Министерство образования и науки Российской Федерации

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

Институт Информационных технологий, математики и механики

Отчёт по лабораторной работе

Операции над полиномами

Выполнил:

студент гр. 381603-01

Жирнов В.Г

Проверил:

ассистент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2018 г

Содержание

[Введение 3](#_Toc515188282)

[1 Постановка задачи 4](#_Toc515188283)

[2 Руководство пользователя 5](#_Toc515188284)

[3 Руководство программиста 6](#_Toc515188285)

[3.1 Структура программы 6](#_Toc515188286)

[3.2 Структуры данных 6](#_Toc515188287)

[3.2.1 Шаблонный класс Node 6](#_Toc515188288)

[3.2.2 Шаблонный класс list 7](#_Toc515188289)

[3.2.3 Класс monom 7](#_Toc515188290)

[3.2.4 Класс polinom 8](#_Toc515188291)

[3.3 Описание алгоритмов 9](#_Toc515188292)

[3.3.1 Реализация циклического односвязного списка 9](#_Toc515188293)

[3.3.2 Алгоритмы класса monom 9](#_Toc515188294)

[3.3.3 Алгоритмы класса polinom 9](#_Toc515188295)

[Заключение 11](#_Toc515188296)

[Литература 12](#_Toc515188297)

[Приложение 13](#_Toc515188298)

[Приложение 1. Программная реализация циклического списка 13](#_Toc515188299)

[Приложение 2. Программная реализация полиномов 16](#_Toc515188300)

# Введение

Таблица – это структура данных, элементом которой является запись.

Понятие таблицы

**Таблица**— набор элементов одинаковой организации, каждый из которых можно представить в виде двойки <K, V>, где K — ключ, а V — тело (информационная часть) элемента.

Ключ уникален для каждого элемента, то есть в таблице нет элементов с одинаковыми ключами. Ключ используется для доступа к элементам при выполнении операций.

Виды таблиц

Существует три вида таблиц: **неупорядоченная, упорядоченная и хеш-таблица.**Рассмотрим подробнее каждую из них.

Неупорядоченная таблица

Элементы такой таблицы не упорядочены по значению ключа. Для поиска элемента с заданным ключом используется алгоритм **линейного** или**быстрого линейного поиска**

Упорядоченная таблица

В такой таблице все элементы **упорядочиваются по возрастанию** значения ключа. Для поиска элемента с заданным ключом применяются алгоритмы **линейного, бинарного** или **блочного** **поиска**. После выполнения операции включения, элементы таблицы должны остаться упорядоченными, поэтому перед выполнением операции включения необходимо найти элемент, после которого нужно включить элемент.

Хеш-таблица

Хеш-таблица — это таблица, в которой положение адреса элемента определяется с помощью некоторой функции **H**(хеш-функции), аргументом которой является значение ключа элемента.

# Постановка задачи

# Руководство пользователя

# Руководство программиста

## Структура программы

## Структуры данных

### Шаблонный класс Node

Поля(public):

valtype data; данные, хранящиеся в элементе списка

node <valtype> \*next; указатель на следующий элемент списка

Методы(public):

node() { next = NULL; } конструктор списка по умолчанию

node(const node<valtype> &n); конструктор копирования

node(const valtype &c); конструктор списка с известными данными

~node() {} конструктор деструктор

const node<valtype>& operator=(const node<valtype> &n); перегрузка присваивания

bool operator==(const node<valtype> &n) const; перегрузка сравнения

### Шаблонный класс list

Поля(private):

node<valtype> \*head; указатель на голову

node<valtype> \*tail; указатель на последний элемент списка

node<valtype> \*current; указатель на текущий элемент списка

Методы(public):

list(); конструктор по умолчанию

list(const list<valtype> &l); конструктор копирования

~list(); деструктор

void gotohead(); передвигает current на голову

void gotonext(); передвигает current на следующий элемент

bool currhead() const; флаг: равен ли current голове?

void clean(); очистка списка

void del(node<valtype> \*n); поиск и удаление данного звена из списка

void insert\_to\_tail(const valtype &d); упорядоченная вставка в конец списка

void insertup(const valtype &d); упорядоченная вставка в начало списка

valtype& getcurdata() const; вывод данных current

node<valtype>\* search(const valtype &d); поиск звена с указанными данными

const list<valtype>& operator=(const list<valtype> &l); перегрузка присваивания

friend ostream & operator<<(ostream &out, const list<valtype> &l) вывод листа на экран

