МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №9**

**«Полиномы»**

**Выполнил:** студент группы 381706-2

Жбанова Надежда Сергеевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Содержание

[1.Введение 3](#_Toc1336360)

[2. Цели и задачи 4](#_Toc1336361)

[2.1. Используемые инструменты 4](#_Toc1336362)

[3. Руководство пользователя 6](#_Toc1336363)

[4. Руководство программиста 7](#_Toc1336364)

[4.1. Описание структуры программы 7](#_Toc1336365)

[4.2. Описание функций и процедур, их алгоритмов 7](#_Toc1336366)

[5. Эксперименты 11](#_Toc1336367)

[6. Заключение 12](#_Toc1336368)

[7. Литература 13](#_Toc1336369)

[8. Приложения 14](#_Toc1336370)

[8.1. Приложение 1:Класс TExсeption 14](#_Toc1336371)

[8.2. Приложение 2:Класс TMonomial 14](#_Toc1336372)

[8.3. Приложение 3:Класс TPolynomial 19](#_Toc1336373)

[8.4. Приложение 4:Код программы тестирования и экспериментов 24](#_Toc1336374)

[8.5. Приложение 5:Тесты для класса 24](#_Toc1336375)

# 1.Введение

В математике, многочлены или полиномы от одной переменной - функции вида

8de2160610dbeafbceb3de7e48c4acdf где ci фиксированные коэффициенты, а x — переменная. Многочлены составляют один из важнейших классов элементарных функций.

Полиномиальные уравнения и их решение составляет едва ли не главный объект «классической алгебры». С изучением многочленов связан целый ряд преобразований в математике: введение в рассмотрение нуля, отрицательных, а затем и комплексных чисел, а также появление теории групп как раздела математики и выделение классов специальных функций в анализе.

Многочлены также играют ключевую роль в алгебраической геометрии, объектом которой являются множества, определённые как решения систем многочленов. Особые свойства преобразования коэффициентов при умножении многочленов используются в алгебраической геометрии, алгебре, теории узлов и других разделах математики для кодирования, или выражения многочленами свойств различных объектов.

Многочлен (или полином) от n переменных — есть конечная формальная сумма вида

3df8c3a2d2083a7dd0d67fd6e4317a73, где I = (i1,i2,...,in) есть набор из целых неотрицательных чисел (называется мультииндекс), cI — число (называемое «коэффициент многочлена»), зависящее только от мультииндекса I.

Лабораторная работа направлена на изучение методов компьютерной обработки полиномов. Основной учебной целью работы является практическое освоение методов организации структур хранения данных с помощью линейных списков.

# 2. Цели и задачи

Для работы с полиномами предлагается реализовать следующие операции:

* конструкторы инициализации и копирования;
* метод присваивания;
* метод сложения полиномов.

Дополнительные операции могут быть определены при разработке спецификации класса для полиномов.

В процессе выполнения лабораторной работы требуется использовать систему контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Выполнение работы предполагает решение следующих задач:

1. Реализация класса TMonomial.
2. Реализация класса TPolynomial, построенного с использованием класса TMonomial.
3. Реализация класса для обработки исключений– TException, которые могут возникнуть при выполнении различных операций, согласно заданному интерфейсу.
4. Обеспечение работоспособности тестов и примера использования.
5. Реализация заготовок тестов, покрывающих все методы созданных классов.
6. Модификация примера использования в тестовое приложение, позволяющее задавать полиномы и осуществлять основные операции над ними.

## 2.1. Используемые инструменты

* Система контроля версий [Git](https://git-scm.com/book/ru/v2). Рекомендуется использовать один из следующих клиентов на выбор студента:
  + [Git](https://git-scm.com/downloads)
  + [GitHub Desktop](https://desktop.github.com/)
* Фреймворк для написания автоматических тестов [Google Test](https://github.com/google/googletest).
* Среда разработки Microsoft Visual Studio (2008 или старше).
* Опционально. Утилита [CMake](http://www.cmake.org/) для генерации проектов по сборке исходных кодов. Может быть использована для генерации решения для среды разработки, отличной от Microsoft Visual Studio 2008 или 2010.

# 3. Руководство пользователя

Запускаем программу sample\_tpolynomial.cpp. Программа выведет пример корректной работы некоторых основных операций с мономами и полиномами. (Рис.1):

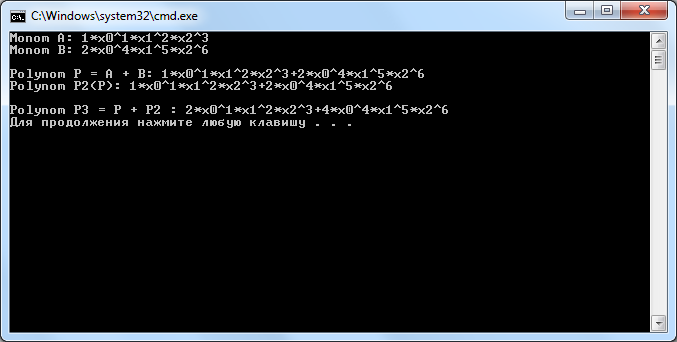


Рис.1.Пример основных операций.

