МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по учебной практике

**«Арифметические операции с полиномами»**

**Выполнил:** студент группы 381603-1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Вихрев И.Б.

**Проверила:** к.т.н., старший преподаватель каф. МОСТ института ИТММ  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кустикова В.Д.

Нижний Новгород

2018

Содержание

[Введение 3](#_Toc515573220)

[1 Постановка учебно-практической задачи 4](#_Toc515573221)

[2 Руководство пользователя 5](#_Toc515573222)

[3 Руководство программиста 6](#_Toc515573223)

[3.1 Описание структуры программы 6](#_Toc515573224)

[3.2 Описание структур данных и алгоритмов. 7](#_Toc515573225)

[3.2.1 Шаблон звена списка. 7](#_Toc515573226)

[3.2.2 Шаблон класса циклический список с “головой”. 7](#_Toc515573227)

[3.2.3 Класс моном 7](#_Toc515573228)

[3.2.4 Класс полином. 7](#_Toc515573229)

[3.3 Описание программной реализации 10](#_Toc515573230)

[Заключение 12](#_Toc515573231)

[Список литературы 13](#_Toc515573232)

[4 Приложение 14](#_Toc515573233)

[4.1 Приложение 1. Заголовочный файл node.h 14](#_Toc515573234)

[4.2 Приложение 2. Заголовочный файл list.h 14](#_Toc515573235)

[4.3 Приложение 3. Заголовочный файл monom.h 16](#_Toc515573236)

[4.4 Приложение 4. Заголовочный файл polinom.h 17](#_Toc515573237)

[4.5 Приложение 5. Файл polinom.cpp 17](#_Toc515573238)

[4.6 Приложение 6. Файл polinom.cpp 22](#_Toc515573239)

[4.7 Приложение 7. Файл test\_Rlist.cpp 23](#_Toc515573240)

[4.8 Приложение 7. Файл test\_polinom.cpp 25](#_Toc515573241)

# Введение

Цель данной лабораторной работы — разработать и реализовать программу, выполняющую арифметические операции над полиномами трёх переменных: сложение, вычитание, умножение на константу и умножение двух полиномов. В реализации применяется циклический односвязный список с головой. Для написания программы используется язык C++.

Реализация подобных математических операций упрощает работу со сложными вычислениями, повышает их точность. Один из методов обработки полиномов на ЭВМ представлен в данной программе. Её элементы могут быть использованы при разработке других программных комплексов и приложений.

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

# Постановка учебно-практической задачи

*Формулировка задачи*:

Разработать программу, выполняющую арифметические операции над полиномами от трех переменных. Полином должен быть составлен из мономов трёх переменных(xyz) со степенью от 0 до 9. Коэффициенты перед мономами – вещественные числа. Допустимые операции: -,+,\*.

Работоспособность программы проверить с помощью Google Tests Framework.

*Исходные данные*:

2 строки, содержащие полином A и полином B соответственно.

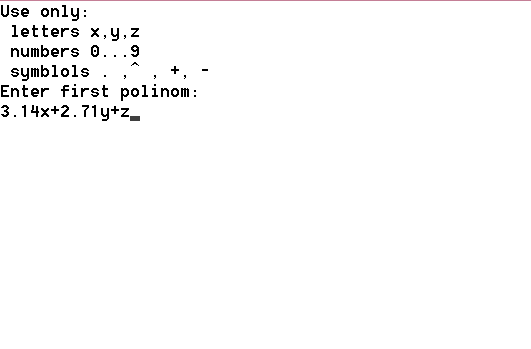
*Требуемый результат*:

Вывести на экран результат применения арифметических операций к полиномам.

# Руководство пользователя

Для выполнения лабораторной работы необходимо запустить файл **sample\_polinom.exe**. На экране вам предложат ввести исходные данные (Рис. 1)*.* Введённый полином не должен содержать пробелов. К вводу допускаются выражения, содержащие только: переменные **x, y, z*,*** цифры **0…9** и символы **. ,^ ,+, -** . Вещественные числа вводятся через точку.

