Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение

высшего образования «Национальный исследовательский Нижегородский

государственный университет им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

**Отчет по учебной практике**

**АНАЛИТИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ ПОЛИНОМОВ ОТ НЕСКОЛЬКИХ ПЕРЕМЕННЫХ (СПИСКИ)**

Выполнил: Власов Максим Сергеевич, студент группы 381806-1

Проверил: к. т. н., доцент кафедры МОСТ Кустикова В. Д.

Нижний Новгород

2019

**Содержание**

[Введение 3](#_Toc27517175)

[1. Постановка задачи 4](#_Toc27517176)

[2. Руководство пользователя 5](#_Toc27517177)

[3. Руководство программиста 7](#_Toc27517178)

[3.1. Структура программы 7](#_Toc27517179)

[3.2. Описание алгоритма 7](#_Toc27517180)

[3.2.1. Структура данных Список 7](#_Toc27517181)

[3.2.2. Полином 7](#_Toc27517182)

[3.3. Описание структур данных и функций 9](#_Toc27517183)

[3.3.1. Класс TList 9](#_Toc27517184)

[3.3.2. Класс TMonomial 14](#_Toc27517185)

[3.3.3. Основная программа 16](#_Toc27517186)

[Заключение 17](#_Toc27517187)

[Список используемых источников 18](#_Toc27517188)

[Приложение 19](#_Toc27517189)

# Введение

В повседневной жизни человеку постоянно необходимо выполнять вычисления различных арифметических выражений разного уровня сложности как в прикладных, так и в научных целях. Для того чтобы избежать ошибок, можно поручить это компьютеру с помощью специальных приложений.

Таким образом, для этого и было создано консольное приложение, работающее с полиномами от нескольких переменных (с некоторыми ограничениями). При этом используется динамическая структура данных Список, на практическое освоение которой и направлена данная лабораторная работа.

# Постановка задачи

**Задача:** разработать и реализовать приложение, выполняющее алгебраические операции (сложение, вычитание, умножение) над полиномами от нескольких переменных.

**Входные данные:** полиномы, операции.

**Выходные данные:** полиномы.

# Руководство пользователя

В данном руководстве содержатся пошаговые инструкции по работе с программой, для того чтобы вы могли как можно быстрее приступить к использованию приложения.

1. Запустите файл **02\_Stack.exe** из папки с программой. Перед вами отобразится приветственный экран с предложением ввести арифметическое выражение, представленное в классической (инфиксной) форме.



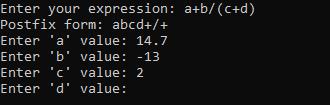
1. Программа после запуска.
2. С помощью клавиатуры введите выражение и нажмите Enter. Если вы указали неверное выражение (например, присутствуют недопустимые символы, то есть, любые, кроме a-z, A-Z, +, -, \*, /, (, ), или допущена логическая ошибка, как-то: более одного операнда рядом, не соблюдена вложенность скобок, бинарные операторы имеют менее двух операндов, или названия переменных содержат более одного символа), то программа сообщит вам об этом соответствующим сообщением об ошибке (см. Рис. 2). Повторите ввод, перезапустив приложение. Если арифметическое выражение корректное, программа преобразует его в постфиксную форму и выведет (см. Рис. 3).



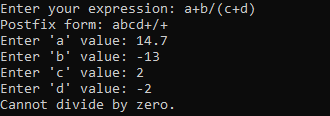
1. Сообщение об ошибке при вводе неверного выражения.



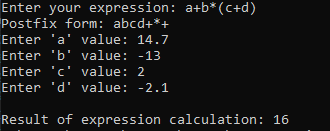
1. Программа после ввода корректного выражения.



1. Программа во время ввода значений переменных.
2. Если арифметическое выражение введено верно (программа не обнаружила ошибок), то далее вам будет предложено ввести значения переменных. Последовательно введите значения указанных переменных, нажимая Enter после ввода значения (см. Рис. 4).



1. Сообщение об ошибке при делении на ноль.
2. После ввода значений переменных программа выполнит попытку вычисления значения выражения. В случае если на каком-то этапе вычислений будет осуществлено деление на ноль, программа покажет сообщение об ошибке (см. Рис. 5). Перезапустите приложение и введите выражение и допустимые значения переменных заново. Если вычисление завершится успешно, программа выведет его (см. Рис. 6). После этого вы можете закрыть программу, для этого нажмите «крестик» в правом верхнем углу окна.