# 4. Руководство программиста

Разработка системы вычисления проводились в среде “Microsoft Visual Studio 2010”.

В данной работе будет использовано 3 класса:

* Класс «Моном» (TMonomial).
* Класс «Полином» (TPolynomial).
* Класс исключения (TExсeption).

## 4.1. Описание структуры программы

Модульная структура программы:

1. tmonomial.h– модуль с классом TMonomial, в котором определен интерфейс класса Моном и реализация его методов.
2. tpolynomial.h– модуль с классом TPolynomial, в котором определен интерфейс класса Полином и реализация его методов.
3. exсeption.h – модуль с классом исключения TExсeption.
4. sample\_tpolynomial.cpp, sample\_performance\_check.cpp– модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.
5. test\_main.cpp, test\_tmonomial\_and\_tpolynomial.cpp – модуль с функциями тестирования для созданных классов. Содержат 23 теста для класса TMonomial и 17 тестов для класса TPolynomial.

## 4.2. Описание функций и процедур, их алгоритмов

Рассмотрим реализацию методов класса TMonomial:

class TMonomial

int\* power - массив степеней переменных монома.

int n - количество переменных в мономе(размерность).

double coeff - коэффициент при мономе.

TMonomial \*next – указатель на следующий моном.

1. Monomial( int \_n = 10 ) - конструктор класса, принимающий количество переменных в мономе. По умолчанию создается моном размера 10.

2. TMonomial( int \_n, int \*\_power, double \_coeff ) - конструктор класса, принимающий количество переменных в мономе, массив степеней переменных монома и коэффициент.

3. TMonomial( TMonomial &A ) - конструктор копирования.

Принимает ссылку на объект класса TMonomial.

4. ~TMonomial() - деструктор. Освобождает выделенную под моном память.

5. void SetPower( int \*\_power ) – метод установления массива степеней монома.

6. void SetN( int \_n ) – метод установления кол-ва переменных монома.

7. void SetCoeff( double \_coeff ) – метод установления коэффициента при мономе.

8. void SetNext( TMonomial\* \_next ) – метод установления адреса следующего монома.

9. int \*GetPower() - метод получения массива степеней монома.

10. int GetN() - метод получения кол-ва переменных монома.

11. double GetCoeff() - метод получения коэффициента при мономе.

12. TMonomial\* GetNext() - метод получения адреса следующего монома.

13. bool ComparePowers(TMonomial &A) – метод сравнения массива степеней монома.

14. TMonomial &operator = ( TMonomial &A ) - перегрузка оператора присваивания одного монома другому.

Присваивает полям первого объекта класса поля второго объекта класса.

15. TMonomial operator + ( TMonomial &A ) - перегрузка оператора сложения мономов.

16. TMonomial &operator += ( TMonomial &A ) - перегрузка оператора “+=” для мономов.

17. TMonomial operator - ( TMonomial &A ) - перегрузка оператора вычитания одного монома из другого.

18. TMonomial &operator -= ( TMonomial &A ) - перегрузка оператора “-=” для мономов.

19. TMonomial operator \* ( TMonomial &A ) - перегрузка оператора умножения мономов.

20. TMonomial &operator \*= ( TMonomial &A ) - перегрузка оператора “\*=” для мономов.

21. bool operator == ( TMonomial &A ) - перегрузка оператора сравнения.

Выполняется проверка мономов на равенство. Возвращает true, если равенство выполняется, false в противном случае.

22. bool operator < ( TMonomial &A ) - перегрузка оператора “<” для мономов.

23. bool operator > ( TMonomial &A ) - перегрузка оператора “>” для мономов.

24. friend istream& operator>>( istream &istr, TMonomial &m ) - дружественный метод. Перегрузка оператора ввода мономов.

25. friend ostream& operator<<( ostream &ostr, TMonomial &m ) - дружественный метод. Перегрузка оператора вывода мономов.

Рассмотрим реализацию методов класса TPolynomial:

class TPolynomial

TMonomial \*start - указатель на первый моном в полиноме.

int n – поле размерности монома в полиноме.

int k – количество мономов в полиноме.

1. TPolynomial() – конструктор класса без параметров.

2. TPolynomial(int \_n) - конструктор класса, принимающий количество переменных в мономе.

3. TPolynomial(const TPolynomial &A) - конструктор копирования.

Принимает ссылку на объект класса TPolynomial.

4. ~TPolynomial() - деструктор. Освобождает выделенную под полином память.

5. int GetN() - метод получения кол-ва переменных монома.

6. int GetSize() – метод получения кол-ва мономов в полиноме.