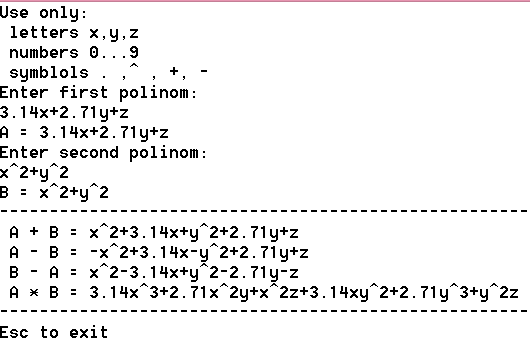
Выражение должно быть осмысленным.



1. Начало работы программы

После ввода каждого из полиномов необходимо нажать клавишу Enter.

На экране появятся результаты применения арифметических операций к двум полиномам: их сумма, разность и произведение(Рис. 2).



1. Результат работы программы

В конце работы программы вам будет предложено либо продолжить вычисления, либо завершить ее. Для продолжения необходимо нажать любую клавишу***,*** для завершения ***Esc***.

# Руководство программиста

## Описание структуры программы

*Структура решения*:

1. Статическая библиотека **gtest**.
2. Статическая библиотека **polinom\_lib**.
3. Проект **polinom\_test**.
4. Проект **sample\_polinom**.

Библиотека **gtest** необходима для работы Google Tests Framework:

Библиотека **polinom\_lib** состоит из нескольких программных модулей:

1. Заголовочный файл **node.h** содержит объявление шаблонного класса **Node** и его реализацию.
2. Заголовочный файл **list.h**содержит объявление шаблонного класса **Rlist** и его реализацию.
3. Файл **monom.h** содержит объявление класса **Monom** и его реализацию.
4. Файл **polinom.cpp** содержит объявление и реализацию класса **Polinom**.

Проект **polinom\_test** содержит тесты функций библиотеки **polinom\_lib**

Проект **sample\_polinom** содержит реализацию пользовательского интерфейса.

## Описание структур данных и алгоритмов.

### Шаблон звена списка.

Содержит 2 поля: данные и указатель на следующее звено списка.

Над звеньями списка определены операции сравнения.

### Шаблон класса циклический список с “головой”.

Содержит поле-указатель на голову списка и поле-указатель на текущий элемент списка.

Используется *алгоритм* упорядоченной вставки в список:

1. Устанавливаем указатель на текущий элемент в начало списка.
2. Перемещаем указатель на следующий элемент списка до тех пор, пока данные звена списка больше данных звена, которое мы вставляем илиъ не конец списка.
3. Вставляем нужное нам звено в место после указателя на текущий.

### Класс моном

Содержит 2 поля:

1. Коэффициент монома – вещественное число.
2. Степень монома. Хранится в свернутом виде — число от 0 до 999. Степень **x** — число сотен , **y**—десятков, **z** – единиц.

Для класса переопределены оператор присваивания и операторы сравнения.

### Класс полином.

В качестве структуры хранения полинома используется зацикленный односвязный список мономов с ненулевыми коэффициентами. Список реализован с фиктивной головой. Элементы списка хранятся упорядоченно.

Поскольку ввод полинома осуществляется с клавиатуры, необходимо преобразовать строковое представление полинома в пригодное для работы с ними. Для такого преобразования используется алгоритм разбиения строки на мономы:

1. На вход подается строка
2. Строка проходится слева-направо до тех пор пока не встретится символ '+', '-' или не случится конец строки.
3. Как только мы встретили искомый символ, мы выделяем подстроку до этого символа и начинаем обрабатывать ее.
4. В подстроке происходит поиск коэффициента и степеней x, y, z. Найденные значения записываются в переменную типа Monom
5. Полученный моном записывается в список.
6. Формируется упорядоченный список из мономов.

Следующий этап в обработке полинома — приведение подобных слагаемых. Это необходимо для того, чтобы все мономы в полиноме были уникальны.

