МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по учебной практике

**Арифметические операции с полиномами**

**Выполнила:** студент группы 381603-1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Захарова Д. Д.

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

Нижний Новгород

2018

**Содержание**

[Введение 3](#_Toc515149244)

[1 Постановка задачи 4](#_Toc515149245)

[2 Руководство пользователя 5](#_Toc515149246)

[3 Руководство программиста 6](#_Toc515149247)

[3.1 Описание структуры программы 6](#_Toc515149248)

[3.2 Описание структуры данных 6](#_Toc515149249)

[3.3 Описание программной реализация 7](#_Toc515149250)

[Заключение 10](#_Toc515149251)

[Список литературы 11](#_Toc515149252)

[Приложение 12](#_Toc515149253)

[Приложение A. node.h 12](#_Toc515149254)

[Приложение B. list.h 12](#_Toc515149255)

[Приложение C. monom.h 14](#_Toc515149256)

[Приложение D. polynom.h 15](#_Toc515149257)

[Приложение E. polynom.cpp 15](#_Toc515149258)

[Приложение F. main\_polynom.cpp](#_Toc515149259) 19

# Введение

Данная работа направлена на изучение способов программной обработки полиномов. Эта программа должна выполнять арифметические операции над полиномами (такие как сложение, вычитание и умножение полиномов, а также умножение полинома на константу). Основная цель данной лабораторной работы- практическое применение таких структур хранения данных, как циклические односвязные списки с головой. В ходе выполнения лабораторной работы разрабатывается общая форма представления циклических списков, разрабатывается программа для работы с ними и практическое приложение списков для работы с полиномами.

# Постановка задачи

*Цель:*

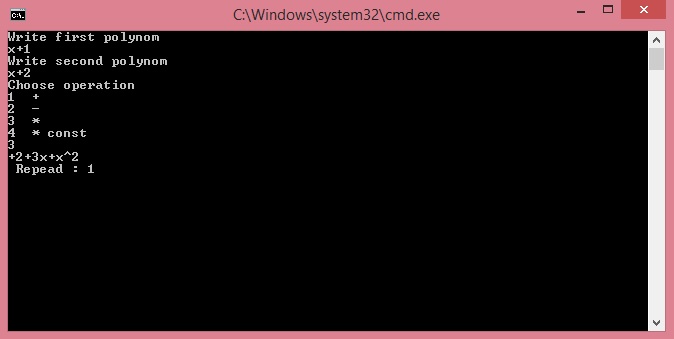
Разработать и реализовать программу, выполняющую арифметические операции над полиномами трёх переменных (x, y, z): сложение, вычитание, умножение полиномов и умножение полинома на константу. Полином – сумма мономов, степень каждого из которых от 0 до 9. Коэффициенты перед мономами — вещественные числа. Для написания программы используется язык C++.

*Исходные данные:*

Вводится 2 строки, содержащие полином\_1 и полином\_2.

*Требуемый результат:*

Правильность выполнения арифметических действий над полиномами.



1. Пример работы программы.

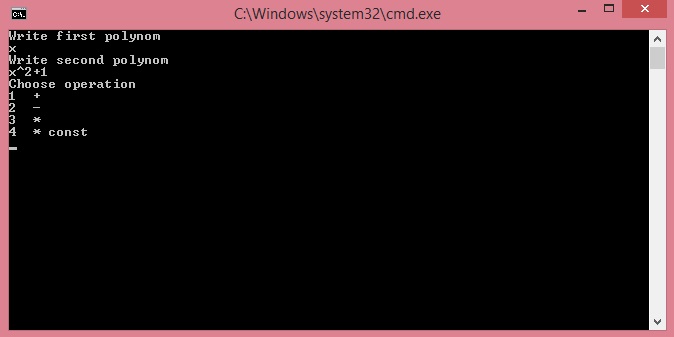
# Руководство пользователя

1. Для начала работы программы необходимо открыть файл .

Допустимые символы: x, y, z

1, 2, 3, 4, 5, 6, 7, 8, 9, 0

+, -, ^

1. Вводится полином\_1, вводится полином\_2.
2. Выбирается операция.
3. Пользователь получает ответ.
4. Выбор арифметической операции

# Руководство программиста

## Описание структуры программы

Программа состоит из 4 проектов.

1. **gtest** – статическая библиотека google-тестов.

gtest.h – заголовочный файл для google—тестов.

gtest.cpp – source-файл для google—тестов.

