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 лабораторной работе

**«Хранение полиномов и арифметические действия над ними»**

**Выполнила:** студентка группы 381603-1

Прохорова О. Д.

**Проверила:** кандидат тех. наук, старший преподаватель каф. МОСТ института ИТММ

Кустикова В.Д.

Нижний Новгород

2018

Оглавление

[Введение 3](#_Toc515414878)

[1. Постановка задачи 4](#_Toc515414879)

[2. Руководство пользователя 5](#_Toc515414880)

[3. Руководство программиста 6](#_Toc515414881)

[1. Описание алгоритмов 6](#_Toc515414882)

[2. Описание структуры программы 8](#_Toc515414883)

[3. Описание структур данных 9](#_Toc515414884)

[Заключение 10](#_Toc515414885)

[Литература 11](#_Toc515414886)

[Приложение 12](#_Toc515414887)

[list.h 12](#_Toc515414888)

[polinom.h 18](#_Toc515414889)

[polinom.cpp 20](#_Toc515414890)

[main.cpp 28](#_Toc515414891)

[test\_list.cpp 31](#_Toc515414892)

[test\_polinom.cpp 37](#_Toc515414893)

# Введение

Изучение полиномиальных уравнений и их решений составляло едва ли не главный объект «классической алгебры».

С изучением полиномов связан целый ряд преобразований в математике: введение в рассмотрение [нуля](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D0%BB%D1%8C_(%D1%87%D0%B8%D1%81%D0%BB%D0%BE)), [отрицательных](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D1%80%D0%B8%D1%86%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE), а затем и [комплексных чисел](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE), а также появление [теории групп](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D1%83%D0%BF%D0%BF) как раздела математики и выделение классов [специальных функций](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8) в анализе.

Полиномы также играют ключевую роль в [алгебраической геометрии](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F), объектом которой являются множества, определённые как решения систем полиномиальных уравнений. Особые свойства преобразования коэффициентов при умножении полиномов используются в алгебраической геометрии, [алгебре](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0), [теории узлов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%83%D0%B7%D0%BB%D0%BE%D0%B2) и других разделах математики для кодирования или выражения полиномами свойств различных объектов.

Лабораторная работа направлена на изучение методов компьютерной обработки

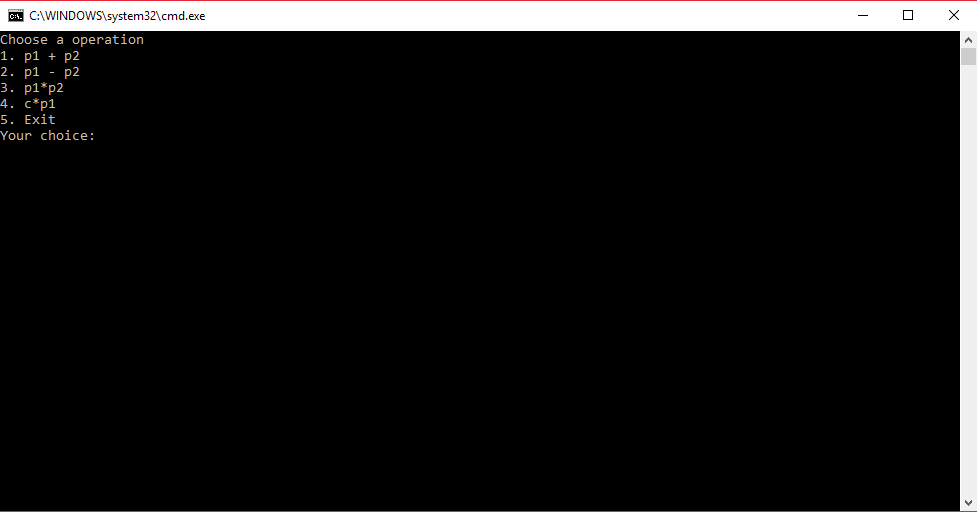
полиномов. С этой целью в лабораторной работе изучаются различные варианты структуры хранения и методы обработки полиномов. В ходе выполнения лабораторной работы разрабатывается общая форма представления циклических списков, разрабатываются программы работы со списками, которые могут быть использованы и в других областях приложений.

