Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский Нижегородский государственный университет

им. Н.И. Лобачевского» (ННГУ)

Институт Информационных технологий, математики и механики

Отчёт по лабораторной работе

**«Арифметические операции с полиномами»**

**Выполнил**:

студент института ИТММ   
гр. 381603-1

Болотов Д.И.

**Проверил**:

Кандидат тех. наук, старший преподаватель каф. МОСТ   
института ИТММ

Кустикова В.Б.

Нижний Новгород

2018 г.

**Содержание**

[Введение 3](#_Toc514109797)

[1 Постановка задачи 4](#_Toc514109798)

[2 Руководство пользователя 5](#_Toc514109799)

[3 Руководство программиста 6](#_Toc514109800)

[3.1 Структура программы 6](#_Toc514109801)

[3.2 Структуры данных 6](#_Toc514109802)

[3.3 Алгоритмы 6](#_Toc514109803)

[3.4 Программная реализация 10](#_Toc514109804)

[3.4.1 Структура TMonom 10](#_Toc514109805)

[3.4.2 Класс TLink 10](#_Toc514109806)

[3.4.3 Класс TRingList 11](#_Toc514109807)

[3.4.4 Класс TPolinom 11](#_Toc514109808)

[Заключение 13](#_Toc514109809)

[Литература 14](#_Toc514109810)

[Приложение 15](#_Toc514109811)

[Приложение А. Программная реализация монома 15](#_Toc514109812)

[Приложение Б. Программная реализация звена 19](#_Toc514109813)

[Приложение В. Программная реализация списка 20](#_Toc514109814)

[Приложение Г. Программная реализация полинома 24](#_Toc514109815)

[Приложение Д. Реализация основного приложения 30](#_Toc514109816)

# Введение

Данная лабораторная посвящена теме “Арифметически операции с полиномами”. Целью работы является разработка консольного приложения, выполняющего следующие операции: сумма, разность и произведение полиномов, а также произведение полинома и константы. Для реализации приложения использовался циклический односвязный список.

Данная программа экономит время пользователя, которое он затратил для вычисления выражения, и выдает точно подсчитанный результат. Программа может использоваться и в других более сложных проектах.

# Постановка задачи

Необходимо разработать программу, выполняющую арифметически операции над полиномами. Операции: сложение, вычитание, умножение и умножение на константу. Полином может состоять из трех переменных (x, y, z). Степень переменных полинома принимает значения от 0 до 9. Коэффициенты могут быть только вещественными числами. После вычислений, программа должна вывести пользователю результат.

# Руководство пользователя

При запуске программы, пользователю будет предложено выбрать операцию, которую он хочет произвести:

1. Сумма полиномов
2. Разность полиномов
3. Произведение полиномов
4. Произведение полинома и числа
5. Произведение числа и полинома

Выбрав одну из операций, пользователю необходимо будет ввести по очереди два полинома (в случае 4 и 5 вводятся полином и число).

Далее программа выводит результат выполнения операции и предлагает повторить работу в программе. При отказе, программа прекращает свою работу.

# Руководство программиста

## Структура программы

Программа состоит из четырех проектов.

*gtest*

Содержит в себе gtest.h и gtest-all.cpp. Файлы содержат google тесты, которые необходимы для проверки корректности реализованных классов.

*polinom\_lib*

Содержит в себе:

1. TLink.h – реализация шаблонного класса звена.
2. TMonom.h – объявление полей и методов, которые отвечают за степень и коэффициент звена
3. TPolinom.h – объявление полей и методов класса полином
4. TRingList.h – реализация шаблонного класса список
5. TMonom.cpp – реализация полей и методов класса Monom
6. TPolinom.cpp – реализация полей и методов класса полином

*polinom\_test*

Содержит в себе test\_main.cpp, test\_polinom.cpp, test\_ringlist.cpp. Реализация тестов для проверки корректности реализованных классов.

*sample*

Содержит main.cpp – реализация меню консольного приложения.