7. TMonomial\* GetStart() - метод получения указателя на первый моном в полиноме.

8. TPolynomial operator-(TPolynomial &A) - перегрузка оператора вычитания полиномов.

9. TPolynomial operator+(TPolynomial &A) - перегрузка оператора сложения полиномов.

10. TPolynomial& operator=(const TPolynomial &A) - перегрузка оператора присваивания одного полинома другому.

Присваивает полям первого объекта класса поля второго объекта класса.

11. bool operator==(const TPolynomial &A) - перегрузка оператора сравнения.

Выполняется проверка полиномов на равенство. Возвращает true, если равенство выполняется, false в противном случае.

12. bool operator!=(const TPolynomial &A) - перегрузка оператора сравнения.

Выполняется проверка полиномов на неравенство. Возвращает true, если неравенство выполняется, false в противном случае.

13. TPolynomial operator\*(TPolynomial &A) - перегрузка оператора умножение полиномов.

14. TPolynomial& operator+=(TMonomial &m) - перегрузка оператора “+=” для сложения полинома с мономом.

15. TPolynomial& operator-=(TMonomial &m) - перегрузка оператора “-=” для вычитания из полинома монома.

16. friend std::ostream& operator<<(std::ostream &ostr, TPolynomial& Tm) - дружественный метод. Перегрузка оператора вывода полиномов.

# 5. Эксперименты

В качестве примера рассмотрим перегрузку оператора присвоения для класса полинома (TPolynomial).

# 6. Заключение

В результате лабораторной работы был разработан класс полиномов, а также освоены такие инструменты разработки программного обеспечения, как система контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Созданный класс был протестированы с использованием Google Tests, а также были проведены эксперименты для сравнения теоретической и практической сложности выполнения операций на методе класса.

# 7. Литература

1. Лабораторный практикум: Учебно-методическое пособие / Мееров И.Б. [и др.] – Нижний Новгород: Нижегородский госуниверситет , 2017. – 105с.
2. Тестирование с использованием Google Test

(<http://neerc.ifmo.ru/wiki/index.php?title=Тестирование_с_использованием_Google_Test#.D0.A4.D1.83.D0.BD.D0.BA.D1.86.D0.B8.D1.8F_main.28.29> )

1. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
2. <https://ru.wikipedia.org/wiki/Многочлен>

# 8. Приложения

## 8.1. Приложение 1:Класс TExсeption

|  |
| --- |
| **exception.h** |
| #pragma once  #include <iostream>  #include <string>  class TException  {  private:  std::string str;  public:  TException(std::string \_str);  void Show();  };  TException::TException(std::string \_str) : str(\_str) {}  void TException::Show()  {  std::cout << "\nWarning! \nMessage: " << str << std::endl;  } |

