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](#_Toc515393618)

[1 Постановка задачи 4](#_Toc515393619)

[2 Руководство пользователя 5](#_Toc515393620)

[3 Руководство программиста 6](#_Toc515393621)

[3.1 Структура программы 6](#_Toc515393622)

[3.2 Описание структуры данных 7](#_Toc515393623)

[3.3 Описание программной реализация 7](#_Toc515393624)

[Заключение 9](#_Toc515393625)

[Список литературы 10](#_Toc515393626)

[Приложение 11](#_Toc515393627)

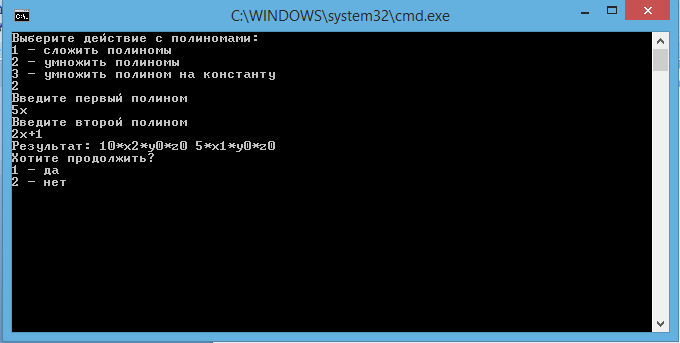
# Введение

Данная работа посвящена теме арифметический операций с полиномами. В математике часто приходится работать с полиномами. Людям необходимо выполнять некоторые преобразования, а так же операции над ними. Разработанная программа помогает выполнять различные арифметические операции над полиномами, такие как:

Сложение и умножение.

# Постановка задачи

Необходимо разработать программу, позволяющую выполнять арифметические операции над полиномами: сложение, умножение, умножение на константу. Каждый моном полинома может состоять из трех переменных, каждая из которых имеет свою степень, коэффициент перед мономом представляет собой вещественное число.

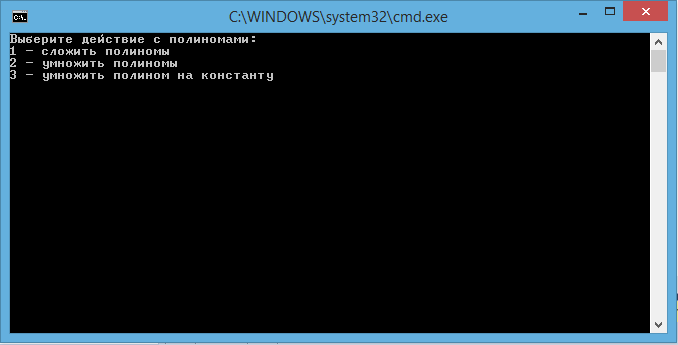
Пример работы программы:

1. Умножение полиномов.

# Руководство пользователя

Для начала работы программы необходимо открыть файлsample.exe.

После запуска программы, перед пользователем появится окно, в котором необходимо будет выбрать арифметическое действие над полиномами

1. Выбор арифметической операции

Затем пользователь должен ввести полиномы, после чего программа выведет на экран полученный результат. В конце выполнения программы, пользователю необходимо выбрать, продолжить вычисления или завершить.

# Руководство программиста

## Структура программы

Программа содержит в себе 4 проекта:

1. gtest (содержит в себе google тесты)
   1. gtest-h (заголовочный файл)
   2. gtest-all
2. polinom\_lib
   1. list.h (содержит в себе объявление и реализацию шаблонного класса list)
   2. monom.h (содержит в себе объявление шаблонного класса monom)
   3. node.h (содержит в себе объявление и реализацию шаблонного класса node)
   4. polinom.h (содержит в себе объявление шаблонного класса polinom)
   5. list.cpp
   6. node.cpp
   7. polinom.cpp (содержит в себе реализацию шаблонного класса polinom)
3. sample
   1. sample\_list.cpp (содержит в себе функцию main)
4. test
   1. test\_list.cpp (содержит в себе тесты для list)
   2. test\_main.cpp (содержит в себе функцию main, которая запускает google - тесты)
   3. test\_polinom.cpp (содержит в себе тесты для monom и polinom)

## Описание структуры данных

В программе используется циклический список с головой, список имеет два поля. С помощью него реализуется polinom. Сам полином состоит из мономов (это звенья списка) каждый моном содержит вещественный коэффициент, а так же степень всех трех переменных. Мономы в списке хранятся по возрастанию степеней. Звено списка реализовано с помощью шаблонного класса node. Список представляет собой структуру с двумя полями: указатель на голову звена и указатель на текущее звено.

