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**

**Отчет по лабораторной работе**

на тему:

**«Полиномы на списках»**

**Выполнил:** студент группы 3822Б1ПР2

Наумов Богдан Александрович

Нижний Новгород

2023

Оглавление

[1. Введение 3](#_Toc153998435)

[2. Постановка задачи 4](#_Toc153998436)

[3. Руководство пользователя 5](#_Toc153998437)

[4. Руководство программиста 6](#_Toc153998438)

[4.1.Описание структуры программы 6](#_Toc153998439)

[4.2.Описание структур данных 6](#_Toc153998440)

[4.3.Описание алгоритмов 10](#_Toc153998441)

[5. Эксперименты 11](#_Toc153998442)

[6. Заключение 12](#_Toc153998443)

[7. Приложение 13](#_Toc153998444)

## Введение

Многочлен представляет собой функцию с переменной, которая играет важную роль в алгебре и математическом анализе.

Он имеет вид P(x)=c\_0+ c\_1 x^1+ c\_2 x^2+⋯+ c\_n x^n, где ci - коэффициенты.

Степень многочлена определяется как наивысшая степень среди всех слагаемых-одночленов. В многочлене может присутствовать несколько независимых переменных, как например в x2y + 2x – 2y. Можно выполнять операции сложения и умножения многочленов, а иногда даже деление одного многочлена на другой. Коэффициенты могут быть не только числовыми. Существует также возможность существования многочлена, не содержащего переменных, и в этом случае его степень считается равной нулю.

Цель данной лабораторной работы заключается в изучении методов обработки многочленов с использованием компьютера путем создания программ для их обработки, а также изучении различных способов хранения многочленов.

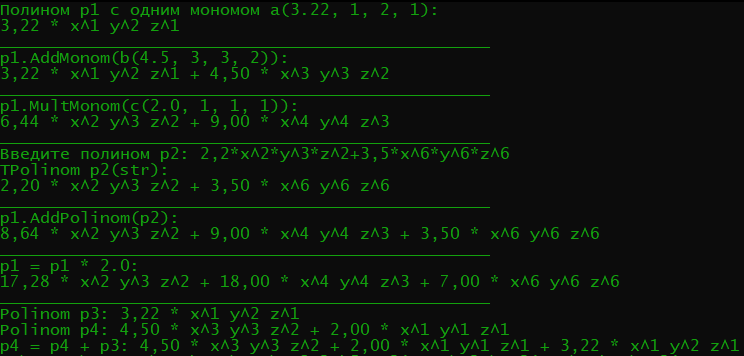
## Постановка задачи

Чтобы выполнить работу нужно решить следующие задачи:

1. Реализовать вспомогательне шаблонные классы TList, THeadList.
2. Реализовать структуру TMonom.
3. Реализовать класс TPolinom.
4. Написать пример программы, которая будет демонстрировать работу класса TPolinom.
5. Написать набор тестов с использованием GoogleC++ TestingFramework для проверки работы класса TPolinom.

## Руководство пользователя

При запуске программы, демонстрирующей работу класса TPolinom, она просит ввести строку по которой построится полином.



*Рис.1 (результат работы программы)*

## 4. Руководство программиста

## 4.1.Описание структуры программы

Программа состоит из следующих модулей:

* Модуль **polinomlib**. Статическая библиотека. Включает в себя заголовочные файлы TMonom.h и TPolinom, в которых описаны методы с реализацией класса Tpolinom и структурой TMonom.
* Модуль **polinomtest**. Набор тестов для класса TPolinom. Включает в себя файл **TestPolinom.cpp**. Реализованы они с помощью использования фреймворка **Google Test**.
* Модуль **polinom** - пример использования классов. Включает в себя файл с реализацией **TPolinom\_Test.cpp**.

## 4.2.Описание структур данных

Класс **TList**:

Поля:

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

Конструкторы и деструктор:

TList()

~TList()

Методы:

int GetLength() – получить длину списка

bool IsEmpty() – проверка на пустоту

void InsertFirst(T item) – вставить элемент в начало

void InsertCurrent(T item) – вставить элемент перед текущим

void InsertLast(T item) - вставить элемент в конец

void DeleteFirst() – удалить первый элемент

void DeleteCurrent() – удалить текущий элемент

void GoNext() – переместить вперед на одну позицию текущий элемент

void GoPrev() - переместить назад на одну позицию текущий элемент

void Reset() – установить на начало списка

bool IsEnd() – завершен ли список

T GetCurrentItem() – получить текущий элемент

void SetCurrentItem(T item) – задать текущий элемент

Структура **TMonom**:

Поля:

double coef – коэффициент монома

int index – индекс (свертка степеней)

Конструкторы:

TMonom()

TMonom(double \_coef, int degX, int degY, int degZ)

Методы:

void SetCoef(double cval) – установить коэффициент

double GetCoef(void) – получить коэффициент

void SetIndex(int ival) – установить индекс

int GetIndex(void) – получить индекс

Перегрузки:

bool operator==(const TMonom& other) - равенство

bool operator>(const TMonom& other) - больше

bool operator<(const TMonom& other) - меньше

Класс **TPolinom:**

Поля:

THeadList<TMonom> monoms – лист мономов

Конструкторы:

TPolinom();

TPolinom(TPolinom& other);

TPolinom(string str);

Методы:

void Parse(string str) – разделяет строку на мономы

void AddMonom(TMonom newMonom) – добавить моном в лист

TPolinom MultMonom(TMonom monom) – умножить полином на моном

TPolinom AddPolinom(TPolinom& other) – сложение полиномов

string ToString() – перевод в строку

bool IsEmpty() – проверка на пустоту

Перегрузки:

TPolinom& operator=(TPolinom& other) – присваивание

TPolinom& operator+(TPolinom& q) – сложение полиномов

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

TPolinom operator\* (TPolinom& other) – умножение полиномов

bool operator==(TPolinom& other) – равны ли полиномы

## 4.3.Описание алгоритмов

**Добавление монома:** Если моном с таким же набором степеней уже присутствует в полиноме, то происходит приведение подобных мономов и сложение их коэффициентов. После этого моном добавляется в список мономов полинома.

**Умножение мономов:** При умножении мономов их коэффициенты перемножаются, а показатели степеней соответствующих переменных суммируются. Если показатель степени превышает 9, то возникает исключение.

**Сложение полиномов**: Все мономы из второго полинома добавляются в первый полином, а затем происходит приведение подобных мономов (происходит суммирование коэффициентов мономов с одинаковыми наборами степеней).

**Умножение полинома на коэффициент:** Каждый коэффициент монома полинома умножается на данный коэффициент.

**Умножение полиномов**: Каждый моном из первого полинома умножается на каждый моном из второго полинома.

## 5. Эксперименты

|  |  |  |
| --- | --- | --- |
| Входная строка | Результат | Ожидаемый результат |
| 1,0\*x^1\*y^1\*z^1 | 1,00 \* x^1 y^1 z^1 | 1,00 \* x^1 y^1 z^1 |
| 15,3\*x^1\*y^2\*z^3 | 15,30 \* x^1 y^2 z^3 | 15,30 \* x^1 y^2 z^3 |
| 5,23\*x^4\*y^2\*z^8 | 5,23 \* x^4 y^2 z^8 | 5,23 \* x^4 y^2 z^8 |
| 2,2\*x^2\*y^3\*z^2+3,5\*x^6\*y^6\*z^6 | 2,20 \* x^2 y^3 z^2 + 3,50 \* x^6 y^6 z^6 | 2,20 \* x^2 y^3 z^2 + 3,50 \* x^6 y^6 z^6 |

## 6. Заключение

В данной лабораторной работе все поставленные задачи были успешно выполнены. Для хранения данных были разработаны две структуры: общий TList и специализированный THeadList. В основе класса полинома лежат структура монома и класс списка. Сильным вкладом в изучение автоматических тестов оказала самостоятельная работа на практике. Благодаря этому можно убедиться, что реализованные классы функционируют корректно. Кроме того, была проведена серия экспериментов для проверки корректности работы класса.

## 7. Приложение

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 GoPrev();

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()

{

length = 0;

pFirst = nullptr;

pCurrent = nullptr;

pPrevious = nullptr;

pLast = nullptr;

pStop = nullptr;

pStop = pLast;

}

template <class T>

TList<T>::~TList()

{

length = 0;

pFirst = nullptr;

pCurrent = nullptr;

pPrevious = nullptr;

pLast = nullptr;

pStop = nullptr;

}

template <class T>

bool TList<T>::IsEmpty()

{

if (length == 0) return true;

else return false;

}

template <class T>

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

{

if (length == 0) {

pFirst = new TNode<T>(item);

pFirst->pNext = pLast;

pCurrent = pFirst;

pLast = pFirst;

pStop = pLast->pNext;