### Класс monom

Поля(public):

double coeff; коэффициент монома

unsigned int xyz; трехзначная переменная, отвечающая за степени переменных монома, первая цифра этого числа – степень х, вторая – степень y, третья – степень z

Методы(public):

monom(const double a = 0.0, const unsigned int b = 0); конструктор по умолчанию

monom(const monom &m); конструктор копирования

monom(const string &s); конструктор, преобразующий введенную строку в моном

~monom() {} деструктор

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

bool operator==(const monom &m) const; перегрузка сравнения

bool operator!=(const monom &m) const; перегрузка сравнения

bool operator<(const monom &m) const; перегрузка сравнения

bool operator>(const monom &m) const; перегрузка сравнения

monom operator\*(const double d) const; перегрузка умножения монома на константу справа

friend monom operator\*(const double d, const monom &m) { return (m \* d); } перегрузка умножения монома на константу слева

monom operator\*(const monom &m) const; перегрузка умножения монома на моном

friend ostream & operator<<(ostream &out, const monom &m); вывод монома на экран

### Класс polinom

Поля(private)

list<monom> pol; циклический список мономов

Методы(public)

polinom() {} конструктор по умолчанию

polinom(const polinom &p); конструктор копирования

polinom(const string &s); конструктор, преобразующий полученную на вход строку в полином

~polinom() {} деструктор

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

polinom operator+(const polinom &p) const; перегрузка сложения полиномов

polinom operator\*(const polinom &p) const; перегрузка умножения полиномов

polinom operator\*(const double a) const; перегрузка умножения полинома на константу справа

friend polinom operator\*(const double d, const polinom &p) { return (p\*d); } перегрузка умножения полинома на константу слева

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

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

friend ostream & operator<<(ostream &out, const polinom &p); вывод полинома на экран

## **Описание алгоритмов**

# Заключение

В данной лабораторный работе был реализован класс полиномов и перегружены операции над ними (+, \*). Программа преобразует строки в полиномы и способна выполнять перечисленные выше операции для них.

# Литература

1. Лекции «Алгоритмы и структуры данных» Гергель В.П. ННГУ им. Лобачевского Н. И.

# Приложение

## Приложение 1. Программная реализация циклического списка

\\-------node.h-------------

#pragma once

#include <iostream>

#include <string>

using namespace std;

template <class valtype>

class node

{

public:

valtype data;

node <valtype> \*next;

node() { next = NULL; }

node(const node<valtype> &n);

node(const valtype &c);

~node() {}

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

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

};

template<class valtype>

node<valtype>::node(const node<valtype>& n)

{

data = n.data;

next = n.next;

};

template<class valtype>

node<valtype>::node(const valtype &c)

{

data = c;

next = NULL;

};

template<class valtype>

const node<valtype>& node<valtype>::operator=(const node<valtype> &n)

{

data = n.data;

next = n.next;

return \*this;

};

template<class valtype>

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

{

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

return true;

else

return false;

};

\\---------------------list.h----------------------

#pragma once

#include "node.h"

template <class valtype>

class list

{

node<valtype> \*head;

node<valtype> \*tail;

node<valtype> \*current;

public:

list();

list(const list<valtype> &l);

~list();

void gotohead();

void gotonext();

bool currhead() const;

void clean();

void del(node<valtype> \*n);

void insert\_to\_tail(const valtype &d);

void insertup(const valtype &d);

valtype& getcurdata() const;

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

const list<valtype>& operator=(const list<valtype> &l);

friend ostream & operator<<(ostream &out, const list<valtype> &l)

{

node<valtype>\* t = l.head->next;

if (t == l.head)

out << "0";

while (t != l.head)

{

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

t = t->next;

}

return out;

}

};

template<class valtype>

list<valtype>::list()

{

head = new node<valtype>();

tail = head;

head->next = head;

};

template<class valtype>

list<valtype>::list(const list<valtype>& l)

{

head = new node<valtype>();

node<valtype>\* k = head;

node<valtype>\* kk = l.head->next;

while (kk != l.head)