## 8.2. Приложение 2:Класс TMonomial

|  |
| --- |
| **tmonomial.h** |
| #pragma once  #include <iostream>  #include "exception.h"  using namespace std;  class TMonomial  {  protected:  int\* power; //массив степеней  int n; //кол-во переменных в мономе  double coeff; ///коэффицтент  TMonomial \*next; //указатель на следующий моном  public:  TMonomial( int \_n = 10 );  TMonomial( int \_n, int \*\_power, double \_coeff );  TMonomial( TMonomial &A );  ~TMonomial();  void SetPower( int \*\_power );  void SetN( int \_n );  void SetCoeff( double \_coeff );  void SetNext( TMonomial\* \_next );  int \*GetPower();  int GetN();  double GetCoeff();  TMonomial\* GetNext();  bool ComparePowers(TMonomial &A);  TMonomial &operator = ( TMonomial &A );  TMonomial operator + ( TMonomial &A );  TMonomial &operator += ( TMonomial &A );  TMonomial operator - ( TMonomial &A );  TMonomial &operator -= ( TMonomial &A );  TMonomial operator \* ( TMonomial &A );  TMonomial &operator \*= ( TMonomial &A );  bool operator == ( TMonomial &A );  bool operator < ( TMonomial &A );  bool operator > ( TMonomial &A );  friend istream& operator>>( istream &istr, TMonomial &m );  friend ostream& operator<<( ostream &ostr, TMonomial &m );  };  TMonomial::TMonomial( int \_n )  {  if ( \_n < 0 )  throw TException( "Error" );  else if ( \_n == 0 )  {  n = 0;  next = NULL;  power = 0;  coeff = 0;  }  else if ( \_n > 0 )  {  n = \_n;  next = NULL;  power = new int[n];  coeff = 0;  }  }  TMonomial::TMonomial( int \_n, int\* \_power, double \_coeff )  {  if ( \_n < 0 )  throw TException( "Error" );  else if ( \_n == 0 )  {  power = 0;  next = NULL;  coeff = \_coeff;  n = \_n;  }  else  {  n = \_n;  coeff = \_coeff;  next = 0;  power = new int[\_n];  for ( int i = 0; i < \_n; i++ )  {  if ( \_power[i] >= 0 )  power[i] = \_power[i];  else  throw TException( "Error" );  }  }  }  TMonomial :: TMonomial( TMonomial &A )  {  n = A.n;  coeff = A.coeff;  next = A.next;  if (n != 0) {  power = new int[n];  for ( int i = 0; i < n; i++ )  {  power[i] = A.power[i];  }  } else  power=NULL;  }  TMonomial::~TMonomial()  {  if ( power )  delete[]power;  n = 0;  coeff = 0;  next = 0;  }  void TMonomial::SetPower( int \*\_power )  {  for ( int i = 0; i < n; i++ )  {  if ( \_power[i] >= 0 )  power[i] = \_power[i];  else  throw TException( "Error" );  }  }  void TMonomial::SetN( int \_n )  {  if ( \_n <= 0 )  throw TException( "Error" );  else if ( \_n == 0 )  {  if ( power != 0 )  delete[] power;  power = NULL;  }  else if (\_n != n)  {  int \*tmp = power;  power = new int[\_n];  int count = n < \_n ? n : \_n;  for (int i = 0; i < count; i++) {  power[i]=tmp[i];  }  delete[] tmp;  }  n = \_n;  }  void TMonomial::SetCoeff( double \_coeff )  {  coeff = \_coeff;  }  void TMonomial::SetNext( TMonomial\* \_next )  {  next = \_next;  }  int \*TMonomial::GetPower()  {  return power;  }  int TMonomial::GetN()  {  return n;  }  double TMonomial::GetCoeff()  {  return coeff;  }  TMonomial\* TMonomial::GetNext()  {  return next;  }  bool TMonomial::ComparePowers(TMonomial &A)  {  if (n != A.n)  return false;  for (int i = 0; i < n; i++)  if (power[i] != A.power[i])  return false;  return true;  };  TMonomial &TMonomial::operator = ( TMonomial &A )  {  coeff = A.coeff;  n = A.n;  next = A.next;  delete[] power;  power = new int[n];  for ( int i = 0; i < n; i++ )  power[i] = A.power[i];  return \*this;  }  TMonomial TMonomial::operator + ( TMonomial &A )  {  TMonomial tmp(\*this);  tmp += A;  return tmp;  }  TMonomial &TMonomial::operator+=( TMonomial & A )  {  if ( n != A.n )  throw TException( "Error" );  for ( int i = 0; i < n; i++ )  if ( power[i] != A.power[i] )  throw TException( "Error" );  coeff += A.coeff;  return \*this;  }  TMonomial TMonomial::operator - ( TMonomial &A )  {  TMonomial tmp(\*this);  tmp -= A;  return tmp;  }  TMonomial &TMonomial::operator-=( TMonomial & A )  {  if ( n != A.n )  throw TException( "Error" );  for ( int i = 0; i < n; i++ )  if ( power[i] != A.power[i] )  throw TException( "Error" );  coeff -= A.coeff;  return \*this;  }  TMonomial TMonomial::operator \* ( TMonomial &A )  {  TMonomial tmp(\*this);  tmp \*= A;  return tmp;  }  TMonomial &TMonomial::operator \*= ( TMonomial & A )  {  if ( n != A.n )  throw TException( "Error" );  coeff \*= A.coeff;  for ( int i = 0; i < n; i++ )  power[i] += A.power[i];  return \*this;  }  bool TMonomial::operator == ( TMonomial &A )  {  if ( n != A.n )  throw TException( "Error" );  if ( coeff != A.coeff )  return false;  for ( int i = 0; i < n; i++ )  if ( power[i] != A.power[i] )  return false;  return true;  }  bool TMonomial::operator > ( TMonomial& A )  {  if ( n != A.n )  throw TException( "Error" );  for ( int i = 0; i < n; i++ )  {  if ( power[i] == A.power[i] )  continue;  else  return power[i] > A.power[i] ? true : false;  }  return coeff > A.coeff ? true : false;  }  bool TMonomial::operator < ( TMonomial& A )  {  if ( n != A.n )  throw TException( "Error" );  for ( int i = 0; i < n; i++ )  {  if ( power[i] == A.power[i] )  continue;  else  return power[i] < A.power[i] ? true : false;  }  return coeff < A.coeff ? true : false;  }  istream& operator>>( istream &istr, TMonomial &m )  {  istr >> m.coeff;  for ( int i = 0; i < m.n; i++ )  istr >> m.power[i];  return istr;  }  ostream& operator<<( ostream &ostr, TMonomial &m )  {  ostr << m.GetCoeff();  for(int i = 0; i < m.GetN(); i++)  ostr <<"\*"<<"x"<<i<<"^"<<m.GetPower()[i];  return ostr;  } |

