МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

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**

Направление подготовки: «Программная инженерия»

Профиль подготовки: «Общий»

**Отчет по лабораторной работе**

на тему:

**«Полином»**

**Выполнил:** студент группы 3822Б1ПР2

Страхов Андрей Андреевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

Нижний Новгород  
2023

**Содержание**

[1. ВВЕДЕНИЕ 3](#_Toc14)

[2. ПОСТАНОВКА ЗАДАЧИ 3](#_Toc15)

[3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 5](#_Toc16)

[4. РУКОВОДСТВО ПРОГРАММИСТА 6](#_Toc17)

[4.1 Описание структуры программы 6](#_Toc18)

[4.2 Описание структур данных 6](#_Toc19)

[5. ЭКСПЕРИМЕНТЫ 10](#_Toc20)

[6. ЗАКЛЮЧЕНИЕ 11](#_Toc21)

[7.ЛИТЕРАТУРА 12](#_Toc22)

[8.ПРИЛОЖЕНИЕ 13](#_Toc23)

# 1. ВВЕДЕНИЕ

Полином – это математическое выражение, которое содержит одну или несколько переменных, а также коэффициенты и степени этих переменных.

Полином (многочлен) от n переменных – это выражение вида

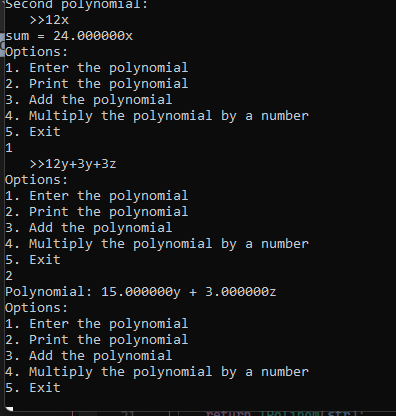
# Мы напишем собственный класс для работы с полиномами, так как язык C++ не включает в себя встроенного класса для их представления. Создание этого класса позволит проводить разнообразные математические операции, такие как сложение, вычитание, умножение и деление, в контексте программирования на C++.

# 2. ПОСТАНОВКА ЗАДАЧИ

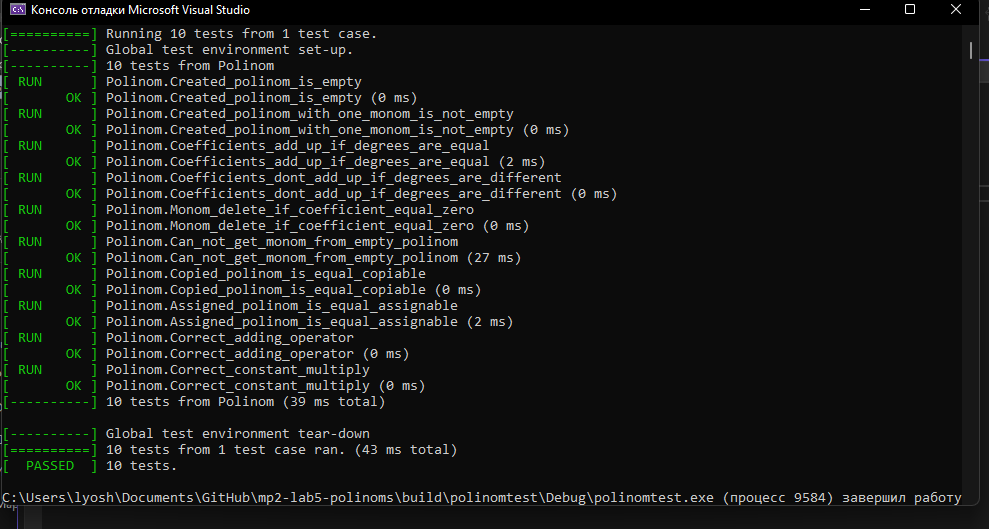
# Цель данной работы заключается в создании шаблонного класса для полиномов и реализации методов для копирования, присваивания, сравнения, а также операций сложения и вычитания.

# 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При запуске файла TPolinom\_Test.cpp из проекта polinom открывается консоль с текстовым меню, которое предлагает сделать выбор между несколькими вариантами, показывающими работоспособность класса.



