const string &A ="");

Polinom(const Polinom &A); //конструктор копирования

~Polinom();

bool operator==(const Polinom& Polinom2) const; //оператор ==

Polinom& operator=(const Polinom& Polinom2); //оператор =

Polinom operator+(const Polinom& Polinom2); //оператор +

Polinom operator-(const Polinom& Polinom2); //оператор -

Polinom operator \* (const Polinom& Polinom2); //опрератор \*

friend ostream& operator<<(ostream& os, const Polinom& P);

//умножение на константу;

friend Polinom operator \* (const Polinom& A, const double a);

friend Polinom operator \* (const double a, const Polinom& A);

private:

list<monom> List;

void Insert(const monom A); //ставляет Node в нужное место (приводит подобные если надо)

};

## Приложение E. Polinom.cpp

#include "polinom.h"

#include <iostream>

using namespace std;

Polinom::Polinom(const string &S)

{

int i = 0;

List.Insert\_to\_tail(); // вставляем нулевую ноду

string MonomStr = "";

while (i + 1 <= S.length())

{

if ((S[i + 1] != '+') && (S[i + 1] != '-') && (S[i + 1] != '\0'))

{

MonomStr.push\_back(S[i]);

i += 1;

}

else

{

MonomStr.push\_back(S[i]);

monom A(MonomStr);

if (A.coeff != 0)

Insert(A);

MonomStr = "";

i++;

}

}

if (List.IsEmpty())

List.Insert\_to\_tail();

}

void Polinom::Insert(const monom N)

{

int i = 0;

int flag = 0;

List.reset(); // ставим current в начало

while ((!List.IsEnd()) && (flag == 0)) //пока лист не пуст

{

monom M = List.getcurrentdate(); // берём текущий моном

if (M.abc == N.abc) // если совпадают степени

{

monom res(M.abc, M.coeff + N.coeff);

if (res.coeff != 0)

{

List.ChangeCurrent(res);

flag = 1;

}

else

{

if (N.coeff != 0)

{

List.Dellcurrent();

flag = 1;

}

}

}

else if ((M.abc < N.abc) && (N.abc<List.getcurrentnextdata().abc) && (flag == 0))

{

if (N.coeff != 0)

List.InsertAfterCurrent(N);

flag = 1;

}

List.getnext();

}

//последняя итерация цикла

if ((List.IsEnd()) && (flag == 0))

{

monom M = List.getcurrentdate(); // берём текущий моном

if (M.abc == N.abc) // если совпадают степени

{

monom res(M.abc, M.coeff + N.coeff);

if (res.coeff != 0)

{

List.ChangeCurrent(res);

flag = 1;

}

else

{

if (N.coeff != 0)

{

List.Dellcurrent();

flag = 1;

}

}

}

else

if (N.coeff != 0)

{

List.InsertAfterCurrent(N);

flag = 1;

}

if (!List.IsEmpty())

List.getnext();

}

}

Polinom::~Polinom()

{

List.clear();

}

Polinom::Polinom(const Polinom &A)

{

List = A.List;

}

Polinom& Polinom :: operator=(const Polinom& Polinom2)

{

List = Polinom2.List;

return \*this;

}

Polinom Polinom :: operator + (const Polinom& Polinom2)

{

Polinom Res;

this->List.reset();

Polinom B(Polinom2);

monom A1 = List.getcurrentdate();

B.List.reset();

monom B1 = B.List.getcurrentdate();

while ((!this->List.IsEnd()) && (!B.List.IsEnd()))

{

if (List.getcurrentdate().abc == B.List.getcurrentdate().abc)

{

monom res(List.getcurrentdate().abc, List.getcurrentdate().coeff + B.List.getcurrentdate().coeff);

if (res.coeff != 0) //при занулении коэффицентов пропускаем звено

{

if (res.abc == 0)

{

Res.Insert(res);

List.getnext();

B.List.getnext();

}

else {

Res.List.Insert\_to\_tail(res);

List.getnext();

B.List.getnext();

