Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение  
высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт Информационных технологий, математики и механики

Отчёт по лабораторной работе

Операции над полиномами

Выполнил:

студент гр. 381603-01

Федотова Е.А.

Проверил:

ассистент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2018 г

Содержание

[Введение 3](#_Toc515262250)

[1 Постановка задачи 4](#_Toc515262251)

[2 Руководство пользователя 5](#_Toc515262252)

[3 Руководство программиста 6](#_Toc515262253)

[3.1 Структура программы 6](#_Toc515262254)

[3.2 Структуры данных 6](#_Toc515262255)

[3.2.1 Описание класса Node 6](#_Toc515262256)

[3.2.2 Описание класса TRinglist 7](#_Toc515262257)

[3.2.3 Описание класса TMonom 7](#_Toc515262258)

[3.2.4 Описание класса TPolinоm 8](#_Toc515262259)

[3.3 Описание алгоритмов 9](#_Toc515262260)

[3.3.1 Алгоритмы класса TPolinom 9](#_Toc515262261)

[Заключение 10](#_Toc515262262)

[Литература 11](#_Toc515262263)

[Приложение 12](#_Toc515262264)

[Приложение 1. Программная реализация звена списка и самого списка 12](#_Toc515262265)

[Приложение 2. Программная реализация монома и полинома 16](#_Toc515262266)

[Приложение 3. Основная программа 22](#_Toc515262267)

[Приложение 4. Тесты 23](#_Toc515262268)

# Введение

Данная лабораторная работа посвящена вычислению арифметических выражений с полиномами. С различными вычислениями человек сталкивается ежедневно. И сложно представить нашу жизнь без каких-либо подсчетов, потому что они встречаются на каждом шагу. Все арифметические выражения являются неотъемлемой частью всех вычислительных программ, поэтому в составе языковых процессоров необходимо иметь алгоритмы, распознающие арифметические выражения с полиномами и вычисляющие их как можно быстрее.

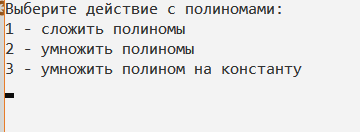
Односвязный циклический список – это структура данных, представляющая собой последовательность элементов, последний элемент которого содержит указатель на первый элемент списка. Основная особенность такой организации состоит в том, что в этом списке нет элементов, содержащих пустые указатели, и, следовательно, нельзя выделить крайние элементы. Эту структуру данных я использую в данной лабораторной работе.

# Постановка задачи

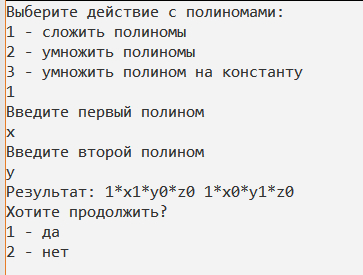
В лабораторной работе нужно реализовать подсчет арифметических выражений с полиномами с помощью циклического списка и нескольких классов, описанных ниже. Для полиномов нужно перегрузить операции сложения, умножения и умножения на константу. Также нужно написать тесты для проверки правильности написанных функций и конструкторов.

# Руководство пользователя

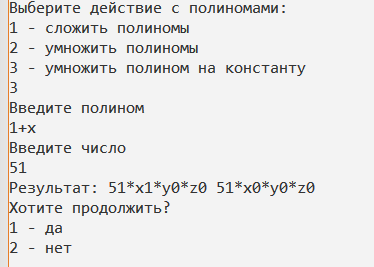
1. При запуске программы пользователь может выбрать, какое действие произвести над полиномами:



1. Варианты действия с полиномами
2. Выбрав одно из них, нажав при этом 1, 2 или 3, он может ввести два полинома (если нажмет 1 или 2) или полином и константу (если нажмет 3):



1. Сложение полиномов



1. Умножение на константу
2. Получив результат, он может либо выйти из программы, нажав 2, либо продолжить работу, нажав 1.

# Руководство программиста

## Структура программы

Программа разделена на несколько файлов: node.h, list.h, monom.h, polinom.h, polinom.cpp, sample\_list.cpp, test\_list.cpp, test\_polinom.cpp.

