МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по лабораторной работе

**«Арифметические операции с полиномами»**

**Выполнил:** студент группы 381603-1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Вятчанин Р.О

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

Нижний Новгород

2018

Содержание

Введение 3

1 Постановка задачи 4

2 Руководство пользователя 5

3 Руководство программиста 6

3.1 Описание структура программы 6

3.2 Описание структуры данных 6

3.3 Программная реализация 6

3.3.1 Класс Node 6

3.3.2 Класс Monom 6

3.3.3 Класс CList 7

3.3.4 Класс Polynom 7

Заключение 9

Литература 10

Приложение 11

Приложение А. Программная реализация списка с головой 11

Приложение Б. Программная реализация монома и звена 13

Приложение В. Программная реализация полинома 14

Приложение Г. Основная программа 18

# Введение

Данная лабораторная работа посвящена разработке программы, реализующей операции над полиномами (сложение, вычитание, умножение и умножение на константу) с использованием циклического списка с головой. Данная программа помогает в вычислениях с полиномами, а также может быть использована в реализации других программ. Реализация подобных операций упрощает сложные вычисления и повышает их точность. В отчете описываются алгоритмы, использующиеся в программе, пример её использования, а также сам код программы, выполняющей поставленную задачу.

# Постановка задачи

Цель:

Разработать программу, выполняющую арифметические операции над полиномами от трех переменных (x, y, z). Полином представляет собой сумму мономов, степень каждого из которых от 0 до 9. Коэффициентами являются вещественные числа. Допустимые операции: сложение, вычитание, умножение двух полиномов, умножение полинома на константу.

Пример:

Поступающие данные: 2 строки, содержащие 2 полинома.

Результат: Программа корректно выводит результат арифметических операций над этими полиномами.

|  |
| --- |
|  |
| Рис.1. Пример работы программы. |

# Руководство пользователя

Для начала работы необходимо запустить файл [**sample.exe**](file:///C:\Users\HP\Desktop\build\lab1_polinom\Lab1_polinom.sln). На экране появится надпись о просьбе ввести первый полином, затем нужно выбрать одну из четырех представленных операций, после ввести второй полином или константу, если введено корректно, то появится результат. После пользователю будет предложено выйти из программы или продолжить вычисления. Для продолжения работы с приложением надо нажать 2, для выхода 1.

|  |
| --- |
|  |
| Рис.2. Ввод первого полинома, выбор операции, ввод второго полинома. |

# Руководство программиста

## Структура программы

Программа состоит из 4 проектов:

1. gtest

Содержит в себе:

[gtest.h](file:///C:\Users\HP\Desktop\gtest\gtest.h) и [gtest-all.cpp](file:///C:\Users\HP\Desktop\gtest\gtest-all.cc). Файлы содержат Google тесты, необходимые для проверки корректности работы реализованных классов.

1. postfix\_lib

Содержит в себе:

[Node.h](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\include\Node.h)- заголовочный файл, содержbn объявление шаблонного класса Node и его реализацию.

[Monom.h](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\include\Monom.h)– заголовочный файл, содержит объявление класса Monom.

[Polinom.h](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\include\Polinom.h)– заголовочный файл, содержит объявление шаблонного класса Polynom

[List.h](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\include\List.h)– заголовочный файл, содержит объявление шаблонного класса CList и его реализацию.

[Monom.cpp](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\src\Monom.cpp)– содержит реализацию класса Monom

[Polinom.cpp](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\src\Polinom.cpp)-содержит реализацию класса Polynom.

1. Tests

Содержит в себе:

[test\_list.cpp](file:///C:\Users\HP\Desktop\test\Test_main.cpp)– реализация тестов для проверки корректности работы класса CList.

[test\_main.cpp](file:///C:\Users\HP\Desktop\test\test_polinoms.cpp)– содержит функцию main(), запускающую Google тесты.

[test\_polynoms.cpp](file:///C:\Users\HP\Desktop\test\test_list.cpp)– реализация тестов для проверки корректности работы класса Polynom.

1. sample

Содержит в себе:

[Main.cpp](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\samples\Main.cpp)– реализация пользовательского интерфейса.

## Структуры данных

### Шаблон класса циклический список

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

Для его реализации его реализации используется шаблонный класс Node-звено списка. Список представляет собой структуру с двумя полями:

-указатель на голову звена.

-указатель на текущее звено.

Описание алгоритмов:

**Вставка в упорядоченный лист.**

На вход передаем данные, для хранения в новом звене списка.