{

k->next = new node<valtype>(kk->data);

tail = k->next;

k = k->next;

kk = kk->next;

}

tail->next = head;

};

template<class valtype>

list<valtype>::~list()

{

clean();

delete head;

};

template<class valtype>

void list<valtype>::gotohead()

{

current = head;

};

template<class valtype>

void list<valtype>::gotonext()

{

current = current->next;

};

template<class valtype>

bool list<valtype>::currhead() const

{

return (current == head);

};

template<class valtype>

void list<valtype>::clean()

{

node<valtype>\* k = head->next;

while (k != head)

{

node<valtype>\* kk = k->next;

delete k;

k = kk;

}

head->next=head;

tail = head;

};

template<class valtype>

void list<valtype>::del(node<valtype> \*n)

{

node<valtype> \*k = head;

while (k->next != n)

k = k->next;

k->next = n->next;

delete n;

};

template<class valtype>

void list<valtype>::insert\_to\_tail(const valtype & d)

{

node<valtype>\* k = new node<valtype>(d);

tail->next = k;

tail = k;

tail->next = head;

};

template<class valtype>

void list<valtype>::insertup(const valtype & d)

{

node<valtype>\* k = head->next;

node<valtype>\* kk = head;

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

{

kk = k;

k = k->next;

}

node<valtype>\* kkk = new node<valtype>(d);

kkk->next = k;

kk->next = kkk;

if (k == head)

tail = kkk;

};

template<class valtype>

valtype & list<valtype>::getcurdata() const

{

return current->data;

};

template<class valtype>

node<valtype>\* list<valtype>::search(const valtype & d)

{

node<valtype> \*k = head->next;

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

k = k->next;

if (k != head)

return k;

else

return NULL;

};

template<class valtype>

const list<valtype>& list<valtype>::operator=(const list<valtype>& l)

{

clean();

head = new node<valtype>();

node<valtype>\* k = head;

node<valtype>\* kk = l.head->next;

tail = head;

while (kk != l.head)

{

k->next = new node<valtype>(kk->data);

tail = k->next;

k = k->next;

kk = kk->next;

}

tail->next = head;

return \*this;

};

## Приложение 2. Программная реализация полиномов

\\-------------monom.h-------------

#pragma once

#include "list.h"

#include "node.h"

using namespace std;

class monom

{

public:

double coeff;

unsigned int xyz;

monom(const double a = 0.0, const unsigned int b = 0);

monom(const monom &m);

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;

friend monom operator\*(const double d, const monom &m) { return (m \* d); }

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

friend ostream & operator<<(ostream &out, const monom &m);

};

\\---------polinom.h-------------

#pragma once

#include "monom.h"

class polinom

{

list<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 a) const;

friend polinom operator\*(const double d, const polinom &p) { return (p\*d); }

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

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

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

};

\\----------------polinom.cpp-----------------------

#include "polinom.h"

monom::monom(const double a, const unsigned int b)

{

coeff = a;

xyz = b;

}

monom::monom(const monom &m)

{

coeff = m.coeff;

xyz = m.xyz;

}

monom::monom(const string & s)

{

string c = s + ' ';

string str;

xyz = 0;

bool flag = true;

int i, j = 1;

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

{

j = -1;

c.erase(0, 1);

}

int len = c.length();

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

{

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

str = str + c[i];

else

{

flag = false;

if (str != "")

coeff = j\*atof(str.c\_str());

else

coeff = j;

}

}

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

{

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

{

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

{

xyz += 100 \* (c[k + 1] - '0');

k++;

}

else

xyz += 100;

}

else

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

{

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

{

xyz += 10 \* (c[k + 1] - '0');

k++;

}

else

xyz += 10;

}

else

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

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

{

xyz += (c[k + 1] - '0');

k++;

}

else

xyz += 1;

}

}

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

{

coeff = m.coeff;

xyz = m.xyz;

return \*this;

}

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

{

return (xyz == m.xyz);

}

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

{

return (xyz != m.xyz);

}

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

{

return (xyz < m.xyz);

}

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

{

return (xyz > m.xyz);

}

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

{

monom m(coeff\*d, xyz);

return m;