# 1. Постановка задачи

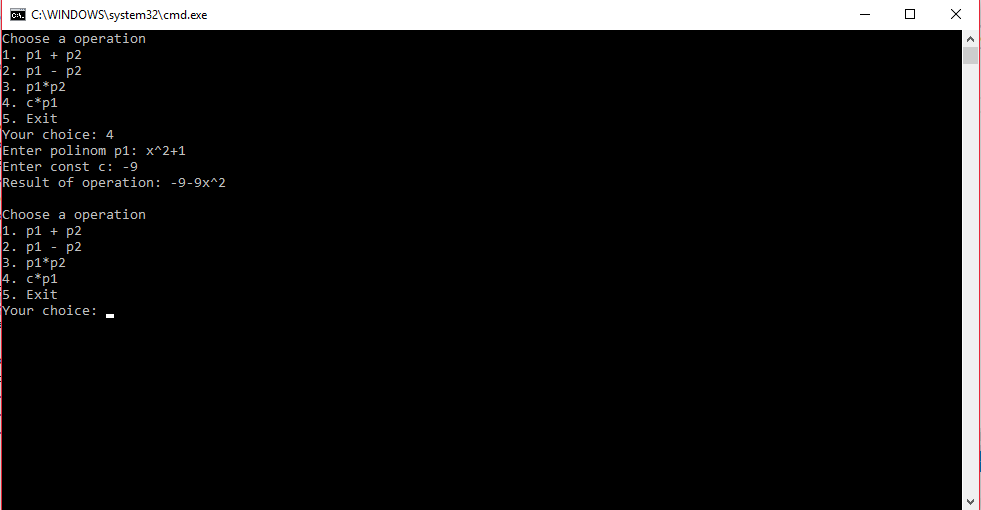
Необходимо написать программу, выполняющую арифметические операции над полиномами от трёх переменных. Допустимые операции: сложение, вычитание, умножение на константу, перемножение двух полиномов. Коэффициенты перед мономами – вещественные числа. Полином – сумма мономов. Степень каждого монома не превышает 9.

# 2. Руководство пользователя

При запуске программы на экран выведется меню, предлагающее пользователю выбрать одно из пяти действий:



При выборе одного из действий, программа предложит ввести данные, а затем выведет результат вычислений и снова откроет меню:



Для выхода из программы необходимо выбрать пятый пункт.

# 3. Руководство программиста

## 1. Описание алгоритмов

**Сложение полиномов**

На вход подаётся два полинома, которые представляют собой циклические списки с головой. Звено списка – моном.

Двигаем указатели обоих списков и списка-результата на звенья, следующие за головой.

Пока в обоих списках не дошли до конца:

Сравниваем звенья списков;

Меньшее звено вставляем в конец списка-результата, сдвигаем указатели списка, из которого взяли элемент и списка-результата на одно звено вперёд;

Звенья равны - создаём моном, коэффициент которого равен сумме коэффициентов мономов, лежащих в этих звеньях, и записываем его в конец списка-результата, сдвигаем указатели обоих списков и списка-результата на одно звено вперёд;

После того, как один из списков кончился, дописываем остаток второго списка в результат;

Возвращаем полином-результат.

**Вычитание полиномов**

Реализуется как сложение полинома с «вычитаемым» полиномом, умноженным на -1.

**Умножение полинома на полином**

На вход подаётся два полинома, которые представляют собой циклические списки с головой. Звено списка – моном.

Сдвигаем указатели обоих списков на звенья, следующие за головой.

Пока в первом списке не дошли до конца:

Создаём полином-копию второго полинома и сдвигаем в нём (рассматривая его как список) указатель на звено, следующее за головой;

Пока в нем не дошли до конца:

Если мономы в текущих звеньях первого списка и списка-копии второго удовлетворяют условию: сумма степеней “x” (“y”, “z”) одного и другого меньше 10, то складываем их обобщенные степени, перемножаем коэффициенты, результат записываем в текущее звено списка-копии второго полинома;

Иначе выводим сообщение об ошибке;

Сдвигаем указатель в списке-копии второго полинома на следующее звено;

Полином-результат = полином-результат + полином-копия второго полинома;

Сдвигаем указатель в первом списке на следующее звено;

Возвращаем полином-результат.

