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 учебной практике

**Арифметические операции с полиномами**

**Выполнил:** студент группы 381603-1

Репин В.И.

**Проверила:** к.т.н.,доцент каф. МОСТ института ИТММ

Кустикова В.Д.

Нижний Новгород  
2018

**Содержание**

[Введение 3](#_Toc514170583)

[1 Постановка задачи 4](#_Toc514170584)

[2 Руководство пользователя 5](#_Toc514170585)

[3 Руководство программиста 6](#_Toc514170586)

[3.1 Описание структуры программы 6](#_Toc514170587)

[3.2 Описание структуры данных 7](#_Toc514170588)

[3.3 Описание программной реализации 9](#_Toc514170589)

[Заключение 12](#_Toc514170590)

[Список литературы 13](#_Toc514170591)

[Приложение 14](#_Toc514170592)

[Приложение А. Программная реализация списка с головой 14](#_Toc514170593)

[Приложение Б. Программная реализация полиномов 16](#_Toc514170594)

[Приложение В. Программная реализация приложения 19](#_Toc514170595)

# Введение

Полиномы являются одними из простейших элементарных функций. Множество задач в математике сводится к работе с полиномами, в том числе приближение ими более сложных классов функций. Разумеется, имеется смысл в упрощении арифметических вычислений с полиномами. Для выполнения такого рода операций с помощью компьютера необходимо в некотором виде хранить и обрабатывать их. Чтобы такое было возможно, необходимо разработать некоторую структуру данных, позволяющую удобным способом это делать.

В отчёте приводится постановка задачи, разработанные алгоритмы и структуры данных, описание программы и правила её использования, а также прилагается код программы, решающей поставленную задачу.

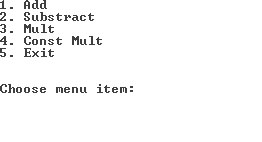
# Постановка задачи

Разработать программу, выполняющую арифметические операции с полиномами трех переменных: сложение, вычитание, умножение на константу, умножение двух полиномов. Считается, что полином составлен из мономов от трех переменных со степенью от 0 до 9. Коэффициенты полинома – вещественные числа. Работоспособность программы необходимо проверить с помощью Google Tests Framework. Кроме того, необходимо разработать пользовательское консольное приложение.

# Руководство пользователя

Для работы с программой необходимо запустить на исполнение файл ui1.exe.

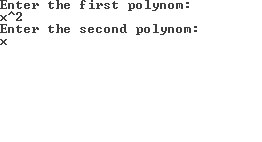
После запуска на экране появится главное меню программы. (рис. 1)



1. Главное меню программы

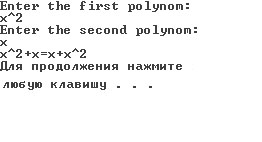
Пункты 1–4 позволяют выполнять соответственно сложение, вычитание, умножение, и умножение на константу полиномов. Для выхода из программы необходимо выбрать 5 пункт меню.

После выбора одного из пунктов 1–4 пользователю предлагается ввести два полинома-операнда (полином и константу, в случае 4 пункта). Предполагается, что полиномы состоят из мономов трёх переменных x,y.z с целой степенью от 0 до 9. В качестве примера приведён пункт 1. (рис. 2)



1. Ввод полиномов для операции сложения

Появится результат выполненной операции и нажатием любой клавиши можно будет вернуться в главное меню. (рис. 3)



1. Результат сложения заданных полиномов

# Руководство программиста

## Описание структуры программы

Описание проектов и файлов, составляющих исходный код программы

RingList – проект, содержащий реализацию шаблонного класса “циклический список с “головой”:

node.h – заголовочный файл, содержащий объявление и реализацию шаблонной структуры “Узел списка”.

list.h – заголовочный файл, содержащий объявление и реализацию шаблонного класса списка.

list.cpp – файл исходного кода класса списка (пустой).

Polynom – проект, содержащий реализацию класса “полином”:

monom.h– заголовочный файл, содержащий объявление класса “моном”.

polynom.h – заголовочный файл, содержащий объявление класса “полином”.

polynom.cpp – файл исходного кода класса полином.

gtest – проект, содержащий Google Tests Framework:

gtest.h – заголовочный файл для Google Tests Framework.

gtest-all.cc – файл с исходным кодом Google Tests Framework.

ui1 – проект, содержащий реализацию консольного приложения:

main\_ui.cpp – файл исходного кода консольного приложения.

tests – проект, содержащий тесты для классов “циклический список с “головой” и “полином”:

test\_main.cpp – файл для запуска тестов с помощью Google Test Framework

test\_ringlist.cpp – файл, содержащий тесты для класса “циклический список с “головой”.

test\_polynom.cpp – файл, содержащий тесты для класса “полином”.