В файле node.h осуществляется описание и реализация шаблоны класса Node. Он является звеном односвязного циклического списка. Этот файл подключен к файлу list.h, в котором описан и реализован шаблонный класс TRinglist. Он представляет собой односвязный циклический список. Файл list.h подключен к файлам monom.h и test\_list.cpp.

В файле monom.h осуществляется описание класса TMonom, а его реализация в файле polinom.cpp, куда подключен файл monom.h. Этот класс представляет собой слагаемое в полиноме, то есть одночлен.

В файле polinom.h осуществляется описание класса TPolinom. Он является многочленом. Реализация этого класса осуществляется в файле polinom.cpp, куда подключен файл polinom.h. Также этот файл подключен к файлу test\_polinom.cpp.

В файлах test\_list.cpp и test\_polinom.cpp находятся тесты, которые проверяют правильность написанных функций, конструкторов и перегрузок в классах.

В файле sample\_list.cpp находится «руководство для пользователя».

## Структуры данных

### Описание класса Node

Поля:

T data; данные в одном звене списка;

Node<T>\* next; указатель на следующее звено;

Методы:

Node() – конструктор по умолчанию;

Node(const Node<T> &n) – конструктор копирования;

Node(const T &d) – конструктор создания звена списка с данными;

~Node()- деструктор;

const Node<T>&operator=(const Node<T> &n) – перегрузка операции равно;

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

### **Описание класса TRinglist**

Поля:

Node<T>\* head – указатель на голову списка;

Node<T>\* curr – «ходилка» по списку;

Node<T>\* tail – указатель на хвост списка;

Методы:

TRinglist()- конструктор по умолчанию;

TRinglist(const TRinglist<T> &l) – конструктор копирования;

~TRinglist()- деструткор;

void Clean()- очистить список;

void Reset()- curr станет head;

void Getnext() – curr перейдет на следующий;

bool Isended() const – проверка на конец списка;

void del(Node<T> \*l) – удаление звена из списка;

T & Getdata() const – возвращает данные у curr;

Node<T>\* search(const T &d) – поиск звена с заданными данными;

void Insert\_to\_tail(const T &d) – вставка в конец;

void Insert\_Up(const T &d) – вставка в упорядоченный список;

bool operator==(const TRinglist<T> &l) const – перегрузка сравнения;

const TRinglist<T> & operator=(const TRinglist<T> &l) – перегрузка оператора равно;

### **Описание класса TMonom**

Поля:

double coeff – коэффициент перед одночленом;

unsigned int abc – степень монома;

Методы:

TMonom(const double cf = 0.0, const unsigned int abc2 = 0) – конструктор по умолчанию;

TMonom(const TMonom &n) – конструктор копирования;

TMonom(const string &s) – конструктор, который введенную строку преобразует в одночлен;

~TMonom()- деструктор;

const TMonom& operator=(const TMonom &m) – перегрузка оператора равно;

bool operator==(const TMonom &m) const – перегрузка сравнения;

bool operator!=(const TMonom &m) const – перегрузка сравнения;

bool operator<(const TMonom &m) const – перегрузка сравнения;

bool operator>(const TMonom &m) const – перегрузка сравнения;

TMonom operator\*(const double d) const – перегрузка умножения монома на константу справа;

TMonom operator\*(const TMonom &m) const – перегрузка умножения мономов;

friend TMonom operator\*(const double d, const TMonom &m) – перегрузка умножения монома на константу слева;

### **Описание класса TPolinоm**

Поля:

TRinglist<TMonom> pol – список из мономов;

Методы:

TPolinom()- конструктор по умолчанию;

TPolinom(const TPolinom &p) – конструктор копирования;

TPolinom(const string &s) – конструктор, который введенную строку преобразует в полином;

~TPolinom()- деструктор;

const TPolinom& operator=(const TPolinom &p) – перегрузка оператора равно;

TPolinom operator+(const TPolinom &p) const – перегрузка сложения полиномов;

TPolinom operator\*(const TPolinom &p) const – перегрузка умножения полиномов;

TPolinom operator\*(const double d) const – перегрузка умножения полинома на константу справа;

bool operator==(const TPolinom &p) const – перегрузка сравнения;

bool operator!=(const TPolinom &p) const – перегрузка сравнения;

friend TPolinom operator\*(const double d, const TPolinom &p) – перегрузка умножения полинома на константу слева;

## Описание алгоритмов

### Алгоритмы класса TPolinom

1. Сложение полиномов:

а) пока какой-либо из двух полиномов не закончился, мы создаем два монома. Один – слагаемое из первого полинома, второй – слагаемое из второго полинома.