1. **polynom** – статическая библиотека для полиномов.

node.h—заголовочный файл, который содержит объявление шаблонного класса node.

monom.h—заголовочный файл, который содержит объявление шаблонного класса monom.

list.h—заголовочный файл, который содержит объявление и реализацию шаблонного класса list.

polynom.h—заголовочный файл, который содержит объявление шаблонного класса polynom.

polynom.cpp—файл, который содержит реализацию шаблонного класса polynom.

1. **samples**—консольное приложение.

main\_polynom.cpp—файл, который содержит функцию main().

1. **test**—консольное приложение.

test\_main.cpp—файл, который содержит реализацию тестов для проверки корректности работы функции main().

test\_list.cpp— файл, который содержит реализацию тестов для проверки корректности работы класса list.

test\_polynom.cpp— файл, который содержит реализацию тестов для проверки корректности работы класса polynom.

## Описание структуры данных

В программе используется циклический односвязный список с головой, который имеет два поля-указатель на голову и указатель на текущий элемент. С помощью циклического списка реализуется класс-полином, который состоит из мономов, т.е. из звеньев списка. При этом каждый моном имеет два поля-степень монома и коэффициент монома. Моном, т.е. звено списка реализовано с помощью шаблонного класса-узла, который содержит два поля-данные в узле и указатель на следующий узел.

**Алгоритмы.**

Вводится с клавиатуры полином, далее он преобразовывается в строку, с помощью алгоритма разбиения строки на мономы.

*Алгоритм разбиения строки на мономы*

Подается строка, которая проходит слева-направо до тех пор, пока не встретится символ '+', '-' или конец строки. Если эти символы встретились, то выделяется подстрока до этого и символа, которая переводится в моном. В подстроке происходит поиск коэффициента и степеней переменных. Эти значения записываются в структуру monom, где степень x – это сотый разряд, степень y — десятый разряд, а степень z – первый разряд. Полученный моном записывается в список, тем самым формируется упорядоченный список из мономов.

*Алгоритм сложения полиномов.*

Алгоритмом осуществляется при помощи слияния упорядоченных массивов.

На вход подаются два упорядоченных списка. Далее происходит сравнивание элементов списков: больший элемент вставляем в результат; двигаем указатель списка, откуда взяли элемент, на одно звено вперед. Если элементы равны, то складываем коэффициенты и записываем в результат; двигаем указатели обоих списков на одно звено вперед. Если один из списков кончился, то записываем остаток второго списка в результат.

*Алгоритм вычитания полиномов.*

Вычитание полиномов выполняется через сложение: к первому полиному прибавляется второй полином, умноженный на -1.

## Описание программной реализации

**Node**:

*Поля:*

TP data;—данные в узле

node<TP>\* next; —указатель на следующий узел.

*Методы:*

node();—конструктор по умолчанию

node(TP a1, node<TP>\* a2 = NULL); —конструктор с параметром

bool operator< (const node<TP>& a) const; —оператор сравнения

bool operator> (const node<TP>& a) const;—оператор сравнения

**Monom:**

*Поля:*

double cff; —коэффициент перед мономом

int svr; —свёрнутая степень

*Методы:*

monom(const monom& a);

monom(const double index\_cff = 0.0, const unsigned int index\_svr = 0); —конструктор

const monom& operator=(const monom& a); —оператор присваивания

bool operator< (const monom& a) const; —оператор сравнения

bool operator> (const monom& a) const— оператор сравнения

bool operator==(const monom& a) const;—перегрузка ==

bool operator!=(const monom& a) const; —перегрузка !=

**List:**

*Поля:*

node<TP>\* head; —голова

node<TP>\* Curr; —указатель на текущий элемент

*Методы:*

list();—конструктор по умолчанию

~list();—деструктор

list(const list<TP>& a); —конструктор

void Clean();—очистка списка

void InsertInOrder(const TP& a); —вставить в упорядоченный список

void InsertAfter(node<TP>\* a1, const TP& a2); —вставить после

void InsertToTail(const TP a); —вставить в конец

bool IsEmpty() const; —проверка на пустоту

TP& GetCurr() const; —получение текущего адреса

void Reset();—установка на начало

void gonext();—переход на следующее звено

bool IsOver();—проверка на конец

const list<TP>& operator=(const list<TP>& a);—перегрузка оператора =

bool operator==(const list<TP>& a) const;—оператор сравнения ==

bool operator!=(const list<TP>& a) const; —оператор сравнения не равно

**Polynom:**

*Поля:*

list<monom> list\_monom;—список мономов

list<monom> such(list <monom> & op);—объединение подобные

*Методы:*

polynom(const polynom& p);

