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 №11**

**«Полиномы»**

**Выполнил:** студент группы 381903-3

Стеценко Данил Михайлович

**Проверил:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

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)

# 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. Обеспечение работоспособности тестов и примера использования.
4. Реализация заготовок тестов, покрывающих все методы созданных классов.
5. 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):

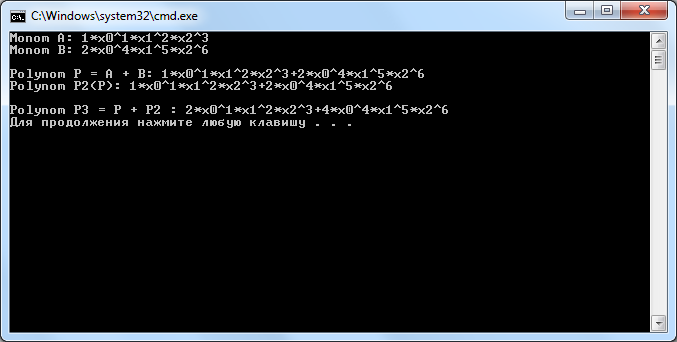


Рис.1.Пример основных операций.

# 4. Руководство программиста

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

В данной работе будут использованы классы:

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

## 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. tpolishl.h– модуль с классом TPolynomial, в котором определен интерфейс класса Полином и реализация его методов.
3. sample\_tpolish.cpp, sample\_performance\_check.cpp– модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.
4. 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

|  |
| --- |
| **Polish.cpp** |
| #include "../PolishLib/Polish.h"  void TOperFactory::Create(char\* s, int& n, IOper\*\*& opers)  {  int l = strlen(s);  for (int i = 0; i < l; i++)  {  if (isdigit(s[i]) == 0)  n++;  }  n++;  n = 2 \* n - 1;  IOper\*\* res = new IOper \* [n];  int start = 0;  int j = 0;  for (int i = 0; i < l; i++)  {  if (i == 0)  {  if (s[i] == '(')  {  res[j] = new TOpenBracket('(');  j++;  start = i + 1;  continue;  }  }  else  {  if (isdigit(s[i - 1]) != 0)  {  char temp[256];  int \_l = i - start + 1;  for (int k = 0; k < \_l - 1; k++)  temp[k] = s[start + k];  temp[\_l - 1] = 0;  res[j] = new TNum(temp);  j++;  }  if (s[i] == '(')  {  res[j] = new TOpenBracket('(');  j++;  start = i + 1;  }  else if (s[i] == ')')  {  res[j] = new TCloseBracket(')');  j++;  start = i + 1;  }  else if (s[i] == '+')  {  res[j] = new TPlus('+');  j++;  start = i + 1;  }  else if (s[i] == '-')  {  res[j] = new TMinus('-');  j++;  start = i + 1;  }  else if (s[i] == '\*')  {  res[j] = new TMultiplication('\*');  j++;  start = i + 1;  }  else if (s[i] == '/')  {  res[j] = new TDivision('/');  j++;  start = i + 1;  }  }  }  char temp[256];  int \_l = l - start + 1;  if (\_l > 1)  {  for (int k = 0; k < \_l - 1; k++)  temp[k] = s[start + k];  temp[\_l - 1] = 0;  res[j] = new TNum(temp);  j++;  }  n = j;  opers = new IOper \* [j];  for (int i = 0; i < j; i++)  opers[i] = res[i];  delete[] res;  }  int TPolish::Calculate(char\* s)  {  int n = 0;  IOper\*\* opers = nullptr;  TOperFactory::Create(s, n, opers);  IOper\*\* res = new IOper \* [n];  TStack<IOper\*> stack(n);  int j = 0;  for (int i = 0; i < n; i++)  {  if (opers[i]->Priority() == -1)  {  res[j] = opers[i];  j++;  }  else  {  if ((opers[i]->Priority() == 1) || (stack.IsEmpty()))  stack.Put(opers[i]);  else if (opers[i]->Priority() == 0)  {  IOper\* temp = stack.Get();  while (temp->Priority() != 1)  {  res[j] = temp;  j++;  temp = stack.Get();  }  }  else if (opers[i]->Priority() != 1 && opers[i]->Priority() != 0)  {  IOper\* temp = stack.Get();  if (opers[i]->Priority() > temp->Priority())  {  stack.Put(temp);  stack.Put(opers[i]);  }  else  {  while (opers[i]->Priority() <= temp->Priority())  {  res[j] = temp;  j++;  if (!stack.IsEmpty())  temp = stack.Get();  else  break;  }  if (!stack.IsEmpty())  stack.Put(temp);  stack.Put(opers[i]);  }  }  }  }  while (!stack.IsEmpty())  {  IOper\* temp = stack.Get();  res[j] = temp;  j++;  }  TStack<IOper\*> ns(n);  for (int i = 0; i < j; i++)  {  if (res[i]->Priority() == -1)  ns.Put(res[i]);  else  {  IOper\* b = ns.Get();  IOper\* a = ns.Get();  IOper\* c = res[i]->Calc(a, b);  ns.Put(c);  }  }  IOper\* r = ns.Get();  delete[] opers;  delete[] res;  return r->GetNumber();  }  TNum::TNum(int \_d)  {  data = \_d;  }  TNum::TNum(TNum& \_v)  {  data = \_v.data;  }  TNum::TNum(char\* s)  {  if (strlen(s) == 0)  throw - 1;  data = atof(s);  }  int TNum::Priority()  {  return -1;  }  int TNum::GetNumber()  {  return data;  }  IOper\* TNum::Calc(IOper\* a, IOper\* b)  {  throw - 1;  }  TOper::TOper(char \_d)  {  data = \_d;  }  TOper::TOper(TOper& \_v)  {  data = \_v.data;  }  int TOper::GetNumber()  {  throw - 1;  }  int TPlus::Priority()  {  return 2;  }  IOper\* TPlus::Calc(IOper\* a, IOper\* b)  {  TNum\* res = new TNum(a->GetNumber() + b->GetNumber());  return res;  }  int TMinus::Priority()  {  return 2;  }  IOper\* TMinus::Calc(IOper\* a, IOper\* b)  {  TNum\* res = new TNum(a->GetNumber() - b->GetNumber());  return res;  }  int TMultiplication::Priority()  {  return 3;  }  IOper\* TMultiplication::Calc(IOper\* a, IOper\* b)  {  TNum\* res = new TNum(a->GetNumber() \* b->GetNumber());  return res;  }  int TDivision::Priority()  {  return 3;  }  IOper\* TDivision::Calc(IOper\* a, IOper\* b)  {  if (b->GetNumber() == 0)  throw - 2;  TNum\* res = new TNum(a->GetNumber() / b->GetNumber());  return res;  }  int TOpenBracket::Priority()  {  return 1;  }  IOper\* TOpenBracket::Calc(IOper\* a, IOper\* b)  {  throw - 1;  }  int TCloseBracket::Priority()  {  return 0;  }  IOper\* TCloseBracket::Calc(IOper\* a, IOper\* b)  {  throw - 1;  } |