1. После парсинга у нас сформировался упорядоченный список мономов – полином.
2. Если список состоит из одного звена – упрощение не требуется.
3. Если состоит из более чем одного звена – идем по этому списку, сравнивая два соседних монома (соседних - т.к. список упорядочен мономы с одинаковой степенью находятся по соседству).
4. Если находятся равные по степеням мономы, мы складываем их коэффициенты и записываем результат в новый список. Уникальные мономы записываем в список без изменений.
5. Полученный список – полином после упрощения.

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

1. На вход подаются два упорядоченных списка.
2. Устанавливаем указатели на текущий элемент в обоих списках на начало
3. Сравниваем элементы списков по порядку с самого начало. :
4. Больший элемент записываем в результат. Двигаем указатель списка, из которого взяли элемент, на одно звено вперед.
5. В случае равенства степеней мономов, складываем коэффициенты и записываем полученный моном в результат. Двигаем указатели обоих списков на одно звено вперед.
6. 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

Умножение полиномов производится по следующему алгоритму:

1. Устанавливаем указатели обоих списков на начало.
2. Умножим моном первого списка на все мономы второго. Запишем полученное в результат.
3. Перемещаем указатель первого списка на одно звено вперед.
4. Продолжать до тех пор, пока первый список не кончится.

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

## Описание программной реализации

1. Шаблон класса **Node** – звено списка.

Поля:

T data; // данные, хранящиеся в звене

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

Методы:

//конструктор с параметрами и конструктор копирования

Node(T DATA = NULL, Node<T>\* NEXT = nullptr);

Node(Node<T> &Node2)

//операторы сравнения:

bool operator< (const Node<T>& N) const;

bool operator> (const Node<T>& N) const;

bool operator!= (const Node<T>& N) const

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

1. Шаблон циклического списка с головой **Rlist**.

Поля:

Node<T>\* head; // голова – фиктивное звено списка

Node<T>\* current; // указатель на текущий элемент

Методы:

//Конструкторы и деструктор

Rlist();

Rlist(const Rlist<T>& ListToCopy);

~Rlist();

Rlist<T>& operator=(const Rlist<T>& ListToCopy);

void InsertAfter(Node<T>\* N, T Data);

void OrderedInsert(T Data);// Вставка в упорядоченный список

void Clean(); // очистка списка

// методы навигации

Node<T>\* GetCurr() const // получить указатель на текущий

void SetNext()// переместить указатель на следующий

void Reset()// переместить указатель в начало

bool IsEnd() const // проверка на конец

//Операторы сравнения

bool operator==(const Rlist<T>& RLst) const;

bool operator!=(const Rlist<T>& RLst) const;

1. Класс **Monom.**

Поля:

double coeff; //коэффициент

unsigned int abc; // степень монома

Методы:

//Конструктор

Monom(double COEFF = 0 , unsigned int ABC = 0) { coeff = COEFF; abc = ABC;}

//Операторы равнение

bool operator< (const Monom& m) const

bool operator> (const Monom& m) const

bool operator==(const Monom& m) const

bool operator!=(const Monom& m) const

Monom& operator=(const Monom& m) //Оператор присваивания

1. Класс **Polinom.**

Поля:

Rlist<Monom> Monoms; // циклический список состоящий из мономов

string name; // имя полинома

//вспомогательные методы

Rlist <Monom> Simplify(Rlist <Monom> POL); // привести подобные слагаемые

Rlist<Monom> Parsing(const string Line);// разбитие строки на мономы

Методы

//Конструкторы

Polinom(const string Line = "" ); // конструктор по строке

Polinom(const Monom m)// конструктор преобразования типа: от монома

Polinom(const Rlist<Monom> &P2) // конструктор преобразования типа: от списка

Polinom(const Polinom& POL)//конструктор копирования

//Операторы

Polinom operator+ (const Polinom&) const;

Polinom operator- (const Polinom& POL) const

Polinom operator\* (const Polinom& POL) const;// умножение полинома на полином

Polinom operator\* (const double c) const; // умножение полинома на константу справа

bool operator== (const Polinom& POL) const

bool operator!= (const Polinom& POL) const

friend Polinom operator\* (const double C , const Polinom& POL)//умножение полинома на константу слева

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

};

# Заключение

В лабораторной работе была реализована программа, выполняющая арифметические операции с полиномами. Был разработан класс полиномов, на основе циклического односвязного списка с головой. Также учтен вырожденный случай – когда на вход вместо полиномов подаются вещественные числа. Таким образом программа может выполнять роль калькулятора для двух вещественных чисел и применять к ним такие операции, как +, -, \*.

