МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное 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

Сарынин С.В.

**Проверил**: к.т.н., доцент каф. МОСТ

института ИТММ

Кустикова В.Б.

Нижний Новгород

2018

**Содержание**

[Введение 3](#_Toc515877211)

[1. Постановка задачи 4](#_Toc515877212)

[2. Руководство пользователя 5](#_Toc515877213)

[3. Руководство программиста 6](#_Toc515877214)

[1. Структура программы 6](#_Toc515877215)

[2. Структура данных 7](#_Toc515877216)

[Класс Node 7](#_Toc515877217)

[Класс Polinom 7](#_Toc515877218)

[Класс List 8](#_Toc515877219)

[Класс Monom 8](#_Toc515877220)

[3. Описание алгоритмов 10](#_Toc515877221)

[Заключение 11](#_Toc515877222)

[Литература 12](#_Toc515877223)

[Приложение 13](#_Toc515877224)

[Приложение А. Программная реализация циклического списка 13](#_Toc515877225)

[Приложение Б. Программная реализация монома 17](#_Toc515877226)

[Приложение В. Программная реализация полинома 18](#_Toc515877227)

[Приложение Г. Реализация основного приложения 25](#_Toc515877228)

[Приложение Д. Тесты для List 26](#_Toc515877229)

[Приложение Е. Тесты для полинома 29](#_Toc515877230)

Введение

Полином(многочлен) представляет собой множество одночленов соединенных между собой знаками(+,-). В данной лабораторной работе необходимо реализовать программу способную выполнять сложение, умножение и вычитание полиномов, для этого в реализации используется циклический односвязный список.

В отчёте приводится постановка задачи, описание использующихся алгоритмов, описание программы и правила её использования, а также прилагается код программы, решающей поставленную задачу.

1. Постановка задачи

Формулировка задачи:

Реализовать программу, выполняющую сложение, умножение и вычитание полиномов от трех переменных. С максимальным порядком=9.

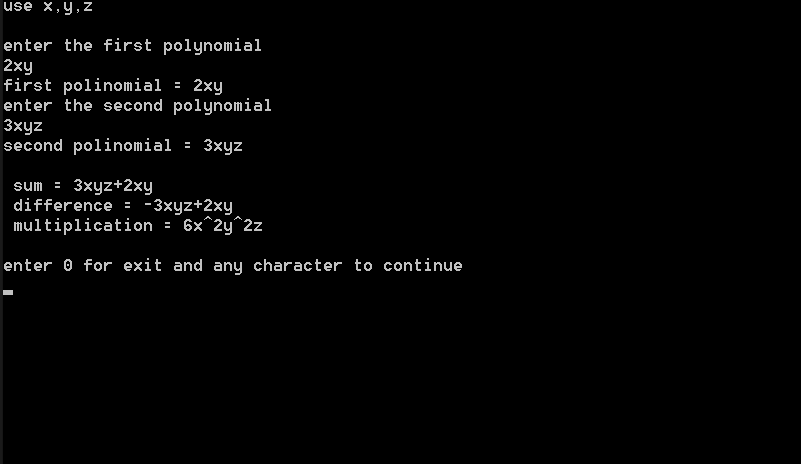
Исходные данные: 2 полинома.

Выходные данные: результат сложения, умножения или вычитания полиномов.

1. Руководство пользователя

Для начала работы с программой необходимо:

1. Открыть файл polinom\_sample.exe.
2. Ввести два полинома, следуя указаниям программы.
3. Программа выведет результат вычисления. Для того, чтобы ввести полиномы заново необходимо нажать любую клавишу после того как выведутся результаты вычислений. Для выхода необходимо нажать 0.



1. демонстрация работы программы
2. Руководство программиста
3. Структура программы

Программа состоит из нескольких файлов:

1. node.h содержит объявление шаблонного класса Node и его реализацию.
2. List.hсодержит объявление шаблонного класса List и его реализацию.
3. monom.h содержит объявление класса Monom и его реализацию.
4. monom.cpp содержит реализацию класса Monom.
5. polinom*.*h содержит объявление класса Polinom.
6. polinom.cpp содержит реализацию класса Polinom.
7. main.cpp содержит реализацию пользовательского интерфейса.
8. test\_list.cpp содержит реализацию тестов для класса List.
9. test\_ poilnom.cpp содержит реализацию тестов для класса Polinom .
10. Структура данных