## 8.3. Приложение 3:Класс TPolynomial

|  |
| --- |
| **tpolynomial.h** |
| #pragma once  #include <iostream>  #include "exception.h"  #include "tmonomial.h"  using namespace std;  class TPolynomial  {  protected:  TMonomial \*start;  int n; //поле размерности монома  int k; //количество элементов  public:  TPolynomial();  TPolynomial(int \_n);  TPolynomial(const TPolynomial &A);  ~TPolynomial();  int GetN();  int GetSize();  TMonomial\* GetStart();  TPolynomial operator-(TPolynomial &A);  TPolynomial operator+(TPolynomial &A);  TPolynomial& operator=(const TPolynomial &A);  bool operator==(const TPolynomial &A);  bool operator!=(const TPolynomial &A);  TPolynomial operator\*(TPolynomial &A);  TPolynomial& operator+=(TMonomial &m);  TPolynomial& operator-=(TMonomial &m);  friend std::ostream& operator<<(std::ostream &ostr, TPolynomial& Tm);  };  TPolynomial::TPolynomial()  {  n = 0;  k = 0;  start = NULL;  }  TPolynomial::TPolynomial(int \_n)  {  if ( \_n <= 0 )  throw TException( "Error" );  n = \_n;  k = 0;  start = NULL;  }  TPolynomial::TPolynomial(const TPolynomial &A)  {  n = A.n;  k = A.k;  if ( A.start == NULL )  start = NULL;  else  {  start = new TMonomial(\*A.start);  for ( TMonomial\* it = A.start->GetNext(); it != NULL; it=it->GetNext() )  {  start->SetNext(new TMonomial(\*it));  }  }  }  TPolynomial::~TPolynomial()  {  if ( start == NULL )  return;  TMonomial\* next\_el;  for ( TMonomial\* it = start; it != NULL; )  {  next\_el = it->GetNext();  it->~TMonomial();  it = next\_el;  }  }  int TPolynomial::GetN()  {  return n;  }  int TPolynomial::GetSize()  {  return k;  }  TMonomial\* TPolynomial::GetStart()  {  return start;  }  TPolynomial& TPolynomial::operator=(const TPolynomial &A)  {  if ( \*this != A )  {  if ( start != NULL )  {  TMonomial\* next\_el;  for ( TMonomial\* it = start; it != NULL; )  {  next\_el = it->GetNext();  it->~TMonomial();  it = next\_el;  }  }  start = new TMonomial( \*A.start );  TMonomial\* src\_el = start;  for ( TMonomial\* it = A.start->GetNext(); it != NULL;it=it -> GetNext() )  {  src\_el -> SetNext( new TMonomial(\*it) );  src\_el = src\_el -> GetNext();  }  n = A.n;  k = A.k;  }  return \*this;  }  TPolynomial &TPolynomial::operator+=(TMonomial &m)  {  if ( n != m.GetN() )  throw TException( "Error" );  if ( m.GetCoeff() == 0 )  return \*this;  if ( start == NULL )  {  start = new TMonomial(m);  k++;  }  else  {  TMonomial\* last\_el = NULL;  for ( TMonomial \*ptr = start; ptr != NULL; ptr=ptr->GetNext() )  {  if ( ptr->ComparePowers(m) )  {  \*ptr += m;  break;  }  else if ( ptr ->GetNext() == NULL )  {  last\_el = ptr;  }  }  if ( last\_el != NULL )  {  last\_el -> SetNext( new TMonomial(m) );  k++;  }  }  return \*this;  }  TPolynomial &TPolynomial::operator-=(TMonomial &m)  {  if ( n != m.GetN() )  throw TException( "Error" );  if ( m.GetCoeff() == 0 )  return \*this;  if ( start == NULL )  {  start = new TMonomial(m);  start -> SetCoeff( start -> GetCoeff() \* (-1) );  k++;  }  else  {  TMonomial\* last\_el = NULL;  for ( TMonomial \*ptr = start; ptr != NULL; ptr=ptr->GetNext() )  {  if ( ptr->ComparePowers(m) )  {  \*ptr -= m;  break;  } else if ( ptr ->GetNext() == NULL )  {  last\_el = ptr;  }  }  if ( last\_el != NULL )  {  last\_el -> SetNext( new TMonomial(m) );  last\_el -> SetCoeff( last\_el ->GetCoeff() \* (-1) );  k++;  }  }  return \*this;  }  bool TPolynomial::operator==(const TPolynomial &A)  {  if ( this->n != A.n )  throw TException( "Error" );  if ( this->k != A.k )  return false;  for ( TMonomial\* src\_ptr = start; src\_ptr != NULL; src\_ptr = src\_ptr -> GetNext() )  for ( TMonomial\* p\_ptr = start; p\_ptr != NULL; p\_ptr = p\_ptr -> GetNext() )  {  if ( \*src\_ptr == \*p\_ptr )  {  break;  }  else if ( p\_ptr -> GetNext() == NULL )  {  return false;  }  }  return true;  }  bool TPolynomial::operator!=(const TPolynomial &A)  {  return !( \*this == A );  }  TPolynomial TPolynomial::operator\*(TPolynomial &A)  {  if ( n != A.n )  throw TException( "Error" );  TPolynomial tmp(n);  for ( TMonomial\* src\_ptr = start; src\_ptr != NULL; src\_ptr = src\_ptr -> GetNext() )  for ( TMonomial\* p\_ptr = start; p\_ptr != NULL; p\_ptr = p\_ptr -> GetNext() )  {  tmp += (\*src\_ptr) \* (\*p\_ptr);  }  return tmp;  }  TPolynomial TPolynomial::operator+(TPolynomial &A)  {  if ( n != A.n )  throw TException( "Error" );  TPolynomial tmp(A);  for ( TMonomial\* p\_ptr = start; p\_ptr != NULL; p\_ptr = p\_ptr -> GetNext() )  {  tmp += \*p\_ptr;  }  return tmp;  }  TPolynomial TPolynomial::operator-(TPolynomial &A)  {  if ( n != A.n )  throw TException( "Error" );  TPolynomial tmp(A);  for ( TMonomial\* p\_ptr = start; p\_ptr != NULL; p\_ptr = p\_ptr -> GetNext() )  {  tmp -= \*p\_ptr;  }  return tmp;  }  ostream& operator<<(ostream &ostr, TPolynomial& Tm)  {  if ( Tm.start == NULL )  ostr << "Empty";  else  {  for ( TMonomial\* ptr = Tm.start; ptr != NULL; ptr = ptr -> GetNext() )  {  ostr << \*ptr;  if ( ptr -> GetNext() != NULL )  ostr << "+";  }  }  return ostr;  } |

