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**

**Отчет по учебной практике**

**«Разработка системы для арифметических действий над многочленами нескольких переменных»**

**Выполнил:** студент группы 381706-1

Денисов Владислав Львович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc295709)

[2. Постановка задачи 4](#_Toc295710)

[3. Руководство пользователя 5](#_Toc295711)

[4. Руководство программиста 6](#_Toc295712)

[4.1 Описание структуры программы 6](#_Toc295713)

[4.2 Описание структур данных 6](#_Toc295714)

[4.3 Описание алгоритмов 8](#_Toc295715)

[5. Заключение 11](#_Toc295716)

[6. Литература 12](#_Toc295717)

# Введение

Лабораторная работа направлена на разработку системы для арифметических действий над многочленами нескольких переменных. Для этого введем несколько необходимых понятий.

Моном (также одночлен) – простое математическое выражение, прежде всего рассматриваемое и используемое в элементарной алгебре, а именно, произведение, состоящее из числового множителя и одной или нескольких переменных, взятых каждая в неотрицательной целой степени.

Мономом также считается каждое отдельное число (без буквенных множителей), причём степень такого одночлена равняется нулю.

Полином (или многочлен) от *n* переменных – это сумма одночленов или, строго, – конечная формальная сумма вида , где – набор всевозможных целых неотрицательных чисел(мультииндекс), - число, (именуемое коэффициент многочлена) зависящее только от мультииндекса *I*.

# Постановка задачи

В рамках лабораторной работы ставится задача эффективной реализации системы для арифметических действий над многочленами нескольких переменных.

Для работы будут разработаны:

* арифметические (+, –, \*) и логические (==, >, <) операции для работы с мономами,
* операция присваивания для мономов,
* перегружены операторы ввода/вывода с помощью консоли для мономов,
* арифметические (+, \*) и логическая (==) для полиномов,
* операция добавления монома к полиному (оператор +=),
* операция присваивания для полиномов,
* перегружен оператор вывода для полиномов,

Программное решение будет выглядеть следующим образом:

1. Класс мономов – TMonom.
2. Класс полиномов – TPolynom,
3. Класс для обработки исключений – TException, которые могут возникнуть при выполнении различных операций.
4. Программа, демонстрирующая работу класса TPolynom.
5. Набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

В качестве примера использования разработанной библиотеки предлагается следующее.

После запуска тестовой программы создается несколько различных мономов. Затем из них собираются 2 полинома разной длины. Производится демонстрация работы операций сложения и умножения полученных полиномов.

Выполняется копирование первого полинома и сравнение адресов исходного и скопированного объектов.

Наконец, пользователю предлагается ввести моном с помощью консоли, чтобы убедиться в том, что оператор ввода перегружен правильно.

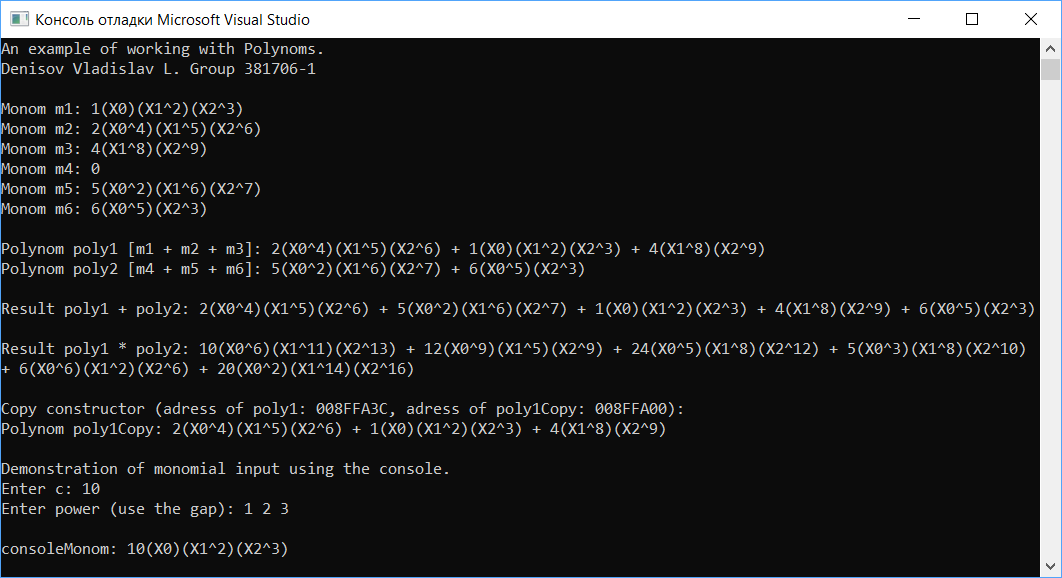


Рисунок 1 Пример работы демонстрационной программы.

# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

* Модуль Polynom. Содержит пример использования полиномов. Реализация в файле *main\_Polynom.cpp.*
* Модуль PolynomLib – статическая библиотека. Содержит файлы *Monom.h* и *Polynom.h*, в которых описан интерфейс классов *TMonom* и *TPolynom* соответственно. А также файлы *Monom.cpp* и *Polynom.cpp*, содержащие реализацию указанных классов.
* Модуль PolynomTest. Содержит суммарно 34 теста (22 для класса мономов, 12 для класса полиномов), описанных в файле *PolynomTest.cpp* и разработанных с помощью использования Google C++ Testing Framework.
* Модуль ExceptionLib – библиотека, позволяющая создавать собственные исключения.

## Описание структур данных

#### Класс TException – класс исключений.

Класс содержит 1 **private** поле *std::string msg* – переменная, хранящая сообщение об ошибке в виде строки.

И содержит 2 **public** элемента:

*TException(std::string \_str) : msg(\_msg)* – конструктор с одним параметром.

*void Print()* – метод отображения ошибки на консоль.

#### Класс TMonom – класс мономов.

Рассмотрим класс *TMonom* подробно.

**Элементы класса, объявленные со спецификатором protected:**

TMonom \*next – указатель на следующий моном.

unsigned \*power – массив степеней переменных в мономе.

double c – коэффициент при мономе.

const int n – количество переменных в мономе.

**Элементы класса, объявленные со спецификатором public:**

TMonom(int \_n, unsigned \*\_power, double c) – конструктор с параметрами.

TMonom(const TMonom &monom) – конструктор копирования.

virtual ~TMonom() – деструктор.

TMonom\* GetNext() – метод получения указателя на следующий моном.

unsigned\* GetPower() – метод получения массива степеней.

double GetC() – метод получения коэффициента.

const int GetN() – метод получения количество переменных.

void SetNext(TMonom \*\_next) метод установки указателя на следующий моном.

void SetPower(unsigned \*\_power) – метод, позволяющий изменить степени.

void SetC(double \_c) – метод, позволяющий изменить коэффициент.

**Перегруженные операторы:**

TMonom& operator=(const TMonom &monom) – оператор присваивания.

TMonom operator+(TMonom &monom) – оператор сложения.

TMonom operator+=(TMonom &monom) – оператор сложения с присваиванием.

TMonom operator-(TMonom &monom) – оператор вычитания.

TMonom operator-=(TMonom &monom) – оператор вычитания с присваиванием.

TMonom operator\*(TMonom &monom) const – оператор умножения.

TMonom operator\*=(TMonom &monom) – оператор умножения с присваиванием.

bool operator==(TMonom &monom) – оператор проверки на равнество.

bool operator>(TMonom &monom) – оператор «больше».

bool operator<(TMonom &monom) – оператор «меньше».

**Дружественные функции:**

friend istream& operator>>(istream &\_s, TMonom &Tm) – оператор ввода монома с помощью консоли.

friend ostream& operator<<(ostream &\_s, TMonom &Tm) – оператор вывода монома на консоль.

#### Класс TPolynom – класс полиномов.

Рассмтрим класс *TPolynom* подробно.

**Элементы класса, объявленные со спецификатором protected:**

TMonom \*start – указатель на начало полинома.

int n – допустимое число переменных в полиноме.

**Элементы класса, объявленные со спецификатором public:**

TPolynom(int \_n) – конструктор с параметром.

TPolynom(TPolynom &p) – конструктор копирования.

virtual ~TPolynom() – деструктор.

TMonom\* GetStart() – метод получения начала полинома.

**Перегруженные операторы:**

TPolynom& operator=(const TPolynom &p) – оператор присваивания.

TPolynom& operator+=(TMonom &m) – оператор добавления монома к полиному.

TPolynom operator+(TPolynom &p) – оператор сложения полиномов.

TPolynom operator\*(TPolynom &p) – оператор умножения полиномов.

bool operator==(const TPolynom &p) – оператор проверки на равенство.

**Дружественная функция:**

friend std::ostream& operator<<(std::ostream& \_s, TPolynom& Tm) – вывод полинома на консоль.