}

else {

TNode<T>\* tmp = new TNode<T>(item, nullptr, pFirst);

pFirst->pPrev = tmp;

pFirst = tmp;

}

length++;

}

template <class T>

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

{

if (length == 0) {

pFirst = new TNode<T>(item);

pFirst->pNext = pLast;

pCurrent = pFirst;

pLast = pFirst;

pStop = pLast->pNext;

}

else {

TNode<T>\* tmp = new TNode<T>(item, pLast, nullptr);

pLast->pNext = tmp;

pLast = tmp;

}

length++;

}

template <class T>

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

{

if (pCurrent == pFirst) {

TNode<T>\* tmp = new TNode<T>(item, nullptr, pFirst);

pFirst->pPrev = tmp;

pFirst = tmp;

}

else {

TNode<T>\* tmp = pCurrent->pPrev;

TNode<T>\* n = new TNode<T>(item, tmp, pCurrent);

tmp->pNext = n;

pCurrent->pPrev = n;

}

length++;

}

template <class T>

void TList<T>::DeleteFirst()

{

if (pCurrent == pFirst) pCurrent = pFirst->pNext;

TNode<T>\* tmp = pFirst;

if (pFirst != nullptr) {

pFirst = pFirst->pNext;

}

if (pFirst != nullptr) {

pFirst->pPrev = nullptr;

}

else pLast = pFirst;

delete tmp;

length--;

}

template <class T>

void TList<T>::DeleteCurrent()

{

if (pCurrent == pFirst)

DeleteFirst();

else {

if (pCurrent == pLast) {

TNode<T>\* tmp = pCurrent->pPrev;

tmp->pNext = nullptr;

TNode<T>\* tmp2 = pCurrent;

pCurrent = tmp;

pLast = tmp;

delete tmp2;

}

else {

TNode<T>\* tmp = pCurrent->pPrev;

tmp->pNext = pCurrent->pNext;

TNode<T>\* tmp2 = pCurrent->pNext;

tmp2->pPrev = pCurrent->pPrev;

tmp = pCurrent;

pCurrent = tmp2;

delete tmp;

}

length--;

}

}

template <class T>

T TList<T>::GetCurrentItem()

{

/\*if (pCurrent == pStop)

throw " ";\*/

return pCurrent->value;

}

template <class T>

void TList<T>::Reset()

{

pCurrent = pFirst;

}

template <class T>

void TList<T>::GoNext()

{

if (pCurrent == pLast) pCurrent = pFirst;

else pCurrent = pCurrent->pNext;

}

template <class T>

void TList<T>::GoPrev()

{

if (pCurrent == pFirst) pCurrent = pLast;

else pCurrent = pCurrent->pPrev;

}

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() : TList<T>()

{

this->pHead = nullptr;

}

template<class T>

THeadList<T>::~THeadList()

{

}

template <class T>

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

{

if (length == 0) {

pFirst = new TNode<T>(item, this->pHead, this->pLast);

pCurrent = pFirst;

pLast = pFirst;

pStop = pLast->pNext;

}

else {

TNode<T>\* tmp = this->pCurrent;

this->pCurrent = this->pFirst;

this->InsertCurrent(item);

this->pCurrent = tmp;

}

length++;

}

template <class T>

void THeadList<T>::DeleteFirst()

{

if (this->pCurrent == this->pFirst) {

TNode<T>\* tmp = this->pFirst->pNext;

this->DeleteCurrent();

this->pCurrent = tmp;

}

else {

TNode<T>\* tmp = this->pCurrent;

this->pCurrent = this->pFirst;

this->DeleteCurrent();

this->pCurrent = tmp;

}

}

TMonom.h

#pragma once

struct TMonom

{

double coef; // коэффициент монома

int index; // индекс (свертка степеней)

TMonom() {

this->coef = 1.0;

this->index = 0;

}

TMonom(double \_coef, int degX, int degY, int degZ) {

if (degX > 9 || degY > 9 || degZ > 9)

throw exception("deg > 9");

SetCoef(\_coef);

index = (degX \* 10 + degY) \* 10 + degZ;