б) Сравниваем эти мономы. Тот, который больше, вставляем в конец созданного полинома, который и будет результатом сложения двух полиномов. Переходим на следующее слагаемое в том полиноме, моном которого оказался больше.

в) Если мономы равны, то есть оказалось, что слагаемые из двух полиномов подобны, то складываем коэффициенты этих мономов и вставляем получившийся моном в конец результирующего полинома. Переходим на следующие слагаемые в каждом из двух полиномов.

г) если какой-либо из двух полиномов закончился, то вставляем в результирующий моном все слагаемые, которые остались в полиноме, который не закончился.

2. Умножение полиномов:

а) Берем моном из первого полинома и умножаем его на все мономы из второго, пока второй полином не закончился. Создаем моном с коэффициентом, который получается в результате умножения двух мономов из двух полиномов, и со степенью, которая получается в результате сложения степени одного монома и другого.

б) Если коэффициент в этом мономе не равен 0, то делаем проверку на подобные в результирующем полиноме. И если такие есть, то складываем у них коэффициенты.

в) Когда второй полином закончился, переходим на следующий моном в первом полиноме и возвращаемся в пункт (а), пока первый полином не закончится.

3. Умножение полинома на константу:

Пока полином не закончился, умножаем коэффициенты в каждом мономе полинома на эту константу.

# Заключение

В данной лабораторной работе мне удалось реализовать метод вычисления арифметических выражений с полиномами. Могу сделать вывод, что он работает правильно, на основе верно выполненных тестов.

# Литература

1. Лекции «Алгоритмы и структуры данных» Гергель В.П. ННГУ им. Лобачевского Н. И.

# Приложение

## Приложение 1. Программная реализация звена списка и самого списка

**Node.h**

#pragma once

#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std;

template <class T>

class Node

{

public:

T data; //данные

Node<T>\* next; //указатель на звено

Node()

{

next = NULL;

}

Node(const Node<T> &n);

Node(const T & d);

~Node()

{

}

const Node<T>&operator=(const Node<T> &n);

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

friend ostream & operator<<(ostream &os, const Node<T> &n)

{

os << n.data;

return os;

}

};

template<class T>

inline Node<T>::Node(const Node<T> &n)

{

data = n.data;

next = n.next;

}

template<class T>

inline Node<T>::Node(const T &d)

{

data = d;

next = NULL;

}

template<class T>

inline const Node<T>& Node<T>::operator=(const Node<T> &n)

{

data = n.data;

next = n.next;

return \*this;

}

template<class T>

inline bool Node<T>::operator==(const Node<T>& n) const

{

if ((data == n.data) && (next == n.next))

return true;

else

return false;

}

**List.h**

#pragma once

#include "node.h"

template <class T>

class TRinglist

{

private:

Node<T>\* head;

Node<T>\* curr; //ходилка

Node<T>\* tail; //указатель на последний эл-т спика

public:

TRinglist();

TRinglist(const TRinglist<T> &l);

~TRinglist();

void Clean(); //очистить список

void Reset(); //curr станет head

void Getnext(); //curr перейдет на следующий

bool Isended() const; //дошли ли до конца?

void del(Node<T> \*l); ////////////////////////

T & Getdata() const; //вернет данные у curr

Node<T>\* search(const T &d);

void Insert\_to\_tail(const T &d); //вставка в конец

void Insert\_Up(const T &d); //вставка в упорядоченный список

bool operator==(const TRinglist<T> &l) const; //сравнение списков

const TRinglist<T> & operator=(const TRinglist<T> &l);/////////

friend ostream& operator<<(ostream &out, const TRinglist<T> &n)

{

Node<T>\* t = n.head->next;

if (t == n.head)

out << "0";

while (t != n.head)