# 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 1. Заголовочный файл node.h

#pragma once

#include <iostream>

template <typename T>

class Node

{

public:

T data;

Node<T>\* next;

Node(T DATA = NULL, Node<T>\* NEXT = nullptr) { data = DATA; next = NEXT; }

Node(Node<T> &Node2) { data = node2.data; next = nullptr; }

//операторы сравнения

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); }

};

## Приложение 2. Заголовочный файл list.h

#pragma once

#include "Node.h"

// Циклический список с головой //

template<typename T>

class Rlist

{

protected:

Node<T>\* head;

Node<T>\* current;

public:

//Конструкторы и деструктор

Rlist();

Rlist(const Rlist<T>& ListToCopy);

~Rlist();

//Методы

Rlist<T>& operator=(const Rlist<T>& ListToCopy);

void InsertAfter(Node<T>\* N, T Data);

void OrderedInsert(T Data); // Вставка в упорядоченный список

void Clean(); // очистка списка

// методы навигации

Node<T>\* GetCurr() const { return current; } // получить указатель на текущий

void SetNext() { current = current->next; }// переместить указатель на следующий

void Reset() { current = head->next; } // переместить указатель в начало

bool IsEnd() const { return current == head; } // проверка на конец

//Операторы сравнения

bool operator==(const Rlist<T>& RLst) const;

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

};

// ............................................................................

template <typename T>

Rlist<T>::Rlist()

{

head = new Node<T>;

head->data = NULL;

head->next = head;

current = head;

}

// ............................................................................

template <typename T>

Rlist<T>::Rlist(const Rlist<T>& ListToCopy)

{

Node<T>\* TempCurr = ListToCopy.head;

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

head->next = head;

current = head;

while (TempCurr->next != ListToCopy.head)

{

TempCurr= TempCurr->next;

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

SetNext();

}

current->next = head;

}

// ............................................................................

template <class T>

Rlist<T>::~Rlist()

{

Clean();

delete head;

}

template <class T>

Rlist<T>& Rlist<T>::operator=(const Rlist<T>& ListToCopy)

{

Clean();

Node<T>\* TempCurr1 = ListToCopy.head;

Node<T>\* TempCurr2 = head;

while (TempCurr1->next != ListToCopy.head)

{

TempCurr1 = TempCurr1->next;

TempCurr2->next = new Node<T>(TempCurr1->data);

TempCurr2 = TempCurr2->next;

}

TempCurr2->next = head;

current = head;

return \*this;

}

// ............................................................................

template <class T>

void Rlist<T>::InsertAfter(Node<T>\* N, T Data)

{

Node<T>\* temp = N->next;

N->next = new Node<T>(Data);

N->next->next = temp;

}

// ............................................................................

template <class T>

void Rlist<T>::OrderedInsert(T Data)

{

Node<T>\* Temp;

current = head;

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

SetNext();

Temp = current->next;

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

current->next->next = Temp;

}

// ............................................................................

template <class T>

void Rlist<T>::Clean()

{

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

Node<T>\* Temp;

while (TempCurr != head)

{

Temp = TempCurr->next;

delete TempCurr;

TempCurr = Temp;

}

head->next = head;

}

// ............................................................................

template<class T>

bool Rlist<T>::operator==(const Rlist<T>& RLst) const

{

bool flag = true;

if (this != &RLst)

{

Node<T>\* temp1 = head->next;

Node<T>\* temp2 = RLst.head->next;

while (temp1 != head && temp2 != RLst.head && temp1->data == temp2->data)

{

temp1 = temp1->next;

temp2 = temp2->next;

}

if (temp1 != head || temp2 != RLst.head)

flag = false;

}

return flag;

}

// ............................................................................

