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](#_Toc513372599)

[1 Постановка задачи 4](#_Toc513372600)

[2 Руководство пользователя 5](#_Toc513372601)

[3 Руководство программиста 6](#_Toc513372602)

[3.1 Описание алгоритмов 6](#_Toc513372603)

[3.2 Описание структуры программы 6](#_Toc513372604)

[3.3 Описание структур данных 7](#_Toc513372605)

[Заключение 9](#_Toc513372606)

[Литература 10](#_Toc513372607)

[Приложения 11](#_Toc513372608)

[Приложение 1. Исходный код программы 11](#_Toc513372609)

# 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 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

## Описание алгоритмов

Вначале пользователь вводит строку. Для дальнейшего вычисление необходимо представить строку в виде полинома. Для этого задается цикл, в котором происходит обработка строки, пока не встретится +, 0 или конец строки. Так мы разбиваем строку на отдельные подстроки. Далее подстроки преобразуются в звенья. На этом этапе происходит определение коэффициента и степени звена. Каждое звено вставляется в список. Когда закончится цикл, мы получим полином, представленный в виде списка.

После этого применяется алгоритм приведения подобных слагаемых. Происходит вставка звеньев в новый пустой список. При этом происходит проверка: если уже есть звено с такой степенью (именно по степени определяется равенство звеньев), то мы не вставляемся это звено, а изменяем коэффициент у уже имеющегося звена. Далее мы записываем в новый список все звенья, у которых коэффициент не оказался равен 0. В результате имеем полином, с приведенными подобными слагаемыми.

Далее происходит выполнение операций над полиномами. Эта часть программы реализована, как слияние двух списков. Запускается цикл (пока оба списка не закончились), в котором мы сравниваем звенья: вставляем звено, которое имеет большую степень. При равенстве степеней происходит вставка звена, у которого коэффициент равен сумме коэффициентов двух сравниваемых звеньев. Отличается алгоритм произведения полиномов. Здесь дополнительно происходит проверка степеней звеньев после умножения, т.к. при перемножении мы могли получить степень больше 9, что не имеет места в нашей программе.

.

## Описание структуры 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:

2.1. TLink.h – реализация шаблонного класса звена.

2.2. TMonom.h – объявление полей и методов, которые отвечают за степень и коэффициент звена

2.3. TPolinom.h – объявление полей и методов класса полином

2.4. TRingList.h – реализация шаблонного класса список

2.5. TMonom.cpp – реализация полей и методов класса Monom

2.6. 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.

## Описание структур данных

Структура TMonom

struct TMonom

{

double cf; // коэффициент монома

unsigned int abc; // степень монов=ма

TMonom(); // конструктор

TMonom(const string s); // конструктор по строке

TMonom(double num, unsigned int num2); // констурктор с параметрами

~TMonom() {} // деструктор

TMonom operator + (const TMonom& monom); // сумма мономов

TMonom operator - (const TMonom& monom); // разность мономов

TMonom operator = (const TMonom& monom); // оператор копирования

// операторы сравнения

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

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

bool operator > (const TMonom& monom) const;

bool operator < (const TMonom& monom) const;

};

Класс TLink

template <typename T>

class TLink

{

public:

T data; // данные

TLink\* pNext; // указатель на следующее звено

TLink(); // конструктор

TLink(const TLink& unit); // конструктор копирования

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

~TLink() {} // деструктор

TLink& operator = (const TLink& unit); // оператор копирования

// операторы сравнения

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

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

};

Класс TRingList

template <typename T>

class TRingList

{

private:

TLink <T> \*head; // голова списка

TLink <T> \*current; // текущей элемент в списке

public:

TRingList(); // конструктор

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); // вставка в упорядоченный список

void Reset(); // current указывает на head

void Next(); // сurrent = current.pNext

bool IsEnded() const ; // current последний в списке ?

TLink<T>\* GetLink(); // вернуть current

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

};

Класс TPolinom

class TPolinom

{

private:

TRingList<TMonom> monoms; // список

public:

TPolinom() {}; // конструктор

TPolinom(const string s); // конструктор по строке

TPolinom(const TPolinom &poli2); // конструктор копирования

~TPolinom() {}; // деструктор

TPolinom operator + (const TPolinom& poli2); // сумма полиномов

TPolinom operator - (const TPolinom& poli2); // разность полиномов

TPolinom operator \* (const TPolinom& poli2); // произведение полиномов

TPolinom operator \* (double cf2); // произведение полинома и числа

friend TPolinom operator \*(double cf2, const TPolinom& poli2); // произведение числа и полинома

TPolinom& operator = (const TPolinom& tmp); // оператор копирвоания

bool operator == (const TPolinom& poli2) const; // оператор сравнения

void PrivPod(); // приведение подобных

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

# Приложения

## Приложение 1. Исходный код программы

TMonom.h

#pragma once

#include <iostream>

#include <stdio.h>

#include <cstring>

#include <string>

#include <cstdlib>

using namespace std;

struct TMonom

{

double cf; // коэффициент монома

unsigned int abc; // степень монов=ма

TMonom(); // конструктор

TMonom(const string s); // конструктор по строке

TMonom(double num, unsigned int num2); // констурктор

~TMonom() {} // деструктор

TMonom operator + (const TMonom& monom); // сумма мономов

TMonom operator - (const TMonom& monom); // разность мономов

TMonom operator = (const TMonom& monom); // конструткор копирования

// операторы сравнения

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

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

bool operator > (const TMonom& monom) const;

bool operator < (const TMonom& monom) const;

};

TLink.h

#pragma once

#include "TMonom.h"

template <typename T>

class TLink

{

public:

T data; // данные

TLink\* pNext; // указатель на следующее звено

TLink(); // конструктор

TLink(const TLink& unit); // конструктор копирования

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

~TLink() {} // деструктор

TLink& operator = (const TLink& unit); // оператор копирования

// операторы сравнения

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;

}

TRingList.h

#pragma once

#include "TLink.h"

template <typename T>

class TRingList

{

private:

TLink <T> \*head; // голова списка

TLink <T> \*current; // текущей элемент в списке

public:

TRingList(); // конструктор

TRingList(const TRingList <T> & TRingList2); // конструктор копирования

~TRingList(); // деструктор

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

void Clean(); // очистка

void Insert(const T& data); // вставка в упорядоченный список

void Reset(); // current указывает на head

void Next(); // сurrent = current.pNext

bool IsEnded() const ; // current последний в списке ?

TLink<T>\* GetLink(); // вернуть current

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

};

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;

}

}

TPolinom.h

#pragma once

#include "TRingList.h"

class TPolinom

{

private:

TRingList<TMonom> monoms; // список

public:

TPolinom() {}; // конструктор

TPolinom(const string s); // конструктор по строке

TPolinom(const TPolinom &poli2); // конструктор копирования

~TPolinom() {}; // деструктор

TPolinom operator + (const TPolinom& poli2); // сумма полиномов

TPolinom operator - (const TPolinom& poli2); // разность полиномов

TPolinom operator \* (const TPolinom& poli2); // произведение полиномов

TPolinom operator \* (double cf2); // произведение полинома и числа

friend TPolinom operator \*(double cf2, const TPolinom& poli2); // произведение числа и полинома

TPolinom& operator = (const TPolinom& tmp); // оператор копирвоания

bool operator == (const TPolinom& poli2) const; // оператор сравнения

void PrivPod(); // приведение подобных

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

};

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;

}

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;

}