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**

**Выполнила:** студентка группы 381603-1

Фадеева А.М.

**Проверила:** доцент каф. МОСТ института ИТММ

Кустикова В.Д.

Нижний Новгород  
2018

**Содержание**

[Введение 3](#_Toc513578490)

[Постановка задачи 4](#_Toc513578491)

[Руководство пользователя 5](#_Toc513578492)

[Руководство программиста 7](#_Toc513578493)

[Описание структуры программы 7](#_Toc513578494)

[Описание структур данных 8](#_Toc513578495)

[Описание алгоритмов 10](#_Toc513578496)

[Заключение 11](#_Toc513578497)

[Список литературы 12](#_Toc513578498)

[Приложение 13](#_Toc513578499)

[RingList 13](#_Toc513578500)

[Polynom 15](#_Toc513578501)

[ui 19](#_Toc513578502)

# Введение

К числу наиболее известных и изученных аналитических функций относятся степенные многочлены — полиномы. Круг объектов, которые могут быть представлены полиномами, достаточно обширен, и полиномиальные преобразования широко используются на практике, в частности, для приближенного представления других функций.

Для того чтобы выполнять операции над полиномами необходимо их хранить и обрабатывать. Для этого нужно разработать структуру данных, которая позволяет это делать. Реализация подобной структуры упрощает работу со сложными вычислениями.

# 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 Test-ов. Кроме того, необходимо разработать пользовательское консольное приложение.

# Руководство пользователя

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

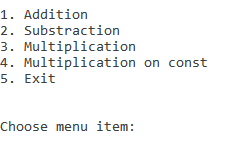


Рис. 1. Меню программы

Далее следует выбрать один из пунктов меню.

Пункты 1-4 позволяют выполнять операции над полиномами (сложение, вычитание, умножение, умножение на константу). Для выхода из программы необходимо выбрать пункт 5.

После выбора пункта 1-4 необходимо ввести первый полином (рис. 2). Предполагается корректное введение полинома. Считается корректным полином, составленный из мономов трех переменных x,y,z с целой степенью от 0 до 9.



Рис. 2

Далее необходимо ввести второй полином (для пунктов 1-3) (рис. 3).

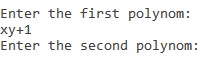


Рис. 3

Для пункта 4 следует ввести константу (рис. 4).

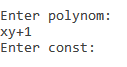


Рис. 4

После ввода второго полинома или константы на экране появится результат выполненной операции (рис. 5).

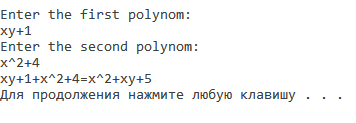


Рис. 5

Далее произойдет возврат в главное меню.

# Руководство программиста

*Описание структуры программы*

Проекты и файлы, из который состоит программа:

1. RingList

node.h, в котором находится шаблон структуры node

list.h, содержит объявление шаблона класса list и его реализацию

list.cpp

1. Polynom

monom.h, в котором находится структура monom

polynom.h, содержит объявление класса polynom

polynom.cpp, содержит реализацию класса polynom

1. gtest

gtest.h заголовочный файл для Google Test

gtest-all.cc файл с кодом Google Test

1. ui

main.cpp, в котором реализован пользовательский интерфейс

1. tests

test\_main.cpp запуск всех тестов

test\_ringlist.cpp тесты для класса list

test\_polynom.cpp тесты для класса polynom

*Описание структур данных*

1. **node –** шаблон структуры, параметризованный type, представляет собой звено списка.

*Поля структуры:*

node\* next: //указатель на следующий элемент

type data; //данные, хранящиеся в звене

node(); //конструктор

node(type a); //конструктор с параметром

bool operator== (const node& a) ; //оператор сравнения

bool operator!=(const node& a) ; //оператор сравнения

1. **list** – шаблон класса, параметризованный type, представляет собой циклический список с «головой».

