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

Кумин А.А

Проверил:

ассистент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2018 г

**Содержание**

[Введение 3](#_Toc514248456)

[1 Постановка задачи 4](#_Toc514248457)

[2 Руководство пользователя 5](#_Toc514248458)

[3 Руководство программиста 6](#_Toc514248459)

[3.1 Структура программы 6](#_Toc514248460)

[3.2 Структуры данных 6](#_Toc514248461)

[3.2.1 Шаблонный класс Node 6](#_Toc514248462)

[3.2.2 Шаблонный класс Ringlist 6](#_Toc514248463)

[3.2.3 Класс Monom 7](#_Toc514248464)

[3.2.4 Класс Polinom 8](#_Toc514248465)

[3.3 Описание алгоритмов 9](#_Toc514248466)

[3.3.1 Циклический односвязный список 9](#_Toc514248467)

[3.3.2 Класс полиномов 9](#_Toc514248468)

[Заключение 11](#_Toc514248469)

[Литература 12](#_Toc514248470)

[Приложение 13](#_Toc514248471)

[Приложение 1. Программная реализация циклического списка 13](#_Toc514248472)

[Приложение 2. Программная реализация многочленов 15](#_Toc514248473)

[Приложение 3. Основная программа 20](#_Toc514248474)

[Приложение 4. Тесты для класса Ringlist 21](#_Toc514248475)

[Приложение 5. Тесты для класса Polinom 23](#_Toc514248476)

# Введение

Многочлены или полиномы представляют собой сумму одночленов. С помощью них в математике представляют ряды, комплексные числа, функции и т.п. Чтобы работать с многочленами необходимо представить их как некоторую структуру и уметь выполнять элементарные операции над ними. Это можно сделать с помощью связного списка.

Связный список – это динамическая структура данных, состоящая из элементов, содержащих, собственно, данные и ссылку на следующий или предыдущий элемент. Его главная особенность в том, что порядок элементов списка может не совпадать с порядком расположения данных этого списка в памяти компьютера, что позволяет эффективно добавлять и удалять элементы.

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

# Постановка задачи

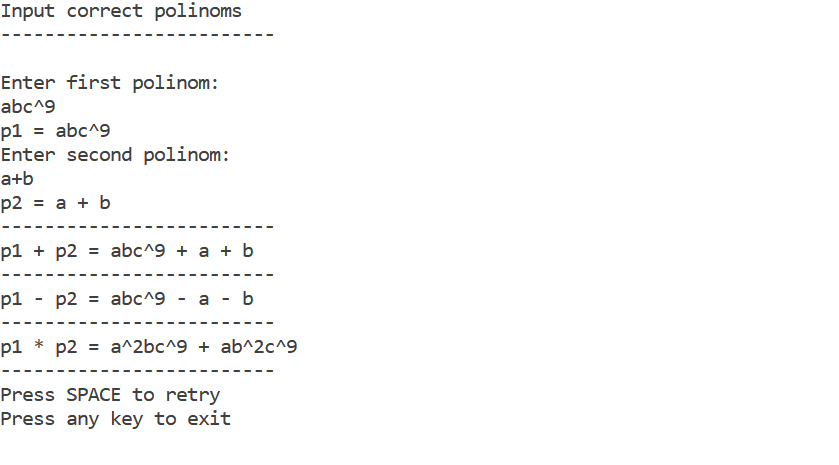
Реализовать класс многочленов трех переменных с максимальным порядком 9, на основе циклического односвязного списка. Перегрузить операции сложения, вычитания, умножения для многочленов. Разработать программу, демонстрирующую работу с многочленами, выполняющую перечисленные операции для двух многочленов.

Исходные данные: два многочлена требуемого вида.

Выходные данные: результат сложения, вычитания и умножения исходных многочленов.

# Руководство пользователя

1. Открыть файл sample\_list.exe.
2. Ввести поочередно два полинома следуя указаниям программы.
3. Программа выведет результат вычисления, при этом, чтобы ввести полиномы заново необходимо нажать клавишу пробел, а чтобы выйти из программы – любую клавишу.