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

polynom(list<monom> &in\_list) : list\_monom(in\_list); —конструктор по списку

polynom(const string op = "");— констурктор по строке

polynom operator-(const polynom& p) const;­— бинарный минус

polynom operator-() const; — унарный минус

bool operator==(const polynom& p) const— оператор сравнения

bool operator!=(const polynom& p) const; — оператор сравнения

polynom operator+(const polynom& p) const; — оператор сложения полиномов

polynom operator\*(const polynom& p) const; — оператор умножения полиномов

polynom operator\*(const double k) const; — умножение на константу

friend polynom operator\*(const double k, const polynom& p); —умножение на константу с другой стороны

friend std::ostream& operator<<(std::ostream &str, const polynom &p); — оператор вставки в поток

# Заключение

В лабораторной работе была реализована программа, выполняющая арифметические операции над полиномами. Хранение полиномов было реализовано с помощью циклического односвязного цикла с головой. Программа выполняет все необходимые операции над полиномами (сложение, умножение, вычитание, умножение на константу)..

# Список литературы

1. Рабочие материалы к учебному курсу «Методы программирования». *Гергель В.П*. 2002 г.
2. Лабораторный практикум. Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017.

# Приложение

## Приложение A. node.h

#pragma once

template<class TP>

class node

{

public:

TP data;

node<TP>\* next;

node() { next = NULL; };

node(TP a1, node<TP>\* a2 = NULL) { data = a1; next = a2; };

bool operator< (const node<TP>& a) const { return (data < a.data); }; bool operator> (const node<TP>& a) const { return (data > a.data); };

## };

## Приложение B. list.h

#pragma once

#include "node.h"

#include <iostream>

using namespace std;

template<class TP>

class list

{

private:

node<TP>\* head; node<TP>\* Curr;

public:

list();

~list(); list(const list<TP>& a); void Clean(); void InsertInOrder(const TP& a); void InsertAfter(node<TP>\* a1, const TP& a2); void InsertToTail(const TP a);

bool IsEmpty() const { return (head->next == head); }; TP& GetCurr() const { return Curr->data; } void Reset() { Curr = head->next; } void gonext() { Curr = Curr->next; }; bool IsOver() { if (Curr->next == head->next) return true; else return false; } const list<TP>& operator=(const list<TP>& a);

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

};

template <class TP>

list<TP>::list()

{

head = new node<TP>();

Curr = head->next;

head->next = head;

}

template <class TP>

list<TP>::~list()

{

Clean();

delete head;

}

template <class TP>

list<TP>::list(const list<TP>& a)

{

head = new node<TP>;

node<TP>\* E1 = a.head;

node<TP>\* E2 = head;

if (E1->next == a.head)

{

head->next = head;

return;

}

while (E1->next != a.head)

{

E1 = E1->next;

E2->next = new node<TP>(E1->data);

E2 = E2->next;

}

E2->next = head;

Curr = head->next;

}

template <class TP>

void list<TP>::Clean()

{

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

while (curr != head)

{

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

delete curr;

curr = temp;

}

head->next = head;

}

template <class TP>

void list<TP>::InsertInOrder(const TP& a)

{

if (IsEmpty())

{

head->next = new node<TP>(a);

head->next->next = head;

}

else

{

node<TP>\* t = new node<TP>(a);

node<TP>\* c = head;

while (c->next != head && (\*(c->next) < \*t))

{

c = c->next;

}

node<TP>\* t1 = c->next;

c->next = t;

c->next->next = t1;

}

}

template <class TP>

void list<TP>::InsertAfter(node<TP>\* a1, const TP& a2)

{

if (head == NULL)

throw "empty list";

else

{

node<TP>\* t = a1->next;

a1->next = new node<TP>(a2, t);

}

}

template<class TP>

void list<TP> ::InsertToTail(const TP a)

{

Reset();

while (Curr->next != head)

gonext();

node<TP>\* t = Curr->next;

Curr->next = new node<TP>(a);

Curr->next->next = t;

}

template <class TP>

const list<TP>& list<TP>::operator=(const list<TP>& a)