*Поля класса (private):*

node<type>\* head; //указатель на "голову"

node<type>\* curr; // указатель на текущий элемент

*Реализованные методы (public):*

list(); //конструктор

~list(); //деструктор

void Clean(); //очистка списка

list(const list<type>& list2); // конструктор копирования

list<type>& operator=(const list<type>& list2); // оператор =

void InsertOrd(type a); //упорядоченная вставка

void Insert(node<type>\*, type );// вставка после

void InsertToTail(type a); //вставка в конец

void Reset(); //переместить указатель на начало

void GetNext(); //переместить указатель на следующий элемент

node<type>\* GetCurr() const; //получить указатель на текущий элемент

bool IsEnded() const; //проверка на конец

bool operator==(const list<type>&) const; //оператор сравнения

bool operator!=(const list<type>& list2) const; //оператор сравнения

1. **monom** – структура, представляет собой звено полинома.

*Поля структуры:*

double cf; //коэффициент

unsigned int abc; //свернутая степень

monom (double cf1 = 0, unsigned int abc1 = 0); //конструктор

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

bool operator> (const monom& a) const; //оператор сравнения

bool operator== (const monom& a) const; //оператор сравнения

bool operator!= (const monom& a) const; //оператор сравнения

1. Класс **polynom**.

*Поля класса(private):*

list<monom> plist; //список из мономов

*Реализованные методы (private):*

static list<monom> parse(const string); //разбиение на мономы

list<monom> reduction(list <monom> p); //приведение подобных

*Реализованные методы (public):*

polynom(const string expr = ""); //конструктор по строке

polynom(list<monom> &expr) : plist(expr) //конструктор преобразования типа

polynom(const polynom& pol) : plist(pol.plist) //конструктор копирования

polynom operator+(const polynom&) const; //сложение полиномов

polynom operator-(const polynom& pol) const ; //разность полиномов

polynom operator\*(const double) const; //умножение полинома на константу

polynom operator\*(const polynom&) const; //произведение полиномов

friend polynom operator\*(const double c, const polynom& pol); //умножение константы на полином

friend ostream& operator<<(ostream& os, const polynom&); //вывод полиномов

bool operator==(const polynom& pol) const ; //оператор сравнения

bool operator!=(const polynom& pol) const ; //оператор сравнения

*Описание алгоритмов*

1. Разбиение строки на мономы.

На вход подается строка. Строка обходится слева направо, пока не кончится или не встретится «+» или «–». Когда встретится символ, выделяем подстроку от начала до этого символа и вырезаем ее. Идем по подстроке до переменной x,y,z и вычисляем коэффициент. Если после переменной идет «^» со степенью, то переводим степень из строки в число, иначе степень считается равной 1. Получили моном, состоящий из коэффициента и степени. Упорядоченно вставляем его в список. Продолжаем обходить строку, если она не кончилась.

На выходе получаем упорядоченный список из мономов.

1. Приведение подобных.

На вход подается упорядоченный список из мономов. Идем по списку. Если у нескольких подряд мономов одинаковая степень, то вместо них создается новый моном со степенью равной степени этих мономов и коэффициентом, который является суммой их коэффициентов. Если степень уникальна, то моном остается без изменений.

1. Сложение полиномов.

На вход подается два полинома. Устанавливаем указатели на начало списков мономов. Сравниваем степени текущих мономов. Наибольших их них записываем результирующий полином и смещаем указатель на следующий. Если степени равны, то записываем моном с их степенью и коэффициентом равным сумме коэффициентов текущих мономов, в обоих списках переходим к следующему моному. Сравнение степеней происходит до тех пор, пока не кончится один из мономов. Мономы из оставшегося списка записываются в результирующий полином.

1. Умножение полиномов.