Алгоритм:

1.Проверяем на пустоту список, если он пустой, то создаем новое звено, которое следует следом за головой, в свою очередь это звено указывает на голову, так как список циклический.

2.Если список не пустой, то создаем 2 указателя на звено, curr присваиваем голову, tmp звено с поступающими на вход данными.

3. До тех пор, пока данные следующего после curr звена меьше входных и список не кончился смещаем curr.

4.Вставляем новое звено после найденного curr.

**Вставка в конец листа.**

На вход передаем данные, для хранения в новом звене списка.

Алгоритм:

1.Делаем указатель на начало списка.

2.До тех пор, пока список не кончился смещаем curr.

3.Создаем указатель на звено и присваиваем ему поступившие на вход данные и зацикливаем список.

### Класс полином

Класс полином является решением данной лабораторной работы. С его помощью реализуется хранение полиномов и операции над ними. У класса одно поле:

-циклический список мономов.

Описание алгоритмов:

**Сложение полиномов**

На вход поступают два полином. Возвращаем результат сложения.

Алгоритм:

1. Создаем полином res для записи результатов сложения.

1. Сдвигаем указатели обоих списков и списка-результата в начало.

2.До тех пор, пока оба полинома не закончатся, сравниваем звенья списка. Если звено одного полинома меньше, то вставляем в конец списка-результата и сдвигаем указатели списка, из которого взяли элемент и списка-результата на одно звено вперёд, аналогично с обратным результатом. В случае если звенья равны, создаём моном, коэффициент которого равен сумме коэффициентов мономов, лежащих в этих звеньях, приводя подобные, и записываем его в конец списка-результата, сдвигаем указатели обоих списков и списка-результата на одно звено вперёд.

3.Если один из списков кончился, записываем остаток второго списка в список-результат.

4.Возвращаем res.

**Разность полиномов**

Реализуется с помощью операции сложения полиномов. Вычитаемый полином умножается на (-1) и складывается с другим полиномом. Возвращаем результат сложения.

**Умножение на константу**

На вход поступает полином и константа. Возвращаем результат умножения.

Алгоритм:

1.Создаем полином-результат res, равный исходному.

3.Сдвигаем указатель res на начало списка.

4.До тех пор, пока полином res не закончится умножаем коэффициент монома, лежащего в текущем звене, на переданную константу и меняем указатель на следующее звено.

5.Возвращаем res.

**Умножение полиномов**

На вход поступает два полинома. Возвращаем результат умножения.

Алгоритм:

1. Сдвигаем указатели обоих списков на звенья, следующие за головой.

2.До тех пор, пока в первый список не закончился, создаём полином-копию второго полинома pol3 и сдвигаем в нём указатель на начало списка, пока он не закончился.

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

5.Иначе выводим сообщение об ошибке "Error.".

6.Сдвигаем указатель в pol3 на следующее звено.

7.Записываем в res сумму самого res и pol3.

8.Сдвигаем указатель res на следующее звено.

9.Возвращаем res.

## Программная реализация

### Класс Node

Поля:

DataType data – данные в звене.

Node<DataType>\* next - указатель на следующее звено.

Методы:

Node() { next = NULL;} –конструктор, создает пустое звено.

Node(DataType d,Node<DataType>\* \_next=NULL) { data = d; next = \_next;}-конструктор с параметром, данные звена, указатель на следующее звено.

bool operator< (const Node& n) const { return (data < n.data);}-перегрузка операции меньше, сравнивает данные звеньев.

bool operator> (const Node& n) const { return (data > n.data);}-перегрузка операции больше, сравнивает данные звеньев.

### Класс Monom

Поля:

double cf - коэффициент монома.

int pw - степень монома.

Методы:

Monom(const double c = 0.0,const int p = 0) - конструктор с параметрами, коэффициент и степень.

Monom(const Monom& m)- конструктор копирования.

Monom& operator=(const Monom& m)- перегрузка оператора присваивания.

Перегрузка операторов меньше и больше, сравнение степени и коэффициентов.

bool operator< (const Monom& m) const

bool operator> (const monom& m) const

Перегрузка операторов сравнения.

bool operator==(const Monom& m) const

bool operator!=(const Monom& m) const

### Класс CList

Поля:

Node<DataType>\* head - указатель на голову списка.

Node<DataType>\* curr - указатель на текущий элемент.

Методы:

CList()- конструктор по умолчанию.

~CList()– деструктор.

void Clean()- очистка списка.

CList(const CList<DataType>& l)- конструктор копирования.