{

Clean();

node<TP>\* p\_a = a.head;

node<TP>\* p\_curr = head;

while (p\_a->next != a.head)

{

p\_a = p\_a->next;

p\_curr->next = new node<TP>(p\_a->data);

p\_curr = p\_curr->next;

}

p\_curr->next = head;

Curr = head->next;

return \*this;

}

template<class TP>

bool list<TP>::operator==(const list<TP>& a) const

{

bool res = true;

if (this != &a)

{

node<TP>\* t1 = a.head->next;

node<TP>\* t2 = head->next;

while (t1->data == t2->data && t2 != head && t1 != a.head)

{

t1 = t1->next;

t2 = t2->next;

}

if (t2 != head || t1 != a.head)

res = false;

}

return res;

}

## Приложение C. monom.h

#pragma once

#include <math.h>

class monom

{

public:

double cff; int svr; monom(const monom& a) { cff = a.cff; svr = a.svr; }

monom(const double index\_cff = 0.0, const unsigned int index\_svr = 0) { cff = index\_cff; svr = index\_svr; }

const monom& operator=(const monom& a) { cff = a.cff; svr = a.svr; return \*this; } bool operator< (const monom& a) const { return (svr<a.svr); }

bool operator> (const monom& a) const { return (svr>a.svr); } bool operator==(const monom& a) const { return((svr == a.svr) && (cff == a.cff));} bool operator!=(const monom& a) const { return !(\*this == a); }

## };

## Приложение D. polynom.h

#pragma once

#include <iostream>

#include <string>

#include <algorithm>

#include "monom.h"

#include "list.h"

#define OFFSET 120

class polynom

{

private:

list<monom> list\_monom; list<monom> such(list <monom> & op);

public:

polynom(const polynom& p);

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

polynom(list<monom> &in\_list) : list\_monom(in\_list) {}

polynom(const string op = ""); polynom operator-(const polynom& p) const { return \*this + p\*(-1); }

polynom operator-() const { return (-1)\*(\*this); } bool operator==(const polynom& p) const { return list\_monom == p.list\_monom; } bool operator!=(const polynom& p) const { return list\_monom != p.list\_monom; } polynom operator+(const polynom& p) const; polynom operator\*(const polynom& p) const; polynom operator\*(const double k) const; friend polynom operator\*(const double k, const polynom& p) { return p\*k; } friend std::ostream& operator<<(std::ostream &str, const polynom &p);

};

## Приложение E. polynom.cpp

#include "polynom.h"

using namespace std;

list<monom> polynom::such(list <monom> & op)

{

list<monom> res;

res.Reset();

op.Reset();

node<monom> mon(op.GetCurr().cff);

list<monom> y(op);

while (!op.IsOver())

{

mon.data.svr = op.GetCurr().svr;

y.gonext();

if (op.GetCurr().svr == y.GetCurr().svr && (y.GetCurr().cff || y.GetCurr().svr))

mon.data.cff += y.GetCurr().cff;

else

{

if (mon.data.cff)

{

res.InsertToTail(mon.data);

res.gonext();

}

mon.data.cff = y.GetCurr().cff;

}

op.gonext();

}

return res;

}

polynom::polynom(const polynom& p)

{

list\_monom = p.list\_monom;

}

polynom::polynom(string op)

{

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

while (op.length())

{

string pt;

monom t;

int pos = 1;

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

pos++;

pt = op.substr(0, pos);

op.erase(0, pos); pos = 0;

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

pos++;

string pstr = pt.substr(0, pos);

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

t.cff = 1;

else if (pstr == "-")

t.cff = -1;

else t.cff = stod(pstr); pt.erase(0, pos);

pt += ' ';

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

{

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

if (pos > -1)

{

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

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

t.svr += d[i] \* stoi(pt.substr(pos + 2, 1)); pt.erase(pos, 3);

}

}

list\_monom.InsertInOrder(t);

}

list\_monom = such(list\_monom);

}

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

{

list\_monom = p.list\_monom;

return \*this;

}

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

{

polynom res;

polynom pthis = \*this;

polynom pol = p;

pthis.list\_monom.Reset();

pol.list\_monom.Reset();

res.list\_monom.Reset();

while (!pthis.list\_monom.IsOver() && !pol.list\_monom.IsOver())