При запуске polinomtest выводится список выполненных тестов.



# 4. РУКОВОДСТВО ПРОГРАММИСТА

## 4.1 Описание структуры программы

Программа состоит из нескольких проектов:

* Проект polinom, содержащий файл TPolinom\_Test.cpp, где находится проверка работоспособности очереди.
* Проект listlib, содержащий файлы TNode, TList и THeadList, реализовывающие шаблонные классы списка и элемента списка.
* Проект polinomlib, содержащий файлы TMonom и TPolinom, где находится реализация классов монома и полинома.
* Проект polinomtest, содержащий тесты для проверки корректности работы класса полинома. Реализован с помощью фреймворка GoogleTest.

## 4.2 Описание структур данных

**TNode**

template<class T>

struct TNode

{

T value; //значение

TNode\* pNext; //указатель на следующий элемент

};

**TList**

template<class T>

class TList

{

protected:

TNode<T>\* pFirst; // первое звено

TNode<T>\* pCurrent; // текущее звено

TNode<T>\* pPrevious; // звено перед текущим

TNode<T>\* pLast; // последнее звено

TNode<T>\* pStop; // значение указателя, означающего конец списка

int length; // количество звеньев в списке

public:

TList(); //конструктор

~TList(); //декструктор

int GetLength() { return length; }//установить длину

bool IsEmpty(); // список пуст ?

// вставка звеньев

void InsertFirst(T item); // перед первым

void InsertCurrent(T item); // перед текущим

void InsertLast(T item); // вставить последним

// удаление звеньев

void DeleteFirst(); // удалить первое звено

void DeleteCurrent(); // удалить текущее звено

void GoNext(); // сдвиг вправо текущего звена

// (=1 после применения GoNext для последнего звена списка)

void Reset(); // установить на начало списка

bool IsEnd(); // список завершен ?

T GetCurrentItem(); //получить текущее значение

void SetCurrentItem(T item) { pCurrent->value = item; } //установить текущее значение

friend ostream& operator<<(ostream& os, const TList<T>& obj); //вывод

};

**THeadList**

template<class T>

class THeadList : public TList<T>

{

protected:

TNode<T>\* pHead; // заголовок, pFirst - звено за pHead

public:

THeadList(); //конструктор

~THeadList(); //деструктор

void InsertFirst(T item); // вставка звеньев после заголовка

void DeleteFirst(); // удалить первое звено

void Clear(); //очистить список

void Reset(); // установить на начало списка

void GoNext(); //перейти к следующему члену

};

**TMonom**

struct TMonom

{

double coef; // коэффициент монома

int index; // индекс (свертка степеней)

TMonom() { } //конструктор

TMonom(double coef, int degX, int degY, int degZ); //конструктор

void SetCoef(int cval); //установить коэффициент

int GetCoef(void); //получить коэффициент

void SetIndex(int ival); //установить индекс

int GetIndex(void); //получить индекс

bool operator==(const TMonom& other); //оператор сравнения

bool operator>(const TMonom& other); //оператор сравнения

bool operator<(const TMonom& other); //оператор сравнения

TMonom operator=(const TMonom& other); //оператор присваивания

friend ostream& operator<<(ostream& os, const TMonom& obj); //вывод

};

**TPolinom**

class TPolinom : public THeadList<TMonom>

{

public:

TPolinom(); //конструктор

TPolinom(TPolinom& other); //конструктор копирования

TPolinom(string str); //конструктор

TPolinom& operator=(TPolinom& other); // присваивание

TPolinom& operator+(TPolinom& q); // сложение полиномов

void AddMonom(TMonom newMonom); // добавление монома

TPolinom MultMonom(TMonom monom); // умножение мономов

TPolinom AddPolinom(TPolinom& other); // добавление полинома

TPolinom operator\*(double coef); // умножение полинома на число

bool operator==(TPolinom& other); // сравнение полиномов на равенство

string ToString(); // перевод в строку

};

# 5. ЭКСПЕРИМЕНТЫ

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 11

2. Процессор: Intel Core i5 8400, 3800 МГц, я: 6, лог пр: 6

3. Версия Visual Studio: 2022

|  |  |
| --- | --- |
| Кол-во элементов (n) | Время работы оператора добавления элементов в полином (ms) |
| 1000 | 9 |
| 4000 | 201 |
| 8000 | 783 |

Возрастание количества элементов коррелирует возрастанию времени работы оператора, причём можно увидеть, что при увеличении элементов в х раз время работы вырастает в х^2 раз, что соответствует расчётной сложности O(n2).