**Умножение полинома на константу**

Если константа не равна нулю:

Создаём полином-результат, равный исходному полиному;

В списке-результате сдвигаем указатель на звено, следующее за головой;

Пока не дошли до конца списка-результата:

Умножаем коэффициент монома, лежащего в текущем звене, на константу;

Сдвигаем указатель на следующее звено;

Возвращаем полином-результат.

## 2. Описание структуры программы

Программа содержит файлы:

* list.hсодержит объявление шаблонных классов TLink и TRingList и их реализацию;
* polinom*.*h содержит объявление классов TMonom и TPolinome;
* polinom.cpp содержит реализацию классов TMonom и TPolinome;
* main.cpp содержит реализацию пользовательского интерфейса;
* test\_list.cpp содержит тесты для класса TRingList;
* test\_ polinom.cpp содержит тесты для классов TMonom и TPolinome.

## 3. Описание структур данных

В качестве структуры хранения полинома используется циклический список с головой. Звенья списка – мономы, однозначно определяемые через коэффициент и обобщенную степень. Голова – фиктивное звено, не содержащее монома. Элементы списка хранятся в порядке возрастания обобщённых степеней.

**Шаблонный класс TLink**

Звено списка. Имеет два конструктора.

Поля: T data – данные, хранящиеся в звене; TLink \*next – указатель на следующее звено.

Методы: operator <, operator > - логические операторы сравнения.

**Шаблонный класс TRingList**

Циклический список с головой. Имеет два конструктора и деструктор.

Поля: TLink <T> \*head, \*curr – голова списка и указатель на текущее звено.

Методы:

void Insert(T data) – упорядоченная вставка;

void InsertTail(T data) – вставка в конец;

TRingList<T>& operator =(const TRingList<T> &a) – оператор присваивания;

bool operator ==(const TRingList<T>& sp) const – оператор сравнения;

bool operator !=(const TRingList<T>& sp) const - оператор сравнения;

void Reset() – перемещение указателя curr в начало списка;

bool IsNotEnded() const – проверка, кончился ли список;

void GetNext() – перемещение указателя curr на одно звено вперёд;

TLink<T>\* GetCurr() const – получение значения указателя curr;

void Clean() – очистка списка.

**Класс TMonom**

Имеет один конструктор.

Поля: double cf – коэффициент монома; unsigned int abc – обобщённая степень монома (Обобщённая степень монома – число, число сотен которого – это степень “x”, число десятков - степень “y”, число единиц - степень “z”.).

Методы:

TMonom& operator =(const TMonom& m) – оператор присваивания;

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

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

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

bool operator !=(const TMonom& m) const – оператор сравнения.

**Класс TPolinome**

Имеет три конструктора.

Поля: TRingList<TMonom> monoms – полином (список из мономов);

Методы:

TRingList<TMonom> similar\_terms(TRingList<TMonom> sp) – приведение подобных слагаемых;

TPolinome operator +(const TPolinome &polinome) const – сложение полиномов;

TPolinome operator \*(const TPolinome &polinome) const – умножение полиномов;

TPolinome operator \*(const double &d) const – умножение полинома на константу справа;

friend TPolinome operator\*(const double d,const TPolinome& polinome) – умножение полинома на константу слева;

TPolinome& operator =(const TPolinome &polinome) – оператор присваивания;

TPolinome operator -(const TPolinome& polinome) const – вычитание полиномов;

TPolinome operator -() const – умножение полинома на -1;

friend ostream& operator<<(ostream &out, const TPolinome &polinome) – вывод полинома;

bool operator==(const TPolinome& polinome) const – оператор сравнения;

bool operator!=(const TPolinome& polinome) const – оператор сравнения.

# Заключение

Была разработана программа, выполняющая арифметические действия над полиномами: сложение, вычитание, умножение на полином и на константу. Хранение полиномов осуществляется при помощи односвязного циклического списка с головой.

# Литература

1. *Тимофеев В. В.* Самоучитель по С++ - М.: Издательство Бином, 2014. -338 с., ил.

2. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: построение и анализ, Москва 2013.

3. Рабочие материалы студента по общему курсу «Методы программирования

# Приложение

## list.h

#pragma once

#include <stdlib.h>

template <typename T>

class TLink

{

public:

T data;

TLink \*next;

TLink();

TLink(T d, TLink<T>\* nx = NULL);

bool operator <(const TLink& lin) const;

bool operator >(const TLink& lin) const;

};

template <typename T>

TLink<T>::TLink()

{

next = NULL;

};

template <typename T>

TLink<T>::TLink(T d, TLink<T>\* nx = NULL)

{

next=nx;

data=d;

};

template <typename T>

bool TLink<T>::operator <(const TLink& lin) const

{

return (data<lin.data);

};

template <typename T>

bool TLink<T>::operator >(const TLink& lin) const

{

return (data>lin.data);

};

template <typename T>

class TRingList

{

private:

TLink <T> \*head, \*curr;

public:

TRingList();

TRingList(const TRingList<T> &list);

~TRingList();

void Insert(T data);

void InsertTail(T data);

TRingList<T>& operator =(const TRingList<T> &a);

bool operator ==(const TRingList<T>& sp) const;

bool operator !=(const TRingList<T>& sp) const;

void Reset();

bool IsNotEnded() const;

void GetNext();

TLink<T>\* GetCurr() const;

void Clean();

};

template <typename T>

TRingList<T>::TRingList()

{

head=new TLink<T>;

head->next=head;

curr=head;

};

template <typename T>

void TRingList<T>::Clean()

{

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

while (tmp != head)

{

TLink<T>\* tmp2 = tmp->next;

delete tmp;

tmp = tmp2;

}

head->next = head;

};

template <typename T>

TRingList<T>::TRingList(const TRingList<T> &list)

{

head = new TLink<T>;

TLink<T>\* A = list.head;

TLink<T>\* B = head;

if ( A->next == list.head)

{

head->next = head;

return;

}

while (A->next != list.head)

{

A = A->next;

B->next = new TLink<T>(A->data);

B = B->next;

}

B->next = head;

curr = head -> next;

};

template <typename T>

TRingList<T>::~TRingList()

{

Clean();

delete head;

};

template <typename T>

void TRingList<T>::Insert(T data)

{

TLink<T>\* tmp = head;

TLink<T>\* elem = new TLink<T>(data);

while ((tmp->next != head) && (\*(tmp->next) < \*elem))

tmp = tmp->next;

TLink<T>\* tmp2 = tmp->next;

tmp->next = elem;

tmp->next->next = tmp2;

};

template <typename T>

void TRingList<T>::Reset()

{

curr = head->next;

};

template <typename T>

bool TRingList<T>::IsNotEnded() const

{

return (!(curr == head));

};

template <typename T>

void TRingList<T>::GetNext()

{

curr = curr->next;

};

template <typename T>

TLink<T>\* TRingList<T>::GetCurr() const

{

return curr;

};

template <typename T>

TRingList<T>& TRingList<T>::operator =(const TRingList<T> &a)

{

Clean();

TLink<T>\* A = a.head;

TLink<T>\* B = head;

while (A->next != a.head)

{

A = A->next;

B->next = new TLink<T>(A->data);

B = B->next;

}

B->next = head;

curr = head;

return \*this;

};

template <typename T>

bool TRingList<T>::operator ==(const TRingList<T>& sp) const

{

bool res = true;

if (this != &sp)

{

TLink<T>\* a = head->next;

TLink<T>\* b = sp.head->next;

while ((a->data == b->data)&&(a != head)&&(b != sp.head))

{

a = a->next;

b = b->next;

}

if ((a != head)||(b != sp.head))

res = false;

}

return res;

};

template <typename T>

bool TRingList<T>::operator !=(const TRingList<T>& sp) const

{

return !(\*this == sp);

};

template <typename T>

void TRingList<T>::InsertTail(T data)

{

Reset();

while (curr ->next != head) GetNext();

TLink<T>\* tmp = curr->next;

curr->next = new TLink<T>(data);

curr->next->next = tmp;

};

## polinom.h

#include "list.h"