1. Успешное завершение программы.

# Руководство программиста

## Структура программы

Исходный код программы содержится в следующих файлах:

1. **02\_Polynomial.cpp** – основной модуль (консольное приложение, использующее модули ниже).
2. **TList.h** – модуль Список (содержит объявление и реализацию класса для работы со структурой данных Список, включая классы узлов, итераторов и исключений).
3. **TMonomial.h** – модуль Моном (содержит объявление и реализацию специализации узла списка, используемого в модуле выше, включая специальные классы исключений).
4. **TPolynomial.h** – модуль Полином (содержит объявление класса для работы с полиномами от нескольких переменных с некоторыми ограничениями на степени, включая классы исключений).
5. **TPolynomial.cpp** – модуль Полином (содержит реализацию методов класса для работы с полиномами).

## Описание алгоритма

### Структура данных Список

Линейный односвязный список – базовая динамическая структура данных, состоящая из узлов, каждый из которых содержит как собственно данные, так и указатель на следующий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

Для работы со списками предлагается реализовать следующие операции:

* методы проверка на пустоту и проверки на полноту списка;
* методы навигации по списку (итератор);
* методы вставки в начало, в конец, после звена с заданным ключи и перед звеном с заданным ключом;
* методы удаления звена с заданным ключом и поиск звена с заданным ключом.

### Полином

Полином можно определить также как выражение из нескольких термов, соединенных знаками сложения или вычитания. Терм включает коэффициент и моном, содержащий одну или несколько переменных, каждая из которых может иметь степень. При выполнении данной лабораторной работы предполагается, что степени переменных в мономах могут принимать значения 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Для организации быстрого доступа может быть использовано упорядоченное хранение мономов. Для задания порядка следования можно принять лексикографическое упорядочивание по степеням переменных, при котором мономы упорядочиваются по степеням первой переменной, потом по второй переменной, и только затем по третьей переменной. В общем виде это правило можно записать как соотношение: моном предшествует моному тогда и только тогда, если

(A1 > A2) ˅ (A1 = A2) & (B1 > B2) ˅ (A1 = A2) & (B1 = B2) & (C1 > C2).

Проверка лексикографического порядка занимает сравнительно много времени. Ее можно существенно упростить при помощи свернутой степени (индекса) монома, образуемой с использованием позиционной системы счисления: для монома со степенями (A, B, C) ставится в соответствие величина

ABC = A \* 100 + B \* 10 + C.

Данное соответствие является взаимно-однозначным. Обратное соответствие определяется при помощи выражений

A= [ABC / 100], B= [ABC % 100 / 10], C=ABC % 10.

Кроме того, введенное соответствие порождает порядок, полностью совпадающий с лексикографическим порядком

> ‎‎↔ ABC1 > ABC2.

Выполненное обсуждение позволяет определить, что наиболее эффективным способом организации структуры хранения полиномов являются линейный (односвязный) список. Тем самым, в рамках лабораторной работы появляется подзадача – разработка структуры хранения в виде линейных списков. Данная разработка должна быть выполнена в некоторой общей постановке с тем, чтобы разработанные программы работы со списками могли быть далее использованы и в других ситуациях, в которых необходимы списковые структуры хранения.

## Описание структур данных и функций

### Класс TList

Объявление класса для работы со списком выглядит следующим образом:

template <typename TKey, typename TData>

class TList

{

public:

class TNode

{

template <typename, typename> friend class TList;

template <typename> friend class TListIterator;

TNode\* pNext;

public:

TKey key;

TData data;

explicit TNode(TKey key = 0, TData data = 0, TNode\* pNext = nullptr);

TNode(const TNode& other);

};

private:

TList::TNode\* pFirst;

public:

TList();

TList(const TList& other);

TList(const TList<TKey, TData>::TNode\* firstNode);