1. Демонстрация работы программы

# Руководство программиста

## Структура программы

1. list.h – содержит объявление и реализацию шаблонного класса Ringlist, представляющий собой односвязный циклический список.
2. node.h – содержит объявление и реализацию шаблонного класса Node, представляющей собой элемент односвязного циклического списка.
3. polinom.h – содержит объявление и реализацию класса Monom, представляющей одночлен(моном) и объявление класса Polinom, представляющей собой многочлены и определяющего операции над ними.
4. polinom.cpp – содержит реализацию класса Polinom.
5. main.cpp – содержит реализацию пользовательского приложения.
6. test\_list.cpp – содержит реализацию тестов для класса Ringlist.
7. test\_polinom.cpp – содержит реализацию тестов для класса Polinom.

## Структуры данных

### Шаблонный класс Node

Поля(public):

**t data** – данные, хранящиеся в элементе списка.

**Node \*next** -указатель на следующий элемент списка.

Методы(**public**):

**Node(t Data = NULL, Node \*Next = NULL)** – конструктор элемента списка(по умолчанию значения NULL).

**Node(const Node &l)** – конструктор копирования.

**bool operator==(const Node<t> &l) const** – перегрузка ==.

**bool operator!=(const Node<t>& l) const** – перегрузка !=.

**bool operator<(const Node<t>& l) const** – перегрузка >.

**bool operator>(const Node<t>& l) const** – перегрузка <.

### Шаблонный класс Ringlist

Поля(private):

**Node<t> \*head** – указатель на первый элемент списка(голову).

**Node<t> \*curr** – указатель на текущий элемент списка.

Методы(public):

**Ringlist()** – конструктор по умолчанию.

**Ringlist(const Ringlist<t> &l)** – конструктор копирования.

**Ringlist<t>& operator= (const Ringlist<t> &l)** – перегрузка =.

**~Ringlist()** - деструктор

**void InsertToOrdered(const t &a)** – упорядоченная вставка элемента со значением a в список(список всегда упорядочен), принимает на вход ссылку на элемент типа t.

**void reset()** – выставляет текущий элемент списка(curr) на второй элемент.

**t& GetCurr()** – выдает значение текущего элемента списка(curr).

**void SetNext()** -выставлет значение текущего элемента списка на следующий за ним.

**int isended()** – проверяет значение текущего элемента, если curr не достиг конца списка то выведет 0, в остальных случаях – константу типа int.

**int operator==(const Ringlist<t>& l) const** – перегрузка ==.

**int operator!=(const Ringlist<t>& l) const** – перегрузка !=.

### Класс Monom

Поля(public):

**double coeff** – коэффициент монома

**int abc** – число < 1000, отвечает за степень переменных монома, первая цифра с конца – степень последней переменной c, цифра посередине – степень b, в самом начале – a.

Методы(public):

**Monom(double coeff1 = NULL, int abc1 = NULL)** – конструктор.

**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 !=.

**Monom& operator=(const Monom& m)** -перегрузка =.

### Класс Polinom

Поля(private):

**Ringlist<Monom> pol** – циклический список, с помощью которого реализуется полином.

Методы(private):

**Ringlist<Monom> Parse(const string& s)** – функция преобразующая строку в полином, принимает на вход строку, выводит полином. Закрытая функция поскольку в ходе ее работы изменяется поле класса.

Методы(public):

**Polinom()** – конструктор по умолчанию.

**Polinom(const string& str)** – конструктор.

**Polinom(const Polinom& p)** - конструктор копирования с полинома.

**Polinom(const Ringlist<Monom>& p)** – конструктор копирования с циклического списка.

**Ringlist<Monom> AddSameAndInsert(const Monom& c, const Ringlist<Monom>& p)** – функция, которая добавляет в многочлен моном и приводит подобные, принимает на вход ссылку на моном который нужно добавить и ссылку на изменяемый полином.