## Описание программной реализация

1. node

val data; - данные;  
 node<val>\* next; - указатель на следующее звено;

node() - создает пустое звено;

{

next = NULL;

}

const node<val>&operator=(const node<val> &n);

bool operator==(const node<val> &n) const;

friend ostream & operator<<(ostream &os, const node<val> &n) - перегрузка вывода

1. list

node<val>\* head; - указатель на голову

node<val>\* current; - текущий элемент

node<val>\* tail; - указатель на хвост

ringlist();

ringlist(const ringlist<val> &l);

~ringlist();

void clean();- очистить список

void reset();- текущий элемент равен голове (перейти к голове)

void getnext();- перейти на следующий

bool isended() const; - проверка на конец

void delet(node<val> \*l); - функция удаления

val & getdata() const; - получить данные

node<val>\* search(const val &d); - поиск звена с конкретными данными

void insert\_to\_tail(const val &d); - вставка в конец

void insert\_up(const val &d); - вставка в упорядоченный список

bool operator==(const ringlist<val> &l) const; - сравнение списков

const ringlist<val> & operator=(const ringlist<val> &l); - перегрузка оператора присваивания

friend ostream& operator<<(ostream &out, const ringlist<val> &n) - вывод

1. monom

double coeff; - коэффициент монома

unsigned int abc; - степень монома

~monom() {}

const monom& operator=(const monom &m); - перегрузка оператора присваивания

bool operator==(const monom &m) const; - перегрузка оператора равенства

bool operator!=(const monom &m) const; - перегрузка оператора неравенства

bool operator<(const monom &m) const; - перегрузка оператора меньше

bool operator>(const monom &m) const; - перегрузка оператора больше

monom operator\*(const double d) const; - перегрузка оператора умножение на const

monom operator\*(const monom &m) const; - перегрузка оператора умножения

friend ostream & operator<<(ostream &os, const monom &n); - перегрузка вывода

1. polinom

ringlist<monom> pol; - список из мономов

const polinom& operator=(const polinom &p); - перегрузка оператора присваивания

polinom operator+(const polinom &p) const; - сложение полиномов

polinom operator\*(const polinom &p) const; - умножение полиномов

polinom operator\*(const double d) const; - умножение на константу

bool operator==(const polinom &p) const; - перегрузка оператора равенства

bool operator!=(const polinom &p) const; - перегрузка оператора неравенства

friend ostream& operator<<(ostream& out, const polinom &p); - перегрузка вывода

# Заключение

В данной работе было реализовано создание полиномов и выполнение арифметических операций над ними, таких как сложение, умножение, умножение на константу.

# Список литературы

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

list.h

#pragma once

#include "node.h"

template <class val>

class ringlist

{

private:

node<val>\* head;

node<val>\* current; //текущий элемент

node<val>\* tail;

public:

ringlist();

ringlist(const ringlist<val> &l);

~ringlist();

void clean(); //очистить список

void reset(); //current = head

void getnext(); //current перейдет на следующий

bool isended() const; //проверка на конец

void delet(node<val> \*l);

val & getdata() const;

node<val>\* search(const val &d);

void insert\_to\_tail(const val &d); //вставка в конец

void insert\_up(const val &d); //вставка в упорядоченный список

bool operator==(const ringlist<val> &l) const; //сравнение списков

const ringlist<val> & operator=(const ringlist<val> &l);

friend ostream& operator<<(ostream &out, const ringlist<val> &n)

{

node<val>\* t = n.head->next;

if (t == n.head)

out << "0";

while (t != n.head)

{

out << t->data << " ";

t = t->next;

}

return out;

}

};

template<class val>

inline ringlist<val>::ringlist()

{

head = new node<val>();

tail = head;

head->next = head;

}

template<class val>

inline ringlist<val>::ringlist(const ringlist<val>& l)

{

head = new node<val>();

node<val>\* t = head;

node<val>\* tt = l.head->next;

while (tt != l.head)

{

t->next = new node<val>(tt->data);

tail = t->next;

t = t->next;

tt = tt->next;

}

tail->next = head;

}

template<class val>

inline ringlist<val>::~ringlist()

{

clean();

delete head;

}

template<class val>

inline void ringlist<val>::clean()

{

node<val>\* t = head->next;

while (t != head)

{

node<val>\* tt = t->next;

delete t;

t = tt;

}

head->next = head;

tail = head;

}

template<class val>

inline void ringlist<val>::delet(node<val> \*l)//удаление звена списка

{

node<val> \*t = head;

while (t->next != l)

t = t->next;

t->next = l->next;

delete l;

}

template<class val>

inline void ringlist<val>::reset()

{

current = head;

}

template<class val>

inline void ringlist<val>::getnext()

{

current = current->next;

}

template<class val>

inline bool ringlist<val>::isended() const

{

return (current == head);

}

template<class val>

inline val & ringlist<val>::getdata() const

{

return current->data;