# 6. ЗАКЛЮЧЕНИЕ

В ходе работы были созданы шаблонные классы списков. Были реализованы классы монома и полинома. Написаны и пройдены тесты, создано консольное приложение.

# 7.ЛИТЕРАТУРА

[Многочлен — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD)

[Список (информатика) — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))

[Pract\_ADS.pdf - Google Диск](https://drive.google.com/file/d/1aZEfnRSgA7IhMccdLTvbJ5noZlXPC1BK/view)

Лабораторный практикум. Составители: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. – 105с.

# 8.ПРИЛОЖЕНИЕ

**TNode.h**

template<class T>

struct TNode

{

T value;

TNode\* pNext;

};

**TList.h**

#pragma once

#include "TNode.h"

#include <iostream>

using namespace std;

template<class T>

class TList

{

protected:

TNode<T>\* pFirst; // первое звено

TNode<T>\* pCurrent; // текущее звено

TNode<T>\* pPrevious; // звено перед текущим

TNode<T>\* pLast; // последнее звено

TNode<T>\* pStop; // значение указателя, означающего конец списка

int length; // количество звеньев в списке

public:

TList();

~TList();

int GetLength() { return length; }

bool IsEmpty(); // список пуст ?

// вставка звеньев

void InsertFirst(T item); // перед первым

void InsertCurrent(T item); // перед текущим

void InsertLast(T item); // вставить последним

// удаление звеньев

void DeleteFirst(); // удалить первое звено

void DeleteCurrent(); // удалить текущее звено

void GoNext(); // сдвиг вправо текущего звена

// (=1 после применения GoNext для последнего звена списка)

void Reset(); // установить на начало списка

bool IsEnd(); // список завершен ?

T GetCurrentItem();

void SetCurrentItem(T item) { pCurrent->value = item; }

};

template <class T>

TList<T>::TList(): pFirst(nullptr), pCurrent(nullptr), pPrevious(nullptr), pLast(nullptr), pStop(nullptr), length(0) { }

template <class T>

TList<T>::~TList()

{

while (!IsEmpty()) {

DeleteFirst();

}

}

template <class T>

bool TList<T>::IsEmpty()

{

return pFirst == nullptr;

}

template <class T>

void TList<T>::InsertFirst(T item)

{

TNode<T>\* New\_Node = new TNode<T>{ item, pFirst };

pFirst = New\_Node;

if (length == 0){ pLast = pFirst}

length++;

}

template <class T>

void TList<T>::InsertLast(T item)

{

TNode<T>\* New\_Node = new TNode<T>{ item, nullptr };

if (IsEmpty()) { pFirst = pLast = New\_Node; }

else {

pLast->pNext = New\_Node;

pLast = New\_Node;

}

length++;

}

template <class T>

void TList<T>::InsertCurrent(T item)

{

if (pCurrent == nullptr || pCurrent == pFirst) {

InsertFirst(item);

return;

}

TNode<T>\* New\_Node = new TNode<T>{ item, pCurrent };

pPrevious->pNext = New\_Node;

length++;

}

template <class T>

void TList<T>::DeleteFirst()

{

if (IsEmpty()) throw runtime\_error("List is empty");

TNode<T>\* temp = pFirst;

pFirst = pFirst->pNext;

if (pFirst == nullptr) pLast = nullptr;

delete temp;

length--;

}

template <class T>

void TList<T>::DeleteCurrent()

{

if (pCurrent == nullptr) throw runtime\_error("Current node is null");

if (pCurrent == pFirst) {

DeleteFirst();

return;

}

if (pCurrent == pLast) {

delete pCurrent;

pPrevious->pNext = nullptr;

pLast = pPrevious;

pCurrent = nullptr;