## Описание алгоритмов

Рассмотрим некоторые алгоритмы, работа которых не очевидна на первый взгляд.

**Добавление монома к полиному.**

Выполняем проверку на то, что количество переменных в полиноме и мономе совпадает, иначе бросаем исключение. Затем смотрим на коэффициент при мономе: если он равен 0, то к полиному ничего не добавляем, возвращаем исходный полином. Если же полином ничего не содержит, то просто в него записываем моном.

Иначе создаем два итератора \_start – начало полинома, \_end – следующий за начальным в полиноме – они позволят нам рассмотреть пары мономов в полиноме и найти необходимую позицию под добавляемый моном. Заметим, что добавляем моном с учетом лексикографического порядка.

Выполняем сравнение \*start и m (имеется в виду добавляемый моном). Если моном m больше первого монома в полиноме, то вставляем его непосредственно перед первым и возвращаем полином.

Иначе выполняем проверку на равенство первому. Если они совпадают, то суммируем их и смотрим на коэффициент при полученном члене. Если он равен нулю, то удаляем первый моном, а start = start->GetNext() и возвращаем полученный полином.

В противном случае добавляемый моном меньше первого в исходном полиноме. Значит в цикле (while (\_end! = 0)) выполним проход всего полинома, рассматривая пары мономов от первой к последней (она будет состоять из предпоследнего и последнего мономов).

Если на очередной итерации моном совпал со вторым из рассматриваемой пары (т.е. \*\_end == m), то выполняем их сложение и смотрим на коэффициент при полученном члене. В случае нуля, а \_start->SetNext(\_end->GetNext()), и удаляем моном по \_end. Возвращаем полученный полином.

Если добавляемый моном оказался между первым и вторым мономом в рассматриваемой паре, то добавляем его между ними, выполняя необходимые связки через SetNext().

Если добавляемый мономом все еще меньше каждого из мономов рассматриваемой пары, то выполняем сдвиг итераторов вправо на следующую и продолжаем цикл.

Если ни одно из условий выше не было выполнено, то добавляем моном просто в самый конец.

**Сложение полиномов.**

Прежде всего выполняем проверку на то, что количество переменных в каждом полиноме совпадает, иначе бросаем исключение.

Создаем временный полином *temp*, куда будем записывать результат сложения. Создаем 3 итератора, чтобы выполнять обход по первому и второму полиномам-слагаемым, а также по полиному-результату.

Пока не закончится один из полиномов-слагаемых, выполняем следующий цикл. Создаем указатель \_monom на моном. Выполняем ряд проверок.

Если текущий рассматриваемый по итератору моном из первого полинома равен текущему моному второго полинома, то выполняем их сложение и записываем по созданному указателю \_monom, увеличиваем итераторы. Переходим к условию (\*).

Если же рассматриваемый моном из перового полинома меньше, чем из второго, то по указателю \_monom записываем моном из второго. Увеличиваем итератор только второго полинома. Переходим к (\*).

В последнем возможном случае текущий моном из первого полинома больше рассматриваемого из второго, следовательно нужно записать по \_monom моном первого, увеличить итератор первого полинома. Затем (\*).

(\*) Если полином-результат пустой, то temp.start = \_monom, итератору по полиному-результату присваиваем значение \_monom. Иначе к полиному-результату добавляем \_monom с помощью i->SetNext(\_monom), а итератор по полиному-результату i = i->GetNext().

После проверки условия (\*) очередная итерация цикла завершается.

По окончании последней итерации цикла выполняем проверку на то, закончился ли первый полином или нет. Если закончился, то итератор второго полинома присваиваем итератору первого. В цикле, пока итератор не будет указывать на 0, выполняем добавление к полиному-результату оставшихся мономов из второго полинома. В конце всех перечисленных действий возвращаем результат.

.

# Заключение

В результате лабораторной работы была разработана библиотека, реализующая классы полиномов и мономов Она позволяет создать объект класса TPolynom и выполнить с ним некоторые операции, задача реализации которых была поставлена в начале данной лабораторной работы.

Были разработаны и доведены до успешного выполнения тесты, разработанные для данного программного проекта с использованием Google C++ Testing Framework.

Программное решение было продемонстрировано с помощью простейшего набора операций над полиномами. Описание примера работы с полнимомами было представлено в разделе «Руководство пользователя».

# Литература

1. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: <https://ru.wikipedia.org/wiki/Моном_(геометрическое_программирование)>
2. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: <https://ru.wikipedia.org/wiki/Многочлен>