Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

# Обратная польская запись

Выполнил:

Студент и-та ИТММ гр. 0823-2

Кипаренко И.В.

Проверил:

ассистент каф. МО ЭВМ, ИИТММ

Сиднев А.А.

Нижний Новгород

2015 г.

Содержание

[Обратная польская запись 1](#_Toc439226084)

[Введение 3](#_Toc439226085)

[Постановка задачи 4](#_Toc439226086)

[Руководство пользователя 5](#_Toc439226087)

[Руководство Программиста 6](#_Toc439226088)

[Используемые инструменты 6](#_Toc439226089)

[Описание структуры программы 6](#_Toc439226090)

[Описание структуры данных 6](#_Toc439226091)

[Дополнительные классы: 8](#_Toc439226092)

[Описание алгоритмов 8](#_Toc439226093)

[Тесты 10](#_Toc439226094)

[Тесты для методов класса List 10](#_Toc439226095)

[Тесты для методов класса Stack 10](#_Toc439226096)

[Тесты для методов класса Expression 11](#_Toc439226097)

[Заключение 12](#_Toc439226098)

[Литература 13](#_Toc439226099)

[Приложения 14](#_Toc439226100)

[Приложение 1. Пример работы sample.exe 14](#_Toc439226101)

# Введение

Потребность вычислять арифметические выражения возникла давно и до сих пор актуальна. В данной лабораторной работе решается задача вычисления некоторых выражений путем преобразования их в обратную польскую запись. Механизм польской записи позволяет решить проблему со скобками и приоритетами операций в выражении. Помимо этого, обратную польскую запись легко вычислять на стеке. Такой способ вычислять выражения активно применялся и применяется в микрокалькуляторах.

# Постановка задачи

Реализовать динамическую структуру данных стек на основе динамической структуры список. С помощью стека реализовать алгоритм перевода инфиксной записи арифметического выражения в постфиксную. Разработать программу, выполняющую вычисление арифметического выражения в постфиксной форме записи. Создать консольное приложение, демонстрирующее работу алгоритма, где входные данные - арифметическое символьное выражение в инфиксном виде и значения каждого параметра, а результат - запись исходного арифметического символьного выражения в постфиксном виде, численный результат. Написать тестирующую программу для каждой структуры данных с помощью Google C++ Testing Framework.

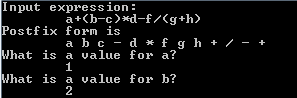
# Руководство пользователя

Данная программа предназначена для вычисления арифметического выражения в инфиксной форме записи, с предварительным переводом в постфиксную форму. Для запуска программы необходимо открыть исполняемый файл Evaluate.

После этого пользователю предлагается ввести выражение, в котором допускаются алгебраические операции (+, -, /, \*, ^), открывающие и закрывающие скобки, буквенные символы, которые обозначают переменные.



Когда арифметическое выражение будет введено, программа выведет постфиксную форму и запросит ввести значения переменных.



Если при вводе была совершена ошибка, программа завершит свою работу с соответствующим сообщением.

# Руководство Программиста

## Используемые инструменты

В ходе лабораторной работы использовались следующие инструменты:

1. Система контроля версий Git.

2. Фреймворк для написания автоматических тестов Google Test.

3. Среда разработки Microsoft Visual Studio.

## Описание структуры программы

Программа состоит из 4 проектов:

1. Pieces - проект, содержащий объявление и реализацию шаблонных классов CNode, List, Stack, а также классов Expression и Symbol .
   1. CNode.h - заголовочный файл, содержащий объявление класса СNode.
   2. List.h - заголовочный файл, содержащий объявление класса List.
   3. Stack.h - заголовочный файл, содержащий объявление класса Stack.
   4. Expression.h - заголовочный файл, содержащий объявление класса Expression.
   5. List.cpp - файл, содержащий реализацию методов класса List.
   6. Stack.cpp - файл, содержащий реализацию методов класса Stack.
   7. Expression.cpp - файл, содержащий реализацию методов класса Expression.
2. Evaluate - консольное приложение, демонстрирующее работу алгоритма перевода из инфиксной формы в постфиксную форму, а так же вычисление значения выражения.
3. Tests - консольное приложение, использующее библиотеку Google Test, проверяющее корректность реализации классов List , Stack и Expression.
4. gtest - фреймворк Google Test.