{

out << t->data << " ";

t = t->next;

}

return out;

}

};

template<class T>

inline TRinglist<T>::TRinglist()

{

head = new Node<T>();

tail = head;

head->next = head;

}

template<class T>

inline TRinglist<T>::TRinglist(const TRinglist<T>& l)

{

head = new Node<T>();

Node<T>\* t = head;

Node<T>\* tt = l.head->next;

while (tt != l.head)

{

t->next = new Node<T>(tt->data);

tail = t->next; /////////

t = t->next;

tt = tt->next;

}

tail->next = head;

}

template<class T>

inline TRinglist<T>::~TRinglist()

{

Clean();

delete head; //////////

}

template<class T>

inline void TRinglist<T>::Clean()

{

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

while (t != head)

{

Node<T>\* tt = t->next;

delete t;

t = tt;

}

head->next = head; //////

tail = head; ///////

}

template<class T> ////////////////////////////

inline void TRinglist<T>::del(Node<T> \*l)

{

Node<T> \*t = head;

while(t->next != l)

t = t->next;

t->next = l->next;

delete l;

}

template<class T>

inline void TRinglist<T>::Reset()

{

curr = head;

}

template<class T>

inline void TRinglist<T>::Getnext()

{

curr = curr->next;

}

template<class T>

inline bool TRinglist<T>::Isended() const

{

return (curr == head);

}

template<class T>

inline T & TRinglist<T>::Getdata() const

{

return curr->data;

}

template<class T>

inline Node<T>\* TRinglist<T>::search(const T &d)

{

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

while ((t != head) && (t->data != d))

{

t = t->next;

}

if (t == head)

return NULL;

else

return t;

}

template<class T>

inline void TRinglist<T>::Insert\_to\_tail(const T & d)

{

Node<T>\* t = new Node<T>(d);

tail->next = t;

tail = t;

tail->next = head;

}

template<class T>

inline void TRinglist<T>::Insert\_Up(const T & d)

{

Node<T>\* t = new Node<T>(d);

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

Node<T>\* pred = head;

while ((tt != head) && (tt->data > d))

{

pred = tt;

tt = tt->next;

}

t->next = tt;

pred->next = t;

if (tt == head)

{

tail = t;

}

}

template<class T>

inline bool TRinglist<T>::operator==(const TRinglist<T> &l) const

{

bool flag = true;

if (this != &l)

{

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

Node<T>\* tt = l.head->next;

while (t->data == tt->data && t != head && tt != l.head)

{

t = t->next;

tt = tt->next;

}

if (t != head || tt != l.head)

flag = false;

}

return flag;

}

template<class T>

const TRinglist<T> & TRinglist<T>::operator=(const TRinglist<T> &l)

{

Clean();

head = new Node<T>();

Node<T>\* t = head;

Node<T>\* tt = l.head->next;

tail = head;

while(tt != l.head)

{

t->next = new Node<T>(tt->data);

tail = t->next;

t = t->next;

tt = tt->next;

}

tail->next = head;

return \*this;

}

## Приложение 2. Программная реализация монома и полинома

**Monom.h**

#pragma once

#include "list.h"

class TMonom

{

public:

double coeff;

unsigned int abc;

TMonom(const double cf = 0.0, const unsigned int abc2 = 0);

TMonom(const TMonom &n);

TMonom(const string &s);

~TMonom() {}

const TMonom& operator=(const TMonom &m);

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

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

bool operator<(const TMonom &m) const;

bool operator>(const TMonom &m) const;

TMonom operator\*(const double d) const;

TMonom operator\*(const TMonom &m) const;

friend TMonom operator\*(const double d, const TMonom &m)

{

return (m\*d);

}

friend ostream & operator<<(ostream &os, const TMonom &n);

};

**Polinom.h**

#pragma once

#include "monom.h"

class TPolinom

{

TRinglist<TMonom> pol;

public:

TPolinom() {}

TPolinom(const TPolinom &p);

TPolinom(const string &s);

~TPolinom() {}

const TPolinom& operator=(const TPolinom &p);

TPolinom operator+(const TPolinom &p) const;

TPolinom operator\*(const TPolinom &p) const;