Const CList<DataType>& operator=(const CList<Datatype>& l)-перегрузка оператора присваивания.

void InsertСДшые(const T& d)- вставка элемента в упорядоченный список.

void InsertToTail(const DataType d)- вставка в конец списка.

Перегрузка операторов сравнения:

bool operator==(const CList<DataType>& l) const

bool operator!=(const CList<DataType>& l) const

Методы навигации:

void First()- сдвиг указателя в начало списка.

void Change()- переход на следующее звено.

DataType& GetCurr() const- получить текущий элемент.

bool End() const- проверка на конец списка.

Bool Empty() const- проверка на пустоту списка.

bool Full() const- проверка на полноту списка.

### Класс Polinom

Поля:

CList<Monom> list - список из мономов.

Методы:

CList<Monom> Bringing\_Sim(CList<Monom>& m)- приведение подобных в полиноме.

Polynom(const Polynom& p) - конструктор копирования.

Polynom(const string s = "") – конструктор по строке.

Polynom operator+(const Polynom& p)const-перегрузка оператора сложения.

Polynom operator\*(const Polynom& p)const- перегрузка оператора умножения.

Polinom operator\*(const double k)const- перегрузка оператора умножения на константу слева.

friend Polynom operator\*(const double a, const Polynom& p)- умножение на константу справа.

Polynom operator-(const Polynom& p) const- перегрузка оператора вычитания.

Polynom operator-() const- унарный минус.

Polynom& operator=(const Polynom& p)- перегрузка оператора присваивания.

friend ostream& operator<<(std::ostream& out, const Polynom& p)- перегрузка оператора вывода.

Перегрузка операторов сравнения:

bool operator==(const Polynom& p) const

bool operator!=(const Polynom& p) const

# Заключение

В лабораторной работе была разработана система для арифметических действий над полиномами с помощью циклического односвязного списка с головой и класса полиномов. Программа может проводить операции с полиномами: сложение, вычитание, умножение на константу и перемножение двух полиномов.

# Литература

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, Москва 2013
2. Гергель В.П. Рабочие материалы к учебному курсу «Методы программирования». 2002 г.

# Приложение

## Приложение А. Программная реализация списка с головой

CList.h

#pragma once

#include "Node.h"

#include <iostream>

using namespace std;

template<class DataType>

class CList

{

private:

Node<DataType>\* head;

Node<DataType>\* curr;

public:

CList();

~CList();

void Clean();

CList(const CList<DataType>& l);

const CList<DataType>& operator=(const CList<DataType>& l);

void InsertCList(const DataType& d);

void InsertToTail(const DataType d);

bool operator==(const CList<DataType>& l) const;

bool operator!=(const CList<DataType>& l) const;

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

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

bool End() const { return curr == head; }

bool Empty() const { return (head->next == head); }

bool Full() const { return(head->next != head); }

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

};

template <class DataType>

void CList<DataType>::Clean()

{

Node<DataType>\* curr = head->next;

while (curr != head)

{

Node<DataType>\* tmp = curr->next;

delete curr;

curr = tmp;

}

head->next = head;

}

template <class DataType>

CList<DataType>::CList()

{

head = new Node<DataType>();

curr = head->next;

head->next = head;

}

template <class DataType>

CList<DataType>::~CList()

{

Clean();

delete head;

}

template <class DataType>

CList<DataType>::CList(const CList<DataType>& l)

{

head = new Node<DataType>;

Node<DataType>\* A = l.head;

Node<DataType>\* B = head;

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

{

head->next = head;

return;

}

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

{

A = A->next;

B->next = new Node<DataType>(A->data);

B = B->next;

}

B->next = head;

curr = head->next;

}

template <class DataType>

const CList<DataType>& CList<DataType>::operator=(const CList<DataType>& l)

{

Clean();

Node<DataType>\* tmp = l.head;

Node<DataType>\* tmp2 = head;

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

{

tmp = tmp->next;

tmp2->next = new Node<DataType>(tmp->data);

tmp2 = tmp2->next;

}

tmp2->next = head;

curr = head->next;

return \*this;

}

template <class DataType>

void CList<DataType>::InsertCList(const DataType& d)

{

if (Empty())

{

head->next = new Node<DataType>(d);

head->next->next = head;

}

else

{

Node<DataType>\* tmp = new Node<DataType>(d);

Node<DataType>\* curr = head;

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

curr = curr->next;

Node<DataType>\* tmp2 = curr->next;

curr->next = tmp;

curr->next->next = tmp2;

}

}

template<class DataType>

inline void CList<DataType>::InsertToTail(const DataType d)

{

First();

while (curr->next != head)

{

Change();