}

}

else

{

List.getnext();

B.List.getnext();

}

}

else if (List.getcurrentdate().abc < B.List.getcurrentdate().abc)

{

monom res(List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

List.getnext();

}

else if (List.getcurrentdate().abc > B.List.getcurrentdate().abc)

{

monom res(B.List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

B.List.getnext();

}

}

// последняя итерация цикла

if (List.getcurrentdate().abc == B.List.getcurrentdate().abc)

{

monom res(List.getcurrentdate().abc, List.getcurrentdate().coeff + B.List.getcurrentdate().coeff);

if (res.coeff != 0) //при занулении коэффицентов пропускаем звено

{

if (res.abc == 0)

{

Res.Insert(res);

List.getnext();

B.List.getnext();

}

else

Res.List.Insert\_to\_tail(res);

List.getnext();

B.List.getnext();

}

else

{

List.getnext();

B.List.getnext();

}

}

else if (List.getcurrentdate().abc < B.List.getcurrentdate().abc)

{

monom res(List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

List.getnext();

}

else if (List.getcurrentdate().abc > B.List.getcurrentdate().abc)

{

monom res(B.List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

B.List.getnext();

}

while ((this->List.getcurrentdate() != A1) && (B.List.getcurrentdate() != B1))

{

if (List.getcurrentdate().abc == B.List.getcurrentdate().abc)

{

monom res(List.getcurrentdate().abc, List.getcurrentdate().coeff + B.List.getcurrentdate().coeff);

if (res.coeff != 0) //при занулении коэффицентов пропускаем звено

{

if (res.abc == 0)

{

Res.Insert(res);

List.getnext();

B.List.getnext();

}

else

Res.List.Insert\_to\_tail(res);

List.getnext();

B.List.getnext();

}

else

{

List.getnext();

B.List.getnext();

}

}

else if (List.getcurrentdate().abc < B.List.getcurrentdate().abc)

{

monom res(List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

List.getnext();

}

else if (List.getcurrentdate().abc > B.List.getcurrentdate().abc)

{

monom res(B.List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

B.List.getnext();

}

}

// обработка крайних случаев

if ((this->List.getcurrentdate() == A1) && (B.List.getcurrentdate() != B1))

{

while (!B.List.IsEnd())

{

monom res(B.List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

B.List.getnext();

}

monom res(B.List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

B.List.getnext();

}

else if ((this->List.getcurrentdate() != A1) && (B.List.getcurrentdate() == B1))

{

while (!List.IsEnd())

{

monom res(List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

List.getnext();

}

monom res(List.getcurrentdate());

Res.List.Insert\_to\_tail(res);

List.getnext();

}

return Res;

}

//

Polinom Polinom :: operator - (const Polinom& Polinom2)

{

Polinom p1 = -1 \* (Polinom2);

return (\*this + p1);

}

//

Polinom Polinom :: operator \* (const Polinom& Polinom2)

{

Polinom Res;

Polinom B(Polinom2);

List.reset();

B.List.reset();

while (!List.IsEnd())

{

while (!B.List.IsEnd())

{

monom res(List.getcurrentdate().abc + B.List.getcurrentdate().abc, List.getcurrentdate().coeff \* B.List.getcurrentdate().coeff);

if (res.abc > 999)

throw "Error";

Res.Insert(res);

B.List.getnext();

}

List.getnext();

B.List.reset();

}

//последняя итерация цикла

B.List.reset();

while (!B.List.IsEnd())

{

monom res(List.getcurrentdate().abc + B.List.getcurrentdate().abc, List.getcurrentdate().coeff \* B.List.getcurrentdate().coeff);

if (res.abc > 999)

throw "Error";

Res.Insert(res);

B.List.getnext();

}

//

monom res(List.getcurrentdate().abc + B.List.getcurrentdate().abc, List.getcurrentdate().coeff \*B.List.getcurrentdate().coeff);

if (res.abc > 999)

throw "Error";;

Res.Insert(res);

List.reset();