TPolinom operator\*(const double d) const;

bool operator==(const TPolinom &p) const;

bool operator!=(const TPolinom &p) const;

friend TPolinom operator\*(const double d, const TPolinom &p)

{

return (p\*d);

}

friend ostream& operator<<(ostream& out, const TPolinom &p);

};

double rank\_number(const double a, const int i);

double converter\_in\_number(const string &s);

**Polinom.cpp**

#include "monom.h"

#include "polinom.h"

TMonom::TMonom(const double cf, const unsigned int abc2)

{

coeff = cf;

abc = abc2;

}

TMonom::TMonom(const TMonom &n)

{

coeff = n.coeff;

abc = n.abc;

}

TMonom::TMonom(const string &s)

{

string c = s + ' ';

bool f = true;

int min = 1, i;

abc = 0;

string cof;

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

{

min = -1;

c.erase(0, 1);

}

int len = c.length();

for (i = 0; f; i++)

{

if (((c[i] >= '0') && (c[i] <= '9')) || (c[i] == '.'))

{

cof = cof + c[i];

}

else

{

f = false;

if (cof != "")

coeff = min \* atof(cof.c\_str());

else

coeff = min;

}

}

for (int j = i-1; j<len; j++)

{

if (c[j] == 'x')

{

if ((c[j + 1] >= '0') && (c[j + 1] <= '9'))

{

abc = abc + 100 \* (c[j + 1] - '0');

j++;

}

else

abc = abc + 100;

}

else

if (c[j] == 'y')

{

if ((c[j + 1] >= '0') && (c[j + 1] <= '9'))

{

abc = abc + 10 \* (c[j + 1] - '0');

j++;

}

else

abc = abc + 10;

}

else

if (c[j] == 'z')

{

if ((c[j + 1] >= '0') && (c[j + 1] <= '9'))

{

abc = abc + (c[j + 1] - '0');

j++;

}

else

abc = abc + 1;

}

}

}

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

{

coeff = m.coeff;

abc = m.abc;

return \*this;

}

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

{

if (abc == m.abc)

return true;

else

return false;

}

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

{

if (abc != m.abc)

return true;

else

return false;

}

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

{

if (abc<m.abc)

return true;

else

return false;

}

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

{

if (abc>m.abc)

return true;

else

return false;

}

TMonom TMonom::operator\*(const double d) const

{

TMonom t(coeff\*d, abc);

return t;

}

TMonom TMonom::operator\*(const TMonom &m) const

{

if (abc + m.abc > 999)

throw "error";

else

{

TMonom t(coeff \* m.coeff, abc + m.abc);

return t;

}

}

ostream & operator<<(ostream &os, const TMonom &n)

{

os << n.coeff << "\*x" << n.abc / 100 << "\*y" << (n.abc / 10) % 10 << "\*z" << n.abc % 10;

return os;

}

//----------------------------------------------

TPolinom::TPolinom(const TPolinom &p)

{

pol = p.pol;

}

TPolinom::TPolinom(const string &s)

{

string c = s + '+';

int len = c.length();

bool min = false;

string mon;

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

{

if ((c[i] != '+') && (c[i] != '-'))

{

mon = mon + c[i];

}

else

{

if (mon != "")

{

if (min)

{

mon = '-' + mon;

}

TMonom b(mon);

mon = "";

Node<TMonom>\* t = pol.search(b);

if (t != NULL)

{

t->data.coeff += b.coeff;

}

else

pol.Insert\_Up(b);

}

if (c[i] == '-')

min = true;

else

min = false;

}

}

}

const TPolinom& TPolinom::operator=(const TPolinom &p)

{

pol = p.pol;

return \*this;

}

TPolinom TPolinom::operator+(const TPolinom &p) const

{

TRinglist<TMonom> Nowpol = pol;

TRinglist<TMonom> Thispol = p.pol;

Nowpol.Reset();

Thispol.Reset();

Nowpol.Getnext();

Thispol.Getnext();

TPolinom summa;

while ((!Nowpol.Isended()) && (!Thispol.Isended()))

{

TMonom a(Nowpol.Getdata());

TMonom b(Thispol.Getdata());

if (a > b)