**Polinom& operator=(const Polinom& p)** – перегрузка =.

**Polinom operator+(const Polinom& p)** – перегрузка +.

**Polinom operator-(const Polinom& p)** – перегрузка -.

**Polinom operator\*(const int& c)** – перегрузка умножения на константу справа.

**Polinom operator\*(const Polinom& p)** – перегрузка \*.

**friend Polinom operator\*(const int c, const Polinom& p)** – перегрузка умножения на константу слева.

**friend ostream& operator<< (ostream& os, const Polinom& p)** – вывод полинома на экран.

**int operator== (const Polinom& p) const** – перегрузка ==.

**int operator!= (const Polinom& p) const** – перегрузка !=.

## Описание алгоритмов

### Циклический односвязный список

Циклический односвязный список представляет собой односвязный список, причем указатель последнего элемента указывает на начальный элемент(head). Перемещение по списку происходит с помощью поля curr. Над списком определены функции навигации (reset, GetCurr, isended, SetNext).

#### Заполнение списка

Заполнение списка производится с помощью функции **InsertToOrdered**. Она работает следующим образом:

На вход подается элемент **а** типа **t**

Находится элемент, данные которого <**a**

а вставляется перед этим элементом

Реализованный список всегда будет упорядочен, поскольку добавлять в него элементы можно только с помощью этой функции.

### Класс полиномов

Класс **Polinom**-это циклический список типа **Monom**.

**Monom** содержит коэффициент и число <1000 отвечающее за степень переменных данного одночлена.

#### Добавление элемента и приведение подобных

Добавление монома и приведение подобных реализуется с помощью функции **AddTheSameAndInsert**:

Если в списке есть элемент, степень переменных которого равна степени переменных монома, то их коэффициенты складываются.

Если нет, то с помощью функции **InsertToOrdered** моном добавляется в список.

#### Преобразование строки в полином (конструктор полинома)

Преобразование строки типа string в полином реализуется функцией **Parse**:

На вход подается строка

Из нее удаляются все пробелы и ^

Создается список типа **Monom Temp**

Строка разбивается на мономы и добавляется в Temp с помощью функции **AddTheSameAndInsert**:

Цикл от 0 до длинны строки (обход строки)

Если символ число, то оно запоминается

Если символ переменная то его степень запоминается (например a^2bc^3 -> abc(поле монома) = 213)

Создается моном на основе полученных данных и добавляется его в список **Temp**

Продолжается обход строки

Полю полинома присваивается **Temp**

#### Сложение, вычитание, умножение

Перегрузка оператора **+ (A + B)**:

элементы списка **B** поочередно добавляются в список **A** и приводятся подобные с помощью функции **AddTheSameAndInsert**.

Перегрузка оператора **- (A - B)**:

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

Перегрузка оператора **\* (A\*B)**:

**A** умножается поочередно на элементы **B** и к новому списку **C** поочередно прибавляется результат, который затем выводится:

Создается полином **Temp**

Создается итератор **i**

Пока B не конец

**A = A\*B[i]**

**Temp = Temp + A**

**i++**

# Заключение

В данной лабораторный работе был реализован класс многочленов и перегружены операции над ним (+, -, \*). Программа преобразует строки в многочлены и способна выполнять перечисленные выше операции для них.

# Литература

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

# Приложение

## Приложение 1. Программная реализация циклического списка

#pragma once

template <typename t>

class Node

{

public:

t data;

Node \*next;

Node(t Data = NULL, Node \*Next = NULL)

{

data = Data;

next = Next;

}

Node(const Node &l) { data = l.data; next = NULL; }

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

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

bool operator< (const Node<t>& l) const { return (data < l.data); }

bool operator> (const Node<t>& l) const { return (data > l.data); }

};

#pragma once

#include <iostream>

#include "node.h"

using namespace std;

template<typename t>

class Ringlist

{

Node<t> \*head;

Node<t> \*curr;

public:

Ringlist();