}

else {

pPrevious->pNext = pCurrent->pNext;

delete pCurrent;

pCurrent = pPrevious->pNext;

}

}

template <class T>

T TList<T>::GetCurrentItem()

{

if (pCurrent == nullptr) throw runtime\_error("Current node is null");

return pCurrent->value;

}

template <class T>

void TList<T>::Reset()

{

pCurrent = pFirst;

pPrevious = nullptr;

}

template <class T>

void TList<T>::GoNext()

{

pPrevious = pCurrent;

pCurrent = pCurrent->pNext;

if (pCurrent == nullptr || pCurrent == pStop) { pLast = pPrevious;}

}

template <class T>

bool TList<T>::IsEnd()

{

return pCurrent == pStop;

}

**THeadList.h**

#pragma once

#include "TList.h"

using namespace std;

template<class T>

class THeadList : public TList<T>

{

protected:

TNode<T>\* pHead; // заголовок, pFirst - звено за pHead

public:

THeadList();

~THeadList();

void InsertFirst(T item); // вставка звеньев после заголовка

void DeleteFirst(); // удалить первое звено

};

template<class T>

THeadList<T>::THeadList()

{

pHead = new TNode<T>();

if (pHead == nullptr) throw bad\_alloc();

this->pLast = pHead;

pHead->pNext = this->pFirst;

}

template<class T>

THeadList<T>::~THeadList()

{

delete pHead;

}

template <class T>

void THeadList<T>::InsertFirst(T item)

{

TNode<T>\* newNode = new TNode<T>{ item, nullptr };

if (newNode == nullptr) throw bad\_alloc();

newNode->pNext = pHead->pNext;

pHead->pNext = newNode;

if (this->IsEmpty()) {

this->pLast = newNode;

}

this->length++;

}

template <class T>

void THeadList<T>::DeleteFirst()

{

if (this->pFirst == nullptr) throw runtime\_error("List is empty");

TNode<T>\* temp = this->pFirst;

this->pFirst = this->pFirst->pNext;

if (this->pFirst == nullptr) {

this->pLast = nullptr;

}

if (this->pCurrent == temp) {

this->pCurrent = this->pFirst;

this->pPrevious = nullptr;

}

delete temp;

this->length--;

}

**TPolinom.h**

#pragma once

#include "THeadList.h"

#include "TMonom.h"

#include <string>

#include<sstream>

using namespace std;

const int nonDisplayedZeros = 4; // Количество неотображаемых нулей при выводе коэффициента полинома

// Кол-во символов после запятой = 6 - nonDisplayedZeros

class TPolinom : public THeadList<TMonom>

{

public:

TPolinom();

TPolinom(TPolinom& other);

TPolinom(string str);

TPolinom& operator=(TPolinom& other); // присваивание

TPolinom& operator+(TPolinom& q); // сложение полиномов

void AddMonom(TMonom newMonom); // добавление монома

TPolinom MultMonom(TMonom monom); // умножение мономов

TPolinom& operator\*(double coef); // умножение полинома на число

bool operator==(TPolinom& other); // сравнение полиномов на равенство

string ToString(); // перевод в строку

};

TPolinom::TPolinom() :THeadList<TMonom>::THeadList() {}

TPolinom::TPolinom(TPolinom& other)

{

pHead = new TNode<TMonom>;

TNode<TMonom>\* h = other.pHead->pNext;

while (h != other.pStop) {

this->AddMonom(h->value);

h = h->pNext;

}

}

TPolinom::TPolinom(string str) {

size\_t pos = 0;

while (pos < str.length()) {

double coef = 0.0;

int degX = 0, degY = 0, degZ = 0;

char var;

if (isdigit(str[pos]) || str[pos] == '-' || str[pos] == '+') {

size\_t nextPos;

coef = stod(str.substr(pos), &nextPos);

pos += nextPos;

}

while (pos < str.length() && (str[pos] == 'x' || str[pos] == 'X' ||

str[pos] == 'y' || str[pos] == 'Y' ||

str[pos] == 'z' || str[pos] == 'Z')) {

var = tolower(str[pos]);

pos++;

if (pos < str.length() && str[pos] == '^') {

pos++;

size\_t nextPos;

int deg = stoi(str.substr(pos), &nextPos);

pos += nextPos;

switch (var) {

case 'x': degX = deg; break;

case 'y': degY = deg; break;

case 'z': degZ = deg; break;