На вход подается два полинома. Устанавливаем указатели на начало списков мономов. Пока второй список не кончится перемножаем коэффициенты текущих мономов и складываем их степени. Степень каждой переменной должна быть меньше 10. Получившийся моном записываем во временный полином. Смещаем указатель на следующий элемент во втором списке. Когда список кончится, прибавляем к результирующему полиному временный. Пока первый список не кончится смещаем в нем указатель на следующий моном, а во втором переместим указатель на начало.

# Заключение

Была разработана программа, выполняющая арифметические операции с полиномами трех переменных: сложение, вычитание, умножение на константу, умножение двух полиномов. Все функции тестируются с помощью Google test-ов. Разработано пользовательское консольное приложение.

# Список литературы

1. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани; Пер. с англ. под ред. А. Шеня. –– М.: МЦНМО, 2014. –– 320 с.
2. Кормен Т., Лейзерсон Ч., Риверст Р., Штайн К. Алгоритмы. Построение и анализ. - М.: Издательский дом "Вильямс". - 2005. - 1290с.
3. Рабочие материалы к учебному курсу «Методы программирования». *Гергель В.П*. 2002 г.

# Приложение

RingList

**node.h**

#pragma once

template <class type>

struct node {

node\* next;

type data;

node() { next = nullptr; }

node(type a) { data = a; next = nullptr; }

bool operator== (const node& a) const { return (data==a.data); }

bool operator!= (const node& a) const { return !(\*this==a); }

};

**list.h**

#pragma once

#include "node.h"

template <class type>

class list {

private:

node<type>\* head;

node<type>\* curr;

public:

list();

~list();

void Clean();

list(const list<type>& list2);

list<type>& operator=(const list<type>& list2);

void InsertOrd(type a);

void Insert(node<type>\*, type );

void InsertToTail(type a);

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

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

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

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

bool operator==(const list<type>&) const;

bool operator!=(const list<type>& list2) const { return !(\*this == list2); }

};

template <class type> list<type>::list() {

head = new node<type>;

head->next = head;

curr = head;

}

template <class type> list<type>::~list() {

Clean();

delete head;

}

template <class type> void list<type>::Clean() {

node <type>\* cur = head->next;

node<type>\* temp;

while (cur != head)

{

temp = cur->next;

delete cur;

cur = temp;

}

head->next = head;

}

template <class type> list<type>::list(const list<type>& list2) {

node<type>\* curr2 = list2.head;

head = new node<type>(curr2->data);

head->next = head;

curr=head;

while (curr2->next != list2.head)

{

curr2 = curr2->next;

curr->next = new node<type>(curr2->data);

GetNext();

}

curr->next = head;

}

template <class type> list<type>& list<type>::operator=(const list<type>& list2) {

Clean();

node<type>\* cur2 = list2.head;

node<type>\* cur = head;

while (cur2->next != list2.head)

{

cur2 = cur2->next;

cur->next = new node<type>(cur2->data);

cur = cur->next;

}

cur->next = head;

curr = head;

return \*this;

}

template <class type> void list<type>::InsertOrd(type a)

{

curr = head;

node<type>\* temp;

while ((curr->next->data > a) && (curr->next != head))

GetNext();

temp = curr->next;

curr->next = new node<type>(a);

curr->next->next = temp;

}

template <class type> bool list<type>::operator==(const list<type>& list2) const {

bool res = true;

if (this != &list2)

{

node<type>\* cur2 = list2.head->next;

node<type>\* cur = head->next;

while (cur->data == cur2->data && cur != head && cur2 != list2.head )

{

cur = cur->next;

cur2 = cur2->next;

}

if (cur != head || cur2 != list2.head)

res = false;

}

return res;

}

template <class type> void list<type>::Insert(node<type>\* p,type d) {

node<type>\* temp = p->next;

p->next = new node<type>(d);

p->next->next = temp;

}

template <class type> void list<type>::InsertToTail(type a)

{

Reset();

while (curr->next != head)

GetNext();

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

curr->next = new node<type>(a);

curr->next->next = temp;