Ringlist(const Ringlist<t> &l);

Ringlist<t>& operator= (const Ringlist<t> &l);

~Ringlist();

void InsertToOrdered(const t &a);

void reset() { curr = head->next; };

t& GetCurr() { return curr->data; };

void SetNext() { curr = curr->next; }

int isended() { return (curr->next == head->next); };

int operator==(const Ringlist<t>& l) const;

int operator!=(const Ringlist<t>& l) const { return !(\*this == l); }

};

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

template<typename t>

Ringlist<t>::Ringlist()

{

head = new Node<t>(NULL);

head->next = head;

curr = head;

}

template<typename t>

Ringlist<t>::Ringlist(const Ringlist<t> &l)

{

head = new Node<t>(l.head->data);

curr = head;

Node<t>\* temp = l.head;

while (temp->next != l.head)

{

temp = temp->next;

curr->next = new Node<t>(temp->data);

curr = curr->next;

}

curr->next = head;

curr = head;

}

template<typename t>

Ringlist<t>::~Ringlist()

{

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

Node<t> \*temp;

while (current != head)

{

temp = current->next;

delete current;

current = temp;

}

head->next = head;

delete head;

}

template<typename t>

Ringlist<t>& Ringlist<t>::operator= (const Ringlist<t> &l)

{

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

Node<t> \*temp1;

while (current != head)

{

temp1 = current->next;

delete current;

current = temp1;

}

head->next = head;

Node<t> \*temp = l.head;

head = new Node<t>(temp->data);

curr = head;

while (temp->next != l.head)

{

temp = temp->next;

curr->next = new Node<t>(temp->data);

curr = curr->next;

}

curr->next = head;

curr = head;

return \*this;

}

template<typename t>

void Ringlist<t>::InsertToOrdered(const t &a)

{

curr = head;

while ((curr->next->data > a) && (curr->next != head))

curr = curr->next;

Node<t>\* temp = curr->next;

curr->next = new Node<t>(a, temp);

}

template<typename t>

int Ringlist<t>::operator==(const Ringlist<t>& l) const

{

int k = 1;

if (this != &l)

{

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

Node<t> \*temp1 = l.head->next;

while ((temp1 != l.head) && (temp != head) && temp->data == temp1->data)

{

temp = temp->next;

temp1 = temp1->next;

while ((temp->data == NULL) && (temp != head))

temp = temp->next;

while ((temp1->data == NULL) && (temp1 != l.head))

temp1 = temp1->next;

}

if (temp1 != l.head || temp != head)

k = 0;

}

return k;

}

## Приложение 2. Программная реализация многочленов

#pragma once

#include "list.h"

#include <iostream>

#include <string>

using namespace std;

class Monom

{

public:

double coeff;

int abc;

Monom(double coeff1 = NULL, int abc1 = NULL) { coeff = coeff1; abc = abc1; }

bool operator< (const Monom& m) const { return (abc<m.abc); }

bool operator> (const Monom& m) const { return (abc>m.abc); }

bool operator==(const Monom& m) const

{

int k = 1;

if (coeff)

k = (abc == m.abc && coeff == m.coeff);

return k;

}

bool operator!=(const Monom& m) const { return !(\*this == m); }

Monom& operator=(const Monom& m) { coeff = m.coeff; abc = m.abc; return \*this; }

};

class Polinom

{

Ringlist<Monom> pol;

Ringlist<Monom> Parse(const string& s);

public:

Polinom();

Polinom(const string& str);

Polinom(const Polinom& p);

Polinom(const Ringlist<Monom>& p) { pol = p; }

Ringlist<Monom> AddSameAndInsert(const Monom& c, const Ringlist<Monom>& p);

Polinom& operator=(const Polinom& p);

Polinom operator+(const Polinom& p);

Polinom operator-(const Polinom& p)

{

Polinom temp(\*this);

temp = temp + (-1) \* p;

return temp;

}

Polinom operator\*(const int& c);