~TList();

typedef TListIterator<TList::TNode> iterator;

typedef TListIterator<TList::TNode> const\_iterator;

typename TNode\* find(TKey needle);

typename TNode\* getFirst();

void insertToStart(TKey key, TData data = 0);

void insertToEnd(TKey key, TData data = 0);

void insertBefore(TKey needle, TKey key, TData data = 0);

void insertBefore(iterator i, TKey key, TData data = 0);

void insertAfter(TKey needle, TKey key, TData data = 0);

void insertAfter(iterator i, TKey key, TData data = 0);

void remove(TKey needle);

void remove(iterator& i);

void removeAll();

size\_t size() const;

bool empty() const;

typename iterator begin();

typename iterator end();

typename const\_iterator begin() const;

typename const\_iterator end() const;

void output(std::ostream& stream, const char\* separator = " ", const char\* ending = "\n") const;

};

namespace TListException

{

class NodeNotFound : ::std::exception

{

const ::std::string whatStr = "Node with given key not found.";

public:

virtual const char\* what() { return whatStr.c\_str(); }

};

};

#### Пользовательские типы данных

TNode – класс узла (элемента) линейного односвязного списка.

#### Классы исключений

NodeNotFound – узел с заданным ключом не найден (при попытке найти, вставить, удалить элемент).

#### Поля класса

TNode\* pFirst;

**Назначение:** хранит указатель на первый узел линейного односвязного списка.

#### Методы класса

TList();

**Назначение:** конструктор по умолчанию.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

TList(const TList& other);

**Назначение:** конструктор копирования.

**Входные параметры:** other – копируемый список.

**Выходные данные:** отсутствуют.

TList(const TNode\* firstNode);

**Назначение:** конструктор преобразования цепи связанных узлов в односвязный список.

**Входные параметры:** firstNode – указатель на первый узел.

**Выходные данные:** отсутствуют.

~ TList();

**Назначение:** деструктор.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

typename TNode\* find(TKey needle);

**Назначение:** поиск элемента в списке по ключу.

**Входные параметры:** needle – ключ искомого элемента.

**Выходные данные:** отсутствуют.

typename TNode\* getFirst();

**Назначение:** получение первого элемента списка.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

void insertToStart(TKey key, TData data);

**Назначение:** вставка элемента в начало списка.

**Входные параметры:** key – ключ вставляемого элемента, data – данные вставляемого элемента.

**Выходные данные:** отсутствуют.

void insertToEnd(TKey key, TData data);

**Назначение:** вставка элемента в конец списка.

**Входные параметры:** key – ключ вставляемого элемента, data – данные вставляемого элемента.

**Выходные данные:** отсутствуют.

void insertBefore(TKey needle, TKey key, TData data);

**Назначение:** вставка элемента перед заданным элементом.

**Входные параметры:** needle – ключ элемента, перед которым нужно вставить новый элемент, key – ключ вставляемого элемента, data – данные вставляемого элемента.

**Выходные данные:** отсутствуют.

void insertBefore(iterator i, TKey key, TData data);

**Назначение:** вставка элемента перед заданным элементом.

**Входные параметры:** i – итератор, установленный на элемент, перед которым нужно вставить новый элемент, key – ключ вставляемого элемента, data – данные вставляемого элемента.

**Выходные данные:** отсутствуют.

void insertAfter(TKey needle, TKey key, TData data);

**Назначение:** вставка элемента после заданного элемента.

**Входные параметры:** needle – ключ элемента, после которого нужно вставить новый элемент, key – ключ вставляемого элемента, data – данные вставляемого элемента.

**Выходные данные:** отсутствуют.

void insertAfter(iterator i, TKey key, TData data);

**Назначение:** вставка элемента после заданного элемента.

**Входные параметры:** i – итератор, установленный на элемент, после которого нужно вставить новый элемент, key – ключ вставляемого элемента, data – данные вставляемого элемента.

**Выходные данные:** отсутствуют.

void remove(TKey needle);

**Назначение:** удаление заданного элемента из списка.

**Входные параметры:** needle – ключ удаляемого элемента.

**Выходные данные:** отсутствуют.

void remove(iterator i);

**Назначение:** удаление заданного элемента из списка.

**Входные параметры:** i – итератор, установленный на удаляемый элемент.

**Выходные данные:** отсутствуют.

void removeAll();

**Назначение:** удаление всех элементов из списка.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

size\_t size() const;

**Назначение:** подсчет количества элементов в списке.

**Входные параметры:** отсутствуют.

**Выходные данные:** количество элементов в списке.

bool empty() const;