}

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

{

if (xyz + m.xyz > 999)

throw "Error <can't multiply>";

else

{

monom mon(coeff \* m.coeff, xyz + m.xyz);

return mon;

}

}

polinom::polinom(const polinom &p)

{

pol = p.pol;

}

polinom::polinom(const string &s)

{

string c = s + '+';

string mon;

bool min = false;

int len = c.length();

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

{

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

mon = mon + c[i];

else

{

if (mon != "")

{

if (min)

mon = '-' + mon;

monom m(mon);

mon = "";

node<monom>\* k = pol.search(m);

if (k == NULL)

pol.insertup(m);

else

k->data.coeff += m.coeff;

}

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

{

list<monom> currpol = pol;

list<monom> thispol = p.pol;

currpol.gotohead();

thispol.gotohead();

currpol.gotonext();

thispol.gotonext();

polinom pp;

while ((!currpol.currhead()) && (!thispol.currhead()))

{

if (currpol.getcurdata() > thispol.getcurdata())

{

pp.pol.insert\_to\_tail(currpol.getcurdata());

currpol.gotonext();

}

else

{

if (currpol.getcurdata() < thispol.getcurdata())

{

pp.pol.insert\_to\_tail(thispol.getcurdata());

thispol.gotonext();

}

else

{

monom m(currpol.getcurdata().coeff + thispol.getcurdata().coeff, currpol.getcurdata().xyz);

if (m.coeff != 0.0)

pp.pol.insert\_to\_tail(m);

currpol.gotonext();

thispol.gotonext();

}

}

}

while (!currpol.currhead())

{

pp.pol.insert\_to\_tail(currpol.getcurdata());

currpol.gotonext();

}

while (!thispol.currhead())

{

pp.pol.insert\_to\_tail(thispol.getcurdata());

thispol.gotonext();

}

return pp;

}

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

{

list<monom> currpol=pol;

list<monom> thispol = p.pol;

currpol.gotohead();

thispol.gotohead();

currpol.gotonext();

thispol.gotonext();

polinom pp;

while (!currpol.currhead())

{

while (!thispol.currhead())

{

monom m(currpol.getcurdata() \* thispol.getcurdata());

if (m.coeff != 0.0)

{

node<monom>\* k = pp.pol.search(m);

if (k == NULL)

pp.pol.insertup(m);

else

{

k->data.coeff += m.coeff;

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

pp.pol.del(k);

}

}

thispol.gotonext();

}

currpol.gotonext();

thispol.gotohead();

thispol.gotonext();

}

return pp;

}

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

{

list<monom> currpol = pol;

polinom p;

if (a != 0)

{

currpol.gotohead();

currpol.gotonext();

while (!currpol.currhead())

{

p.pol.insert\_to\_tail(currpol.getcurdata()\*a);

currpol.gotonext();

}

}

return p;

}

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

{

if (this != &p)

{

bool flag = true;

polinom copypolinom(\*this), copyp(p);

copypolinom.pol.gotohead();

copypolinom.pol.gotonext();

copyp.pol.gotohead();

copyp.pol.gotonext();

while ((flag) && (!copypolinom.pol.currhead()) && (!copyp.pol.currhead()))

{

monom a = copypolinom.pol.getcurdata();

monom b = copyp.pol.getcurdata();

if (a == b)

{

if (a.coeff != b.coeff)

flag = false;

}

else

flag = false;

copypolinom.pol.gotonext();

copyp.pol.gotonext();

}

if ((!copypolinom.pol.currhead()) || (!copyp.pol.currhead()))

flag = false;

return flag;

}

else

return true;

}

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

{

return !(\*this == p);

}

ostream & operator<<(ostream & out, const monom & m)

{

out << m.coeff << "\*x" << m.xyz / 100 << "\*y" << (m.xyz / 10) % 10 << "\*z" << m.xyz % 10;

return out;

}

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

{

out << p.pol;

return out;

}

**Приложение 3. Основная программа**

#include "polinom.h"

using namespace std;

int main()

{

int f;

do

{

cout << "Actions with polinoms" << endl;

cout << "1 - addition of polinoms" << endl;

cout << "2 - multiplication of polinoms" << endl;

cout << "3 - multiplication of a polinom on the number" << endl;

cin >> f;

cin.ignore();

if (f == 1)