## Структуры данных

В программе реализована структура TMonom (моном), класс TLink (звено), класс, TRingList (список), класс TPolinom (полином). Полином состоит из зацикленного списка с головой. Список состоит из звеньев. Полями списка является звено – голова и звено – текущий элемент в списке. ­Звенья хранят в себе указатель на звено и моном. Моном состоит из числа двойной точности – коэффициент и беззнакового целого числа – степень монома.

## Алгоритмы

Конструктор полинома по строке:

Определяется длина строки, и запускается цикл. Цикл будет выполняться, пока не кончится строка. Внутри цикла реализован следующий алгоритм:

1. Если мы встретили на i–ой позиции число, или переменную, или ‘-‘, то заходим внутрь инструкции if, иначе увеличивается переменная цикла i (снова происходит проверка i–ой позиции).
2. В инструкции if создается новая переменная j.
3. Запускается цикл while. Если на i + 1 + j позиции находится ‘+’, ‘-‘ или конец строки, то увеличиваем переменную j. Иначе, выходим из цикла.
4. Следующим шагом в новую строку мы копируем исходную строку с i–ой позиции j + 1 символ. Таким образом, мы выделяем каждое слагаемое в строке.
5. Передаем подстроку в конструктор монома.
6. В список вставляем полученный моном.
7. Переменную цикла увеличиваем на i + j.
8. После завершения цикла вызывается функция приведения подобных слагаемых.

Сложение полиномов:

Создается новый полином. В списках у полиномов, которые являются первым и вторым слагаемым соответственно, указатель на текущий элемент переводиться на следующее за головой звено.

1. Запускается цикл while, условием которого является то, что оба списка не кончились. Если один из списков закончился, переходим к пункту 3.
2. Внутри цикла while сравниваем текущие звенья в списках. В новый список нового полинома вставляется моном, имеющее большее значение степени. Указатель передвигается на следующее звено. При равенстве степеней вставляется сумма двух моном. Указатель передвигается у обоих списков.
3. Происходит проверка. Если закончился первый список, то в новый список вставляется все оставшиеся звенья из второго списка. Аналогично в противоположном случае.
4. Возвращаем полученный новый полином.

Умножение полиномов:

Создается новый полином (1). У списка первого множителя указатель текущего элемента устанавливается на следующее звено после головы:

1. Запускается цикл while, условием которого является то, что список первого множителя не кончился. Если условие не выполняется, переходим к пункту 10.
2. Внутри цикла while у списка второго множителя указатель на текущий элемент устанавливается на следующее звено после головы. Запускается еще один цикл while, условием которого является то, что список второго множителя не кончился. Если условие не выполняется, переходим к пункту 9.
3. Создается моном
4. Коэффициентом нового монома является произведение коэффициентов двух мономов текущих звеньев.
5. Если степень не равно нулю, то создается новая переменная, отвечающая за ступень нового монома. Иначе, переходим к пункту 8. Она равно сумме степеней двух мономов текущих звеньев. Если эта переменная лежит в пределе от 0 до 999, то в степень монома сохраняем значение этой переменной. Иначе, выдается ошибка.
6. Создается новый полином. В него вставляется созданный моном.
7. К полиному (1) прибавляем только что созданный полином, и результат суммы записывается в (1).
8. Передвигаем указатель у списка второго множителя.
9. Передвигаем указатель у списка первого множителя.
10. Возвращаем значение (1).

Умножение полинома на константу:

Создается новый полином. Указатель у списка первого множителя устанавливается на следующий элемент после головы.

1. Запускается цикл while, условием которого является то, что список первого множителя не кончился. Иначе, переходим к пункту 4.
2. У монома текущего звена коэффициент умножаем на константу.
3. Вставляем в новый полином моном текущего звена списка первого множителя. Передвигаем указатель.
4. Возвращаем полученный полином.

Умножение константы на полином реализовано аналогично.

Вычитание полиномов:

Создается новый полином. Полином, являющийся вычитаемым, умножается на константу (-1). В созданный полином сохраняем сумма двух данных полиномов. Возвращаем полученный полином.