Класс Node

поля

T data;

Node<T>\* next;

методы

Node(T d = NULL, Node<T>\* l = NULL) –конструктор копирования

bool operator< (const Node<T>& n) const – перегрузка <

bool operator> (const Node<T>& n) const – перегрузка >

bool operator!= (const Node<T>& n) const – перегрузка !=

bool operator==(const Node<T>& n) const – перегрузка ==

Класс Polinom

поля

List<Monom> list\_m;

методы

Polinom(const string Line = "" ) – конструктор по строке

Polinom(const Monom m) – конструктор преобразования типа от монома

Polinom(const List<Monom> &p2) – конструктор преобразования типа от списка

Polinom(const Polinom& pol) – конструктор копирования

Polinom operator+ (const Polinom&) const – перегрузка сложения

Polinom operator- (const Polinom& pol) const – перегрузка разности

Polinom operator\* (const Polinom& pol) const –перегрузка умножения полиномов

Polinom operator\* (const double c) const – перегрузка умножения н кнстанту

bool operator== (const Polinom& pol) const – перегрузка ==

bool operator!= (const Polinom& pol) const – перегрузка !=

friend ostream& operator<< (ostream& os, const Polinom&) – вывод полинома

Класс List

поля

Node<T> \*head;

Node<T> \*current;

методы

List()– конструктор по умолчанию

List(const List<T> &lst)– конструктор копирования

~List()– деструктор

void Clean()– удалить все звенья

List<T>& operator=(const List<T>& lst)– оператор присваивания

Node<T>\* GetCurr()– получить указатель на текущий элемент

void InsertAfter(Node<T>\* n, T d) – вставка после выбраннго элемента, элемента d.

void InsertOrdered(T d) – всавка в упорядоченный список

void GetNext()–выставляет значение текущего элемента списка на следующий.

void Reset()– возращение указателя на начало

bool IsEnded() const – проверка на конец списка

bool operator==(const List<T>& lst) const–перегрузка ==

bool operator!=(const List<T>& lst) const–перегрузка!=

Класс Monom

поля

double cf;

unsigned int abc;

методы

Monom(double a= 0 , unsigned int b = 0)– конструктор по умолчанию

Monom& operator=(const Monom& m) – перегрузка =

bool operator< (const Monom& m) const– перегрузка <

bool operator> (const Monom& m) const– перегрузка >

bool operator==(const Monom& m) const– перегрузка ==

bool operator!=(const Monom& m) const– перегрузка !=

1. Описание алгоритмов

Заполнение списка

* На вход подаем элемент **A**, типа **T**.
* Устанавливаем указатель на начало.
* Перемещаем указатель до тех пор, пока **A** меньше элементов списка в который мы вставляем или пока не конец списка.
* Вставляем нужный элемент после указателя на текущий.

Сложение полиномов:

* На вход подаем два полинома, которые представляют собой циклический список.
* Сравниваем мономы(элементы списков).
* Больший моном записываем в результат и передвигаем указатель списка, из которого взяли элемент, на одно звено вперед.
* Если мономы равны то складываем их коэффициенты, записываем в результат, передвигаем указатели списков на одно звено вперед.
* Когда один из полиномов закончился, остаток второго записываем в результат.

Вычитание полиномов:

* Выполняем при помощи сложения полиномов с умножением одного из них на минус один.
* a=b-c=b+(-1)\*c.

Умножение полинома на константу:

* На вход подается полином и константа
* Создаем полином результата и приравниваем его к исходному полиному.
* Идем по полиному результата пока он не закончился и умножаем каждый коэффициент монома на константу.

Умножение полиномов:

* На вход подаются два полинома.
* Создается полином результата(res).
* Идем по одному из полиномов пока он не закончился и каждый его элемент умножаем на каждый элемент другого полинома(так же идем по другому полиному пока он не закончится), записываем результат в res.

Заключение

В лабораторной работе был реализован класс полином, который представляет собой циклический список с головой. Были реализованы арифметические операции над полиномами(+,-,\*).

Литература

1. Рабочие материалы к учебному курсу «Методы программирования». *Гергель В.П*. 2002 г.