#include <string>

#include <iostream>

using namespace std;

class TMonom

{

public:

double cf;

unsigned int abc;

TMonom(double c = 0, unsigned int a = 0);

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;

};

class TPolinome

{

private:

TRingList<TMonom> monoms;

public:

TRingList<TMonom> similar\_terms(TRingList<TMonom> sp);

TPolinome(string str="");

TPolinome(const TPolinome &polinome);

TPolinome operator +(const TPolinome &polinome) const;

TPolinome operator \*(const TPolinome &polinome) const;

TPolinome operator \*(const double &d) const;

friend TPolinome operator\*(const double d,const TPolinome& polinome)

{ return polinome\*d; };

TPolinome& operator =(const TPolinome &polinome);

TPolinome operator -(const TPolinome& polinome) const;

TPolinome operator -() const;

friend ostream& operator<<(ostream &out, const TPolinome &polinome);

bool operator==(const TPolinome& polinome) const;

bool operator!=(const TPolinome& polinome) const;

TPolinome(TRingList<TMonom> &list) : monoms(list) {};

## polinom.cpp

#include "polinom.h"

using namespace std;

TMonom::TMonom(double c, unsigned int a)

{

cf = c;

abc = a;

};

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

{

cf = m.cf;

abc = m.abc;

return \*this;

};

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

{

if (abc >= m.abc) return false;

return true;

};

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

{

if (abc <= m.abc) return false;

return true;

};

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

{

if ((abc != m.abc)||(cf != m.cf)) return false;

return true;

};

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

{

return !(\*this == m);

};

TRingList<TMonom> TPolinome::similar\_terms(TRingList<TMonom> sp)

{

TRingList<TMonom> res;

res.Reset();

sp.Reset();

TLink<TMonom> mon(sp.GetCurr()->data.cf);

while (sp.IsNotEnded())

{

mon.data.abc = sp.GetCurr()->data.abc;

if (sp.GetCurr()->data.abc == sp.GetCurr()->next->data.abc &&(sp.GetCurr()->next->data.cf || sp.GetCurr()->next->data.abc))

mon.data.cf += sp.GetCurr()->next->data.cf;

else

{

if (mon.data.cf)

{

res.InsertTail(mon.data);

res.GetNext();

}

mon.data.cf = sp.GetCurr()->next->data.cf;

}

sp.GetNext();

}

return res;

};

TPolinome::TPolinome(string str)

{

TRingList<TMonom> res;

while (str.length())

{

string part;

TMonom temp;

int pos = 1;

while ((pos < str.length())&&(str[pos] != '+')&&(str[pos] != '-')) pos++;

part = str.substr(0, pos);

str.erase(0, pos);

pos = 0;

while ((part[pos] != 'x')&&(part[pos] != 'y')&&(part[pos] != 'z')&&(pos < part.length())) pos++;

string c = part.substr(0,pos);

if ((c == "+")||(c.length() == 0)) temp.cf = 1;

else

if (c == "-") temp.cf = -1;

else

temp.cf = stod(c);

part.erase(0, pos);

part += ' ';

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

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

{

pos = part.find((char)(120 + i));

if (pos > -1)

{

if (part[pos + 1] != '^') part.insert(pos + 1, "^1");

temp.abc += a[i] \* stoi(part.substr(pos + 2, 1));

part.erase(pos, 3);

}

}

monoms.Insert(temp);

}

monoms = similar\_terms(monoms);

};

TPolinome::TPolinome(const TPolinome &polinome)

{

monoms = polinome.monoms;

};

TPolinome TPolinome::operator +(const TPolinome &polinome) const

{

TPolinome res;

TPolinome pthis = \*this;

TPolinome p = polinome;

pthis.monoms.Reset();

p.monoms.Reset();

res.monoms.Reset();

while ((pthis.monoms.IsNotEnded()) && (p.monoms.IsNotEnded()))

{

if (pthis.monoms.GetCurr()->data > p.monoms.GetCurr()->data)

{

res.monoms.InsertTail(p.monoms.GetCurr()->data);

p.monoms.GetNext();

res.monoms.GetNext();

}

else

if (pthis.monoms.GetCurr()->data < p.monoms.GetCurr()->data)