Polinom operator\*(const Polinom& p);

friend Polinom operator\*(const int c, const Polinom& p)

{

Polinom temp(p);

temp = temp \* c;

return temp;

}

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

int operator== (const Polinom& p) const { return pol == p.pol; }

int operator!= (const Polinom& p) const { return pol != p.pol; }

};

#include "polinom.h"

Ringlist<Monom> Polinom::AddSameAndInsert(const Monom& c,const Ringlist<Monom>& p)

{

Ringlist<Monom> temp = p;

temp.reset();

int k = 0;

while (!temp.isended())

{

if (c.abc == temp.GetCurr().abc)

{

temp.GetCurr().coeff += c.coeff;

k++;

}

temp.SetNext();

}

temp.reset();

if (!k)

temp.InsertToOrdered(c);

return temp;

}

Ringlist<Monom> Polinom::Parse(const string& s)

{

Ringlist<Monom> r;

string temp = s;

int l = temp.length();

for (int i = 0; i < temp.length(); i++)

if ((temp[i] == ' ') || (temp[i] == '^'))

{

temp.erase(i, 1);

i--;

}

for (int i = 0; i < temp.length();)

{

string a;

int abc = 0;

int j = i;

if ((temp[i] == '-') || (temp[i] == '+'))

{

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

{

i++;

j++;

while (isdigit(temp[j]) || temp[j] == '.')

j++;

if (j - i)

{

a = "-" + temp.substr(i, j - i);

i = j;

}

else

a = "-1";

}

else

{

i++;

j++;

while (isdigit(temp[j]) || temp[j] == '.')

j++;

if (j - i)

{

a = temp.substr(i, j - i);

i = j;

}

else

a = "1";

}

}

else

{

while (isdigit(temp[j]) || temp[j] == '.')

j++;

if (j - i)

{

a = temp.substr(i, j - i);

i = j;

}

else

a = "1";

}

while ((temp[i] != '+') && (temp[i] != '-') && (i < temp.length()))

{

switch (temp[i])

{

case 'a':

{

if (isdigit(temp[i + 1]))

{

i++;

int st = temp[i] - '0';

abc += 100 \* st;

}

else

abc += 100;

break;

}

case 'b':

{

if (isdigit(temp[i + 1]))

{

i++;

int st = temp[i] - '0';

abc += 10 \* st;

}

else

abc += 10;

break;

}

case 'c':

{

if (isdigit(temp[i + 1]))

{

i++;

int st = temp[i] - '0';

abc += st;

}

else

abc += 1;

break;

}

}

i++;

}

Monom c(stod(a), abc);

r = AddSameAndInsert(c, r);

}

return r;

}

Polinom::Polinom()

{

Ringlist<Monom> A;

pol = A;

}

Polinom::Polinom(const string& s)

{

pol = Parse(s);

}

Polinom::Polinom(const Polinom& p)

{

Ringlist<Monom> A(p.pol);

pol = A;

}

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

{

Ringlist<Monom> A(p.pol);

pol = A;

return \*this;

}

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

{

Polinom temp(\*this);

Ringlist<Monom> p1 = p.pol;

p1.reset();

temp.pol.reset();

while (!p1.isended())

{

temp.pol = AddSameAndInsert(p1.GetCurr(), temp.pol);

p1.SetNext();

}

return temp;

}

Polinom Polinom::operator\*(const int& a)

{

Polinom temp(\*this);

temp.pol.reset();

while (!temp.pol.isended())

{

temp.pol.GetCurr().coeff \*= a;

temp.pol.SetNext();

}

temp.pol.reset();

return temp;

}

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

{

Polinom temp;

Ringlist<Monom> p1 = p.pol;

p1.reset();

while (!p1.isended())

{

Polinom temp1(\*this);

temp1.pol.reset();

int abc = p1.GetCurr().abc;

if (abc)

{

while (!temp1.pol.isended())

{

int tmp = temp1.pol.GetCurr().abc;