## Приложение 3. Заголовочный файл monom.h

#pragma once

// Моном

class Monom

{

public:

double coeff; //коэффициент

unsigned int abc; // степень

//Конструктор

Monom(double COEFF = 0 , unsigned int ABC = 0) { coeff = COEFF; abc = ABC;}

//Сравнение

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 { return (abc == m.abc && coeff == m.coeff); } //

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

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

};

## Приложение 4. Заголовочный файл polinom.h

#pragma once

#include <iostream>

#include <string>

#include <algorithm>

#include "Monom.h"

#include "list.h"

using std::string;

using std::ostream;

using std::cout;

using std::cin;

using std::endl;

class Polinom {

protected:

Rlist<Monom> Monoms;

string name;

//вспомогательные методы

Rlist <Monom> Simplify(Rlist <Monom> POL); // привести подобные слагаемые

Rlist<Monom> Parsing(const string Line);// разбитие строки на мономы

public:

//Конструкторы

Polinom(const string Line = "" ); // конструктор по строке

Polinom(const Monom m) { Monoms.InsertAfter(Monoms.GetCurr(),m); }// конструктор преобразования типа: от монома

Polinom(const Rlist<Monom> &P2) : Monoms(P2) {}; // конструктор преобразования типа: от списка

Polinom(const Polinom& POL) : Monoms(POL.Monoms) {}; //конструктор копирования

//Операторы

Polinom operator+ (const Polinom&) const;

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

Polinom operator\* (const Polinom& POL) const;// умножение полинома на полином

Polinom operator\* (const double c) const; // умножение полинома на константу справа

bool operator== (const Polinom& POL) const { return Monoms == POL.Monoms; }

bool operator!= (const Polinom& POL) const { return Monoms != POL.Monoms; }

friend Polinom operator\* (const double C , const Polinom& POL) { return POL\*C; } // умножение полинома на константу слева

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

};

## Приложение 5. Файл polinom.cpp

#include "polinom.h"

//............................................................................

Polinom::Polinom(const string Line)

{

Monoms = Parsing(Line);

Monoms = Simplify(Monoms);

}

//............................................................................

Rlist<Monom> Polinom::Simplify(Rlist<Monom> POL)

{

Rlist<Monom> res;

POL.Reset(); // устанавливаем указатели current в начало списка

res.Reset();

Monom tmp;

// проверяем, что полином не состоит из одного звена - иначе нет смысла упрощать

Node<Monom>\* p = POL.GetCurr();

POL.SetNext();

if (!(POL.IsEnd()))

{

POL.Reset();

while (!(POL.IsEnd()))

{

tmp.coeff = POL.GetCurr()->data.coeff;

tmp.abc = POL.GetCurr()->data.abc;

while ((POL.GetCurr()->data.abc == POL.GetCurr()->next->data.abc) && !(POL.IsEnd()))

{

tmp.coeff += POL.GetCurr()->next->data.coeff;

POL.SetNext();

}

if (tmp.coeff)

{

res.OrderedInsert(tmp);

res.SetNext();

}

if (!(POL.IsEnd()))

POL.SetNext();

}

res.Reset();

//если при упрощении получился 0

if (res.GetCurr()->data.abc == 0 && res.GetCurr()->data.coeff == 0)

res.InsertAfter(res.GetCurr(), NULL);

}

else

res = POL;

return res;

}

//............................................................................