{

res.monoms.InsertTail(pthis.monoms.GetCurr()->data);

pthis.monoms.GetNext();

res.monoms.GetNext();

}

else

{

double newcf = pthis.monoms.GetCurr()->data.cf + p.monoms.GetCurr()->data.cf;

if (newcf)

{

TMonom temp(newcf, pthis.monoms.GetCurr()->data.abc);

res.monoms.InsertTail(temp);

res.monoms.GetNext();

}

pthis.monoms.GetNext();

p.monoms.GetNext();

}

}

while (pthis.monoms.IsNotEnded())

{

res.monoms.InsertTail(pthis.monoms.GetCurr()->data);

pthis.monoms.GetNext();

res.monoms.GetNext();

}

while (p.monoms.IsNotEnded())

{

res.monoms.InsertTail(p.monoms.GetCurr()->data);

p.monoms.GetNext();

res.monoms.GetNext();

}

return res;

};

TPolinome TPolinome::operator \*(const TPolinome &polinome) const

{

TPolinome res;

TPolinome pthis = \*this;

TPolinome p =polinome;

pthis.monoms.Reset();

p.monoms.Reset();

while (pthis.monoms.IsNotEnded())

{

double pthiscf = pthis.monoms.GetCurr()->data.cf;

int pthisabc = pthis.monoms.GetCurr()->data.abc;

TPolinome temp(polinome);

temp.monoms.Reset();

while (temp.monoms.IsNotEnded())

{

int tempabc = temp.monoms.GetCurr()->data.abc;

if ((tempabc % 10 + pthisabc % 10) < 10 && (tempabc/10 % 10 + pthisabc/10 % 10) < 10 && (tempabc/100 + pthisabc/100) < 10)

{

temp.monoms.GetCurr()->data.abc += pthisabc;

temp.monoms.GetCurr()->data.cf \*= pthiscf;

}

else

throw "large index";

temp.monoms.GetNext();

}

res = res + temp;

pthis.monoms.GetNext();

}

return res;

};

TPolinome TPolinome::operator \*(const double &d) const

{

TPolinome res;

if (d)

{

res = \*this;

res.monoms.Reset();

while (res.monoms.IsNotEnded())

{

res.monoms.GetCurr()->data.cf \*= d;

res.monoms.GetNext();

}

}

return res;

};

TPolinome TPolinome::operator -() const

{

return (\*this)\*(-1.0);

};

TPolinome& TPolinome::operator =(const TPolinome &polinome)

{

monoms = polinome.monoms;

return \*this;

};

TPolinome TPolinome::operator -(const TPolinome& polinome) const

{

return \*this+polinome\*(-1.0);

};

bool TPolinome::operator==(const TPolinome& polinome) const

{

return (monoms == polinome.monoms);

};

bool TPolinome::operator!=(const TPolinome& polinome) const

{

return !(\*this == polinome);

};

ostream& operator<<(ostream &out, const TPolinome &polinome)

{

TPolinome p = polinome;

p.monoms.Reset();

while (p.monoms.IsNotEnded())

{

TMonom temp = p.monoms.GetCurr()->data;

if (temp.cf > 0)

{

out << "+";

if (temp.cf != 1)

out << temp.cf;

}

else

out << temp.cf;

int a = temp.abc / 100;

if (a>1)

out << "x^" << a;

else

if (a == 1)

out << "x";

a = temp.abc / 10 % 10;

if (a>1)

out << "y^" << a;

else

if (a == 1)

out << "y";

a = temp.abc % 10;

if (a>1)

out<< "z^" << a;

else

if (a == 1)

out << "z";

p.monoms.GetNext();

}

return out;

};

## main.cpp

#include "polinom.h"

#include <iostream>

#include <string>

using namespace std;

char menu()

{

char choice;

cout<<"Choose a operation"<<endl;

cout<<"1. p1 + p2"<<endl;

cout<<"2. p1 - p2"<<endl;

cout<<"3. p1\*p2"<<endl;

cout<<"4. c\*p1"<<endl;

cout<<"5. Exit"<<endl;

cout << "Your choice: ";

cin >> choice;

return choice;

}

int main()

{

string s1,s2;