**Назначение:** проверяет, является ли список пустым.

**Входные параметры:** отсутствуют.

**Выходные данные:** истина или ложь.

typename iterator begin();

**Назначение:** получение начального итератора для списка.

**Входные параметры:** отсутствуют.

**Выходные данные:** итератор.

typename iterator end();

**Назначение:** получение конечного итератора для списка.

**Входные параметры:** отсутствуют.

**Выходные данные:** итератор.

typename const\_iterator begin() const;

**Назначение:** получение начального итератора для константного списка.

**Входные параметры:** отсутствуют.

**Выходные данные:** итератор.

typename const\_iterator end() const;

**Назначение:** получение конечного итератора для константного списка.

**Входные параметры:** отсутствуют.

**Выходные данные:** итератор.

void output(std::ostream& stream, const char\* separator, const char\* ending) const;

**Назначение:** вывод элементов списка в поток.

**Входные параметры:** stream – поток вывода, separator – разделитель между элементами списка, ending – символы, выводимые за последним элементом.

**Выходные данные:** отсутствуют.

### Класс TMonomial

Для работы с арифметическими выражениями и их постфиксными формами был спроектирован класс PostfixFormProcessor, содержащий набор статических методов, а также вспомогательные пользовательские типы данных и перечисления. Объявление представлено ниже.

template<>

class TList<unsigned, double>::TNode

{

public:

class DegreeOverflow : std::exception

{

const std::string whatStr = "Degree cannot be greater than 9.";

public:

virtual const char\* what() { return whatStr.c\_str(); }

};

class DegreeUnequality : std::exception

{

const std::string whatStr = "Degrees are not equal.";

public:

virtual const char\* what() { return whatStr.c\_str(); }

};

private:

template <typename, typename> friend class TList;

template <typename> friend class TListIterator;

friend class TPolynomial;

friend std::ostream& operator<<(std::ostream& stream, const TPolynomial& polynomial);

unsigned key;

double data;

TNode\* pNext;

bool checkDegrees(unsigned degrees) const;

bool checkDegrees(unsigned x, unsigned y, unsigned z) const;

public:

explicit TNode(unsigned degree = 0U, double coefficient = 0., TNode\* pNext\_ = nullptr);

TNode(const TNode&) = default;

TNode(double coefficient, unsigned degree);

TNode operator+(const TNode& other) const;

TNode operator-(const TNode& other) const;

TNode operator\*(const TNode& other) const;

TNode operator\*(double number) const;

TNode operator+() const;

TNode operator-() const;

};

#### Классы исключений

1. DegreeOverflow – одна или несколько степеней переменных монома превышает 9.
2. DegreeUnequality – степени мономов различаются (при сложении, вычитании).

#### Методы класса

bool checkDegrees(unsigned degrees) const;

**Назначение:** проверка свертки степени монома.

**Входные параметры:** degrees – свертка.

**Выходные данные:** истина или ложь.

bool checkDegrees(unsigned x, unsigned y, unsigned z) const;

**Назначение:** проверка степеней монома.

**Входные параметры:** x – степень при переменной x, y – степень при переменной y, z – степень при переменной z.

**Выходные данные:** истина или ложь.

explicit TNode(unsigned degree, double coefficient, TNode\* pNext);

**Назначение:** конструктор ().

**Входные параметры:** degree.

**Выходные данные:** отсутствуют.

### Основная программа

int main()

**Назначение:** основная функция (точка входа).

**Входные параметры:** отсутствуют.

# Заключение

В ходе выполнения практической работы «Аналитические преобразования полиномов от нескольких переменных (списки)» было разработано и реализовано консольное приложение для работы с полиномами от трех переменных, включая выполнение алгебраических операций над ними, использующее динамическую структуру данных Список.

# Список используемых источников

1. Лабораторный практикум. Барышева И. В., Мееров И. Б., Сысоев А. В., Шестакова Н. В. Под ред. Гергеля В. П. Учебно-методическое пособие. – Нижний Новгород (ННГУ), 2017. – 105 с.
2. Рабочие материалы к учебному курсу «Методы программирования» (часть 1). Гергель В. П. Нижний Новгород, 2015. – 100 с.

# Приложение

**Приложение 1.** Исходный код основной программы

#include <iostream>

#include "PostfixFormProcessor.h"

int main()

{