Функция вставка в список:

1. Если список пуст, создаем новое звено, которое указывает на голову списка. Назначаем текущим элементом новое звено. Голова списка указывает на текущий элемент.
2. Если список пуст. Текущий элемент устанавливаем на голову.
3. Создаем цикл while, условием которого является то, что если данные звена, на которое указывает текущий элемент, больше, чем данные, которые нужно вставить, то передвигаем указатель на следующее звено после текущего. Как только выйдем из цикла, создается новое звено, которое указывает на следующий элемент после текущего. Назначаем новое звено следующим элементом после текущего.

Конструктор монома по строке:

Создается 3 переменные a, b, c, отвечающие за степень переменной x, y, z соответственно. Создается переменная cf равная единице. Создается переменная abc. Определяется длина строки.

1. Запускается цикл for, условием которого является то, что строка не кончилась.
2. Если на i–ой позиции оказалось число. В коэффициент сохраняем число с помощью функции atof. Создаем переменную j. Создается цикл while. Пока на i + 1 + j позиции находится число или не конец строки, то увеличиваем переменную j. Передвигаем индекс i на i + j. Переходим к следующему пункту.
3. Если на i–ой позиции оказался ‘-‘. Если на следующей позиции оказалась переменная, то коэффициент назначаем равным (-1). Если на следующей позиции оказалось число, то коэффициент назначаем равным этому числу с помощью функции atof. Создается переменная j. Создается цикл while. Пока на i + 1 + j позиции находится число или не конец строки, то увеличиваем переменную j. Передвигаем индекс i на i + j. Переходим к следующему пункту.
4. Если на i–ой позиции оказалась переменная ‘x’. Если на следующей позиции находится знак степени ‘^’. Переходим к следующему пункту. Иначе переходим к пункту 6.
5. В переменную a сохраняем число, которое начинается с позиции i + 2. Создается переменная j. Пока на i + 2 + j позиции не встретился знак ‘\*’ или не конец строки, увеличиваем переменную j. В переменную i сохраняем значение i + j + 2. Переходим к пункту 8.
6. В переменную a сохраняем 1.
7. Аналогично выполняется для переменной ‘y’ и ‘z’.
8. В обобщенную степень сохраняем сумму произведений.

## Программная реализация

### Структура TMonom

Поля:

double cf – коэффициент монома

unsigned int abc – степень монома

Методы:

TMonom() – конструктор по умолчанию

TMonom(const string s) – конструктор по строке

TMonom(double num, unsigned int num2) – конструктор по параметрам

~TMonom() – деструктор

TMonom operator + (const TMonom& monom) – сумма мономов

TMonom operator – (const TMonom& monom) – разность мономов

TMonom operator = (const TMonom& monom) – оператор копирования

bool operator == (const TMonom& monom) const – сравнение мономов

bool operator != (const TMonom& monom) const – сравнение мономов

bool operator > (const TMonom& monom) const – сравнение мономов

bool operator < (const TMonom& monom) const – сравнение мономов

### Класс TLink

Поля:

T data – данные

TLink\* pNext – указатель на следующее звено

Методы:

TLink() – конструктор по умолчанию

TLink(const TLink& unit) – конструктор копирования

TLink(T d, TLink\* unit = NULL) – конструктор по параметрам

~TLink() – деструктор

TLink& operator = (const TLink& unit) – оператор копирования

bool operator == (const TLink& unit) const – оператор сравнения

bool operator != (const TLink& unit) const – оператор сравнения

### Класс TRingList

Поля:

TLink <T> \*head – голова списка

TLink <T> \*current – текущий элемент списка

Методы:

TRingList() – конструктор по умолчанию

TRingList(const TRingList <T> & TRingList2) – конструктор копирования

~TRingList() – деструктор

TRingList<T>& operator = (const TRingList<T>& TRingList2) – оператор копирования

void Clean() – очистка списка

void Insert(const T& data) – упорядоченная вставка в список

void Reset() – текущий элемент указывает следующее звено после головы

void Next() – двигаем указатель текущего элемента на следующий

bool IsEnded() const – проверка на конце списка

TLink<T>\* GetLink() – вернуть текущее звено

bool operator == (const TRingList<T>& TRingList2) – оператор сравнения

### Класс TPolinom

Поля:

TRingList<TMonom> monoms – список

Методы:

TPolinom() – конструктор по умолчанию

TPolinom(const string s) – конструктор по строке

TPolinom(const TPolinom &poli2) – конструткор копирования

~TPolinom() – деструктор

TPolinom operator + (const TPolinom& poli2) – оператор суммы

TPolinom operator - (const TPolinom& poli2) – оператор разности

TPolinom operator \* (const TPolinom& poli2) – оператор произведения

TPolinom operator \* (double cf2) – оператор произведения на константу

friend TPolinom operator \* (double cf2, const TPolinom& poli2) – оператор произведения константы и полинома

TPolinom& operator = (const Tpolinom& tmp) – оператор копирвоания

bool operator == (const TPolinom& poli2) – оператор сравнения

void PrivPod() – функция приведения подобных звеньев

friend ostream& operator <<(ostream& ostr, const TPolinom& poli2) – перегрузка вывода полинома

# Заключение

В данной программе было реализовано вычисление суммы, разности, и произведения полиномов, а также произведение полиномов на константу. Для реализации алгоритмов использовался односвязный список с головой.

# Литература

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

# Приложение

## Приложение А. Программная реализация монома

TMonom.h

#pragma once

#include <iostream>

#include <stdio.h>

#include <cstring>

#include <string>

#include <cstdlib>

using namespace std;

struct TMonom

{

double cf;

unsigned int abc;

TMonom();

TMonom(const string s);

TMonom(double num, unsigned int num2);

~TMonom() {}

TMonom operator + (const TMonom& monom);

TMonom operator - (const TMonom& monom);

TMonom operator = (const TMonom& monom);

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

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

bool operator > (const TMonom& monom) const;

bool operator < (const TMonom& monom) const;

};

TMonom.cpp

#include "TMonom.h"

TMonom::TMonom()

{

cf = NULL;

abc = NULL;

}

TMonom::TMonom(const string s)

{

int a = 0;

int b = 0;

int c = 0;

cf = 1;

abc = 0;

int k = s.length();

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

{

if (isdigit(s[i]))

{

this->cf = atof(&s[i]);

int j = 0;

while ((s[i + 1 + j] != '\0') && (isdigit(s[i + 1 + j])))

{

j++;

}

i = i + j;

}

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

{

if (isalpha(s[i + 1]))

{

this->cf = -1;

}

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

{

cf = atof(&s[i]);

this->cf = cf;

}

int j = 0;

while ((s[i + 1 + j] != '\0') && (isdigit(s[i + 1 + j])))

{

j++;

}

i = i + j;

}

if (s[i] == 'x')

{

if (s[i + 1] == '^')

{

a = atof(&s[i + 2]);

int j = 0;

while ((s[i + 2 + j] != '\*') && (s[i + 2 + j] != '\0') && (s[i + 2 + j] == '+') && (s[i + 2 + j] == '-'))

{

j++;

}

i = i + j + 2;

}

else

{

a = 1;

}

}

if (s[i] == 'y')

{

if (s[i + 1] == '^')

{

b = atof(&s[i + 2]);

int j = 0;

while ((s[i + 2 + j] != '\*') && (s[i + 2 + j] != '\0') && (s[i + 2 + j] == '+') && (s[i + 2 + j] == '-'))

{

j++;

}

i = i + j + 2;

}

else

{

b = 1;

}

}

if (s[i] == 'z')

{

if (s[i + 1] == '^')

{

c = atof(&s[i + 2]);

int j = 0;

while ((s[i + 2 + j] != '\*') && (s[i + 2 + j] != '\0') && (s[i + 2 + j] == '+') && (s[i + 2 + j] == '-'))

{

j++;

}

i = i + j + 2;

}

else

{

c = 1;

}

}

}

this->abc = a \* 100 + b \* 10 + c;