//

while (!List.IsEnd())

{

monom res(List.getcurrentdate().abc + B.List.getcurrentdate().abc, List.getcurrentdate().coeff \* B.List.getcurrentdate().coeff);

if (res.abc > 999)

throw "Error";

Res.Insert(res);

List.getnext();

}

return Res;

}

ostream& operator<<(ostream& os, const Polinom& P)

{

int flag = 0;

Polinom P1(P);

P1.List.reset();

//os << "0";

while (!P1.List.IsEnd())

{

if (P1.List.getcurrentdate().coeff == 0)

P1.List.getnext();

else {

flag = 1;

if (P1.List.getcurrentdate().coeff > 0)

os << '+';

os << P1.List.getcurrentdate().coeff;

string ABC = to\_string(P1.List.getcurrentdate().abc);

if (ABC.length() == 1) // если abc =1

{

if (ABC == "0")

P1.List.getnext();

else {

os << "z^" << ABC[0];

P1.List.getnext();

}

}

else if (ABC.length() == 2) // если abc =2

{

os << "y^" << ABC[0];

if (ABC[1] != '0')

os << "z^" << ABC[1];

P1.List.getnext();

}

else if (ABC.length() == 3) // если abc =3

{

os << "x^" << ABC[0];

if (ABC[1] != '0')

os << "y^" << ABC[1];

if (ABC[2] != '0')

os << "z^" << ABC[2];

P1.List.getnext();

}

}

}

// последняя итерация цикла

if (P1.List.getcurrentdate().coeff == 0)

P1.List.getnext();

else {

flag = 1;

if (P1.List.getcurrentdate().coeff > 0)

os << '+';

os << P1.List.getcurrentdate().coeff;

string ABC = to\_string(P1.List.getcurrentdate().abc);

if (ABC.length() == 1) // если abc =1

{

if (ABC == "0")

P1.List.getnext();

else {

os << "z^" << ABC[0];

P1.List.getnext();

}

}

else if (ABC.length() == 2) // если abc =2

{

os << "y^" << ABC[0];

if (ABC[1] != '0')

os << "z^" << ABC[1];

P1.List.getnext();

}

else if (ABC.length() == 3) // если abc =3

{

os << "x^" << ABC[0];

if (ABC[1] != '0')

os << "y^" << ABC[1];

if (ABC[2] != '0')

os << "z^" << ABC[2];

P1.List.getnext();

}

}

if (flag == 0)

os << "0";

os << endl;

return os;

}

bool Polinom :: operator==(const Polinom& Polinom2) const //оператор ==

{

return (this->List == Polinom2.List);

}

Polinom operator \* (const Polinom& A, const double a)

{

Polinom Res;

Polinom A1(A);

A1.List.reset();

while (!A1.List.IsEnd())

{

monom res(A1.List.getcurrentdate().abc, A1.List.getcurrentdate().coeff \* a);

if (res.coeff != 0)

Res.Insert(res);

A1.List.getnext();

}

// последняя итерация цикла

monom res(A1.List.getcurrentdate().abc, A1.List.getcurrentdate().coeff \* a);

if (res.coeff != 0)

Res.Insert(res);

A1.List.getnext();

return Res;

}

Polinom operator \* (const double a, const Polinom& A)

{

return A \* a;

}

## Приложение F. main.cpp

#include "monom.h"

#include "polinom.h"

#include <iostream>

#include <ostream>

using namespace std;

void main()

{

int flag = 1;

int k = 0;

while (flag == 1)

{

system("cls");

cout << "Please, write your first string" << endl;

string s1;

cin >> s1;

Polinom S1(s1);

cout << "Please, write your second string" << endl;

string s2;

cin >> s2;

Polinom S2(s2);

cout << "Please, choose operation" << endl;

cout << "1 +" << endl;

cout << "2 -" << endl;

cout << "3 \*" << endl;

cout << "4 \* on constant" << endl;

cin >> k;

switch (k)

{

case 1:

{

Polinom res = S1 + S2;

cout << res;

break;