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

{

Polinom res;

Rlist<Monom> P1 = Monoms;

Rlist<Monom> P2 = POL.Monoms;

P1.Reset();

P2.Reset();

res.Monoms.Reset();

// складываем мономы

while (!P1.IsEnd() && !P2.IsEnd())

{

if ((P1.GetCurr()->data > P2.GetCurr()->data) && P1.GetCurr()->data.coeff) // сравниваем мономы

{

res.Monoms.InsertAfter(res.Monoms.GetCurr(), P1.GetCurr()->data);

P1.SetNext();

res.Monoms.SetNext();

}

else

if ((P1.GetCurr()->data < P2.GetCurr()->data) && P2.GetCurr()->data.coeff)

{

res.Monoms.InsertAfter(res.Monoms.GetCurr(), P2.GetCurr()->data);

P2.SetNext();

res.Monoms.SetNext();

}

else // случай равенства степеней мономов

{

double cf = P1.GetCurr()->data.coeff + P2.GetCurr()->data.coeff;

if (cf) // если коэфф не ноль

{

Monom temp(cf, P1.GetCurr()->data.abc);

res.Monoms.InsertAfter(res.Monoms.GetCurr(), temp);

res.Monoms.SetNext();

}

P1.SetNext();

P2.SetNext();

}

}

//добавляем оставшиеся

while (!P1.IsEnd())

{

if (P1.GetCurr()->data.coeff)

{

res.Monoms.InsertAfter(res.Monoms.GetCurr(), P1.GetCurr()->data);

res.Monoms.SetNext();

}

P1.SetNext();

}

while (!P2.IsEnd())

{

if (P2.GetCurr()->data.coeff)

{

res.Monoms.InsertAfter(res.Monoms.GetCurr(), P2.GetCurr()->data);

res.Monoms.SetNext();

}

P2.SetNext();

}

res.Monoms.Reset();

//если при сложении получился 0

if (res.Monoms.GetCurr()->data.abc == 0 && res.Monoms.GetCurr()->data.coeff == 0)

res.Monoms.InsertAfter(res.Monoms.GetCurr(), NULL);

return res;

}

//............................................................................

Rlist<Monom> Polinom::Parsing(const string Line)

{

Rlist<Monom> List;

string SMonom, SPol = Line;

int k = 1;

int d[] = { 100,10,1 };

while (SPol.length())

{

Monom tmp;

k = 1;

// выделяем моном

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

k++;

SMonom = SPol.substr(0, k);

SPol.erase(0, k);

// проверяем наличие + или - в начале

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

{

tmp.coeff = -1;

SMonom.erase(0, 1);

}

else

{

tmp.coeff = 1;

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

SMonom.erase(0, 1);

}

//преобразуем коэффициент

if (isdigit(SMonom[0]))

{

k = 0;

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

k++;

tmp.coeff \*= std::stod(SMonom.substr(0, k));

SMonom.erase(0, k);

}

//находим степени

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

{

int pos = SMonom.find("xyz"[i]);

if (pos != -1)

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

{

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

}

else

tmp.abc += d[i];

}

List.OrderedInsert(tmp);

}

return List;

}

//............................................................................

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

{

Polinom res;

if (c)

{

res = \*this;

res.Monoms.Reset();

while (!(res.Monoms.IsEnd()))

{

res.Monoms.GetCurr()->data.coeff \*= c;

res.Monoms.SetNext();

}

}

else

{

res.Monoms.InsertAfter(res.Monoms.GetCurr(), NULL);

}

return res;

}

//............................................................................

Polinom Polinom::operator\*(const Polinom& POL) const {

Polinom res;

Polinom P1 = Monoms, P2 = POL;

int abc;

double coeff;

P1.Monoms.Reset();

P2.Monoms.Reset();

while (!P2.Monoms.IsEnd())

{

Monom tmp;

P1.Monoms.Reset();

while (!P1.Monoms.IsEnd())

{

abc = P1.Monoms.GetCurr()->data.abc;

coeff = P1.Monoms.GetCurr()->data.coeff;

tmp = P2.Monoms.GetCurr()->data;

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

{

tmp.coeff \*= coeff;

tmp.abc += abc;

res = res + tmp;

}

else

{

throw "Degree>10";

}

P1.Monoms.SetNext();

}

P2.Monoms.SetNext();

}

return res;

}

//............................................................................