Приложение

Приложение А. Программная реализация циклического списка

#pragma once

#include <iostream>

template <typename T>

class Node

{

public:

T data;

Node<T>\* next;

Node(T d = NULL, Node<T>\* l = NULL)

{

data = d;

next = l;

}

bool operator< (const Node<T>& n) const { return (data<n.data); }

bool operator> (const Node<T>& n) const { return (data>n.data); }

bool operator!= (const Node<T>& n) const { return !(\*this == n); }

bool operator==(const Node<T>& n) const { return (data == n.data && next == n.next); }

};

#pragma once

#include "node.h"

template<typename T>

class List

{

protected:

Node<T> \*head;

Node<T> \*current;

public:

List();

List(const List<T> &lst);

~List();

void Clean();

List<T>& operator=(const List<T>& lst);

Node<T>\* GetCurr() const { return current; }

void InsertAfter(Node<T>\* n, T d);

void InsertOrdered(T d);

void GetNext() { current = current->next; }

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

bool IsEnded() const { return current == head; }

bool operator==(const List<T>& lst) const;

bool operator!=(const List<T>& lst) const;

};

template <typename T>

List<T>::List()

{

head = new Node<T>;

head->next = head;

current = head->next;

}

template <typename T>

List<T>::List(const List<T>& lst)

{

Node<T>\* tmp = lst.head;

head = new Node<T>(tmp->data);

head->next = head;

current = head;

while (tmp->next != lst.head)

tmp = tmp->next;

current->next = new Node<T>(tmp->data);

GetNext();

}

current->next = head;

}

template <class T>

List<T>::~List()

{

Clean();

delete head;

}

template <class T>

List<T>& List<T>::operator=(const List<T>& lst)

{

Clean();

Node<T>\* tmp = lst.head;

Node<T>\* tmp2 = head;

while (tmp->next != lst.head)

{

tmp = tmp->next;

tmp2->next = new Node<T>(tmp->data);

tmp2 = tmp2->next;

}

tmp2->next = head;

current = head->next;

return \*this;

}

template <class T>

void List<T>::InsertAfter(Node<T>\* n, T d)

{

Node<T>\* tmp = n->next;

n->next = new Node<T>(d);

n->next->next = tmp;

}

template <class T>

void List<T>::InsertOrdered(T d)

{

Node<T>\* tmp;

current = head;

while ((current->next->data > d) && current->next != head)

{

GetNext();

}

tmp = current->next;

current->next = new Node<T>(d);

current->next->next = tmp;

}

template <class T>

void List<T>::Clean()

{

Node<T>\* tmp = head->next;

Node<T>\* tmp2;

while (tmp != head)

{

tmp2 = tmp->next;

delete tmp;

tmp = tmp2;

}

head->next = head;

}

template<class T>

bool List<T>::operator==(const List<T>& lst) const

{

bool res = true;

if (this != &lst)

{

Node<T>\* tmp1 = head->next;

Node<T>\* tmp2 = lst.head->next;

while (tmp1->data == tmp2->data && tmp1 != head && tmp2 != lst.head)

{

tmp1 = tmp1->next;

tmp2 = tmp2->next;

}

if (tmp1 != head || tmp2 != lst.head)

res = false;

}

return res;

}

template<class T>

bool List<T>::operator!=(const List<T>& lst)const

{

return !(\*this == lst);

}

Приложение Б. Программная реализация монома

#pragma once

class Monom

{

public:

double cf;

unsigned int abc;

Monom(double a= 0 , unsigned int b = 0) { cf = a; abc = b;}

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;

};

#include "monom.h"

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

{

cf = m.cf;

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

{

return (abc == m.abc && cf == m.cf);

}

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

{

return !(\*this == m);

}

Приложение В. Программная реализация полинома

#pragma once

#include <iostream>

#include <string>

#include <algorithm>

#include "monom.h"

#include "List.h"

#define X 120

using namespace std;

class Polinom

{

private:

List<Monom> list\_m;

public:

Polinom(const string Line = "" );

Polinom(const Monom m) { list\_m.InsertAfter(list\_m.GetCurr(),m); }

Polinom(const List<Monom> &p2) { this->list\_m = p2; };

Polinom(const Polinom& pol) { this->list\_m = pol.list\_m; };

Polinom operator+ (const Polinom&) const;

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

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

Polinom operator\* (const double c) const;

bool operator== (const Polinom& pol) const { return list\_m == pol.list\_m; }

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

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

};

#include "polinom.h"

Polinom::Polinom(const string Line)