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

{

temp1.pol.GetCurr().coeff \*= p1.GetCurr().coeff;

temp1.pol.GetCurr().abc += abc;

}

else

throw "abc > 9";

temp1.pol.SetNext();

}

}

else

temp1 = temp1 \* p1.GetCurr().coeff;

temp1.pol.reset();

temp = temp + temp1;

p1.SetNext();

}

return temp;

}

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

{

Polinom tempP = p;

tempP.pol.reset();

Monom tempM;

int k = 0;

while (!tempP.pol.GetCurr().coeff && !tempP.pol.isended())

tempP.pol.SetNext();

if (!tempP.pol.isended())

{

Monom FNode = tempP.pol.GetCurr();

while (!tempP.pol.isended())

{

tempM = tempP.pol.GetCurr();

if (tempM.coeff > 0 && tempM != FNode)

os << " + ";

if (tempM.coeff != 1 && tempM.coeff != -1 && tempM.coeff != 0)

os << ' ' << tempM.coeff;

else

if (tempM.coeff == -1 && tempM.abc != 0)

os << " - ";

else

if ((tempM.coeff == 1 || tempM.coeff == -1) && tempM.abc == 0)

os << ' ' << tempM.coeff;

if (tempM.coeff)

{

if (tempM.abc / 100)

{

if (tempM.abc / 100 == 1)

os << "a";

else

os << "a^" << tempM.abc / 100;

}

if (((tempM.abc % 100) - (tempM.abc % 10)) / 10)

{

if (((tempM.abc % 100) - (tempM.abc % 10)) / 10 == 1)

os << 'b';

else

os << "b^" << ((tempM.abc % 100) - (tempM.abc % 10)) / 10;

}

if (tempM.abc % 10)

{

if (tempM.abc % 10 == 1)

os << 'c';

else

os << "c^" << tempM.abc % 10;

}

}

tempP.pol.SetNext();

}

}

else

os << " 0";

tempP.pol.reset();

return os;

}

## Приложение 3. Основная программа

#include "polinom.h"

#include <conio.h>

void main()

{

cout << "Input correct polinoms" << endl;

char c = 0;

string pol1;

string pol2;

while (!c || (c == ' '))

{

cout << "-------------------------" << endl;

cout << endl;

cout << "Enter first polinom:" << endl;

getline(cin, pol1);

if (pol1 == "")

getline(cin, pol1);

Polinom p1(pol1);

cout << "p1 = " << p1 << endl;

cout << "Enter second polinom:" << endl;

getline(cin, pol2);

if (pol2 == "")

getline(cin, pol2);

Polinom p2(pol2);

cout << "p2 = " << p2 << endl;

cout << "-------------------------" << endl;

Polinom res = p1 + p2;

cout << "p1 + p2 = " << res << endl;

cout << "-------------------------" << endl;

res = p1 - p2;

cout << "p1 - p2 = " << res << endl;

cout << "-------------------------" << endl;

res = p1 \* p2;

cout << "p1 \* p2 = " << res << endl;

cout << "-------------------------" << endl;

cout << "Press SPACE to retry" << endl;

cout << "Press any key to exit" << endl;

c = \_getch();

}

}

## Приложение 4. Тесты для класса Ringlist

#include "list.h"

#include "node.h"

#include <gtest.h>

TEST(Node, can\_create\_node)

{

ASSERT\_NO\_THROW(Node<double> l(0));

}

TEST(Ringlist, can\_create\_ringlist)

{

ASSERT\_NO\_THROW(Ringlist<double> l);

}

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

class EmptyList : public testing::Test

{

protected:

Ringlist<double> l;

public:

EmptyList() {};

~EmptyList() {};

};

TEST\_F(EmptyList, can\_GetCurr\_from\_empty\_list)

{

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

int a = l.GetCurr();

l.reset();

EXPECT\_EQ(a, l.GetCurr());

}

TEST\_F(EmptyList, can\_SetNext\_from\_empty\_list)

{

ASSERT\_NO\_THROW(l.SetNext());