}

void SetCoef(double cval) { coef = cval; }

double GetCoef(void) { return coef; }

void SetIndex(int ival) { index = ival; }

int GetIndex(void) { return index; }

bool operator==(const TMonom& other) { return (coef == other.coef) && (index == other.index); }

bool operator>(const TMonom& other) {

int degX, degY, degZ, o\_degX, o\_degY, o\_degZ;

degX = this->index / 100;

degY = (this->index / 10) % 10;

degZ = this->index % 10;

o\_degX = other.index / 100;

o\_degY = (other.index / 10) % 10;

o\_degZ = other.index % 10;

if (degX == o\_degX) {

if (degY == o\_degY) {

if (degZ == o\_degZ) return false;

else return degZ > o\_degZ;

}

else return degY > o\_degY;

}

else return degX > o\_degX;

}

bool operator<(const TMonom& other) { return !(this->operator>(other) && (this->operator==(other))); }

};

TPolinom.h

#pragma once

#include "THeadList.h"

#include "TMonom.h"

#include <string>

#include <vector>

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

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

class TPolinom : public THeadList<TMonom>

{

public:

THeadList<TMonom> monoms;

void Parse(string str);

TPolinom();

TPolinom(TPolinom& other);

TPolinom(string str);

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

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

// дополнительно можно реализовать:

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

TPolinom operator\* (TPolinom& other); // умножение полиномов

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

string ToString(); // перевод в строку

bool IsEmpty();

};

bool TPolinom::IsEmpty() {

return this->monoms.GetLength() == 0;

}

void TPolinom::Parse(string str) {

string s\_coef = "";

double d\_coef = 0;

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

int ch = 0;

while (ch < str.size() - 1)

{

while (str[ch] != '\*') {

s\_coef += str[ch];

ch++;

}

d\_coef = std::stod(s\_coef);

ch += 3;

degX = str[ch] - '0';

ch += 4;

degY = str[ch] - '0';

ch += 4;

degZ = str[ch] - '0';

}

TMonom m(d\_coef, degX, degY, degZ);

monoms.InsertLast(m);

}

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

{

}

TPolinom::TPolinom(TPolinom& other)

{

monoms = other.monoms;

}

TPolinom::TPolinom(string str)

{

vector<string> monoms\_to\_parse;

int i = 0;

string monom;

while (i < str.size())

{

monom = "";

while ((str[i] != '+') && (i < str.size()))

{

monom.push\_back(str[i]);

i++;

}

monoms\_to\_parse.push\_back(monom);

i++;

}

for (int j = 0; j < monoms\_to\_parse.size(); j++) {

Parse(monoms\_to\_parse[j]);

}

}

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

{

this->monoms = other.monoms;

return \*this;

}

void TPolinom::AddMonom(TMonom m)

{

bool flag = true;

monoms.Reset();

for (int i = 0; i < monoms.GetLength(); i++) {

if (m.GetIndex() == monoms.GetCurrentItem().GetIndex()) {

double coef = (this->monoms.GetCurrentItem().coef) + m.coef;

int degX = (this->monoms.GetCurrentItem().GetIndex() / 100);

int degY = ((this->monoms.GetCurrentItem().GetIndex() / 10) % 10);

int degZ = (this->monoms.GetCurrentItem().GetIndex() % 10);

TMonom tmp(coef, degX, degY, degZ);

monoms.DeleteCurrent();

if (i == monoms.GetLength() && monoms.GetLength() != 0) monoms.InsertLast(tmp);

else {

if (monoms.GetLength() == 0) monoms.InsertFirst(tmp);

else monoms.InsertCurrent(tmp);

}

flag = false;

break;

}

monoms.GoNext();

}

if(flag) monoms.InsertLast(m);

}

TPolinom TPolinom::MultMonom(TMonom monom)

{

THeadList<TMonom> tmp;

monoms.Reset();

double coef;

int degX, degY, degZ;

int mdegX = monom.GetIndex() / 100;

int mdegY = (monom.GetIndex() / 10) % 10;

int mdegZ = monom.GetIndex() % 10;

for (int i = 0; i < monoms.GetLength(); i++) {

coef = (this->monoms.GetCurrentItem().coef) \* monom.coef;

degX = (this->monoms.GetCurrentItem().GetIndex() / 100) + mdegX;

degY = ((this->monoms.GetCurrentItem().GetIndex() / 10) % 10) + mdegY;

degZ = (this->monoms.GetCurrentItem().GetIndex() % 10) + mdegZ;

if (degX > 9 || degY > 9 || degZ > 9) throw - 1;

TMonom m(coef, degX, degY, degZ);

tmp.InsertLast(m);

this->monoms.GoNext();

}

this->monoms = tmp;

return \*this;