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

{

Polinom tmpPol = POL;

tmpPol.Monoms.Reset();

Node<Monom>\* FirstNode =tmpPol.Monoms.GetCurr(); // указатель на первое звено спика

Node<Monom> tmpMon;

while (!(tmpPol.Monoms.IsEnd()))

{

tmpMon = tmpPol.Monoms.GetCurr()->data;

if (tmpMon.data.coeff > 0 && tmpPol.Monoms.GetCurr() != FirstNode)

os << '+';

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

os << tmpMon.data.coeff;

else

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

os << '-';

else

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

os << tmpMon.data.coeff;

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

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

{

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

{

os << "xyz"[i];

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

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

}

}

tmpPol.Monoms.SetNext();

}

tmpPol.Monoms.Reset();

if (tmpPol.Monoms.GetCurr()->data.abc == 0 && tmpPol.Monoms.GetCurr()->data.coeff == 0 && !(tmpPol.Monoms.IsEnd()))

{

os << "0";

}

return os;

}

// ............................................................................

## Приложение 6. Файл polinom.cpp

#include "polinom.h"

#include <conio.h>

#include <Windows.h>

int main()

{

char ch = 0;

string exp1;

string exp2;

system("color F0");

cout << "Use only: " << endl << " letters x,y,z" << endl << " numbers 0...9" << endl << " symblols . ,^ , +, -" << endl ;

while (ch != 27)

{

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

cin >> exp1;

Polinom A(exp1);

cout << "A = " <<A << endl;

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

cin >> exp2;

Polinom B(exp2);

cout <<"B = " << B << endl << "-----------------------------------------------------------------------------" << endl;

Polinom res = A + B;

cout << " A + B = ";

cout << res << endl;

res = A - B;

cout << " A - B = ";

cout << res << endl;

res = B - A;

cout << " B - A = ";

cout << res << endl;

res = A \* B;

cout << " A \* B = ";

cout << res << endl;

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

cout << "Esc to exit " << endl;

ch = \_getch();

}

return 0;

}

## Приложение 7. Файл test\_Rlist.cpp

#include <gtest.h>

#include "node.h"

#include "list.h"

//тесты для циклического списка с головой

//............................................................................

TEST(Node, can\_create\_node)

{

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

}

//............................................................................

TEST(Rlist, can\_create\_Rlist)

{

ASSERT\_NO\_THROW(Rlist<double> L);

}

//............................................................................

class EmptyRlist : public testing::Test

{

protected:

Rlist<int> L;

public:

EmptyRlist() {};

~EmptyRlist() {};

};

//............................................................................

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

{

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

}

//............................................................................

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

{

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

}

//............................................................................

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

{

Rlist<int> L2(L);

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

}

//............................................................................

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

{

Rlist<int> L2;

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

}

//............................................................................

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

{

Rlist<int> L2;

ASSERT\_NO\_THROW(L = L2);

}

//............................................................................

TEST\_F(EmptyRlist, can\_assign\_empty\_list\_to\_itself)

{

ASSERT\_NO\_THROW(L = L);

}

//............................................................................

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

{

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

}

//............................................................................

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

{

L.Clean();

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

}

//............................................................................

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

{

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

}

//............................................................................

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

{

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

}

//............................................................................

TEST\_F(EmptyRlist, reset\_empty\_list\_is\_correct)

{

L.Reset();

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

}

//............................................................................

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

{

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

}

//............................................................................

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

{

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

}

//............................................................................

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

{

L.Reset();

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

}

//............................................................................

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

{

L.Reset();

L.InsertAfter(L.GetCurr(), 1);

L.Reset();

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

EXPECT\_EQ(1, tmp);

}

//............................................................................

class FilledRlist : public testing::Test

{

protected:

Rlist<int> L;

public:

FilledRlist()

{

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

L.OrderedInsert(i);

};

~FilledRlist() {};

};

//............................................................................

TEST\_F(FilledRlist, can\_copy\_filled\_list)

{

ASSERT\_NO\_THROW(Rlist<int> A(L));

}

//............................................................................