}

TMonom::TMonom(double num, unsigned int num2)

{

cf = num;

abc = num2;

}

TMonom TMonom::operator + (const TMonom& monom)

{

cf = cf + monom.cf;

return \*this;

}

TMonom TMonom::operator - (const TMonom& monom)

{

cf = cf - monom.cf;

return \*this;

}

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

{

cf = monom.cf;

abc = monom.abc;

return \*this;

}

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

{

int k = 0;

if (abc != monom.abc)

k++;

if (k == 0)

return true;

else

return false;

}

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

{

int k = 0;

if (abc == monom.abc)

k++;

if (k == 0)

return true;

else

return false;

}

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

{

int k = 0;

if (abc > monom.abc)

k++;

if (k != 0)

return true;

else

return false;

}

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

{

int k = 0;

if (abc < monom.abc)

k++;

if (k != 0)

return true;

else

return false;

}

## Приложение Б. Программная реализация звена

TLink.h

#pragma once

#include "TMonom.h"

template <typename T>

class TLink

{

public:

T data;

TLink\* pNext;

TLink();

TLink(const TLink& unit);

TLink(T d, TLink\* unit= NULL);

~TLink() {}

TLink& operator = (const TLink& unit);

bool operator == (const TLink& unit) const;

bool operator != (const TLink& unit) const;

};

template <typename T>

TLink<T>::TLink()

{

pNext = NULL;

}

template <typename T>

TLink<T>::TLink(const TLink& unit)

{

data = unit.data;

pNext = unit.pNext;

}

template <typename T>

TLink<T>::TLink(T d, TLink\* unit)

{

data = d;

pNext = unit;

}

template <typename T>

TLink<T>& TLink<T>::operator=(const TLink& unit)

{

data = unit.data;

pNext = unit.pNext;

return \*this;

}

template <typename T>

bool TLink<T>::operator== (const TLink& unit) const

{

int k = 0;

if (data != unit.data)

k++;

if (pNext != unit.pNext)

k++;

if (k == 0)

return true;

else

return false;

}

template <typename T>

bool TLink<T>::operator!= (const TLink& unit) const

{

int k = 0;

if (data == unit.data)

k++;

if (pNext == unit.pNext)

k++;

if (k != 0)

return true;

else

return false;

}

## Приложение В. Программная реализация списка

TRingList.h

#pragma once

#include "TLink.h"

template <typename T>

class TRingList

{

private:

TLink <T> \*head;

TLink <T> \*current;

public:

TRingList();

TRingList(const TRingList <T> & TRingList2);

~TRingList();

TRingList<T>& operator = (const TRingList<T>& TRingList2);

void Clean();

void Insert(const T& data);

void Reset();

void Next();

bool IsEnded() const ;

TLink<T>\* GetLink();

bool operator == (const TRingList<T>& TRingList2) const;

};

template <typename T>

TRingList<T>::TRingList()

{

head = new TLink<T>();

head->pNext = head;

current = head;

}

template <typename T>

void TRingList<T>::Clean()

{

if (head->pNext != head)

{

TLink<T> \*tmp = head;

TLink<T> \*tmp2 = head->pNext;

while (tmp2 != head)

{

tmp = tmp2->pNext;

delete tmp2;

tmp2 = tmp;

}

}

}

template <typename T>

TRingList<T>::~TRingList()

{

this->Clean();

delete head;

}

template <typename T>

TRingList<T>& TRingList<T>::operator = (const TRingList<T>& TRingList2)

{

if (this != &TRingList2)

{

this->Clean();

if (TRingList2.head->pNext == TRingList2.head)

{

head = new TLink<T>();

head->pNext = head;

current = head;

}

else

{

head = new TLink<T>(TRingList2.head->data, TRingList2.head->pNext);

current = head;

TLink<T> \*tmp = TRingList2.head->pNext;

while (tmp != TRingList2.head)

{

current->pNext = new TLink<T>(tmp->data);

current = current->pNext;

tmp = tmp->pNext;

}

current->pNext = head;

}

}

return \*this;