}

}

else {

switch (var) {

case 'x': degX = 1; break;

case 'y': degY = 1; break;

case 'z': degZ = 1; break;

}

}

}

this->AddMonom(TMonom(coef, degX, degY, degZ));

}

}

TPolinom& TPolinom::operator=(TPolinom& other)

{

if (this != &other) {

while (!this->IsEmpty()) {

this->DeleteFirst();

}

TNode<TMonom>\* current = other.pHead->pNext;

while (current != nullptr) {

this->AddMonom(current->value);

current = current->pNext;

}

}

return \*this;

}

void TPolinom::AddMonom(TMonom m)

{

if (m.coef == 0) throw invalid\_argument("Cannot add monom with a null coefficient");

this->Reset();

bool isAdded = false;

while (!this->IsEnd()) {

if (this->pCurrent->value.index == m.index) {

this->pCurrent->value.coef += m.coef;

if (fabs(this->pCurrent->value.coef) < 1e-6) {

this->DeleteCurrent();

}

isAdded = true;

break;

}

this->GoNext();

}

if (!isAdded) this->InsertLast(m);

pHead->pNext = this->pFirst;

}

TPolinom TPolinom::MultMonom(TMonom monom)

{

TPolinom res(\*this);

TNode<TMonom>\* current = res.pHead->pNext;

while (current != nullptr) {

current->value.coef \*= monom.coef;

current->value.index += monom.index;

current = current->pNext;

}

return res;

}

TPolinom& TPolinom::operator+(TPolinom& other)

{

if (other.IsEmpty()) throw invalid\_argument("Cannot add an empty polynomial");

TNode<TMonom>\* current = other.pHead->pNext;

while (current != nullptr) {

this->AddMonom(current->value);

current = current->pNext;

}

return \*this;

}

TPolinom& TPolinom::operator\*(double coef)

{

if (this->IsEmpty()) throw invalid\_argument("Cannot multiply an empty polynomial");

TNode<TMonom>\* current = this->pHead->pNext;

while (current != nullptr) {

current->value.coef \*= coef;

if (current->pNext == nullptr) break;

current = current->pNext;

}

return \*this;

}

bool TPolinom::operator==(TPolinom& other) {

if (this->GetLength() != other.GetLength()) return false;

TNode<TMonom>\* thisCurrent = this->pHead->pNext;

TNode<TMonom>\* otherCurrent = other.pHead->pNext;

while (thisCurrent != nullptr && otherCurrent != nullptr) {

if (!(thisCurrent->value == otherCurrent->value)) { return false; }

thisCurrent = thisCurrent->pNext;

otherCurrent = otherCurrent->pNext;

}

return thisCurrent == otherCurrent;

}

string TPolinom::ToString() {

string result;

TNode<TMonom>\* current = this->pHead->pNext;

while (current != nullptr) {

int degX = current->value.index % 10;

int degY = (current->value.index / 10) % 10;

int degZ = current->value.index / 100;

if (!result.empty()) result += (current->value.coef > 0) ? " + " : " - ";

else if (current->value.coef < 0) { result += "-"; }

result += to\_string(abs(current->value.coef));

if (degX > 0) {

result += "x";

if (degX > 1) {

result += "^" + to\_string(degX);

}

}

if (degY > 0) {

result += "y";

if (degY > 1) {

result += "^" + to\_string(degY);

}

}

if (degZ > 0) {

result += "z";

if (degZ > 1) {

result += "^" + to\_string(degZ);

}

}

current = current->pNext;

}

return result;

}

**TPolinom\_Test.cpp**

#include <iostream>

#include "TList.h"

#include "TPolinom.h"

using namespace std;

void PrintMenu() {

cout << "Options:\n";

cout << "1. Enter the polynomial\n";

cout << "2. Print the polynomial\n";

cout << "3. Add the polynomial\n";

cout << "4. Multiply the polynomial by a number\n";

cout << "5. Exit\n";

}

TPolinom ReadPolinom() {

cout << " >>";

string str;

getline(cin, str);

return TPolinom(str);