}

Node<DataType>\* tmp = curr->next;

curr->next = new Node<DataType>(d);

curr->next->next = tmp;

}

template<class DataType>

bool CList<DataType>::operator==(const CList<DataType>& l) const

{

bool k = true;

if (this != &l)

{

Node<DataType>\* tmp = l.head->next;

Node<DataType>\* tmp2 = head->next;

while (tmp->data == tmp2->data && tmp2 != head && tmp != l.head)

{

tmp = tmp->next;

tmp2 = tmp2->next;

}

if (tmp != l.head || tmp2 != head)

k = false;

}

return k;

}

template<class DataType>

inline bool CList<DataType>::operator!=(const CList<DataType>& l) const

{

return !(\*this == l);

## }

## Приложение Б. Программная реализация монома и звена

Monom.h

#pragma once

using namespace std;

class Monom

{

public:

double cf;

int pw;

Monom(const double c = 0.0, const int p = 0);

Monom(const Monom& m);

bool operator==(const Monom& m)const;

bool operator!=(const Monom& m)const;

bool operator< (const Monom& m)const;

bool operator> (const Monom& m)const;

const Monom& operator= (const Monom& m);

};

Monom.cpp

#include "Monom.h"

Monom::Monom(const double c, const int p)

{

cf = c;

pw = p;

}

Monom::Monom(const Monom & m)

{

cf = m.cf;

pw = m.pw;

}

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

{

return ((pw == m.pw) && (cf == m.cf));

}

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

{

return !(\*this == m);

}

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

{

return (pw<m.pw);

}

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

{

return (pw>m.pw);

}

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

{

cf = m.cf;

pw = m.pw;

return \*this;

## }Приложение В. Программная реализация полинома

Polynom.h

#pragma once

#include "CList.h"

#include"Monom.h"

#include <algorithm>

#include <iostream>

#include <string>

class Polynom

{

private:

CList<Monom> list;

CList<Monom> Bringing\_Sim(CList<Monom>& m);

public:

Polynom(const Polynom& p);

Polynom(const string s = "");

Polynom operator+(const Polynom& p) const;

Polynom operator\*(const Polynom& p) const;

Polynom operator\*(const double k) const;

Polynom operator-(const Polynom& p) const;

Polynom operator-() const;

const Polynom& operator=(const Polynom &p);

friend Polynom operator\*(const double k, const Polynom& p);

bool operator==(const Polynom& p) const;

bool operator!=(const Polynom& p) const;

friend std::ostream& operator<<(std::ostream& out, const Polynom& pol);

};

Polynom.cpp

#include "Polynom.h"

using namespace std;

Polynom::Polynom(const Polynom & p)

{

list = p.list;

}

CList<Monom> Polynom::Bringing\_Sim(CList<Monom>& m)

{

CList<Monom> l;

l.First();

m.First();

Node<Monom> monom(m.GetCurr().cf);

CList<Monom> y(m);

while (!m.End())

{

monom.data.pw = m.GetCurr().pw;

y.Change();

if (m.GetCurr().pw == y.GetCurr().pw && (y.GetCurr().cf || y.GetCurr().pw))

monom.data.cf += y.GetCurr().cf;

else

{

if (monom.data.cf)

{

l.InsertToTail(monom.data);

l.Change();

}

monom.data.cf = y.GetCurr().cf;

}

m.Change();

}

return l;

}

Polynom::Polynom(string s)

{

CList<Monom> res;

while (s.length())

{

string str;

Monom m;

int p = 1;

while (p < s.length() && s[p] != '+' && s[p] != '-')

p++;

str = s.substr(0, p);

s.erase(0, p);

p = 0;

while (str[p] != 'x' && str[p] != 'y' && str[p] != 'z' && p < str.length())

p++;

string c = str.substr(0, p);

if (c == "+" || c.length() == 0)

m.cf = 1;

else

if (c == "-")

m.cf = -1;

else

m.cf = stod(c);

str.erase(0, p);

str += ' ';

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

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

{

p = str.find((char)(120 + i));

if (p > -1)

{

if (str[p + 1] != '^')

str.insert(p + 1, "^1");

m.pw += a[i] \* stoi(str.substr(p + 2, 1));

str.erase(p, 3);

}

}

list.InsertCList(m);

}

list = Bringing\_Sim(list);

}

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

{

Polynom res;

Polynom pol1 = \*this;

Polynom pol2 = p;

res.list.First();

pol1.list.First();

pol2.list.First();

while (!pol1.list.End() && !pol2.list.End())