}

template <typename T>

TRingList<T>::TRingList(const TRingList<T> & TRingList2)

{

if (TRingList2.head->pNext == TRingList2.head)

{

head = new TLink<T>();

head->pNext = head;

current = head;

}

else

{

head = new TLink<T>(TRingList2.head->data, TRingList2.head->pNext);

current = head;

TLink<T> \*tmp = TRingList2.head->pNext;

while (tmp != TRingList2.head)

{

current->pNext = new TLink<T>(tmp->data);

current = current->pNext;

tmp = tmp->pNext;

}

current->pNext = head;

}

}

template <typename T>

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

{

if (head->pNext == head)

{

current = new TLink<T>(data, head);

head->pNext = current;

}

else

{

current = head;

while (current->pNext->data > data)

{

this->Next();

}

current->pNext = new TLink<T>(data, current->pNext);

}

}

template <typename T>

void TRingList<T>::Reset()

{

current = head->pNext;

}

template <typename T>

void TRingList<T>::Next()

{

current = current->pNext;

}

template <typename T>

bool TRingList<T>::IsEnded() const

{

if (current == head)

{

return true;

}

else

{

return false;

}

}

template <typename T>

TLink<T>\* TRingList<T>::GetLink()

{

return current;

}

template <typename T>

bool TRingList<T>::operator == (const TRingList<T>& TRingList2) const

{

TRingList<T> tmp(\*this);

TRingList<T> tmp2(TRingList2);

tmp.Reset();

tmp2.Reset();

int k = 0;

while ((tmp.IsEnded() == false) && (tmp2.IsEnded() == false))

{

if (tmp.GetLink()->data != tmp2.GetLink()->data)

{

k++;

}

tmp.Next();

tmp2.Next();

}

if ((tmp.IsEnded() == false) && (tmp2.IsEnded() == true))

{

k++;

tmp.Next();

}

if ((tmp.IsEnded() == true) && (tmp2.IsEnded() == false))

{

k++;

tmp2.Next();

}

if (k == 0)

{

return true;

}

else

{

return false;

}

}

## Приложение Г. Программная реализация полинома

TPolinom.h

#pragma once

#include "TRingList.h"

class TPolinom

{

private:

TRingList<TMonom> monoms;

public:

TPolinom() {}; // конструктор

TPolinom(const string s);

TPolinom(const TPolinom &poli2);

~TPolinom() {};

TPolinom operator + (const TPolinom& poli2);

TPolinom operator - (const TPolinom& poli2);

TPolinom operator \* (const TPolinom& poli2);

TPolinom operator \* (double cf2);

friend TPolinom operator \*(double cf2, const TPolinom& poli2);

TPolinom& operator = (const TPolinom& tmp);

bool operator == (const TPolinom& poli2) const;

void PrivPod();

friend ostream& operator <<(ostream& ostr, const TPolinom& poli2);

};

TPolinom.cpp

#include "TPolinom.h"

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

{

monoms = tmp.monoms;

return \*this;

}

TPolinom::TPolinom(const string s)

{

int k = s.length();

string c;

if (k != 0)

{

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

{

if ((isdigit(s[i])) || (isalpha(s[i])) || (s[i] == '-'))

{

int j = 0;

while ((s[i + 1 + j] != '+') && (s[i + 1 + j] != '-') && ((s[i + 1 + j] != '\0')))

{

j++;

}

c = s.substr(i, j + 1);

TMonom tmp(c);

monoms.Insert(tmp);

i = i + j;

}

}

}

this->PrivPod();

}

TPolinom::TPolinom(const TPolinom &poli2)

{

monoms = poli2.monoms;

}

TPolinom TPolinom::operator + (const TPolinom& poli2)

{

TPolinom C;

TPolinom tmp(poli2);

monoms.Reset();

tmp.monoms.Reset();

while ((monoms.IsEnded() == false) && (tmp.monoms.IsEnded() == false))

{

if (monoms.GetLink()->data > tmp.monoms.GetLink()->data)