}

case 2:

{

Polinom res = S1 - S2;

cout << res;

break;

}

case 3:

{

Polinom res = S1 \* S2;

cout << res;

break;

}

case 4:

{

//string con;

//cin >> con;

double a;

cout << "Please, write const for multyplication" << endl;

cin >> a;

//Polinom C(con);

Polinom res = S1\*a;

cout << res;

break;

}

default:

break;

}

cout << "Repeat - 1" << endl;

cin >> flag;

}

}

## Приложение G. test\_polinom.cpp. Тесты программы

#include "gtest.h"

#include "polinom.h"

#include "monom.h"

#include "list.h"

int i = -1;

int j = -1;

int k = -1;

int n = -1;

int l = -1;

// тесты мономов и полиномов

TEST(list, insert\_to\_end) // вставка в конец

{

monom a(12, 14);

monom b(14, 43);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.reset();

monom res(l.getcurrentdate());

EXPECT\_EQ(res,a);

}

TEST(list, can\_create\_list) // может создать лист

{

EXPECT\_NO\_THROW(list<monom> l);

}

TEST(list, delete\_last\_not\_empty\_list) // Удаление не пустого листа

{

monom a(12, 14);

monom b(14, 43);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.clear();

EXPECT\_EQ(l.IsEmpty(), 1);

}

TEST(list, can\_check\_is\_empty) // проверка на пустоту не пустого листа

{

monom a(12, 14);

monom b(14, 43);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

EXPECT\_EQ(0, l.IsEmpty());

}

TEST(list, can\_check\_is\_empty\_if\_empty) // проверка на пустоту пустого листа

{

list<monom> l;

EXPECT\_EQ(1, l.IsEmpty());

}

TEST(list, operator\_ravno\_for\_list) // проверка оператора =

{

monom a(12, 14);

monom b(14, 43);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

list<monom> z = l;

EXPECT\_EQ(z,l);

}

TEST(list, operator\_ravno\_for\_list2)

{

monom a(12, 14);

monom b(14, 43);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

list<monom> z = l;

if (l == z)

EXPECT\_EQ(l, z);

}

TEST(list, operator\_ravno\_for\_list3)

{

monom a(12, 14);

monom b(14, 43);

monom c(11, 123);

monom d(12, 1232);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

list<monom> z = l;

if (l == z)

EXPECT\_EQ(l, z);

}

//dellcurent

TEST(list, Dellcurent\_first)

{

monom a(12, 14);

monom b(14, 43);

monom c(11, 123);

monom d(12, 1232);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

l.reset();

l.Dellcurrent();

EXPECT\_EQ(l.getcurrentdate(), b);

}

TEST(list, Dellcurent\_last)

{

monom a(12, 14);

monom b(14, 43);

monom c(11, 123);

monom d(12, 1232);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

l.getnext();

l.Dellcurrent();

EXPECT\_EQ(l.getcurrentdate(), c);

}

TEST(list, Dellcurent\_middle)

{

monom a(12, 14);

monom b(14, 43);

monom c(11, 123);

monom d(12, 1232);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

l.Dellcurrent();

EXPECT\_EQ(l.getcurrentdate(), b);

}

TEST(list, Dellcurent\_first\_must\_empty)

{

monom a(12, 14);

list<monom> l;

l.Insert\_to\_tail(a);

l.reset();

l.Dellcurrent();

EXPECT\_EQ(l.IsEmpty(),1);

}

TEST(list, can\_get\_currnextdata)

{

monom a(12, 14);

monom b(14, 43);

monom c(11, 123);

monom d(12, 1232);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

EXPECT\_EQ(l.getcurrentnextdata(), d);

}

TEST(list, can\_get\_currdata)

{

monom a(12, 14);

monom b(14, 43);

monom c(11, 123);

monom d(12, 1232);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

EXPECT\_EQ(l.getcurrentdate(), c);

}

TEST(list, can\_getnext\_node)