## Описание структуры данных

Структура данных список

Односвязный линейный список - динамическая структура данных, состоящая из однотипных "узлов", каждый из которых содержит данные определенного типа и указатель на последующий узел списка. Указатель последнего элемента списка равен нулю, что является признаком конца списка. Указателем на список является указатель на его первый элемент (start).



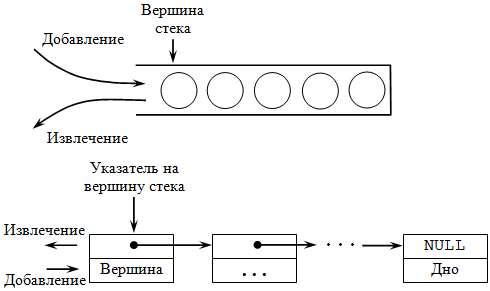
В данной лабораторной работе структура данных "список" представлена в виде класса list, который содержит в себе следующие методы:

1. Конструктор по умолчанию.
2. Конструктор копирования списков.
3. Деструктор.
4. find - поиск элемента списка с заданным ключом
5. add1 - вставка нового элемента в начало списка.
6. add - вставка нового элемента в конец списка.
7. add - вставка нового элемента после заданного элемента.
8. del1 – удаление первого элемента списка.
9. del – удаление элемента с заданным ключом.
10. del – удаление элемента после заданного элемента.
11. GetFirst - метод, возвращающий указатель на первый элемент списка.

Структура данных стек

Стек - динамическая структура данных, представляющая собой список элементов, организованных по принципу FILO (англ. first in - last out, «первым пришёл - последним вышел»).

В данной лабораторной работе структура данных "стек" реализована на основе односвязного линейного списка, то есть каждый элемент содержит помимо хранимой информации в стеке указатель на следующий элемент стека.



В данной лабораторной работе структура данных "стек" представлена в виде класса stack, который содержит следующие методы:

1. Конструктор по умолчанию, который явно вызывает конструктор класса list.

2. Конструктор копирования.

3. Деструктор.

4. isEmpty - метод проверки стека на пустоту.

5. isFull - метод проверки стека на полноту.

6. push - добавление элемента на вершину стека.

7. pop - изъятие элемента с вершины стека.

9. Оператор == - перегрузка оператора сравнения.

# Дополнительные классы:

1. **Expression:**

Этот класс хранит постфиксное представление некой функции, может вычислять её запрашивая параметры у пользователя , или же принимать их все сразу.

Его методы:

1. Конструктор, принимающий строку в инфиксной записи, и превращающий её в постфиксную.
2. Метод eval , в двух версиях: принимающий список переменных со значениями или нет (в этом случае запрашивает их значения у пользователя с клавиатуры), и вычисляющие знчение этой функции при данных значения переменных.
3. **Symbol:**

Этот класс отвечает за хранение и взаимодействие различных объектов в стеке, для хранения и вычисления арифметических выражений.

Его методы:

1. Перегруженный оператор равенства.
2. Перегруженный оператор сравнения .
3. Функция me ,для определения какому из наследником принадлежит указатель на базовый класс.
4. Виртуальная функция печати в поток вывода print .
5. Виртуальная функция вычисления текущего значения value.

Также у каждого из классов наследников : Numb (хранит вещественные числа) , Var (хранит имена и значения переменных) , Op(хранит операции и их приоритеты) , есть

1. Конструктор.
2. Перегруженный оператор сравнения.
3. Перегрузка виртуальной функции печати print.
4. Перегрузка виртуальной функции «узнавания» me.
5. Перегрузка виртуальной функции вычисления value.