{

C.monoms.Insert(monoms.GetLink()->data);

monoms.Next();

}

else if (monoms.GetLink()->data < tmp.monoms.GetLink()->data)

{

C.monoms.Insert(tmp.monoms.GetLink()->data);

tmp.monoms.Next();

}

else if (monoms.GetLink()->data == tmp.monoms.GetLink()->data)

{

if ((monoms.GetLink()->data.cf + tmp.monoms.GetLink()->data.cf) != 0)

{

C.monoms.Insert(monoms.GetLink()->data + tmp.monoms.GetLink()->data);

}

monoms.Next();

tmp.monoms.Next();

}

}

while (monoms.IsEnded() == false)

{

C.monoms.Insert(monoms.GetLink()->data);

monoms.Next();

}

while (tmp.monoms.IsEnded() == false)

{

C.monoms.Insert(tmp.monoms.GetLink()->data);

tmp.monoms.Next();

}

return C;

}

TPolinom TPolinom::operator - (const TPolinom& poli2)

{

TPolinom C;

TPolinom tmp(poli2);

tmp = tmp \* (-1);

C = \*this + tmp;

return C;

}

TPolinom TPolinom::operator \* (const TPolinom& poli2)

{

TPolinom C;

TPolinom tmp(poli2);

monoms.Reset();

while (monoms.IsEnded() == false)

{

tmp.monoms.Reset();

while (tmp.monoms.IsEnded() == false)

{

TMonom monom;

monom.cf = tmp.monoms.GetLink()->data.cf \* monoms.GetLink()->data.cf;

if (monom.cf != 0)

{

int ABC = tmp.monoms.GetLink()->data.abc + monoms.GetLink()->data.abc;

if ((ABC / 100 < 10) && (ABC / 10 % 10 < 10) && (ABC % 10 < 10))

monom.abc = ABC;

else

throw "Error";

TPolinom tmp2;

tmp2.monoms.Insert(monom);

C = C + tmp2;

}

tmp.monoms.Next();

}

monoms.Next();

}

return C;

}

TPolinom TPolinom::operator \* (double cf2)

{

TPolinom C;

monoms.Reset();

while (monoms.IsEnded() == false)

{

monoms.GetLink()->data.cf = cf2 \* monoms.GetLink()->data.cf;

C.monoms.Insert(monoms.GetLink()->data);

monoms.Next();

}

return C;

}

TPolinom operator \*(double cf2, const TPolinom& poli2)

{

TPolinom C;

TPolinom tmp(poli2);

tmp.monoms.Reset();

while (tmp.monoms.IsEnded() == false)

{

tmp.monoms.GetLink()->data.cf = cf2 \* tmp.monoms.GetLink()->data.cf;

C.monoms.Insert(tmp.monoms.GetLink()->data);

tmp.monoms.Next();

}

return C;

}

ostream& operator <<(ostream& ostr, const TPolinom& poli2)

{

TPolinom B(poli2);

B.monoms.Reset();

if (B.monoms.IsEnded() == false)

{

while (B.monoms.IsEnded() == false)

{

if (B.monoms.GetLink()->data.cf > 0)

{

if (B.monoms.GetLink()->data.abc != 0)

{

if (B.monoms.GetLink()->data.cf == 1)

{

ostr << "+";

}

else

{

ostr << "+" << B.monoms.GetLink()->data.cf;

}

}

else

{

ostr << "+" << B.monoms.GetLink()->data.cf;

}

}

if (B.monoms.GetLink()->data.cf < 0)

{

ostr << B.monoms.GetLink()->data.cf;

}

if (B.monoms.GetLink()->data.abc != 0)

{

int a = B.monoms.GetLink()->data.abc / 100;

if (a > 1)

{

ostr << "x^" << a;

}

if (a == 1)

{

ostr << "x";

}

a = B.monoms.GetLink()->data.abc / 10 % 10;

if (a > 1)

{

ostr << "y^" << a;

}

if (a == 1)

{

ostr << "y";

}

a = B.monoms.GetLink()->data.abc % 10;

if (a > 1)