{

summa.pol.Insert\_to\_tail(a);

Nowpol.Getnext();

}

else

if (a < b)

{

summa.pol.Insert\_to\_tail(b);

Thispol.Getnext();

}

else

{

TMonom c(a.coeff + b.coeff, a.abc);

if (c.coeff != 0)

summa.pol.Insert\_to\_tail(c);

Nowpol.Getnext();

Thispol.Getnext();

}

}

while (!Nowpol.Isended())

{

summa.pol.Insert\_to\_tail(pol.Getdata());

Nowpol.Getnext();

}

while (!Thispol.Isended())

{

summa.pol.Insert\_to\_tail(Thispol.Getdata());

Thispol.Getnext();

}

return summa;

}

TPolinom TPolinom::operator\*(const TPolinom &p) const

{

TRinglist<TMonom> Nowpol = pol;

TRinglist<TMonom> Thispol = p.pol;

Nowpol.Reset();

Thispol.Reset();

Nowpol.Getnext();

Thispol.Getnext();

TPolinom mult;

while (!Nowpol.Isended())

{

while (!Thispol.Isended())

{

TMonom c(Nowpol.Getdata() \* Thispol.Getdata());

if (c.coeff != 0.0)

{

Node<TMonom>\* t = mult.pol.search(c);

if (t != NULL)

{

t->data.coeff += c.coeff;

if (t->data.coeff == 0.0)

mult.pol.del(t);

}

else

{

mult.pol.Insert\_Up(c);

}

}

Thispol.Getnext();

}

Nowpol.Getnext();

Thispol.Reset();

Thispol.Getnext();

}

return mult;

}

TPolinom TPolinom::operator\*(const double d) const

{

TRinglist<TMonom> Nowpol = pol;

TPolinom multip;

if (d != 0)

{

Nowpol.Reset();

Nowpol.Getnext();

while (!Nowpol.Isended())

{

multip.pol.Insert\_to\_tail(Nowpol.Getdata()\*d);

Nowpol.Getnext();

}

}

return multip;

}

bool TPolinom::operator==(const TPolinom &p) const

{

if (this != &p)

{

bool flag = true;

TPolinom pol1(\*this), pol2(p);

pol1.pol.Reset();

pol1.pol.Getnext();

pol2.pol.Reset();

pol2.pol.Getnext();

while ((flag) && (!pol1.pol.Isended()) && (!pol2.pol.Isended()))

{

TMonom a = pol1.pol.Getdata();

TMonom b = pol2.pol.Getdata();

if (a == b)

{

if (a.coeff != b.coeff)

flag = false;

}

else

flag = false;

pol1.pol.Getnext();

pol2.pol.Getnext();

}

if ((!pol1.pol.Isended()) || (!pol2.pol.Isended()))

flag = false;

return flag;

}

else

return true;

}

bool TPolinom::operator!=(const TPolinom &p) const

{

return !(\*this == p);

}

ostream & operator<<(ostream & out, const TPolinom &p)

{

out << p.pol;

return out;

}

## Приложение 3. Основная программа

**sample\_list.main**

**#include "polinom.h"**

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

{

int f;

do

{

cout << "Выберите действие с полиномами:" << endl;

cout << "1 - сложить полиномы" << endl;

cout << "2 - умножить полиномы" << endl;

cout << "3 - умножить полином на константу" << endl;

cin >> f;

cin.ignore();

if (f == 1)

{

string s;

cout << "Введите первый полином" << endl;

getline(cin, s);

TPolinom p1(s);

cout << "Введите второй полином" << endl;

getline(cin, s);

TPolinom p2(s);

TPolinom p3 = p1 + p2;

cout << "Результат: " << p3 << endl;

}

else

if (f == 2)

{

string s;

cout << "Введите первый полином" << endl;

getline(cin, s);

TPolinom p1(s);

cout << "Введите второй полином" << endl;

getline(cin, s);

TPolinom p2(s);

TPolinom p3 = p1 \* p2;

cout << "Результат: " << p3 << endl;

}

else

if (f == 3)

{

string s;

cout << "Введите полином" << endl;

getline(cin, s);