}

list.cpp

#include "list.h"

#include "node.h"

Polynom

**monom.h**

#pragma once

struct monom{

double cf;

unsigned int abc;

monom(double cf1 = 0, unsigned int abc1 = 0) { cf = cf1; abc = abc1; }

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

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

bool operator== (const monom& a) const { return ((abc == a.abc) && (cf==a.cf)); }

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

};

**polynom.h**

#pragma once

#include "monom.h"

#include "list.h"

#include <string>

using namespace std;

class polynom {

private:

list<monom> plist;

static list<monom> parse(const string);

list<monom> reduction(list <monom> p);

public:

polynom(const string expr = "");

polynom(list<monom> &expr) : plist(expr) {}

polynom(const polynom& pol) : plist(pol.plist) { }

polynom operator+(const polynom&) const;

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

polynom operator\*(const double) const;

polynom operator\*(const polynom&) const;

friend polynom operator\*(const double c, const polynom& pol) { return pol\*c; }

friend ostream& operator<<(ostream& os, const polynom&);

bool operator==(const polynom& pol) const { return plist == pol.plist; }

bool operator!=(const polynom& pol) const { return plist != pol.plist; }

};

**polinom.cpp**

#include "polynom.h"

polynom::polynom(const string expr) {

plist = parse(expr);

plist = reduction(plist);

};

list<monom> polynom::reduction(list <monom> p) {

list<monom> res;

p.Reset();

res.Reset();

node<monom> temp(p.GetCurr()->data.cf);

while (!(p.IsEnded()))

{

temp.data.abc = p.GetCurr()->data.abc;

if (p.GetCurr()->data.abc == p.GetCurr()->next->data.abc && (p.GetCurr()->next->data.cf || p.GetCurr()->next->data.abc))

{

temp.data.cf += p.GetCurr()->next->data.cf;

}

else

{

if (temp.data.cf)

{

//res.Insert(res.GetCurr(), temp.data);

res.InsertToTail(temp.data);

res.GetNext();

}

temp.data.cf = p.GetCurr()->next->data.cf;

}

p.GetNext();

}

return res;

}

list<monom> polynom::parse(const string expr) {

const string xyz = "xyz";

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

list<monom> list;

string str = expr, part;

int pos;

while (str.length())

{

monom m;

pos = 1;

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

pos++;

part = str.substr(0, pos);

str.erase(0, pos);

pos = 0;

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

pos++;

if (pos == 1 && (part[0] == '-' || part[0] == '+') || pos == 0)

if (part[0] == '-')

m.cf = -1;

else

m.cf = 1;

else

m.cf = stod(part.substr(0, pos));

part.erase(0, pos);

part += ' ';

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

{

pos = part.find(xyz[i]);

if (pos > -1)

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

m.abc += k[i];

else m.abc += k[i] \* stoi(part.substr(pos+2,1));

}

list.InsertOrd(m);

}

return list;

}

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

{

polynom res;

list<monom> list2 = pol.plist, list1 = plist;

list1.Reset();

list2.Reset();

res.plist.Reset();

while (!list1.IsEnded() && !list2.IsEnded())

{

if (list1.GetCurr()->data > list2.GetCurr()->data)

{

//res.plist.Insert(res.plist.GetCurr(), list1.GetCurr()->data);

res.plist.InsertToTail(list1.GetCurr()->data);

list1.GetNext();

res.plist.GetNext();

}

else

if (list1.GetCurr()->data < list2.GetCurr()->data)

{

//res.plist.Insert(res.plist.GetCurr(), list2.GetCurr()->data);

res.plist.InsertToTail(list2.GetCurr()->data);

list2.GetNext();

res.plist.GetNext();

}

else

{

double rcf = list1.GetCurr()->data.cf + list2.GetCurr()->data.cf;

if (rcf)

{

monom temp(rcf, list1.GetCurr()->data.abc);