TPolinome p1, p2;

double c;

char number;

do

{

number = menu();

switch(number)

{

case '1':

{

cout << "Enter polinom p1: ";

cin >> s1;

p1 = s1;

cout << "Enter polinom p2: ";

cin >> s2;

p2 = s2;

cout << "Result of operation: " << p1 + p2 << endl << endl;

break;

}

case '2':

{

cout << "Enter polinom p1: ";

cin >> s1;

p1 = s1;

cout << "Enter polinom p2: ";

cin >> s2;

p2 = s2;

cout << "Result of operation: " << p1 - p2 << endl << endl;

break;

}

case '3':

{

cout << "Enter polinom p1: ";

cin >> s1;

p1 = s1;

cout << "Enter polinom p2: ";

cin >> s2;

p2 = s2;

cout << "Result of operation: " << p1 \* p2 << endl << endl;

break;

}

case '4':

{

cout << "Enter polinom p1: ";

cin >> s1;

p1 = s1;

cout << "Enter const c: ";

cin >> c;

cout << "Result of operation: " << p1\*c << endl << endl;

break;

}

case '5':

{

break;

}

}

}

while (number != '5');

return 0;

}

## test\_list.cpp

#include "gtest.h"

#include "list.h"

class TestList : public ::testing::Test

{

protected:

TRingList<int> list1;

TRingList<int> list2;

public:

TestList() { list2.Insert(6); list2.Insert(1);}

~TestList() {}

};

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

{

ASSERT\_NO\_THROW(TRingList<int> list);

}

TEST\_F(TestList, empty\_list\_is\_empty)

{

EXPECT\_EQ(true, !list1.IsNotEnded());

}

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

{

ASSERT\_NO\_THROW(TRingList<int> list(list1));

}

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

{

TRingList<int> list(list1);

EXPECT\_NE(list1.GetCurr(), list.GetCurr());

}

TEST\_F(TestList, can\_assign\_empty\_list)

{

ASSERT\_NO\_THROW(TRingList<int> list = list1);

}

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

{

TRingList<int> list,list2;

EXPECT\_EQ(true, list == list2);

}

TEST\_F(TestList, assigned\_empty\_list\_is\_correct)

{

TRingList<int> list = list1;

EXPECT\_NE(list1.GetCurr(), list.GetCurr());

}

TEST\_F(TestList, can\_compare\_empty\_list\_with\_itself)

{

EXPECT\_EQ(true, list1 == list1);

}

TEST\_F(TestList, can\_compare\_list\_with\_itself)

{

EXPECT\_EQ(true, list2 == list2);

}

TEST\_F(TestList, can\_insert\_in\_empty\_list)

{

list1.Insert(4);

list1.Reset();

EXPECT\_EQ(list1.GetCurr(), list1.GetCurr() -> next -> next );

}

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

{

list1.Insert(4);

list1.Reset();

EXPECT\_EQ(4, list1.GetCurr() -> data);

}

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

{

ASSERT\_NO\_THROW(TRingList<int> list(list2));

}

TEST\_F(TestList, not\_empty\_list\_is\_copied\_correctly\_1)

{

TRingList<int> list(list2);

EXPECT\_NE(list2.GetCurr(), list.GetCurr());

}

TEST\_F(TestList, not\_empty\_list\_is\_copied\_correctly\_2)

{

TRingList<int> list(list2);

EXPECT\_EQ(list.GetCurr(), list.GetCurr()->next->next->next);

}

TEST\_F(TestList, not\_empty\_list\_is\_copied\_correctly\_3)

{

TRingList<int> list(list2);

list.Reset();

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

EXPECT\_EQ(6, list.GetCurr()->next->data);

}

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

{

ASSERT\_NO\_THROW(TRingList<int> list = list2);

}

TEST\_F(TestList, assigned\_not\_empty\_list\_is\_correct\_1)

{

TRingList<int> list = list2;

EXPECT\_NE(list.GetCurr(), list2.GetCurr());

}

TEST\_F(TestList, assigned\_not\_empty\_list\_is\_correct\_2)

{

TRingList<int> list = list2;

EXPECT\_EQ(list.GetCurr(), list.GetCurr()->next->next->next);