TPolinom p(s);

cout << "Введите число" << endl;

double d;

cin >> d;

p = p \* d;

cout << "Результат: " << p << endl;

}

else

cout << "Выберите еще раз" << endl;

cout << "Хотите продолжить?" << endl;

cout << "1 - да" << endl;

cout << "2 - нет" << endl;

cin >> f;

} while (f == 1);

return 0;

}

}

## Приложение 4. Тесты

**Test\_list.cpp**

#include "list.h"

#include <gtest.h>

TEST(Node, can\_create\_empty\_node)

{

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

}

TEST(Node, can\_create\_node\_with\_data)

{

Node<int> a(20);

EXPECT\_EQ(a.data, 20);

}

TEST(Node, can\_create\_copied\_node)

{

Node<int> a;

Node<int> b(a);

EXPECT\_EQ(b, a);

}

TEST(Node, can\_assign)

{

Node<int> a;

Node<int> b(5);

a = b;

EXPECT\_EQ(a.data, 5);

}

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

{

ASSERT\_NO\_THROW(TRinglist<int> a);

}

TEST(TRinglist, can\_compare\_equall)

{

TRinglist<int> a;

a.Insert\_Up(2);

TRinglist<int> b;

b = a;

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

}

TEST(TRinglist, can\_create\_copied)

{

TRinglist<int> b;

b.Insert\_Up(2);

TRinglist<int> c(b);

EXPECT\_EQ(c, b);

}

TEST(TRinglist, can\_compare\_not\_equall)

{

TRinglist<int> a, b;

a.Insert\_Up(2);

b.Insert\_Up(3);

EXPECT\_NE(b == a, true);

}

TEST(TRinglist, can\_insert\_up)

{

TRinglist<int> a;

a.Insert\_Up(2);

a.Reset();

a.Getnext();

EXPECT\_EQ(a.Getdata(), 2);

}

TEST(TRinglist, can\_insert\_to\_tail)

{

TRinglist<int> a;

a.Insert\_to\_tail(2);

a.Reset();

a.Getnext();

EXPECT\_EQ(a.Getdata(), 2);

}

TEST(TRinglist, can\_clean)

{

TRinglist<int> a;

a.Insert\_to\_tail(5);

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

}

TEST(TRinglist, can\_delete\_elem)

{

TRinglist<int> a;

a.Insert\_Up(2);

a.Insert\_Up(3);

a.del(a.search(3));

a.Reset();

a.Getnext();

EXPECT\_EQ(a.Getdata(), 2);

}

TEST(TRinglist, can\_assign)

{

TRinglist<int> a, b;

a.Insert\_to\_tail(3);

b.Insert\_to\_tail(1);

b = a;

b.Reset();

b.Getnext();

EXPECT\_EQ(b.Getdata(), 3);

}

**Test\_polinom.cpp**

#include "polinom.h"

#include <gtest.h>

TEST(TMonom, can\_create)

{

TMonom a;

EXPECT\_EQ(a.abc, 0);

EXPECT\_EQ(a.coeff, 0.0);

}

TEST(TMonom, can\_create\_copied)

{

TMonom a(2.0), b(a);

EXPECT\_EQ(b.coeff, 2.0);

EXPECT\_EQ(b.abc, 0);

}

TEST(TMonom, can\_create2)

{

string a = "3x2y3z";

TMonom b(a);

EXPECT\_EQ(b.coeff, 3.0);

EXPECT\_EQ(b.abc, 231);

}

TEST(TMonom, can\_assign)

{

TMonom a(2.0, 3), b(1.0);

a = b;

EXPECT\_EQ(a.coeff, 1.0);

EXPECT\_EQ(a.abc, 0);

}

TEST(TMonom, can\_compare\_equall)

{

TMonom a(2.0, 3), b(1.0);

EXPECT\_NE(a == b, true);

}

TEST(TMonom, can\_compare\_not\_equall)

{

TMonom a(2.0, 3), b(1.0);

EXPECT\_NE(a, b);

}

TEST(TMonom, can\_compare1)

{

TMonom a(2.0, 3), b(1.0);

EXPECT\_EQ(a > b, true); //

}

TEST(TMonom, can\_mult\_const1)

{

TMonom a(2.0, 3);