Также у класса Var , дополнительно есть конструктор копирования и деструктор.

## Описание алгоритмов

Алгоритм перевода в постфиксную форму записи

1. У каждой операции есть свой приоритет.

1. Операции возведения в степень ^ приоритет 4.
2. Операции умножения \* и деления /  приоритет 3.
3. Операции сложения + и вычитания - приоритет 2.
4. Операции открывающей скобки ( и закрывающей ) приоритет 1.
5. Операции равенства = приоритет 0.

2. Используется два стека: стек операций op и результирующий стек out.

3. Выражение просматривается слева - направо. При этом возможны 4 ситуации:

а) Встретился операнд. Тогда он добавляется в стек out.

б) Встретилась операция, приоритет которой выше, чем приоритет операции, лежащей на вершине стека op или стек op пуст. В этом случае операция добавляется в стек op.

в) Встретилась операция, приоритет которой равен или ниже приоритета операции, лежащей на вершине стека op. В этом случае все операции, приоритет которых выше данной перекладываются в стек out до тех пор, пока на вершине стека op не появится операция с меньшим приоритетом или operation не станет пустым. Новая операция добавляется в стек op.

г) Встретилась операция закрывающая скобка. В этом случае из стека operation перекладываются все операции в стек out до первого вхождения операции открывающая скобка. Операция открывающая скобка удаляется из стека op.

4. Если выражение закончилось, то все операции из стека op перекладываются в стек out.

Алгоритм подсчета выражения в постфиксной форме записи

1. Создается один стек operand.

2. Выражение рассматривается слева - направо. Возможны 2 ситуации:

а) Встретился операнд. В таком случае у пользователя запрашивается его значение и добавляется на вершину стека operand.

б) Встретилась операция. Тогда из стека operand изымаются 2 операнда, над ними производится операция, результат операции снова добавляется в стек.

3. При достижении конца арифметического выражения, в стеке будет находиться численный результат выражения.

# Тесты

## Тесты для методов класса List

1. TEST(List, can\_create\_list) – тест, проверяющий возможно ли создать список.
2. TEST(List, can\_create\_copied\_list) – тест, проверяющий можно ли скопировать список.
3. TEST(List, copied\_list\_is\_equal\_to\_sourse\_one) – тест, проверяющий равен ли скопированный список исходному.
4. TEST(List, copied\_list\_has\_its\_own\_memory) – тест, проверяющий есть ли у скопированного списка своя выделенная память.
5. TEST(List, can\_find\_element) – тест, проверяющий метод поиска элемента списка с заданным ключом.
6. TEST(List, can\_delete\_element) – тест, проверяющий возможно ли удалить элемент списка с заданным ключом.
7. TEST(List, return\_null\_when\_delete\_not\_actual\_value) – тест, проверяющий возвращает ли ноль метод удаления элемента, если элемент не был найден.
8. TEST(List, can\_delete\_first\_element) – тест, проверяющий возможно ли удалить первый элемент списка.
9. TEST(List, throws\_when\_try\_delete\_in\_empty\_list) - тест, проверяющий возвращает ли метод удаления элемента исключение, если удаление происходит в пустом списке.
10. TEST(List, can\_insert\_first\_element) – тест, проверяющий возможно ли вставить элемент в список на первую позицию.
11. TEST(List, can\_insert\_last\_element) – тест, проверяющий возможно ли вставить элемент в список на последнюю позицию.
12. TEST(List, can\_get\_first) – тест, проверяющий возможно ли получить указатель на первый элемент списка.