{

List<Monom> List;

string stringMonom, stringPol = Line;

int k = 1;

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

while (stringPol.length() != 0)

{

Monom tmp;

k = 1;

while (stringPol[k] != '-' && stringPol[k] != '+' && k < stringPol.length())

{

k++;

}

stringMonom = stringPol.substr(0, k);

stringPol.erase(0, k);

if (stringMonom[0] == '-')

{

tmp.cf = -1;

stringMonom.erase(0, 1);

}

else

{

tmp.cf = 1;

if (stringMonom[0] == '+')

stringMonom.erase(0, 1);

}

if (isdigit(stringMonom[0]))

{

k = 0;

while (isdigit(stringMonom[k]) || stringMonom[k] == '.')

{

k++;

}

tmp.cf \*= std::stod(stringMonom.substr(0, k));

stringMonom.erase(0, k);

}

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

{

int pos = stringMonom.find((char)(X + i));

if (pos != -1)

if (stringMonom[pos + 1] == '^')

{

tmp.abc += cof[i] \* std::stoi(stringMonom.substr(pos + 2, 1));

}

else

tmp.abc += cof[i];

}

List.InsertOrdered(tmp);

}

list\_m = List;

}

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

{

Polinom res;

List<Monom> p1 = list\_m;

List<Monom> p2 = pol.list\_m;

p1.Reset();

p2.Reset();

res.list\_m.Reset();

while (!p1.IsEnded() && !p2.IsEnded())

{

if ((p1.GetCurr()->data > p2.GetCurr()->data) && p1.GetCurr()->data.cf)

{

res.list\_m.InsertAfter(res.list\_m.GetCurr(), p1.GetCurr()->data);

p1.GetNext();

res.list\_m.GetNext();

}

else

if ((p1.GetCurr()->data < p2.GetCurr()->data) && p2.GetCurr()->data.cf)

{

res.list\_m.InsertAfter(res.list\_m.GetCurr(), p2.GetCurr()->data);

p2.GetNext();

res.list\_m.GetNext();

}

else

{

double cf = p1.GetCurr()->data.cf + p2.GetCurr()->data.cf;

if (cf)

{

Monom temp(cf, p1.GetCurr()->data.abc);

res.list\_m.InsertAfter(res.list\_m.GetCurr(), temp);

res.list\_m.GetNext();

}

p1.GetNext();

p2.GetNext();

}

}

while (!p1.IsEnded())

{

if (p1.GetCurr()->data.cf)

{

res.list\_m.InsertAfter(res.list\_m.GetCurr(), p1.GetCurr()->data);

res.list\_m.GetNext();

}

p1.GetNext();

}

while (!p2.IsEnded())

{

if (p2.GetCurr()->data.cf)

{

res.list\_m.InsertAfter(res.list\_m.GetCurr(), p2.GetCurr()->data);

res.list\_m.GetNext();

}

p2.GetNext();

}

res.list\_m.Reset();

if (res.list\_m.GetCurr()->data.abc == 0 && res.list\_m.GetCurr()->data.cf == 0)

res.list\_m.InsertAfter(res.list\_m.GetCurr(), NULL);

return res;

}

Polinom Polinom::operator\*(const double c) const

{

Polinom res;

if (c)

{

res = \*this;

res.list\_m.Reset();

while (!(res.list\_m.IsEnded()))

{

res.list\_m.GetCurr()->data.cf \*= c;

res.list\_m.GetNext();

}

}

else

{

res.list\_m.InsertAfter(res.list\_m.GetCurr(), NULL);

}

return res;

}

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

{

Polinom res;

Polinom p1 = list\_m, p2 = pol;

int abc;

double cf;

p1.list\_m.Reset();

p2.list\_m.Reset();

while (!p2.list\_m.IsEnded())

{

Monom tmp;

p1.list\_m.Reset();

while (!p1.list\_m.IsEnded())

{

abc = p1.list\_m.GetCurr()->data.abc;

cf = p1.list\_m.GetCurr()->data.cf;

tmp = p2.list\_m.GetCurr()->data;

if ( (tmp.abc % 10 + abc % 10) < 10 && (tmp.abc /10 % 10 + abc / 10 % 10) < 10 && (tmp.abc / 100 + abc / 100 ) <10)

{

tmp.cf \*= cf;

tmp.abc += abc;

res = res + tmp;

}

else

{

throw "Degree>10";