{

ostr << "z^" << a;

}

if (a == 1)

{

ostr << "z";

}

}

B.monoms.Next();

}

}

else

{

ostr << "0";

}

return ostr;

}

bool TPolinom::operator == (const TPolinom& poli2) const

{

TPolinom A(\*this);

TPolinom B(poli2);

return A.monoms == B.monoms;

}

void TPolinom::PrivPod()

{

TPolinom A(\*this);

TPolinom res;

A.monoms.Reset();

while (A.monoms.IsEnded() == false)

{

res.monoms.Reset();

if (res.monoms.IsEnded() == false)

{

while ((res.monoms.IsEnded() == false) && (res.monoms.GetLink()->data.abc != A.monoms.GetLink()->data.abc))

{

res.monoms.Next();

}

if (res.monoms.IsEnded() == true)

{

res.monoms.Insert(A.monoms.GetLink()->data);

}

if (res.monoms.GetLink()->data.abc == A.monoms.GetLink()->data.abc)

{

res.monoms.GetLink()->data.cf = res.monoms.GetLink()->data.cf + A.monoms.GetLink()->data.cf;

}

}

else

{

res.monoms.Insert(A.monoms.GetLink()->data);

}

A.monoms.Next();

}

res.monoms.Reset();

TPolinom B;

while (res.monoms.IsEnded() == false)

{

// удаляем, когда cf = 0

if (res.monoms.GetLink()->data.cf != 0)

{

B.monoms.Insert(res.monoms.GetLink()->data);

}

res.monoms.Next();

}

\*this = B;

}

## Приложение Д. Реализация основного приложения

main.cpp

#include "TPolinom.h"

#include <iostream>

#include <stdio.h>

#include <cstring>

#include <string>

using namespace std;

void main()

{

setlocale(LC\_ALL, "Russian");

TPolinom A;

TPolinom B;

TPolinom C;

string s;

string c;

int i = 1;

int j = 0;

int k;

while (i == 1)

{

cout << "Что хотите сделать с полиномами ?" << endl;

cout << "1. Сумма полиномов." << endl;

cout << "2. Разность полиномов." << endl;

cout << "3. Произведение полиномов." << endl;

cout << "4. Произведение полинома и числa." << endl;

cout << "5. Произведение числа и полинома." << endl;

cout << "Ваш выбор: ";

cin >> j;

cout << endl;

if (j == 1)

{

cout << "Введите первый полином: ";

cin.ignore();

getline(cin, s);

cout << endl;

cout << "Введите второй полином: ";

getline(cin, c);

cout << endl;

TPolinom A(s);

TPolinom B(c);

C = A + B;

cout << C;

cout << endl;

}

if (j == 2)

{

cout << "Введите первый полином: ";

cin.ignore();

getline(cin, s);

cout << endl;

cout << "Введите второй полином: ";

getline(cin, c);

cout << endl;

TPolinom A(s);

TPolinom B(c);

C = A - B;

cout << C;

cout << endl;

}

if (j == 3)

{

cout << "Введите первый полином: ";

cin.ignore();

getline(cin, s);

cout << endl;

cout << "Введите второй полином: ";

getline(cin, c);

cout << endl;

TPolinom A(s);

TPolinom B(c);

C = A \* B;

cout << C;

cout << endl;

}

if (j == 4)

{

cout << "Введите полином: ";

cin.ignore();

getline(cin, s);

cout << endl;

cout << "Введите число: ";

cin >> k;

TPolinom A(s);

C = A \* k;

cout << C;

cout << endl;

}

if (j == 5)

{

cout << "Введите число: ";

cin >> k;

cout << endl;

cout << "Введите полином: ";

cin.ignore();

getline(cin, s);

cout << endl;

TPolinom A(s);

C = k \* A;

cout << C;

cout << endl;

}

cout << endl;

cout << "Повторить работу в программе ? " << endl;

cout << "1. Да" << endl;

cout << "2. Нет" << endl;

cout << "Ваш выбор: ";

cin >> i;

cout << endl;

}

cin.get();

}