## Описание структур данных

### Шаблон класса циклический список с «головой»

Поля класса:

* указатель на «голову» списка;
* указатель на текущий элемент списка.

Описание алгоритмов:

*Упорядоченная вставка в циклический список с «головой»*

Вход: данные, для хранения в новом узле списка.

Алгоритм:

1. Создать указатель curr, установить на голову списка;
2. Создать указатель temp, выделяя память под новый узел;
3. До тех пор пока данные следующего после curr узла меньше входных и список не кончился смещать curr;
4. Вставить узел temp после curr.

Выход: нет.

### Класс полином

Поля класса:

* список из мономов.

Описание алгоритмов:

*Разбиение строки на мономы*

Вход: строка, полином в текстовом виде.

Алгоритм:

1. Найти в данной строке “+”, “-” или конец. Если фрагмент до этого ненулевой, то выделить его как подстроку-моном part, иначе – закончить алгоритм;
2. Найти в part “x”, “y” или “z” и выделить символы до него как коэффициент. Если отсутствуют или “+”, считать коэффициент единице, если “-”, то минус единице;
3. Если коэффициент не равен нулю, то поочерёдно найти “x”, “y”, и “z” и выбрать соответствующие степени после знака “^”, если он отсутствует, то считать степень равной единице;
4. Сформированный моном вставить в упорядоченный список мономов res;
5. Если строка не кончилась, то выполнить пункт 1.

Выход: упорядоченный список из мономов.

*Сложение полиномов*

Вход: два полинома.

Алгоритм:

1. Создать результирующий полином res;
2. Установить указатели на начала списков мономов;
3. Сравнить степени текущих мономов;
4. Если степени не равны, то моном с наибольшей степенью вставить в res и сместить указатель в нём;
5. Если степени равны, то записать в res моном с этой степенью и суммой коэффициентов. В обоих списках перейти к следующему моному;
6. Если ни один из мономов не закончился, то перейти к пункту 2;
7. Мономы из оставшегося списка вставить в res.

Выход: приведённый полином res, являющийся суммой входных.

*Умножение полиномов*

Вход: два полинома (left,right).

Алгоритм:

1. Создать результирующий полином res;
2. Установить указатели на начало списка мономов right;
3. Скопировать полином left во временный полином temp;
4. Умножить каждый моном temp на текущий моном right, затем сместить указатель в нём;
5. Сложить полиномы res и temp (таким образом произойдёт приведение);
6. Вернуться к пункту 3, если список мономов right не кончился.

Выход: приведённый полином res, являющийся произведением входных.

## Описание программной реализации

### Структура node

node – шаблонная структура, является узлом списка, с параметром dtype – тип данных, хранящийся в узле.

*Поля:*

dtype data – данные в узле

node\* next – указатель на следующий узел

*Методы:*

node() – конструктор по умолчанию

node(dtype in, node<dtype>\* nxt = NULL) – конструктор с параметрами

bool operator< (const node& n) – оператор сравнения

bool operator> (const node& n) – оператор сравнения

### Класс list

list – шаблонный класс, циклический список с “головой”, с параметром dtype – тип данных, хранящийся в списке

*Поля:*

node<dtype>\* head – указатель на “голову” списка

node<dtype>\* curr – указатель на текущий элемент

*Методы:*

list() – констуктор по умолчанию

~list() – деструктор

void Clean() – очистка списка

list(const list<dtype>& src) – конструктор копирования

const list<dtype>& operator=(const list<dtype>& src) – оператор присваивания

void InsertInOrder(dtype elem) – вставка в упорядоченный список

void InsertAfter(node<dtype>\* nd, const dtype& d) – вставка после указанного элемента

bool operator==(const list<dtype>& rlst) const – оператор сравнения

bool operator!=(const list<dtype>& rlst) const – оператор сравнения

void Reset() – установка на начало списка

void Move() – сдвиг указателя на следующий узел

bool IsEnded() const – проверка на окончания списка

node<dtype>\* GetCurr() const – получить указатель на текущий узел

### Класс monom

monom –класс, представляет собой моном, из которых составлен полином

*Поля:*

double cf – коэффициент перед мономом

int abc – свёрнутая степень

*Методы:*

monom(double icf = 0, unsigned int iabc = 0) – конструктор с параметром(по умолчанию)