}

p1.list\_m.GetNext();

}

p2.list\_m.GetNext();

}

return res;

}

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

{

Polinom tmpPol = pol;

tmpPol.list\_m.Reset();

Node<Monom>\* FirstNode =tmpPol.list\_m.GetCurr();

Node<Monom> tmpMon;

while (!(tmpPol.list\_m.IsEnded()))

{

tmpMon = tmpPol.list\_m.GetCurr()->data;

if (tmpMon.data.cf > 0 && tmpPol.list\_m.GetCurr() != FirstNode)

os << '+';

if (tmpMon.data.cf != 1 && tmpMon.data.cf != -1 && tmpMon.data.cf !=0)

os << tmpMon.data.cf;

else

if (tmpMon.data.cf == -1 && tmpMon.data.abc != 0)

os << '-';

else

if ((tmpMon.data.cf == 1 || tmpMon.data.cf == -1 )&& tmpMon.data.abc == 0)

os << tmpMon.data.cf;

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

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

{

if (tmpMon.data.abc / cof[i] % 10 != 0)

{

os << (char)(X + i);

if (tmpMon.data.abc / cof[i] % 10 != 1)

os << '^' << tmpMon.data.abc / cof[i] % 10;

}

}

tmpPol.list\_m.GetNext();

}

tmpPol.list\_m.Reset();

if (tmpPol.list\_m.GetCurr()->data.abc == 0 && tmpPol.list\_m.GetCurr()->data.cf == 0 && !(tmpPol.list\_m.IsEnded()))

{

os << "0";

}

return os;

}

Приложение Г. Реализация основного приложения

#include "polinom.h"

#include "List.h"

int main()

{

cout << "use x,y,z " << endl << endl;

int f = 1;

string pol1;

string pol2;

Polinom res;

while (f != 0)

{

cout << "enter the first polynomial" << endl;

cin >> pol1;

Polinom A(pol1);

cout << "first polinomial = " << A << endl;

cout << "enter the second polynomial" << endl;

cin >> pol2;

Polinom B(pol2);

cout << "second polinomial = " << B << endl << endl;

res = A + B;

cout << " sum = ";

cout << res << endl;

res = A - B;

cout << " difference = ";

cout << res << endl;

res = A \* B;

cout << " multiplication = ";

cout << res << endl;

cout << endl;

cout << "enter 0 for exit and any character to continue " << endl;

cin >> f;

cout << endl;

}

return 0;

}

Приложение Д. Тесты для List

#include <gtest.h>

#include "node.h"

#include "list.h"

TEST(Node, can\_create\_node)

{

ASSERT\_NO\_THROW(Node<int> N);

}

TEST(List, can\_create\_List)

{

ASSERT\_NO\_THROW(List<int> L);

}

class EmptyList : public testing::Test

{

protected:

List<int> L;

public:

EmptyList() {};

~EmptyList() {};

};

TEST\_F(EmptyList, new\_list\_is\_empty)

{

EXPECT\_EQ(NULL, L.GetCurr()->data);

}

TEST\_F(EmptyList, can\_copy\_empty\_list)

{

ASSERT\_NO\_THROW(List<int> L2(L));

}

TEST\_F(EmptyList, copied\_empty\_list\_is\_correct)

{

List<int> L2(L);

EXPECT\_EQ(NULL, L2.GetCurr()->data);

}

TEST\_F(EmptyList, two\_empty\_lists\_are\_eq)

{

List<int> L2;

EXPECT\_EQ(true, L == L2);

}

TEST\_F(EmptyList, can\_assign\_empty\_lists)

{

List<int> L2;

ASSERT\_NO\_THROW(L = L2);

}

TEST\_F(EmptyList, can\_assign\_itself\_an\_empty\_list)

{

ASSERT\_NO\_THROW(L = L);

}

TEST\_F(EmptyList, can\_clean\_empty\_list)

{

ASSERT\_NO\_THROW(L.Clean());

}

TEST\_F(EmptyList, clean\_empty\_list\_is\_correct)

{

L.Clean();

EXPECT\_EQ(NULL, L.GetCurr()->data);

}

TEST\_F(EmptyList, can\_get\_current\_from\_empty\_list)

{

ASSERT\_NO\_THROW(L.GetCurr());

}

TEST\_F(EmptyList, can\_reset\_empty\_list)

{

ASSERT\_NO\_THROW(L.Reset());