TMonom c;

double d = 5.0;

c = a\*d;

EXPECT\_EQ(c.coeff, 10.0);

}

TEST(TMonom, can\_mult\_const2)

{

TMonom a(2.0, 3);

TMonom c;

double d = 5.0;

c = d\*a;

EXPECT\_EQ(c.coeff, 10.0);

}

TEST(TMonom, can\_mult)

{

TMonom a(2.0, 3), b(3.0, 21), c;

c = a\*b;

EXPECT\_EQ(c.coeff, 6.0);

EXPECT\_EQ(c.abc, 24);

}

TEST(TMonom, can\_check)

{

TMonom a(2.0, 800), b(3.0, 200), c;

ASSERT\_ANY\_THROW(TMonom c = a\*b);

}

TEST(TPolinom, can\_create)

{

ASSERT\_NO\_THROW(TPolinom a);

}

TEST(TPolinom, can\_create2)

{

string s = "xy-1";

ASSERT\_NO\_THROW(TPolinom a(s));

}

TEST(TPolinom, can\_create\_copied)

{

TPolinom a("x+1");

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

}

TEST(TPolinom, can\_create\_copied2)

{

TPolinom a("x+1");

TPolinom b(a);

EXPECT\_EQ(b, a);

}

TEST(TPolinom, can\_assign)

{

TPolinom a("xy+z");

TPolinom b("1+x");

b = a;

EXPECT\_EQ(b, a);

}

TEST(TPolinom, can\_compare\_not\_equal)

{

TPolinom a("xy+z");

TPolinom b("1+x");

EXPECT\_EQ(a != b, true);

}

TEST(TPolinom, can\_compare\_equal)

{

TPolinom a("xy+z");

TPolinom b("xy+z");

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

}

TEST(TPolinom, can\_addition1)

{

TPolinom a("-xy+z");

TPolinom b("xy+z");

TPolinom c("2z");

EXPECT\_EQ(c, a + b);

}

TEST(TPolinom, can\_addition2)

{

TPolinom a("2x2+3xyz+1");

TPolinom b("2+x2+z");

TPolinom c("3+z+3xyz+3x2");

EXPECT\_EQ(c, a + b);

}

TEST(TPolinom, can\_mult1)

{

TPolinom a("-xy+z");

TPolinom b("xy+z");

TPolinom c("z2-x2y2");

EXPECT\_EQ(c, a\*b);

}

TEST(TPolinom, can\_mult2)

{

TPolinom a("-xy+z+1");

TPolinom b("xy+z");

TPolinom c("z+xy+z2-x2y2");

EXPECT\_EQ(c, a\*b);

}

TEST(TPolinom, can\_mult\_const1)

{

TPolinom a("-xy+z+1");

double d = 0.0;

TPolinom c;

EXPECT\_EQ(c, a\*d);

}

TEST(TPolinom, can\_mult\_const2)

{

TPolinom a("-xy+z+1");

double d = 2.0;

TPolinom c("-2xy+2z+2");

EXPECT\_EQ(c, a\*d);

}

TEST(TPolinom, can\_mult\_const3)

{

TPolinom a("-xy+z+1");

double d = 2.0;

TPolinom c("-2xy+2z+2");

EXPECT\_EQ(c, d\*a);

}

//подобные в полиноме

TEST(TPolinom, can\_pod1)

{

TPolinom a("-xy+2-1+4xy+z");

TPolinom b("x+1");

TPolinom c1("2+x+z+3xy");

EXPECT\_EQ(c1, a + b);

}

TEST(TPolinom, can\_pod2)

{

TPolinom a("-xy+2-1+4xy+z");

TPolinom b("x+1+2x-3");

TPolinom c1("-1+3x+z+3xy");

EXPECT\_EQ(c1, a + b);

}

TEST(TPolinom, can\_pod3)

{

TPolinom a("-xy+2-1+4xy");

TPolinom b("x");

TPolinom c1("x+3x2y");

EXPECT\_EQ(c1, a \* b);

}

TEST(TPolinom, can\_pod4)

{

TPolinom a("-x+x");

TPolinom b("x");

TPolinom c1;

EXPECT\_EQ(c1, a\*b);

}