//res.plist.Insert(res.plist.GetCurr(), temp);

res.plist.InsertToTail(temp);

res.plist.GetNext();

}

list1.GetNext();

list2.GetNext();

}

}

while (!list1.IsEnded())

{

//res.plist.Insert(res.plist.GetCurr(), list1.GetCurr()->data);

res.plist.InsertToTail(list1.GetCurr()->data);

list1.GetNext();

res.plist.GetNext();

}

while (!list2.IsEnded())

{

//res.plist.Insert(res.plist.GetCurr(), list2.GetCurr()->data);

res.plist.InsertToTail(list2.GetCurr()->data);

list2.GetNext();

res.plist.GetNext();

}

return res;

}

polynom polynom::operator\*(const double c) const {

polynom res;

if (c)

{

res = \*this;

res.plist.Reset();

while (!(res.plist.IsEnded()))

{

res.plist.GetCurr()->data.cf \*= c;

res.plist.GetNext();

}

}

return res;

}

polynom polynom::operator\*(const polynom& pol) const {

polynom res;

double l1cf;

int l1abc;

list<monom> list1 = plist, list2 = pol.plist;

list1.Reset();

list2.Reset();

while (!(list1.IsEnded()))

{

l1cf = list1.GetCurr()->data.cf;

l1abc = list1.GetCurr()->data.abc;

polynom temp(pol);

temp.plist.Reset();

while (!(temp.plist.IsEnded()))

{

int tabc = temp.plist.GetCurr()->data.abc;

if ((tabc % 10 + l1abc % 10) < 10 && (tabc / 100 + l1abc / 100) < 10 && (tabc / 10 % 10 + l1abc / 10 % 10) < 10)

{

temp.plist.GetCurr()->data.abc += l1abc;

temp.plist.GetCurr()->data.cf \*= l1cf;

}

else

throw "Too much degree";

temp.plist.GetNext();

}

res = res + temp;

list1.GetNext();

}

return res;

}

ostream& operator<<(ostream& os, const polynom& pol)

{

const string xyz = "xyz";

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

polynom p = pol;

node<monom> temp;

p.plist.Reset();

node<monom>\*start = p.plist.GetCurr();

if (p.plist.IsEnded())

os << '0';

while (!(p.plist.IsEnded()))

{

temp = p.plist.GetCurr()->data;

if (temp.data.cf > 0 && p.plist.GetCurr()!=start)

os << '+';

if ((temp.data.cf != 1 && temp.data.cf != -1) || temp.data.abc == 0)

os << temp.data.cf;

else

if (temp.data.cf == -1)

os << '-';

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

{

if (temp.data.abc / k[i] % 10 != 0)

{

os << xyz[i];

if (temp.data.abc / k[i] % 10 != 1)

os << '^' << temp.data.abc / k[i] % 10;

}

}

p.plist.GetNext();

}

return os;

}

ui

**main.cpp**

#include "polynom.h"

#include <iostream>

char main\_menu()

{

system("cls");

char res;

cout << "1. Addition\n";

cout << "2. Substraction\n";

cout << "3. Multiplication\n";

cout << "4. Multiplication on const\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 = main\_menu();

switch (pmenu)

{

case '1':

{

system("cls");

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

cin >> s1;

p1 = s1;

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

cin >> s2;

p2 = s2;

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

system("pause");

break;

}

case '2':

{

system("cls");

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

cin >> s1;

p1 = s1;

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

cin >> s2;

p2 = s2;

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

system("pause");

break;

}

case '3':

{

system("cls");

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

cin >> s1;

p1 = s1;

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

cin >> s2;

p2 = s2;

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

system("pause");

break;

}

case '4':

{

system("cls");

cout << "Enter polynom:\n";

cin >> s1;

p1 = s1;

cout << "Enter const:\n";

cin >> c;

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

system("pause");

break;

}

case '5': break;

}

} while (pmenu != '5');

}