}

template<class val>

inline node<val>\* ringlist<val>::search(const val &d)//поиск звена с даными d

{

node<val>\* t = head->next;

while ((t != head) && (t->data != d))

{

t = t->next;

}

if (t == head)

return NULL;

else

return t;

}

template<class val>

inline void ringlist<val>::insert\_to\_tail(const val & d)

{

node<val>\* t = new node<val>(d);

tail->next = t;

tail = t;

tail->next = head;;

}

template<class val>

inline void ringlist<val>::insert\_up(const val & d)

{

node<val>\* t = new node<val>(d);

node<val>\* tt = head->next;

node<val>\* pred = head;

while ((tt != head) && (tt->data > d))

{

pred = tt;

tt = tt->next;

}

t->next = tt;

pred->next = t;

if (tt == head)

{

tail = t;

}

}

template<class val>

inline bool ringlist<val>::operator==(const ringlist<val> &l) const

{

bool flag = true;

if (this != &l)

{

node<val>\* t = head->next;

node<val>\* tt = l.head->next;

while (t->data == tt->data && t != head && tt != l.head)

{

t = t->next;

tt = tt->next;

}

if (t != head || tt != l.head)

flag = false;

}

return flag;

}

template<class val>

const ringlist<val> & ringlist<val>::operator=(const ringlist<val> &l)

{

clean();

head = new node<val>();

node<val>\* t = head;

node<val>\* tt = l.head->next;

tail = head;

while (tt != l.head)

{

t->next = new node<val>(tt->data);

tail = t->next;

t = t->next;

tt = tt->next;

}

tail->next = head;

return \*this;

}

monom.h

#pragma once

#include "node.h"

#include "list.h"

class monom

{

public:

double coeff;

unsigned int abc;

monom(const double cf = 0.0, const unsigned int abc2 = 0);

monom(const monom &n);

monom(const string &s);

~monom() {}

const monom& operator=(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;

monom operator\*(const double d) const;

monom operator\*(const monom &m) const;

friend monom operator\*(const double d, const monom &m)

{

return (m\*d);

}

friend ostream & operator<<(ostream &os, const monom &n);

};

node.h

#pragma once

#include <stdlib.h>

#include <iostream>

#include <string>

using namespace std;

template <class val>

class node

{

public:

val data;

node<val>\* next;

node()

{

next = NULL;

}

node(const node<val> &n);

node(const val & d);

~node()

{

}

const node<val>&operator=(const node<val> &n);

bool operator==(const node<val> &n) const;

friend ostream & operator<<(ostream &os, const node<val> &n)

{

os << n.data;

return os;

}

};

template<class val>

inline node<val>::node(const node<val> &n)

{

data = n.data;

next = n.next;

}

template<class val>

inline node<val>::node(const val &d)

{

data = d;

next = NULL;

}

template<class val>

inline const node<val>& node<val>::operator=(const node<val> &n)

{

data = n.data;

next = n.next;

return \*this;

}

template<class val>

inline bool node<val>::operator==(const node<val>& n) const

{

if ((data == n.data) && (next == n.next))

return true;

else

return false;

}

polinom.h

#pragma once

#include "monom.h"

class polinom

{

ringlist<monom> pol;

public:

polinom() {}

polinom(const polinom &p);

polinom(const string &s);

~polinom() {}

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

polinom operator+(const polinom &p) const;

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

polinom operator\*(const double d) const;

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

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

friend polinom operator\*(const double d, const polinom &p)

{

return (p\*d);

}

friend ostream& operator<<(ostream& out, const polinom &p);

};

polinom.cpp

#include "monom.h"

#include "polinom.h"

monom::monom(const double cf, const unsigned int abc2)

{

coeff = cf;

abc = abc2;

}

monom::monom(const monom &n)

{

coeff = n.coeff;

abc = n.abc;

}

monom::monom(const string &s)//создание монома

{

string c = s + ' ';

bool f = true;

int min = 1, i;

abc = 0;

string cof;

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

{

min = -1;

c.erase(0, 1); //удаление элемента

}

int len = c.length();

for (i = 0; f; i++)

{

if (((c[i] >= '0') && (c[i] <= '9')) || (c[i] == '.'))