{

string s;

cout << "Write first polinom" << endl;

getline(cin, s);

polinom p1(s);

cout << "Write second polinom" << endl;

getline(cin, s);

polinom p2(s);

polinom p3 = p1 + p2;

cout << "Addition of polinoms: " << p3 << endl;

}

else

if (f == 2)

{

string s;

cout << "Write first polinom" << endl;

getline(cin, s);

polinom p1(s);

cout << "Write second polinom" << endl;

getline(cin, s);

polinom p2(s);

polinom p3 = p1 \* p2;

cout << "Multiplication of polinoms: " << p3 << endl;

}

else

if (f == 3)

{

string s;

cout << "Write a polinom" << endl;

getline(cin, s);

polinom p(s);

cout << "Write a number" << endl;

double d;

cin >> d;

p = p \* d;

cout << "Multiplication of a polinom on the number: " << p << endl;

}

else

cout << "There is no such choice" << endl;

cout << "Want to continue" << endl;

cout << "1 - yes" << endl;

cout << "2 - no" << endl;

cin >> f;

}

while (f == 1);

return 0;

}

**Приложение 4. Тесты для list и node**

#include "node.h"

#include "list.h"

#include <gtest.h>

TEST(node, can\_create\_empty)

{

ASSERT\_NO\_THROW(node<int> a);

}

TEST(node, can\_create\_copied)

{

node<int> a;

node<int> b(a);

EXPECT\_EQ(b, a);

}

TEST(node, can\_create\_with\_data)

{

node<int> a(1);

EXPECT\_EQ(a.data, 1);

}

TEST(node, can\_assign)

{

node<int> a;

node<int> b(12);

a = b;

EXPECT\_EQ(a.data, 12);

}

//========================================================================================//

TEST(list, can\_create\_empty)

{

ASSERT\_NO\_THROW(list<int> l);

}

TEST(list, can\_create\_copied)

{

list<int> a;

a.insertup(20);

ASSERT\_NO\_THROW(list<int> b(a));

}

TEST(list, can\_get\_currdata)

{

list<int> a;

a.insertup(20);

a.gotohead();

a.gotonext();

EXPECT\_EQ(a.getcurdata(), 20);

}

TEST(list, can\_insert\_up\_and\_to\_tail)

{

list<int> a;

a.insertup(11);

a.insert\_to\_tail(12);

a.gotohead();

a.gotonext();

EXPECT\_EQ(a.getcurdata(), 11);

a.gotonext();

EXPECT\_EQ(a.getcurdata(), 12);

}

class TestList : public ::testing::Test

{

protected:

list<int> l1;

list<int> l2;

public:

TestList()

{

l1.insertup(1); l2.insertup(3);

l1.insert\_to\_tail(2); l2.insert\_to\_tail(4);

}

~TestList()

{

l1.clean();

l2.clean();

}

};

TEST\_F(TestList, can\_clean)

{

ASSERT\_NO\_THROW(l1.clean());

}

TEST\_F(TestList, can\_delete\_element)

{

l1.del(l1.search(1));

l1.gotohead();

l1.gotonext();

EXPECT\_EQ(l1.getcurdata(), 2);

}

TEST\_F(TestList, can\_search\_data)

{

ASSERT\_NO\_THROW(l1.search(1));

}

TEST\_F(TestList, can\_assign)

{

l2 = l1;

l2.gotohead();

l2.gotonext();

EXPECT\_EQ(l2.getcurdata(), 1);

l2.gotonext();

EXPECT\_EQ(l2.getcurdata(), 2);

}

**Приложение 5. Тесты для polinom и monom**

#include "polinom.h"

#include <gtest.h>

TEST(monom, can\_create\_with\_null\_param)

{

monom a;

EXPECT\_EQ(a.coeff, 0.0);

EXPECT\_EQ(a.xyz, 0);

}

TEST(monom, can\_create\_copied\_monom)

{

monom a(2.0, 4);

monom b(a);

EXPECT\_EQ(b.coeff, 2.0);

EXPECT\_EQ(b.xyz, 4);

}

TEST(monom, can\_create\_from\_string)

{

string s = "xyz";

monom a(s);

EXPECT\_EQ(a.coeff, 1.0);