const monom& operator=(const monom& src) – оператор присваивания

bool operator< – оператор сравнения

bool operator> – оператор сравнения

bool operator== – оператор сравнения

bool operator!= – оператор сравнения

### Класс polynom

polynom –класс, представляет собой полином

*Поля:*

list<monom> pmlist – список из мономов

*Методы:*

static list<monom> CreateInstance(std::string ipm = "") – создание экземпляра полинома по строке

polynom(const polynom& src) – конструктор копирования

const polynom& operator=(const polynom &src) – оператор присваивания

friend std::ostream& operator<<(std::ostream &ostr, const polynom &pm) – оператор вставки в поток

std::string str() const – получить строковое представление полинома

friend polynom operator\*(const double mp, const polynom& pol) – оператор умножения константы на полином

polynom operator-(const polynom& pml) const – бинарный минус

polynom operator-() const – унарный минус

polynom(list<monom> &inlist) – конструктор по списку

bool operator==(const polynom& pml) const – оператор сравнения

bool operator!=(const polynom& pml) const – оператор сравнения

polynom(std::string ipm = "") – конструктор по строке

polynom operator+(const polynom& pml) const – оператор сложения

polynom operator\*(const polynom& pml) const – оператор умножения полинома на полином

polynom operator\*(const double mp) const – оператор умножения полинома на константу

# Заключение

В ходе учебной практики была разработана программа, полностью удовлетворяющая поставленным требованиям в смысле функциональности. Получившееся консольное приложение обладает дружественным интерфейсом, работоспособность всех его функций была протестирована с помощью Google Test Framework.

# Список литературы

1. Кормен Т., Лейзерсон Ч., Риверст Р., Штайн К. Алгоритмы. Построение и анализ. – М.: Издательский дом "Вильямс". – 2005. – 1290с.
2. Гергель В.П. и др. Методы программирования. Учебное пособие. – Н.Новгород.: ННГУ. – 2016. – 211с.
3. Страуструп Б. Язык программирования С++. – М.:Бином. – 2001. – 1099с.

# Приложение

## Приложение А. Программная реализация списка с головой

**node.h**

#pragma once

template<class dtype>

struct node

{

public:

dtype data;

node\* next;

bool operator< (const node& n) { return (data<n.data); }

bool operator> (const node& n) { return (data>n.data); }

node(dtype in, node<dtype>\* nxt = NULL) { data = in; next = nxt; }

node() { next = NULL; }

};

**list.h**

#pragma once

#include "node.h"

template<class dtype>

class list

{

private:

node<dtype>\* head;

node<dtype>\* curr;

public:

list();

~list();

void Clean();

list(const list<dtype>& src);

const list<dtype>& operator=(const list<dtype>& src);

void InsertInOrder(dtype elem);

void InsertAfter(node<dtype>\* nd, const dtype& d);

bool operator==(const list<dtype>& rlst) const;

bool operator!=(const list<dtype>& rlst) const { return !(\*this == rlst); }

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

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

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

node<dtype>\* GetCurr() const { return curr; }

};

template <class dtype>

void list<dtype>::Clean()

{

node<dtype>\* curr = head->next;

while (curr != head)

{

node<dtype>\* temp = curr->next;

delete curr;

curr = temp;

}

head->next = head;

}

template <class dtype>

list<dtype>::list()

{

head = new node<dtype>;

head->next = head;

curr = head->next;

}

template <class dtype>

list<dtype>::~list()

{

Clean();

delete head;

}

template <class dtype>

list<dtype>::list(const list<dtype>& src)

{

head = new node<dtype>;

head->next = head;

curr = head->next;

node<dtype>\* psrc = src.head;

node<dtype>\* pcur = head;

while (psrc->next != src.head)

{

psrc = psrc->next;

pcur->next = new node<dtype>(psrc->data);

pcur = pcur->next;

}

pcur->next = head;

curr = head->next;

}

template <class dtype>

const list<dtype>& list<dtype>::operator=(const list<dtype>& src)

{

Clean();

node<dtype>\* psrc = src.head;

node<dtype>\* pcur = head;

while (psrc->next != src.head)

{

psrc = psrc->next;

pcur->next = new node<dtype>(psrc->data);

pcur = pcur->next;

}

pcur->next = head;

curr = head->next;

return \*this;

}

template <class dtype>

void list<dtype>::InsertInOrder(dtype elem)

{

node<dtype>\* temp = new node<dtype>(elem);

node<dtype>\* curr = head;

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

curr = curr->next;

node<dtype>\* pn = curr->next;

curr->next = temp;

curr->next->next = pn;