}

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

{

ASSERT\_NO\_THROW(l.reset());

}

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

{

EXPECT\_EQ(NULL, l.GetCurr());

}

TEST\_F(EmptyList, can\_isended\_from\_empty\_list)

{

ASSERT\_NO\_THROW(l.isended());

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

}

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

{

ASSERT\_NO\_THROW(Ringlist<double> l1(l));

}

TEST\_F(EmptyList, empty\_list\_copy\_is\_correct)

{

Ringlist<double> l1(l);

EXPECT\_EQ(l.GetCurr(), l1.GetCurr());

EXPECT\_EQ(NULL, l1.GetCurr());

}

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

{

Ringlist<double> l1;

ASSERT\_NO\_THROW(l = l1);

l = l1;

EXPECT\_EQ(l.GetCurr(), l1.GetCurr());

}

TEST\_F(EmptyList, can\_insert\_to\_ordered\_for\_empty\_list)

{

l.InsertToOrdered(1);

l.InsertToOrdered(2);

l.reset();

EXPECT\_EQ(2, l.GetCurr());

l.SetNext();

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

}

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

class ThreeNodesList : public testing::Test

{

protected:

Ringlist<double> l;

public:

ThreeNodesList()

{

l.InsertToOrdered(1);

l.InsertToOrdered(2);

l.InsertToOrdered(3);

}

~ThreeNodesList() {};

};

TEST\_F(ThreeNodesList, can\_copy\_not\_empty\_list)

{

ASSERT\_NO\_THROW(Ringlist<double> l1(l));

}

TEST\_F(ThreeNodesList, copy\_of\_not\_empty\_list\_is\_correct)

{

Ringlist<double> l1(l);

l.reset();

l1.reset();

EXPECT\_EQ(3, l1.GetCurr());

l1.SetNext();

l.SetNext();

EXPECT\_EQ(2, l1.GetCurr());

l1.SetNext();

l.SetNext();

EXPECT\_EQ(1, l1.GetCurr());

l1.SetNext();

l.SetNext();

}

TEST\_F(ThreeNodesList, can\_assign\_not\_empty\_list)

{

ASSERT\_NO\_THROW(Ringlist<double> l1 = l);

}

TEST\_F(ThreeNodesList, assign\_of\_not\_empty\_list\_is\_correct)

{

Ringlist<double> l1 = l;

l.reset();

l1.reset();

EXPECT\_EQ(3, l1.GetCurr());

l1.SetNext();

l.SetNext();

EXPECT\_EQ(2, l1.GetCurr());

l1.SetNext();

l.SetNext();

EXPECT\_EQ(1, l1.GetCurr());

l1.SetNext();

l.SetNext();

}

## Приложение 5. Тесты для класса Polinom

#include <gtest.h>

#include "polinom.h"

#include <vector>

using std::vector;

TEST(Polinom, can\_create\_polinom)

{

ASSERT\_NO\_THROW(Polinom p);

}

TEST(Polinom, can\_create\_polinom\_str)

{

string s = "abc";

Ringlist<Monom> b;

b.InsertToOrdered(Monom(1, 111));

Polinom c(b);

Polinom a(s);

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

EXPECT\_EQ(a, c);

}

TEST(Polinom, can\_copy\_polinoms)

{

Polinom b("abc");

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

Polinom a(b);

EXPECT\_EQ(a, b);

}

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

class PolTestParse

{

public:

string s;

Polinom res;

PolTestParse(string str, vector<Monom> &M)

{

s = str;

Ringlist<Monom> temp;

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

temp.InsertToOrdered(M[i]);

res = Polinom(temp);

}

};

class TestParsing : public ::testing::TestWithParam< PolTestParse>

{

protected:

Polinom P1;

public:

TestParsing() : P1(GetParam().s) {}

~TestParsing() {}

};

TEST\_P(TestParsing, test\_polinom\_parsing)

{

EXPECT\_EQ(GetParam().res, P1);