}

TEST\_F(EmptyList, is\_end\_is\_correct)

{

EXPECT\_EQ(true, L.IsEnded());

}

TEST\_F(EmptyList, can\_set\_next\_empty\_list)

{

ASSERT\_NO\_THROW(L.GetNext());

}

TEST\_F(EmptyList, can\_insert\_after\_in\_empty\_list)

{

L.Reset();

ASSERT\_NO\_THROW(L.InsertAfter(L.GetCurr(), 1));

}

TEST\_F(EmptyList, insert\_after\_in\_empty\_list\_is\_correct)

{

L.Reset();

L.InsertAfter(L.GetCurr(), 5);

L.Reset();

int tmp = L.GetCurr()->data;

EXPECT\_EQ(5, tmp);

}

Приложение Е. Тесты для полинома

#include "gtest.h"

#include "polinom.h"

#include <vector>

using std::vector;

TEST(Monom, can\_create\_monom)

{

ASSERT\_NO\_THROW(Monom M);

}

TEST(Polinom, can\_create\_polinom)

{

ASSERT\_NO\_THROW(Polinom p);

}

TEST(Polinom, can\_copy\_polinoms)

{

Polinom B("x^2y^3z");

ASSERT\_NO\_THROW(Polinom A(B));

}

struct string\_test

{

string s;

Polinom res;

string\_test(string str, const vector<Monom> &vec)

{

s = str;

List<Monom> tmp;

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

tmp.InsertOrdered(vec[i]);

res = Polinom(tmp);

}

};

class testpars : public ::testing::TestWithParam<string\_test>

{

protected:

Polinom P;

public:

testpars() : P(GetParam().s){}

~testpars() {}

};

TEST\_P(testpars, correct\_parse\_of\_polynom\_strings)

{

EXPECT\_EQ(GetParam().res, P);

}

INSTANTIATE\_TEST\_CASE\_P(tst1,testpars,::testing::Values(

string\_test("0", vector<Monom> {Monom()}),

string\_test("-zyx", vector<Monom> {Monom(-1, 111)}),

string\_test("-2x^2", vector<Monom> {Monom(-2, 200)}),

string\_test("z^2", vector<Monom> {Monom(1, 2)}),

string\_test("x^2yz", vector<Monom> {Monom(1, 211)}),

string\_test("x+y", vector<Monom> {Monom(1, 100), Monom(1, 10)}),

string\_test("17.9xy", vector<Monom> {Monom(17.9, 110)})

));

class Operat

{

public:

string P1, P2, res;

Operat(string r, string a, string b)

{

P1 = a;

P2 = b;

res = r;

}

};

class TestSum : public ::testing::TestWithParam<Operat>

{

public:

Polinom pol1, pol2, Res;

TestSum() : pol1(GetParam().P1), pol2(GetParam().P2), Res(GetParam().res) {};

~TestSum() {};

};

TEST\_P(TestSum, sum)

{

EXPECT\_EQ(Res, pol1 + pol2);

}

INSTANTIATE\_TEST\_CASE\_P(tst2, TestSum, ::testing::Values(

Operat("0", "1", "-1"),

Operat("0", "x", "-x"),

Operat("0", "-xyz", "xyz"),

Operat("2xyz", "yzx", "xyz"),

Operat("x+y", "x", "y"),

Operat("3.14x", "3x", "0.14x"),

Operat("18x^2y^2", "10x^2y^2+x^2", "8x^2y^2-x^2")

));

class TestMult : public ::testing::TestWithParam<Operat>

{

public:

Polinom pol1, pol2, Res;

TestMult() : pol1(GetParam().P1), pol2(GetParam().P2), Res(GetParam().res) {}

~TestMult() {}

};

TEST\_P(TestMult, mult)

{

EXPECT\_EQ(Res, pol1 \* pol2);

}

INSTANTIATE\_TEST\_CASE\_P(tst3, TestMult, ::testing::Values(

Operat("2", "2", "1"),

Operat("xyz", "xyz", "1"),

Operat("xyz^2", "xz", "yz"),

Operat("200x^3", "10x", "20x^2"),

Operat("12xyz", "6xy", "2z"),

Operat("x+y+z", "1", "x+y+z"),

Operat("x^2-y^2", "x-y", "x+y"),

Operat("x^3+y^3", "x+y", "y^2-xy+x^2")

));