}

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

{

other.monoms.Reset();

for (int i = 0; i < other.monoms.GetLength(); i++) {

AddMonom(other.monoms.GetCurrentItem());

other.monoms.GoNext();

}

return \*this;

}

TPolinom TPolinom::AddPolinom(TPolinom& other)

{

other.monoms.Reset();

for (int i = 0; i < other.monoms.GetLength(); i++) {

AddMonom(other.monoms.GetCurrentItem());

other.monoms.GoNext();

}

return \*this;

}

TPolinom TPolinom::operator\*(double \_coef)

{

THeadList<TMonom> tmp;

monoms.Reset();

double coef;

int degX, degY, degZ;

for (int i = 0; i < monoms.GetLength(); i++) {

coef = (this->monoms.GetCurrentItem().coef) \* \_coef;

degX = (this->monoms.GetCurrentItem().GetIndex() / 100);

degY = ((this->monoms.GetCurrentItem().GetIndex() / 10) % 10);

degZ = (this->monoms.GetCurrentItem().GetIndex() % 10);

if (degX > 9 || degY > 9 || degZ > 9) throw - 1;

TMonom m(coef, degX, degY, degZ);

tmp.InsertLast(m);

this->monoms.GoNext();

}

this->monoms = tmp;

return \*this;

}

TPolinom TPolinom::operator\*(TPolinom& other)

{

other.monoms.Reset();

for (int j = 0; j < other.monoms.GetLength(); j++) {

TMonom cur = other.monoms.GetCurrentItem();

this->MultMonom(cur);

other.monoms.GoNext();

}

return \*this;

}

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

{

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

else {

monoms.Reset();

for (int i = 0; i < monoms.GetLength(); i++) {

if (!(this->monoms.GetCurrentItem() == other.monoms.GetCurrentItem()))

return false;

other.monoms.GoNext();

monoms.GoNext();

}

}

return true;

}

string TPolinom::ToString()

{

string result = "";

string tmp;

string coef;

int degX, degY, degZ, ind;

monoms.Reset();

for (int i = 0; i < monoms.GetLength(); i++) {

ind = monoms.GetCurrentItem().GetIndex();

degX = ind / 100;

degY = (ind / 10) % 10;

degZ = ind % 10;

coef = std::to\_string(monoms.GetCurrentItem().GetCoef());

for (int j = 0; j < nonDisplayedZeros; j++) coef.pop\_back();

tmp = coef + " \* x^" + std::to\_string(degX) + " y^" + std::to\_string(degY) + " z^" + std::to\_string(degZ);

result += tmp;

if(monoms.GetLength() - i > 1) result += " + ";

monoms.GoNext();

}

return result;

}

TPolinom\_Test.cpp

#include <iostream>

#include "TList.h"

#include "TPolinom.h"

using namespace std;

//1,0\*x^1\*y^1\*z^1

//15,3\*x^1\*y^2\*z^3

//5,23\*x^4\*y^2\*z^8

//2,2\*x^2\*y^3\*z^2+3,5\*x^6\*y^6\*z^6

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Полином p1 с одним мономом a(3.22, 1, 2, 1):" << endl;

TPolinom p1;

TMonom a(3.22, 1, 2, 1);

TMonom b(4.5, 3, 3, 2);

TMonom c(2.0, 1, 1, 1);

p1.AddMonom(a);

cout << p1.ToString() << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

p1.AddMonom(b);

cout << "p1.AddMonom(b(4.5, 3, 3, 2)):" << endl << p1.ToString() << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

p1.MultMonom(c);

cout << "p1.MultMonom(c(2.0, 1, 1, 1)):" << endl << p1.ToString() << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

string str;

cout << "Введите полином p2: ";

cin >> str;

TPolinom p2(str);

cout << "TPolinom p2(str):" << endl << p2.ToString() << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

p1.AddPolinom(p2);

cout << "p1.AddPolinom(p2):" << endl << p1.ToString() << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

p1 = p1 \* 2.0;

cout << "p1 = p1 \* 2.0:" << endl << p1.ToString() << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

TPolinom p3;

TPolinom p4;

p3.AddMonom(a);

p4.AddMonom(b);

p4.AddMonom(c);

cout << "Polinom p3: " << p3.ToString() << endl;

cout << "Polinom p4: " << p4.ToString() << endl;

p4 = p4 + p3;

cout << "p4 = p4 + p3: " << p4.ToString();

}