}

INSTANTIATE\_TEST\_CASE\_P(t1, TestParsing, ::testing::Values(

PolTestParse("0", vector<Monom> {Monom()}),

PolTestParse("abc", vector<Monom> {Monom(1, 111)}),

PolTestParse("cba", vector<Monom> {Monom(1, 111)}),

PolTestParse("bac", vector<Monom> {Monom(1, 111)}),

PolTestParse("a", vector<Monom> {Monom(1, 100)}),

PolTestParse("b", vector<Monom> {Monom(1, 10)}),

PolTestParse("c", vector<Monom> {Monom(1, 1)}),

PolTestParse("a^2", vector<Monom> {Monom(1, 200)}),

PolTestParse("b^2", vector<Monom> {Monom(1, 20)}),

PolTestParse("c^2", vector<Monom> {Monom(1, 2)}),

PolTestParse("a^2bc", vector<Monom> {Monom(1, 211)}),

PolTestParse("a+b", vector<Monom> {Monom(1, 100), Monom(1, 10)}),

PolTestParse("a-b", vector<Monom> {Monom(1, 100), Monom(-1, 10)}),

PolTestParse("a^2-b^2", vector<Monom> {Monom(1, 200), Monom(-1, 20)}),

PolTestParse("abc+abc", vector<Monom> {Monom(2, 111)}),

PolTestParse("5.1ab", vector<Monom> {Monom(5.1, 110)}),

PolTestParse("a^2b^2c^2", vector<Monom> {Monom(1, 222)}),

PolTestParse("a+1-a-1", vector<Monom> {Monom(0, 0)})

));

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

class TGeneralPol

{

public:

string P1, P2, res;

TGeneralPol(string RES, string POL1, string POL2)

{

P1 = POL1;

P2 = POL2;

res = RES;

}

};

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

{

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(t2, TestSum, ::testing::Values(

TGeneralPol("0", "1", "-1"),

TGeneralPol("0", "a", "-a"),

TGeneralPol("0", "-abc", "abc"),

TGeneralPol("2abc", "bca", "abc"),

TGeneralPol("a+b", "a", "b"),

TGeneralPol("3.14a", "3a", "0.14a"),

TGeneralPol("18a^2b^2", "10a^2b^2+c^2", "8a^2b^2-c^2")

));

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

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

{

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(t3, TestMult, ::testing::Values(

TGeneralPol("2", "2", "1"),

TGeneralPol("abc", "abc", "1"),

TGeneralPol("abc^2", "ac", "cb"),

TGeneralPol("15abc+15ab-15c^2", "15", "abc+ab-c^2"),

TGeneralPol("15abc+15ab-15c^2", "abc+ab-c^2", "15"),

TGeneralPol("100a^2", "50a", "2a"),

TGeneralPol("111abc", "55.5ab", "2c"),

TGeneralPol("a+b+c", "1", "a+b+c"),

TGeneralPol("a^2-b^2", "a-b", "a+b"),

TGeneralPol("a^3+b^3", "a+b", "a^2-ab+b^2"),

TGeneralPol("a^3-b^3", "a-b", "a^2+ab+b^2")

));

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

class TestSubstr : public ::testing::TestWithParam<TGeneralPol>

{

public:

Polinom pol1, pol2, Res;

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

~TestSubstr() {};

};

TEST\_P(TestSubstr, substr)

{

EXPECT\_EQ(Res, pol1 - pol2);

}

INSTANTIATE\_TEST\_CASE\_P(t4, TestSubstr, ::testing::Values(

TGeneralPol("2", "1", "-1"),

TGeneralPol("2a", "a", "-a"),

TGeneralPol("-2abc", "-abc", "abc"),

TGeneralPol("0", "bca", "abc"),

TGeneralPol("a-b", "a", "b"),

TGeneralPol("2a", "3a", "a"),

TGeneralPol("2a^2b^2+2c^2", "10a^2b^2+c^2", "8a^2b^2-c^2")

));