{

if (pthis.list\_monom.GetCurr() > pol.list\_monom.GetCurr()) {

res.list\_monom.InsertToTail(pol.list\_monom.GetCurr());

pol.list\_monom.gonext();

res.list\_monom.gonext();

}

else if (pthis.list\_monom.GetCurr() < pol.list\_monom.GetCurr())

{

res.list\_monom.InsertToTail(pthis.list\_monom.GetCurr());

pthis.list\_monom.gonext();

res.list\_monom.gonext();

}

else

{

double new\_c = pthis.list\_monom.GetCurr().cff + pol.list\_monom.GetCurr().cff;

if (new\_c)

{

res.list\_monom.InsertToTail(monom(new\_c, pthis.list\_monom.GetCurr().svr));

res.list\_monom.gonext();

}

pthis.list\_monom.gonext();

pol.list\_monom.gonext();

}

}

while (!pthis.list\_monom.IsOver())

{

res.list\_monom.InsertToTail(pthis.list\_monom.GetCurr());

pthis.list\_monom.gonext();

res.list\_monom.gonext();

}

while (!pol.list\_monom.IsOver())

{

res.list\_monom.InsertToTail(pol.list\_monom.GetCurr());

pol.list\_monom.gonext();

res.list\_monom.gonext();

}

return res;

}

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

{

polynom res;

polynom pth = \*this;

polynom pol = p;

pth.list\_monom.Reset();

pol.list\_monom.Reset();

while (!pth.list\_monom.IsOver())

{

double pth\_cff = pth.list\_monom.GetCurr().cff;

int pth\_svr = pth.list\_monom.GetCurr().svr;

polynom tp(p);

tp.list\_monom.Reset();

while (!tp.list\_monom.IsOver())

{

int tp\_svr = tp.list\_monom.GetCurr().svr;

if ((tp\_svr % 10 + pth\_svr % 10) < 10 && (tp\_svr / 10 % 10 + pth\_svr / 10 % 10) < 10 && (tp\_svr / 100 + pth\_svr / 100) < 10) //остаток от деления

{

tp.list\_monom.GetCurr().svr += pth\_svr;

tp.list\_monom.GetCurr().cff \*= pth\_cff;

}

else

throw "Error";

tp.list\_monom.gonext();

}

res = res + tp;

pth.list\_monom.gonext();

}

return res;

}

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

{

polynom res;

res = \*this;

res.list\_monom.Reset();

while (!res.list\_monom.IsOver())

{

res.list\_monom.GetCurr().cff \*= k;

res.list\_monom.gonext();

}

return res;

}

ostream& operator<<(ostream &str, const polynom& p)

{

polynom pl = p;

pl.list\_monom.Reset();

while (!pl.list\_monom.IsOver())

{

monom tp = pl.list\_monom.GetCurr();

if (tp.cff > 0)

{

str << "+";

if (tp.cff == 1 && tp.svr == 0)

str << "1";

else

if (tp.cff != 1)

str << tp.cff;

}

else

str << tp.cff;

int a = tp.svr / 100;

if (a>1)

str << "x^" << a;

else

if (a == 1)

str << "x";

a = tp.svr / 10 % 10;

if (a>1)

str << "y^" << a;

else

if (a == 1)

str << "y";

a = tp.svr % 10;

if (a>1)

str << "z^" << a;

else

if (a == 1)

str << "z";

pl.list\_monom.gonext();

}

return str;

}

## Приложение F. main\_polynom.cpp

#include "monom.h"

#include "polynom.h"

#include <iostream>

using namespace std;

void main()

{

int flag = 1;

int k = 0;

while (flag == 1)

{

system("cls");

cout << "Write first polynom" << endl;

string s1;

cin >> s1;

polynom S1(s1);

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

string s2;

cin >> s2;

polynom S2(s2);

cout << "Choose operation" << endl;

cout << "1 +" << endl;

cout << "2 -" << endl;

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

cout << "4 \* const" << endl;

cin >> k;

switch (k)

{

case 1:

{

polynom res = S1 + S2;

cout << res;

break;

}

case 2:

{

polynom res = S1 - S2;

cout << res;

break;

}

case 3:

{

polynom res = S1 \* S2;

cout << res;

break;

}

case 4:

{

string con;

cout << " Write const" << endl;

cin >> con;

polynom C(con);

polynom res = C\*S1;

cout << res;

break;

}

default:

break;

}

cout << " \n Repead : 1" << endl;

cin >> flag;

}

}