## 8.4. Приложение 4:Код программы тестирования и экспериментов

|  |
| --- |
| **sample\_tpolynomial.cpp** |
| #include <iostream>  #include <locale.h>  #include "tpolynomial.h"  using namespace std;  int main()  {  try  {  int a[] = { 1, 2, 3 };  int b[] = { 4, 5, 6 };  TMonomial A(3, a, 1);  TMonomial B(3, b, 2);  cout << "Monom A: " << A << endl;  cout << "Monom B: " << B << endl;  TPolynomial P(3);  P += A;  P += B;  cout << "\nPolynom P = A + B: " << P << endl;  TPolynomial P2(P);  cout << "Polynom P2(P): " << P2 << endl;  TPolynomial P3(3);  P3 = P + P2;  cout << "\nPolynom P3 = P + P2 : " << P3 << endl;  }  catch (TException exp)  {  exp.Show();  }  return 0;  } |

|  |
| --- |
| **sample\_performance\_check.cpp** |
|  |

## 8.5. Приложение 5:Тесты для класса

|  |
| --- |
| **test\_main.cpp** |
| #include <gtest.h>  int main(int argc, char \*\*argv)  {  ::testing::InitGoogleTest(&argc, argv);  return RUN\_ALL\_TESTS();  } |

|  |
| --- |
| **test\_tmonomial\_and\_tpolynomial.cpp** |
| #include "tpolynomial.h"  #include <gtest.h>  TEST(TMonomial, can\_create\_default\_monomial)  {  ASSERT\_NO\_THROW(TMonomial A);  }  TEST(TMonomial, can\_create\_monomial\_wiht\_custom\_size)  {  ASSERT\_NO\_THROW(TMonomial A(3));  }  TEST(TMonomial, throws\_when\_create\_monomial\_with\_negative\_size)  {  ASSERT\_ANY\_THROW(TMonomial A(-3));  }  TEST(TMonomial, throws\_when\_create\_monom\_with\_negative\_power)  {  int a[3] = { 1,-2,3 };  ASSERT\_ANY\_THROW(TMonomial A(3, a, 1));  }  TEST(TMonomial, can\_create\_monomial\_with\_parameters)  {  int a[3] = { 1,2,3 };  ASSERT\_NO\_THROW(TMonomial A(3, a, 3.34));  }  TEST(TMonomial, can\_create\_copy\_of\_the\_monomial)  {  int a[3] = { 1,2,3 };  TMonomial A(3, a, 3.34);  TMonomial B(A);  ASSERT\_EQ(B.GetCoeff(), A.GetCoeff());  ASSERT\_EQ(B.GetN(), A.GetN());  ASSERT\_EQ(B.GetNext(), A.GetNext());  int \* power = B.GetPower();  for ( int i = 0; i < 3; i++ )  {  ASSERT\_EQ(power[i], a[i]);  }  }  TEST(TMonomial, can\_set\_next)  {  TMonomial A(2);  TMonomial B(3);  ASSERT\_NO\_THROW(A.SetNext(&B));  ASSERT\_EQ(A.GetNext(), &B);  }  TEST(TMonomial, can\_set\_and\_get\_coeff)  {  TMonomial A;  ASSERT\_NO\_THROW(A.SetCoeff(3.14));  ASSERT\_EQ(3.14, A.GetCoeff());  }  TEST(TMonomial, can\_set\_and\_get\_power)  {  int a[3] = { 1,2,3 };  TMonomial A(3);  ASSERT\_NO\_THROW(A.SetPower(a));  ASSERT\_EQ(1, A.GetPower()[0]);  ASSERT\_EQ(2, A.GetPower()[1]);  ASSERT\_EQ(3, A.GetPower()[2]);  }  TEST(TMonomial, can\_set\_and\_get\_size)  {  TMonomial A(3);  ASSERT\_NO\_THROW(A.SetN(5));  ASSERT\_EQ(5, A.GetN());  }  TEST(TMonomial, can\_assign\_monomials)  {  int a[3] = { 1,2,3 };  TMonomial A(3, a, 3.12);  TMonomial B(6);  B = A;  ASSERT\_EQ(B.GetCoeff(), A.GetCoeff());  ASSERT\_EQ(B.GetN(), A.GetN());  ASSERT\_EQ(B.GetNext(), A.GetNext());  int \* power = B.GetPower();  for ( int i = 0; i < 3; i++ )  {  