{

monom a(12, 14);

monom b(14, 43);

monom c(11, 123);

monom d(12, 1232);

list<monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

l.getnext();

EXPECT\_EQ(l.getcurrentdate(), d);

}

// тесты для полинома, параметризированные

struct t //структура для проверки создания полинома и последнего монома

{

string s;

monom lastmonom;

t(string mys = "", monom f = monom(0,0))

{

s = mys;

lastmonom = f;

}

};

class parameterizedstringforPolinom : public ::testing::TestWithParam<t>

{

protected:

struct t TESTcase;

public:

parameterizedstringforPolinom() : TESTcase(GetParam())

{

}

~parameterizedstringforPolinom() {}

};

struct t1 //структура для проверки создания полинома и последнего монома

{

string s;

t1(string mys = "")

{

s = mys;

}

};

class parameterizedstringforPolinom1 : public ::testing::TestWithParam<t1>

{

protected:

struct t1 TESTcase;

public:

parameterizedstringforPolinom1() : TESTcase(GetParam())

{

}

~parameterizedstringforPolinom1() {}

};

TEST\_P(parameterizedstringforPolinom1, can\_throw\_when\_degree\_normal)

{

ASSERT\_NO\_THROW(Polinom a(GetParam().s));

}

TEST\_P(parameterizedstringforPolinom1, can\_oper\_ravno)

{

Polinom a(GetParam().s);

Polinom b = a;

EXPECT\_EQ(b == a,1);

}

t1 myarray1[] = { t1("3+z^1+y+x") , t1("3+z+y^1"), t1("12.12+x+y+xyz"), t1("1+x^2+z^3+xyz^2"), t1("12+3xyz+x^2y^4"), t1("111+323xyz+23.32xyz+2.232+12x^9"), t1("1+x+y+x^2yz") };

INSTANTIATE\_TEST\_CASE\_P(instantiation2,

parameterizedstringforPolinom1,

::testing::ValuesIn(myarray1));

class parameterizedstringforPolinom2 : public parameterizedstringforPolinom1 {};

TEST\_P(parameterizedstringforPolinom2, can\_throw\_when\_degree)

{

ASSERT\_ANY\_THROW(Polinom a(GetParam().s));

}

t1 myarray2[] = { t1("3+z^10+y+x") , t1("3+z^12+y^1"), t1("12.12+x^12+y+xyz"), t1("1+x^2+z^31+xyz^2"), t1("12+3xyz+x^2y^41"), t1("111+323xyz+23.32xyz+2.232+12x^91"), t1("1+x+y^12+x^2yz") };

INSTANTIATE\_TEST\_CASE\_P(instantiation3,

parameterizedstringforPolinom2,

::testing::ValuesIn(myarray2));

class parameterizedstringforPolinom3 : public parameterizedstringforPolinom1 {}; // класс для проверки сложения

struct t2 //структура для проверки создания полинома и последнего монома

{

string s;

t2(string mys = "")

{

s = mys;

}

};

t2 myarray4[] = { t2("2+2x"), t2("22+2x+4x^2+2z^7") };

TEST\_P(parameterizedstringforPolinom3, can\_plus)

{

Polinom a(GetParam().s);

Polinom b(GetParam().s);

Polinom c = a + b;

Polinom z(myarray4[i+1].s);

i++;

EXPECT\_EQ(z, c);

}

t1 myarray3[] = { t1("1+x"), t1("11+x+2x^2+z^7") };

INSTANTIATE\_TEST\_CASE\_P(instantiation4,

parameterizedstringforPolinom3,

::testing::ValuesIn(myarray3));

class parameterizedstringforPolinom4 : public parameterizedstringforPolinom1 {}; // класс для проверки вычитания

t2 myarray6[] = { t2("0") , t2("0"), t2("1"), t2("12x^3-11x^4y+21.4xyz"), t2("-1")};

t2 arrayforminus [] = { t2("1+x+y"), t2("12+x^9+y^1+z^7"), t2("10+x+y+z"), t2("0"), t2("2+x+yx+z")};