## Тесты для методов класса Stack

1. TEST(Stack, can\_create\_stack) - тест, проверяющий возможно ли создать стек.
2. TEST(Stack, can\_create\_copied\_stack) - тест, проверяющий можно ли скопировать стек.
3. TEST(Stack, copied\_stack\_is\_equal\_to\_sourse\_one) - тест, проверяющий равен ли скопированный стек исходному.
4. TEST(Stack, copied\_stack\_has\_its\_own\_memory) - тест, проверяющий есть ли у скопированного стека своя выделенная память.
5. TEST(Stack, IsEmpty\_returns\_1\_when\_stack\_is\_empty) - тест, проверяющий, что метод проверки стека на пустоту возвращает 1, если стек пуст.
6. TEST(Stack, IsEmpty\_returns\_0\_when\_stack\_isnt\_empty) - тест, проверяющий, что метод проверки стека на пустоту возвращает 0, если стек не пуст.
7. TEST(Stack, IsFull\_returns\_0\_when\_stack\_isnt\_full) - тест, проверяющий, что метод проверки стека на полноту возвращает 0, если стек не полон.
8. TEST(Stack, can\_push\_new\_element) - тест, проверяющий возможно ли добавить новый элемент в стек.
9. TEST(Stack, cant\_pop\_element\_fron\_empty\_stack) - тест, проверяющий, что метод изъятия элемента из стека возвращает исключение, если стек пуст.
10. TEST(Stack, can\_pop\_element\_from\_not\_empty\_stack) - тест, проверяющий возможно ли изъять элемент из стека.
11. TEST(Stack, poped\_element\_is\_equal\_to\_pushed\_one) - тест, проверяющий, что первый изъятый элемент равен последнему добавленному.

## Тесты для методов класса Expression

1. TEST(Expression, can\_create\_expression) - тест, проверяющий возможно ли перевести арифметическое выражение в постфиксную форму.
2. TEST(Expression, can\_calculate\_expression\_in\_a\_postfix\_notation) - тест, проверяющий возможно ли вычислить арифметическое выражение в постфиксной форме.
3. TEST(Expression, throws\_when\_input\_invalid) - тест, проверяющий, что метод перевода арифметического выражения в постфиксную форму возвращает исключение, если выражение было введено неверно.
4. TEST(Expression, throws\_when\_try\_convert\_empty\_writing) - тест, проверяющий, что метод перевода арифметического выражения в постфиксную форму возвращает исключение, если выражение не было введено.

# Заключение

В ходе лабораторной работы была разработана программа, удовлетворяющая поставленным задачам. Структура стек и список были реализованы с использованием шаблонных классов, так как этого требовал алгоритм преобразования записи выражения. Написаны примеры использования списков и стеков, демонстрирующие работу методов соответствующих классов.

В процессе было написано 28тестов. Все тесты успешно пройдены.

Реализован алгоритм перевода арифметического выражения из инфиксной формы в постфиксную и вычисление его результата.

# Литература

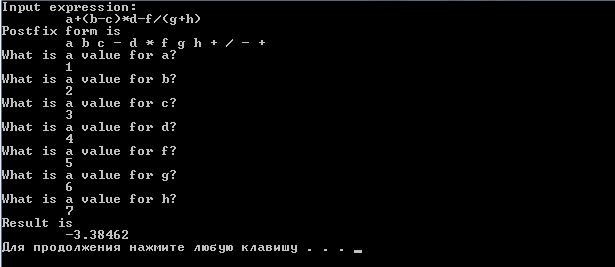
1. Обратная польская запись –[http://habrahabr.ru/post/100869/].

2. Альфред В. Ахо, Джон Хопкрофт, Джеффри Д. Ульман. Структуры данных и алгоритмы = Data Structures and Algorithms. — М.: Вильямс, 2000. — 384 с.

3. Майкл Мейн, Уолтер Савитч. Структуры данных и другие объекты в C++ = Data Structures and Other Objects Using C++. — 2-е изд. — М.: Вильямс, 2002. — 832 с.

# Приложения

## Приложение 1. Пример работы sample.exe

******