}

template <class dtype>

void list<dtype>::InsertAfter(node<dtype>\* nd, const dtype& d)

{

node<dtype>\* temp = nd->next;

nd->next = new node<dtype>(d, temp);

}

template<class dtype>

bool list<dtype>::operator==(const list<dtype>& rlst) const

{

bool res = true;

if (this != &rlst)

{

node<dtype>\* pr = rlst.head->next;

node<dtype>\* pl = head->next;

while (pr->data == pl->data && pl != head && pr != rlst.head)

{

pr = pr->next;

pl = pl->next;

}

if (pl != head || pr != rlst.head)

res = false;

}

return res;

}

## Приложение Б. Программная реализация полиномов

**monom.h**

#pragma once

#include <cmath>

#define EPS 1e-15

class monom

{

public:

double cf;

int abc;

bool operator< (const monom& m) { return (abc<m.abc); }

bool operator> (const monom& m) { return (abc>m.abc); }

bool operator==(const monom& m) { return (abs(abc - m.abc) < EPS && cf == m.cf); }

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

monom(double icf = 0, unsigned int iabc = 0) { cf = icf; abc = iabc; }

const monom& operator=(const monom& src) { cf = src.cf; abc = src.abc; return \*this; }};

**polynom.h**

#pragma once

#include <iostream>

#include <sstream>

#include <string>

#include <algorithm>

#include "monom.h"

#include "list.h"

#define OFFSET 120

class polynom

{

private:

list<monom> pmlist;

static list<monom> CreateInstance(std::string ipm = "");

public:

polynom(const polynom& src) : pmlist(src.pmlist) {}

const polynom& operator=(const polynom &src) { pmlist = src.pmlist; return \*this; }

friend std::ostream& operator<<(std::ostream &ostr, const polynom &pm);

std::string str() const;

friend polynom operator\*(const double mp, const polynom& pol) { return pol\*mp; } polynom operator-(const polynom& pml) const { return \*this + pml\*(-1); }

polynom operator-() const { return (-1)\*(\*this); }

polynom(list<monom> &inlist) : pmlist(inlist) {}

bool operator==(const polynom& pml) const { return pmlist == pml.pmlist; }

bool operator!=(const polynom& pml) const { return pmlist != pml.pmlist; }

polynom(std::string ipm = "");

polynom operator+(const polynom& pml) const;

polynom operator\*(const polynom& pml) const;

polynom operator\*(const double mp) const;

};

**polinom.cpp**

#include "polynom.h"

using namespace std;

list<monom> polynom::CreateInstance(string ipm)

{

list<monom> res;

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

while (ipm.length())

{

string part;

monom temp;

int pos = 1;

while (pos < (int)ipm.length() && ipm[pos] != '+' && ipm[pos] != '-')

pos++;

part = ipm.substr(0, pos);

ipm.erase(0, pos);

pos = 0;

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

pos++;

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

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

temp.cf = 1;

else if (coef == "-")

temp.cf = -1;

else temp.cf = stod(coef);

if (abs(temp.cf) > EPS)

{

part.erase(0, pos);

part += ' ';

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

{

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

if (pos > -1)

{

if (part[pos + 1] != '^')

part.insert(pos + 1, "^1");

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

part.erase(pos, 3);

}

}

res.InsertInOrder(temp);

}

}

return res;

}

string polynom::str() const

{

stringstream buffer;

buffer << \*this;

return buffer.str();

}

polynom::polynom(string ipm)

{

pmlist = CreateInstance(ipm);

}

polynom polynom::operator+(const polynom& pmr) const

{

polynom res;

polynom left = \*this;

polynom right = pmr;

while (!left.pmlist.IsEnded() && !right.pmlist.IsEnded())

{

if (left.pmlist.GetCurr()->data < right.pmlist.GetCurr()->data)

{

res.pmlist.InsertAfter(res.pmlist.GetCurr(), left.pmlist.GetCurr()->data);

left.pmlist.Move();

res.pmlist.Move();

}

else if (left.pmlist.GetCurr()->data > right.pmlist.GetCurr()->data)

{

res.pmlist.InsertAfter(res.pmlist.GetCurr(), right.pmlist.GetCurr()->data);

right.pmlist.Move();

res.pmlist.Move();

}

else

{

double coef = left.pmlist.GetCurr()->data.cf + right.pmlist.GetCurr()->data.cf;

if (abs(coef) > EPS)

{

res.pmlist.InsertAfter(res.pmlist.GetCurr(), monom(coef,left.pmlist.GetCurr()->data.abc));

res.pmlist.Move();