}

int main()

{

TPolinom p1, p2, result;

int choice;

double coef;

while (true) {

PrintMenu();

cin >> choice;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

switch (choice) {

case 1:

p1 = ReadPolinom();

break;

case 2:

cout << "Polynomial: " << p1.ToString() << endl;

break;

case 3:

cout << "Second polynomial:\n";

p2 = ReadPolinom();

result = p1 + p2;

cout << "sum = " << result.ToString() << endl;

break;

case 4:

cout << "Enter the number:";

cin >> coef;

result = p1 \* coef;

cout << "Product = " << result.ToString() << endl;

break;

case 5:

return 0;

default:cout << "\n";

}

}

}

**TestPolinom.cpp**

#include "TPolinom.h"

#include <gtest.h>

TEST(Polinom, Created\_polinom\_is\_empty) {

TPolinom p;

EXPECT\_TRUE(p.IsEmpty());

}

TEST(Polinom, Created\_polinom\_with\_one\_monom\_is\_not\_empty) {

TPolinom p;

TMonom m(1.0, 2, 3, 4);

p.AddMonom(m);

EXPECT\_TRUE(p.IsEmpty() == false);

}

TEST(Polinom, Coefficients\_add\_up\_if\_degrees\_are\_equal) {

TPolinom p;

TMonom m1(1.5, 2, 3, 4);

TMonom m2(2.0, 2, 3, 4);

p.AddMonom(m1);

p.AddMonom(m2);

EXPECT\_EQ(p.GetLength(), 1);

EXPECT\_EQ(p.GetCurrentItem().coef, 3.5);

}

TEST(Polinom, Coefficients\_dont\_add\_up\_if\_degrees\_are\_different) {

TPolinom polynom;

TMonom monom1(1.0, 2, 1, 3);

TMonom monom2(2.0, 2, 1, 3);

polynom.AddMonom(monom1);

polynom.AddMonom(monom2);

EXPECT\_EQ(polynom.GetLength(), 1);

EXPECT\_EQ(polynom.GetCurrentItem().coef, 3.0);

}

TEST(Polinom, Monom\_delete\_if\_coefficient\_equal\_zero) {

TPolinom polynom;

TMonom monom1(1.0, 2, 1, 3);

TMonom monom2(-1.0, 2, 1, 3);

polynom.AddMonom(monom1);

polynom.AddMonom(monom2);

EXPECT\_EQ(polynom.GetLength(), 0);

}

TEST(Polinom, Can\_not\_get\_monom\_from\_empty\_polinom) {

TPolinom polynom;

EXPECT\_THROW(polynom.GetCurrentItem(), std::runtime\_error);

}

TEST(Polinom, Copied\_polinom\_is\_equal\_copiable) {

TPolinom originalPolinom;

originalPolinom.AddMonom(TMonom(1.0, 3, 0, 0));

TPolinom copiedPolinom(originalPolinom);

bool flag = originalPolinom == copiedPolinom;

EXPECT\_EQ(flag, true);

}

TEST(Polinom, Assigned\_polinom\_is\_equal\_assignable) {

TPolinom originalPolinom;

originalPolinom.AddMonom(TMonom(1.0, 3, 0, 0));

TPolinom assignedPolinom;

assignedPolinom = originalPolinom;

EXPECT\_EQ(assignedPolinom.ToString(), originalPolinom.ToString());

}

TEST(Polinom, Correct\_adding\_operator) {

TPolinom temp;

temp.AddMonom(TMonom(1.0, 3, 0, 0));

temp.AddMonom(TMonom(1.0, 3, 0, 0));

TPolinom temp2;

temp2.AddMonom(TMonom(2.0, 3, 0, 0));

EXPECT\_EQ(temp.ToString(), temp2.ToString());

}

TEST(Polinom, Correct\_constant\_multiply) {

TPolinom temp;

temp.AddMonom(TMonom(1.0, 3, 0, 0));

temp = temp \* 4.0;

TPolinom temp2;

temp2.AddMonom(TMonom(4.0, 3, 0, 0));

EXPECT\_EQ(temp.ToString(), temp2.ToString());

}