}

TEST\_F(TestList, assigned\_not\_empty\_list\_is\_correct\_3)

{

TRingList<int> list = list2;

list.Reset();

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

EXPECT\_EQ(6, list.GetCurr()->next->data);

}

TEST\_F(TestList, not\_empty\_lists\_are\_equal)

{

TRingList<int> list(list2);

EXPECT\_EQ(true, list == list2);

}

TEST\_F(TestList, not\_empty\_and\_empty\_lists\_are\_not\_equal)

{

EXPECT\_EQ(false, list2 == list1);

}

TEST\_F(TestList, insert\_to\_the\_beginning\_1)

{

list2.Insert(0);

EXPECT\_EQ(list2.GetCurr()->next->next->next->next, list2.GetCurr());

}

TEST\_F(TestList, insert\_to\_the\_beginning\_2)

{

list2.Insert(0);

list2.Reset();

EXPECT\_EQ(0, list2.GetCurr()->data);

EXPECT\_EQ(1, list2.GetCurr()->next->data);

EXPECT\_EQ(6, list2.GetCurr()->next->next->data);

}

TEST\_F(TestList, insert\_in\_the\_middle\_1)

{

list2.Insert(3);

EXPECT\_EQ(list2.GetCurr()->next->next->next->next, list2.GetCurr());

}

TEST\_F(TestList, insert\_in\_the\_middle\_2)

{

list2.Insert(3);

list2.Reset();

EXPECT\_EQ(1, list2.GetCurr()->data);

EXPECT\_EQ(3, list2.GetCurr()->next->data);

EXPECT\_EQ(6, list2.GetCurr()->next->next->data);

}

TEST\_F(TestList, insert\_in\_the\_end\_1)

{

list2.Insert(8);

EXPECT\_EQ(list2.GetCurr()->next->next->next->next, list2.GetCurr());

}

TEST\_F(TestList, insert\_in\_the\_end\_2)

{

list2.Insert(8);

list2.Reset();

EXPECT\_EQ(1, list2.GetCurr()->data);

EXPECT\_EQ(6, list2.GetCurr()->next->data);

EXPECT\_EQ(8, list2.GetCurr()->next->next->data);

}

## test\_polinom.cpp

#include "gtest.h"

#include "polinom.h"

#include <vector>

using namespace std;

TEST(TMonom, can\_create\_monom)

{

ASSERT\_NO\_THROW(TMonom m);

}

TEST(TPolinome, can\_create\_polinom)

{

ASSERT\_NO\_THROW(TPolinome p);

}

TEST(TPolinome, can\_copy\_polinoms)

{

TPolinome a("x^4");

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

}

TEST(TPolinome, copy\_polinoms\_correct)

{

TPolinome a("x^4");

TPolinome b(a);

EXPECT\_EQ(a, b);

}

TEST(TPolinome, can\_assign\_polinoms)

{

TPolinome a("x^2");

TPolinome b;

ASSERT\_NO\_THROW(b = a);

}

TEST(TPolinome, assign\_polinoms\_correct)

{

TPolinome a("x^2");

TPolinome b = a;

EXPECT\_EQ(a, b);

}

TEST(TPolinome, can\_sum\_polinoms)

{

TPolinome a("x^2+z-1");

TPolinome b("x^2-z-1");

TPolinome c("2x^2-2");

EXPECT\_EQ(c, a + b);

}

TEST(TPolinome, can\_dif\_polinoms)

{

TPolinome a("x^2+y^3+z-2");

TPolinome b("x^2-y^3+z-1");

TPolinome c("2y^3-1");

EXPECT\_EQ(c, a - b);

}

TEST(TPolinome, can\_mul\_polinoms)

{

TPolinome a("x^2+y^1-1");

TPolinome b("z^2");

TPolinome c("x^2\*z^2+y^1\*z^2-z^2");

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

}

TEST(TPolinome, can\_mul\_polinom\_on\_const)

{

TPolinome a("x^2-1");

int k = 10;

TPolinome c("10x^2-10");

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

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

}

TEST(TPolinome, degree\_not\_more\_than\_9)

{

TPolinome a("x^5"), b("x^5");

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

}