}

left.pmlist.Move();

right.pmlist.Move();

}

}

while (!left.pmlist.IsEnded())

{

res.pmlist.InsertAfter(res.pmlist.GetCurr(), left.pmlist.GetCurr()->data);

left.pmlist.Move();

res.pmlist.Move();

}

while (!right.pmlist.IsEnded())

{

res.pmlist.InsertAfter(res.pmlist.GetCurr(), right.pmlist.GetCurr()->data);

right.pmlist.Move();

res.pmlist.Move();

}

return res;

}

polynom polynom::operator\*(const double mp) const

{

polynom res;

if (abs(mp) > EPS)

{

res = \*this;

while (!res.pmlist.IsEnded())

{

res.pmlist.GetCurr()->data.cf \*= mp;

res.pmlist.Move();

}

}

return res;

}

polynom polynom::operator\*(const polynom& pmr) const

{

polynom res;

polynom right = pmr;

while (!right.pmlist.IsEnded())

{

polynom temp(\*this);

while (!temp.pmlist.IsEnded())

{

temp.pmlist.GetCurr()->data.cf \*= right.pmlist.GetCurr()->data.cf;

int nabc = temp.pmlist.GetCurr()->data.abc + right.pmlist.GetCurr()->data.abc;

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

temp.pmlist.GetCurr()->data.abc = nabc;

else

throw "Too large exponent";

temp.pmlist.Move();

}

res = res + temp;

right.pmlist.Move();

}

return res;

}

ostream& operator<<(ostream &ostr,const polynom& pm)

{

polynom pmt = pm;

while (!pmt.pmlist.IsEnded())

{

monom temp = pmt.pmlist.GetCurr()->data;

if (abs(temp.cf - 1) > EPS && abs(temp.cf + 1) > EPS || temp.abc == 0)

ostr << temp.cf;

else if (abs(temp.cf + 1) < EPS && abs(temp.cf - 1) > EPS)

ostr << '-';

int p = temp.abc / 100;

if(p>1)

ostr << "x^" << p;

else if (p == 1) ostr << "x";

p = temp.abc / 10 % 10;

if(p>1)

ostr << "y^" << p;

else if (p == 1) ostr << "y";

p = temp.abc % 10;

if(p>1)

ostr<< "z^" << p;

else if (p == 1) ostr << "z";

pmt.pmlist.Move();

if(!pmt.pmlist.IsEnded() && pmt.pmlist.GetCurr()->data.cf>0)

ostr << "+";

}

return ostr;

}

## Приложение В. Программная реализация приложения

**main.cpp**

#include "polynom.h"

#include <iostream>

using namespace std;

char menu()

{

system("cls");

char res;

cout << "1. Add\n";

cout << "2. Substract\n";

cout << "3. Mult\n";

cout << "4. Const Mult\n";

cout << "5. Exit\n";

cout << "\n\n";

cout << "Choose menu item: ";

cin >> res;

return res;

}

void main() {

char pmenu;

polynom p1, p2;

string s1, s2;

double c;

do {

pmenu = menu();

switch (pmenu)

{

case '1':

{

system("cls");

cout << "Enter the first polynom:\n";

cin >> s1;

p1 = polynom(s1);

cout << "Enter the second polynom:\n";

cin >> s2;

p2 = polynom(s2);

cout << p1 << '+' << p2 << '=' << p1 + p2 << '\n';

system("pause");

break;

}

case '2':

{

system("cls");

cout << "Enter the first polynom:\n";

cin >> s1;

p1 = polynom(s1);

cout << "Enter the second polynom:\n";

cin >> s2;

p2 = polynom(s2);

cout << '(' << p1 << ')' << '-' << '(' << p2 << ')' << '=' << p1 - p2 << '\n';

system("pause");

break;

}

case '3':

{

system("cls");

cout << "Enter the first polynom:\n";

cin >> s1;

p1 = polynom(s1);

cout << "Enter the second polynom:\n";

cin >> s2;

p2 = polynom(s2);

cout << '(' << p1 << ')' << '\*' << '(' << p2 << ')' << '=' << p1 \* p2 << '\n';

system("pause");

break;

}

case '4':

{

system("cls");

cout << "Enter polynom:\n";

cin >> s1;

p1 = polynom(s1);

cout << "Enter const:\n";

cin >> c;

cout << '(' << p1 << ')' << '\*' << c << '=' << p1 \* c << '\n';

system("pause");

break;

}

case '5': break;

}

} while (pmenu != '5');

}