|  |
| --- |
| **Polish.h** |
| #ifndef \_POLISH\_H\_  #define \_POLISH\_H\_  #include "../StackLib/Stack.h"  #include <math.h>  #include <stdlib.h>  #include <ctype.h>  #include <stdio.h>  #include <string.h>  class IOper  {  public:  virtual int Priority() = 0;  virtual int GetNumber() = 0;  virtual IOper\* Calc(IOper\* a, IOper\* b) = 0;  };  class TNum : public IOper  {  protected:  int data;  public:  TNum(int \_d);  TNum(TNum& \_v);  TNum(char\* s);  virtual int Priority();  virtual int GetNumber();  virtual IOper\* Calc(IOper\* a, IOper\* b);  };  class TOper : public IOper  {  protected:  char data;  public:  TOper(char \_d);  TOper(TOper& \_v);  virtual int GetNumber();  };  class TPlus : public TOper  {  public:  TPlus(char \_d) : TOper(\_d) {}  virtual int Priority();  virtual IOper\* Calc(IOper\* a, IOper\* b);  };  class TMinus : public TOper  {  public:  TMinus(char \_d) : TOper(\_d) {}  virtual int Priority();  virtual IOper\* Calc(IOper\* a, IOper\* b);  };  class TMultiplication : public TOper  {  public:  TMultiplication(char \_d) : TOper(\_d) {}  virtual int Priority();  virtual IOper\* Calc(IOper\* a, IOper\* b);  };  class TDivision : public TOper  {  public:  TDivision(char \_d) : TOper(\_d) {}  virtual int Priority();  virtual IOper\* Calc(IOper\* a, IOper\* b);  };  class TOpenBracket : public TOper  {  public:  TOpenBracket(char \_d) : TOper(\_d) {}  virtual int Priority();  virtual IOper\* Calc(IOper\* a, IOper\* b);  };  class TCloseBracket : public TOper  {  public:  TCloseBracket(char \_d) : TOper(\_d) {}  virtual int Priority();  virtual IOper\* Calc(IOper\* a, IOper\* b);  };  class TOperFactory  {  public:  static void Create(char\* s, int& n, IOper\*\*& opers);  };  class TPolish  {  public:  static int Calculate(char\* s);  };  #endif |