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**

**ЛАБОРАТОРНАЯ РАБОТА №10**

**«Полиномы»**

**Выполнил:** студент группы 381903-3

Микийчук Дмитрий Алексеевич

**Проверил:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2020

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 25](#_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):

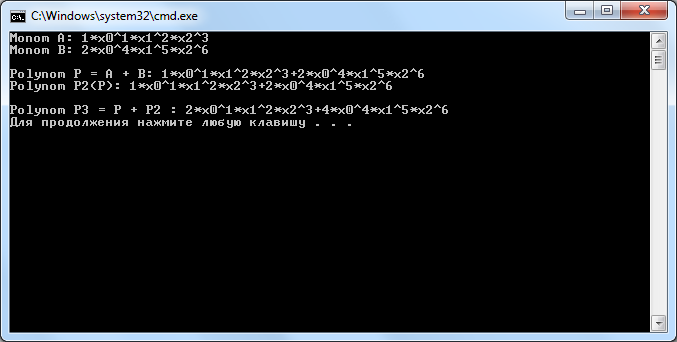


Рис.1.Пример основных операций.

# 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(const 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() const - метод получения кол-ва переменных монома.

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

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

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

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

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

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

15. TPolynomial& operator-=( const 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 “+=” для сложения полинома с мономом для класса полинома (TPolynomial).

Теоретическая сложность выполнения алгоритма O(1).

Мы провели измерение прибавляя к полиному мономы с разными кол-вами переменных: 10, 100, …, 1000000 переменных. Ниже вы можете увидеть график зависимости времени выполнения операции “+=” от количества переменных монома. (Рис.2) По приведенным данным можно сделать вывод, что практическая сложность выполнения алгоритма равна теоретической.

Рис.2. График зависимости времени выполнения операции “+=”.

По горизонтали – количество переменных монома.

По вертикали - время выполнения программы.

# 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;

int n;

double coeff;

## 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+=(const TMonomial &m);

TPolynomial& operator-=(const 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() )

{

## 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**

#include <iostream>

#include "tpolynomial.h"

#include "time.h"

using namespace std;

int main()

{

int max\_count = 0;

cout << "Enter number of checks - ";

cin >> max\_count;

clock\_t time;

clock\_t average\_time;

for ( unsigned size = 10; size < 1000000; size \*= 10 )

{

int\* powers = new int[size];

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

{

powers[i] = i;

}

TMonomial A(size, powers, 1);

TPolynomial p1(size);

average\_time = 0;

for ( unsigned count = 0; count < max\_count; count++ )

{

time = clock();

p1 += A;

average\_time += clock() - time;

}

average\_time /= max\_count;

cout<< "Time is: " <<average\_time<<endl;

}

return 0;

}

## 8.5. Приложение 5:Тесты для класса

**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));

}