{

cof = cof + c[i];

}

else

{

f = false;

if (cof != "")

{

double cof1 = atof(cof.c\_str());

coeff = min \* cof1;

}

else

coeff = min;

}

}

for (int j = i - 1; j<len; j++)

{

if (c[j] == 'x')

{

if ((c[j + 1] >= '0') && (c[j + 1] <= '9'))

{

abc = abc + 100 \* (c[j + 1] - '0');

j++;

}

else

abc = abc + 100;

}

else

if (c[j] == 'y')

{

if ((c[j + 1] >= '0') && (c[j + 1] <= '9'))

{

abc = abc + 10 \* (c[j + 1] - '0');

j++;

}

else

abc = abc + 10;

}

else

if (c[j] == 'z')

{

if ((c[j + 1] >= '0') && (c[j + 1] <= '9'))

{

abc = abc + (c[j + 1] - '0');

j++;

}

else

abc = abc + 1;

}

}

}

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

{

coeff = m.coeff;

abc = m.abc;

return \*this;

}

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

{

if (abc == m.abc)

return true;

else

return false;

}

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

{

if (abc != m.abc)

return true;

else

return false;

}

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

{

if (abc<m.abc)

return true;

else

return false;

}

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

{

if (abc>m.abc)

return true;

else

return false;

}

monom monom::operator\*(const double d) const

{

monom t(coeff\*d, abc);

return t;

}

monom monom::operator\*(const monom &m) const

{

if (abc + m.abc > 999)

throw "error";

else

{

monom t(coeff \* m.coeff, abc + m.abc);

return t;

}

}

ostream & operator<<(ostream &os, const monom &n)

{

os << n.coeff << "\*x" << n.abc / 100 << "\*y" << (n.abc / 10) % 10 << "\*z" << n.abc % 10;

return os;

}

polinom::polinom(const polinom &p)

{

pol = p.pol;

}

polinom::polinom(const string &s)

{

string c = s + '+';

int len = c.length();

bool min = false;

string mon;

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

{

if ((c[i] != '+') && (c[i] != '-'))

{

mon = mon + c[i];

}

else

{

if (mon != "")

{

if (min)

{

mon = '-' + mon;

}

monom b(mon);

mon = "";

node<monom>\* t = pol.search(b);//ищем звено с данными b

if (t != NULL)

{

t->data.coeff += b.coeff; // считаем подобные

}

else

pol.insert\_up(b);

}

if (c[i] == '-')

min = true;

else

min = false;

}

}

}

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

{

pol = p.pol;

return \*this;

}

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

{

ringlist<monom> pol1 = pol;

ringlist<monom> thispol = p.pol;

pol1.reset(); //запись в head

thispol.reset();

pol1.getnext();

thispol.getnext();

polinom summa;

while ((!pol1.isended()) && (!thispol.isended())) //пока полиномы не закончились

{

monom a(pol1.getdata());

monom b(thispol.getdata());

if (a > b)

{

summa.pol.insert\_to\_tail(a);

pol1.getnext();

}

else

if (a < b)

{

summa.pol.insert\_to\_tail(b);

thispol.getnext();

}

else

{

monom c(a.coeff + b.coeff, a.abc);

if (c.coeff != 0)

summa.pol.insert\_to\_tail(c);

pol1.getnext();

thispol.getnext();

}

}

while (!pol1.isended())

{

summa.pol.insert\_to\_tail(pol.getdata());

pol1.getnext();

}

while (!thispol.isended())

{

summa.pol.insert\_to\_tail(thispol.getdata());

thispol.getnext();

}

return summa;

}

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

{

ringlist<monom> pol1 = pol;

ringlist<monom> thispol = p.pol;

pol1.reset();

thispol.reset();

pol1.getnext();

thispol.getnext();

polinom mult;

while (!pol1.isended())

{

while (!thispol.isended())

{

monom c(pol1.getdata() \* thispol.getdata());

if (c.coeff != 0.0)

{

node<monom>\* t = mult.pol.search(c);

if (t != NULL)

{

t->data.coeff += c.coeff;

if (t->data.coeff == 0.0)

mult.pol.delet(t);

}

else

{

mult.pol.insert\_up(c);

}

}

thispol.getnext();

}

pol1.getnext();

thispol.reset();

thispol.getnext();

}

return mult;

}

polinom polinom::operator\*(const double d) const

{

ringlist<monom> pol1 = pol;

polinom multip;

if (d != 0)

{

pol1.reset();

pol1.getnext();

while (!pol1.isended())

{

multip.pol.insert\_to\_tail(pol1.getdata()\*d);

pol1.getnext();

}

}

return multip;

}

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

{

if (this != &p)

{

bool flag = true;

polinom pol1(\*this), pol2(p);

pol1.pol.reset();

pol1.pol.getnext();

pol2.pol.reset();

pol2.pol.getnext();

while ((flag) && (!pol1.pol.isended()) && (!pol2.pol.isended()))

{

monom a = pol1.pol.getdata();

monom b = pol2.pol.getdata();

if (a == b)

{

if (a.coeff != b.coeff)

flag = false;

}

else

flag = false;

pol1.pol.getnext();

pol2.pol.getnext();

}

if ((!pol1.pol.isended()) || (!pol2.pol.isended()))

flag = false;

return flag;

}

else

return true;

}

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

{

return !(\*this == p);

}

ostream & operator<<(ostream & out, const polinom &p)

{

out << p.pol;

return out;

}