{

if (pol1.list.GetCurr() < pol2.list.GetCurr())

{

res.list.InsertToTail(pol1.list.GetCurr());

pol1.list.Change();

res.list.Change();

}

else if (pol1.list.GetCurr() > pol2.list.GetCurr())

{

res.list.InsertToTail(pol2.list.GetCurr());

pol2.list.Change();

res.list.Change();

}

else

{

double c = pol1.list.GetCurr().cf + pol2.list.GetCurr().cf;

if (c)

{

res.list.InsertToTail(Monom(c, pol1.list.GetCurr().pw));

res.list.Change();

}

pol1.list.Change();

pol2.list.Change();

}

}

while (!pol1.list.End())

{

res.list.InsertToTail(pol1.list.GetCurr());

pol1.list.Change();

res.list.Change();

}

while (!pol2.list.End())

{

res.list.InsertToTail(pol2.list.GetCurr());

pol2.list.Change();

res.list.Change();

}

return res;

}

Polynom Polynom::operator\*(const double k) const

{

Polynom res;

res = \*this;

res.list.First();

while (!res.list.End())

{

res.list.GetCurr().cf \*= k;

res.list.Change();

}

return res;

}

Polynom Polynom::operator-(const Polynom & p) const

{

return \*this + p\*(-1);

}

Polynom Polynom::operator-() const

{

return (-1)\*(\*this);

}

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

{

list = p.list;

return \*this;

}

bool Polynom::operator==(const Polynom & p) const

{

return list == p.list;

}

bool Polynom::operator!=(const Polynom & p) const

{

return list != p.list;

}

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

{

Polynom res;

Polynom pol = \*this;

Polynom pol2 = p;

pol.list.First();

pol2.list.First();

while (!pol.list.End())

{

double pol\_cf = pol.list.GetCurr().cf;

int pol\_pw = pol.list.GetCurr().pw;

Polynom pol3(p);

pol3.list.First();

while (!pol3.list.End())

{

int pol3\_pw = pol3.list.GetCurr().pw;

if ((pol3\_pw % 10 + pol\_pw % 10) < 10 && (pol3\_pw / 10 % 10 + pol\_pw / 10 % 10) < 10 && (pol3\_pw / 100 + pol\_pw / 100) < 10)

{

pol3.list.GetCurr().pw += pol\_pw;

pol3.list.GetCurr().cf \*= pol\_cf;

}

else

throw "Error.";

pol3.list.Change();

}

res = res + pol3;

pol.list.Change();

}

return res;

}

Polynom operator\*(const double k, const Polynom & p)

{

return p\*k;

}

ostream& operator<<(ostream &out, const Polynom& pol)

{

Polynom p = pol;

p.list.First();

while (!p.list.End())

{

Monom temp = p.list.GetCurr();

if (temp.cf > 0)

{

out << "+";

if (temp.cf == 1 && temp.pw == 0)

out << "1";

else

if (temp.cf != 1)

out << temp.cf;

}

else

out << temp.cf;

int a = temp.pw / 100;

if (a>1)

out << "x^" << a;

else

if (a == 1)

out << "x";

a = temp.pw / 10 % 10;

if (a>1)

out << "y^" << a;

else

if (a == 1)

out << "y";

a = temp.pw % 10;

if (a>1)

out << "z^" << a;

else

if (a == 1)

out << "z";

p.list.Change();

}

return out;

## }

## Приложение Г. Основная программа

void main()

{

string s1;

string s2;

int q = 2;

int k;

while (q == 2)

{

cout << "Write one polynom." << endl;

cin >> s1;

Polynom P1(s1);

cout << "Choose." << endl;

cout << "1) +" << endl;

cout << "2) -" << endl;

cout << "3) \*" << endl;

cout << "4) \* d" << endl;

cin >> k;

switch (k)

{

case 1:

{

cout << "Write second polynom." << endl;

cin >> s2;

Polynom P2(s2);

cout << "Result:" << P1 + P2;

break;

}

case 2:

{

cout << "Write second polynom." << endl;

cin >> s2;

Polynom P2(s2);

cout << "Result:" << P1 - P2;

break;

}

case 3:

{

cout << "Write second polynom." << endl;

cin >> s2;

Polynom P2(s2);

cout << "Result:" << P1 \* P2;

break;

}

case 4:

{

double d;

cout << " Write d" << endl;

cin >> d;

cout << "Result:" << P1\*d;

break;

}

}

cout << "\n Exit: " << endl;

cout << " 1)Yes. " << endl;

cout << " 2)No. " << endl;

cin >> q;

}

}