TEST\_F(FilledRlist, copied\_filled\_list\_is\_correct)

{

Rlist<int> A(L);

A.Reset();

EXPECT\_EQ(2, A.GetCurr()->data);

EXPECT\_EQ(1, A.GetCurr()->next->data);

EXPECT\_EQ(0, A.GetCurr()->next->next->data);

EXPECT\_NE(L.GetCurr(), A.GetCurr());

}

//............................................................................

TEST\_F(FilledRlist, can\_assign\_filled\_list)

{

ASSERT\_NO\_THROW(Rlist<int> A = L);

}

//............................................................................

TEST\_F(FilledRlist, assigned\_filled\_list\_is\_correct)

{

Rlist<int> A = L;

A.Reset();

L.Reset();

EXPECT\_EQ(2, A.GetCurr()->data);

EXPECT\_EQ(1, A.GetCurr()->next->data);

EXPECT\_EQ(0, A.GetCurr()->next->next->data);

EXPECT\_NE(L.GetCurr(), A.GetCurr());

}

//............................................................................

## Приложение 8. Файл test\_polinom.cpp

#include "gtest.h"

#include "polinom.h"

#include <vector>

//тесты для мономов и полиномов

using std::vector;

string Expr[] = {"0", "xyz","zxy","yzx","x","y","z","x^2","y^2","z^2", "x^2yz", "x+y", "x-y", "x^2-y^2","xyz+xyz","x^2y^2z^2"};

Monom M[] = { Monom() ,Monom(1, 111),Monom(1, 111),Monom(1, 111), Monom(1, 100),Monom(1, 10),Monom(1, 1), Monom(1, 200),Monom(1, 20),Monom(1, 2),Monom(1, 211) };

//............................................................................

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 a("x^5");

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

}

//............................................................................

class PolTestParse

{

public:

string Line;

Polinom res;

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

{

Line = str;

Rlist<Monom> temp;

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

temp.OrderedInsert(M[i]);

res = Polinom(temp);

}

};

//............................................................................

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

{

protected:

Polinom P1;

public:

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

~TestParsing() {}

};

//............................................................................

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

{

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

}

//............................................................................

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

PolTestParse(Expr[0], vector<Monom> {M[0]}),

PolTestParse(Expr[1], vector<Monom> {M[1]}),

PolTestParse(Expr[2], vector<Monom> {M[2]}),

PolTestParse(Expr[3], vector<Monom> {M[3]}),

PolTestParse(Expr[4], vector<Monom> {M[4]}),

PolTestParse(Expr[5], vector<Monom> {M[5]}),

PolTestParse(Expr[6], vector<Monom> {M[6]}),

PolTestParse(Expr[7], vector<Monom> {M[7]}),

PolTestParse(Expr[8], vector<Monom> {M[8]}),

PolTestParse(Expr[9], vector<Monom> {M[9]}),

PolTestParse(Expr[10], vector<Monom> {M[10]}),

PolTestParse(Expr[11], vector<Monom> {M[4], M[5]}),

PolTestParse(Expr[12], vector<Monom> {Monom(1,100),Monom(-1,10)}),

PolTestParse(Expr[13], vector<Monom> {Monom(1, 200), Monom(-1, 20)}),

PolTestParse(Expr[14], vector<Monom> {Monom(2, 111)}),

PolTestParse(Expr[15], vector<Monom> {Monom(1,222)})

));

//............................................................................

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(SECOND, TestSum, ::testing::Values(

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

TGeneralPol("0", "x", "-x"),

TGeneralPol("0", "-xyz", "xyz"),

TGeneralPol("2xyz", "yzx", "xyz"),

TGeneralPol("x+y", "x", "y"),

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

TGeneralPol("18x^2y^2", "10x^2y^2+x^2", "8x^2y^2-x^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(THIRD, TestMult, ::testing::Values(

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

TGeneralPol("xyz", "xyz", "1"),

TGeneralPol("xyz^2", "xz", "zy"),

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

TGeneralPol("111xyz", "55.5xy", "2z"),

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

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

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

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

));