TEST\_P(parameterizedstringforPolinom4, can\_minus)

{

Polinom a(GetParam().s);

Polinom b(arrayforminus[j + 1].s);

Polinom c;

c = a - b;

Polinom z(myarray6[j + 1].s);

j++;

EXPECT\_EQ(z, c);

}

t1 myarray5[] = { t1("1+x+y"), t1("x^9+y^1+z^7+12"), t1("11+x+y+z") , t1("12x^3-11x^4y+21.4xyz"), t1("1+x+yx+z") };

INSTANTIATE\_TEST\_CASE\_P(instantiation5,

parameterizedstringforPolinom4,

::testing::ValuesIn(myarray5));

TEST(Polinom, can\_create\_polinom)

{

ASSERT\_NO\_THROW(Polinom a("1+x+y"));

}

TEST(Polinom, can\_plus)

{

Polinom a("1+x+y");

Polinom b("1+x+x^3");

ASSERT\_NO\_THROW(a+b);

}

TEST(Polinom, can\_minus)

{

ASSERT\_NO\_THROW(Polinom a("1+x+y"));

}

TEST(Polinom, too\_big\_degree)

{

Polinom a("1+x+y");

Polinom b("1+x^9+y");

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

}

class parameterizedstringforPolinom5 : public parameterizedstringforPolinom1 {}; // класс для проверки вычитания

t2 myarray7[] = { t2("1-x^2") , t2("1+2x+x^2"), t2("1-x^3"), t2("0"), t2("-1-x-xy-z"), t2("x^2-y^2"), t2("x^2-2xz+z^2"), t2("x^2+2zx+z^2") };

t2 arrayformultyply[] = { t2("1-x"), t2("1+x"), t2("1+x+x^2"), t2("0"), t2("-1") , t2("x-y"), t2("x-z"), t2("x+z") };

TEST\_P(parameterizedstringforPolinom5, muityply\_between\_Polinoms)

{

Polinom a(GetParam().s);

Polinom b(arrayformultyply[k + 1].s);

Polinom c;

c = a \*b;

Polinom z(myarray7[k + 1].s);

k++;

EXPECT\_EQ(z, c);

}

t1 myarray8[] = { t1("1+x"), t1("1+x"), t1("1-x") , t1("12x^3-11x^4y+21.4xyz"), t1("1+x+yx+z") , t1("x+y"), t1("x-z"), t1("x+z") };

INSTANTIATE\_TEST\_CASE\_P(instantiation6,

parameterizedstringforPolinom5,

::testing::ValuesIn(myarray8));

class parameterizedstringforPolinom6 : public parameterizedstringforPolinom1 {}; // класс для проверки вычитания

t2 arrayonconst[] = { t2("1+x") , t2("-1-x"), t2("-2+2x"), t2("24x^3-22x^4y+42.8xyz"), t2("100+100x+100yx+100z"), t2("0.5x+0.5y") };

double arrayformultyply1[] = { 1, -1,-2,2,100, 0.5 };

TEST\_P(parameterizedstringforPolinom6, muityply\_between\_Polinoms\_right)

{

Polinom a(GetParam().s);

Polinom c;

c = a \* arrayformultyply1[n + 1];

Polinom z(arrayonconst[n + 1].s);

n++;

EXPECT\_EQ(z, c);

}

TEST\_P(parameterizedstringforPolinom6, muityply\_between\_Polinoms\_left)

{

Polinom a(GetParam().s);

Polinom c;

c = arrayformultyply1[l + 1]\*a;

Polinom z(arrayonconst[l + 1].s);

l++;

EXPECT\_EQ(z, c);

}

t1 myarray9[] = { t1("1+x"), t1("1+x"), t1("1-x") , t1("12x^3-11x^4y+21.4xyz"), t1("1+x+yx+z") , t1("x+y") };

INSTANTIATE\_TEST\_CASE\_P(instantiation7,

parameterizedstringforPolinom6,

::testing::ValuesIn(myarray9));