EXPECT\_EQ(a.xyz, 111);

}

TEST(monom, can\_assign)

{

monom a(2.0, 4);

monom b(3.0, 2);

a = b;

EXPECT\_EQ(a.coeff, 3.0);

EXPECT\_EQ(a.xyz, 2);

}

TEST(monom, work\_equall)

{

monom a(2.0, 4);

monom b(2.0, 4);

EXPECT\_EQ(a == b, true);

}

TEST(monom, work\_not\_equall)

{

monom a(3.0, 4);

monom b(2.0, 5);

EXPECT\_EQ(a != b, true);

}

TEST(monom, work\_less)

{

monom a(2.0, 4);

monom b(3.0, 5);

EXPECT\_EQ(a < b, true);

}

TEST(monom, work\_more)

{

monom a(3.0, 5);

monom b(2.0, 4);

EXPECT\_EQ(a > b, true);

}

TEST(monom, can\_multiply\_on\_number\_from\_right)

{

monom a(2.0, 4);

a = a \* 2;

EXPECT\_EQ(a.coeff, 4.0);

}

TEST(monom, can\_multiply\_on\_number\_from\_left)

{

monom a(2.0, 4);

a = 2 \* a;

EXPECT\_EQ(a.coeff, 4.0);

}

TEST(monom, can\_multiply\_on\_another\_monom)

{

monom a(2.0, 111);

monom b(3.0, 222);

monom c;

c = a\*b;

EXPECT\_EQ(c.coeff, 6.0);

EXPECT\_EQ(c.xyz, 333);

}

//====================================================================//

TEST(polinom, can\_create\_empty)

{

ASSERT\_NO\_THROW(polinom a);

}

TEST(polinom, can\_create\_from\_string)

{

string s = "xyz+2xy";

ASSERT\_NO\_THROW(polinom a(s));

}

TEST(polinom, can\_create\_copied\_polinom)

{

polinom a("x-y");

ASSERT\_NO\_THROW(polinom b(a));

}

TEST(polinom, can\_assign)

{

polinom a("x+y");

polinom b("z-x");

a = b;

EXPECT\_EQ(a, b);

}

TEST(polinom, right\_addition\_1)

{

polinom a("x+y");

polinom b("x");

polinom c("2x+y");

EXPECT\_EQ(c, a + b);

}

TEST(polinom, right\_addition\_2)

{

polinom a("-x+y");

polinom b("x");

polinom c("y");

EXPECT\_EQ(c, a + b);

}

TEST(polinom, right\_addition\_3)

{

polinom a("1+x+y");

polinom b("x");

polinom c("1+2x+y");

EXPECT\_EQ(c, a + b);

}

TEST(polinom, right\_addition\_4)

{

polinom a("xy");

polinom b("xyz-2x2yz");

polinom c("xy+xyz-2x2yz");

EXPECT\_EQ(c, a + b);

}

TEST(polinom, right\_addition\_5)

{

polinom a("x+y");

polinom b("-x-y");

polinom c;

EXPECT\_EQ(c, a + b);

}

TEST(polinom, right\_multiply\_on\_another\_polinom\_1)

{

polinom a("x");

polinom b("y");

polinom c("xy");

EXPECT\_EQ(c, a\*b);

}

TEST(polinom, right\_multiply\_on\_another\_polinom\_2)

{

polinom a("x-y");

polinom b("x+y");

polinom c("x2-y2");

EXPECT\_EQ(c, a \* b);

}

TEST(polinom, right\_multiply\_on\_another\_polinom\_3)

{

polinom a("x");

polinom b("-x");

polinom c("-x2");

EXPECT\_EQ(c, a \* b);

}

TEST(polinom, right\_multiply\_on\_number\_1)

{

polinom a("x-y");

double b = 10.0;

polinom c("10x-10y");

EXPECT\_EQ(c, a \* b);

}

TEST(polinom, right\_multiply\_on\_number\_2)

{

polinom a("x-y");

double b = -10.0;

polinom c("-10x+10y");

EXPECT\_EQ(c, a \* b);

}

TEST(polinom, right\_multiply\_on\_number\_3)

{

polinom a("x-y");

double b = -10.0;

polinom c("-10x+10y");

EXPECT\_EQ(c, b \* a);

}