ASSERT\_EQ(power[i], a[i]);  }  }  TEST(TMonomial, can\_add\_monomials\_with\_equal\_size)  {  int a[3] = { 1,2,3 };  TMonomial A(3, a, 3.1);  TMonomial B(3, a, 0.5);  ASSERT\_NO\_THROW(A + B);  TMonomial C(3);  C = A + B;  ASSERT\_EQ(3.6, C.GetCoeff());  }  TEST(TMonomial, trows\_when\_add\_monomials\_with\_different\_size\_and\_powers)  {  int a[3] = { 1,2,3 };  TMonomial A(3);  TMonomial B(3);  TMonomial C(4);  A.SetPower(a);  ASSERT\_ANY\_THROW(A + B);  ASSERT\_ANY\_THROW(A + C);  }  TEST(TMonomial, can\_subtract\_monomials\_with\_equal\_size)  {  int a[3] = { 1,2,3 };  TMonomial A(3, a, 3.1);  TMonomial B(3, a, 0.5);  ASSERT\_NO\_THROW(A - B);  TMonomial C(3);  C = A - B;  ASSERT\_EQ(2.6, C.GetCoeff());  }  TEST(TMonomial, trows\_when\_subtract\_monomials\_with\_different\_size\_and\_power)  {  int a[3] = { 1,2,3 };  TMonomial A(3);  TMonomial B(3);  TMonomial C(4);  A.SetPower(a);  ASSERT\_ANY\_THROW(A - B);  ASSERT\_ANY\_THROW(A - C);  }  TEST(TMonomial, can\_multiply\_monomials\_with\_equal\_size)  {  int a[3] = { 1,2,3 };  int a2[3] = { 3,2,1 };  TMonomial A(3, a, 3.1);  TMonomial B(3, a2, 0.5);  ASSERT\_NO\_THROW(A \* B);  TMonomial C(3);  C = A \* B;  ASSERT\_EQ(3.1\*0.5, C.GetCoeff());  for ( int i =0; i < 3; i++ )  ASSERT\_EQ(C.GetPower()[i], 4);  }  TEST(TMonomial, trows\_when\_multiply\_monomials\_with\_different\_size)  {  int a[3];  for ( int i = 0; i < 3; i++ )  a[i] = i + 1;  TMonomial A(3);  TMonomial C(4);  A.SetPower(a);  ASSERT\_ANY\_THROW(A \* C);  }  TEST(TMonomial, can\_equivalence\_equal\_monomials)  {  int a[3] = { 1,2,3 };  TMonomial A(3, a, 3.1);  TMonomial B(3, a, 3.1);  ASSERT\_TRUE(A == B);  }  TEST(TMonomial, trows\_when\_equivalence\_monomials\_with\_different\_size)  {  TMonomial A(3);  TMonomial B(4);  ASSERT\_ANY\_THROW(A == B);  }  TEST(TMonomial, equivalence\_different\_monomials\_with\_equal\_size)  {  int a[3] = { 1,2,3 };  int b[3] = { 1,3,2 };  TMonomial A(3);  TMonomial B(3);  A.SetPower(a);  B.SetPower(b);  ASSERT\_EQ(false, A == B);  B.SetPower(a);  A.SetCoeff(3.1);  B.SetCoeff(3.4);  ASSERT\_EQ(false, A == B);  }  TEST(TMonomial, can\_compare\_monomials\_with\_equal\_size)  {  int a[3] = { 1,2,3 };  int b[3] = { 1,3,2 };  TMonomial A(3,a,1);  TMonomial B(3,b,1);  ASSERT\_EQ(false, A > B);  ASSERT\_EQ(true, A < B);  }  TEST(TMonomial, trows\_when\_compare\_monomials\_with\_different\_size)  {  TMonomial A(3);  TMonomial B(4);  ASSERT\_ANY\_THROW(A > B);  ASSERT\_ANY\_THROW(A < B);  }  TEST(TMonomial, compare\_powers\_of\_monomials)  {  int a[3] = { 1,2,3 };  TMonomial A(3, a, 4);  TMonomial B(3, a, 5);  ASSERT\_EQ(true, A.ComparePowers(B));  }  TEST(TPolynomial, can\_create\_polynomial)  {  ASSERT\_NO\_THROW(TPolynomial A);  }  TEST(TPolynomial, can\_create\_polynomial\_with\_positive\_n)  {  ASSERT\_NO\_THROW(TPolynomial A(3));  }  TEST(TPolynomial, throws\_when\_create\_polynomial\_with\_negative\_n)  {  ASSERT\_ANY\_THROW(TPolynomial A(-3));  }  TEST(TPolynomial, can\_create\_copy\_polynomial)  {  TPolynomial A(3);  ASSERT\_NO\_THROW(TPolynomial B(A));  TPolynomial C(A);  ASSERT\_EQ(C.GetSize(), 0);  ASSERT\_EQ(C.GetN(), 3);  if ( C.GetStart() != NULL )  ASSERT\_TRUE(false);  }  TEST(TPolynomial, can\_get\_size\_polynomial)  {  int a[] = { 1, 2, 3 };  int b[] = { 1, 0, 3 };  TMonomial A(3, a, 1);  TMonomial B(3, b, 2);  TPolynomial P(3);  P += A;  P += B;  ASSERT\_EQ(P.GetSize(), 2);  }  TEST(TPolynomial, can\_get\_n\_of\_polynomial)  {  TPolynomial P(3);  ASSERT\_EQ(P.GetN(), 3);  }  TEST(TPolynomial, trows\_when\_add\_polynomials\_with\_different\_n)  {  TPolynomial P1(2);  TPolynomial P2(3);  ASSERT\_ANY\_THROW(P1 + P2);  }  TEST(TPolynomial, trows\_when\_subtract\_polynomials\_with\_different\_n)  {  TPolynomial P1(2);  TPolynomial P2(3);  ASSERT\_ANY\_THROW(P1 - P2);  }  TEST(TPolynomial, add\_polynomials)  {  int a[] = { 1, 0, 3 };  TMonomial M1(3, a, 1);  TPolynomial P(3);  P += M1;  int b[] = { 1, 2, 3 };  TMonomial M2(3, b, 4);  TPolynomial R(3);  R += M2;  TPolynomial S(3);  S = R + P;  TMonomial \* ptr = S.GetStart();  ASSERT\_TRUE( (\*ptr == M1 && \*ptr->GetNext() == M2) ||  (\*ptr == M2 && \*ptr->GetNext() == M1) );  ptr = ptr -> GetNext();  if ( ptr -> GetNext() != NULL )  ASSERT\_TRUE(false);  }  TEST(TPolynomial, sub\_polynomials)  {  int a[] = { 1, 0, 3 };  TMonomial M1(3, a, 1);  TPolynomial P(3);  P += M1;  int b[] = { 1, 2, 3 };  TMonomial M2(3, b, 4);  TPolynomial R(3);  R += M2;  TPolynomial S(3);  S = R - P;  TMonomial \* ptr = S.GetStart();  M1.SetCoeff(M1.GetCoeff() \* (-1));  ASSERT\_TRUE( (\*ptr == M1 && \*ptr->GetNext() == M2) ||  (\*ptr == M2 && \*ptr->GetNext() == M1) );  ptr = ptr -> GetNext();  if ( ptr -> GetNext() != NULL )  ASSERT\_TRUE(false);  }  TEST(TPolynomial, throws\_when\_assign\_monomials\_with\_different\_n)  {  TPolynomial P1(4);  TPolynomial P2(3);  ASSERT\_ANY\_THROW(P1 = P2);  }  TEST(TPolynomial, trows\_when\_equivalence\_polynomials\_with\_different\_n)  {  TPolynomial P1(3);  TPolynomial P2(4);  ASSERT\_ANY\_THROW(P1 == P2);  }  TEST(TPolynomial, trows\_when\_multiply\_different\_polynomials)  {  TPolynomial P1(3);  TPolynomial P2(4);  ASSERT\_ANY\_THROW(P1 \* P2);  }  TEST(TPolynomial, trows\_when\_plus\_eq\_different\_polynomials)  {  int a[] = { 1, 0, 3, 6 };  TMonomial M(4, a, 1);  TPolynomial P(3);  ASSERT\_ANY\_THROW(P += M);  }  TEST(TPolynomial, trows\_when\_minus\_eq\_different\_polynomials)  {  int a[] = { 1, 0, 3, 6 };  TMonomial M(4, a, 1);  TPolynomial P(3);  ASSERT\_ANY\_THROW(P -= M);  }  TEST(TPolynomial, can\_add\_polynomials\_with\_equal\_n)  {  TPolynomial P1(2);  TPolynomial P2(2);  ASSERT\_NO\_THROW(P1 + P2);  }  TEST(TPolynomial, can\_subtract\_polynomials\_with\_equal\_n)  {  TPolynomial P1(2);  TPolynomial P2(2);  ASSERT\_NO\_THROW(P1 - P2);  } |