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

Жорин С.М.

Проверил:

ассистент каф. МО ЭВМ, ИИТММ

Сиднев А.А.

Нижний Новгород

2015 г.

Содержание

[Обратная польская запись 1](#_Toc439150667)

[Введение 3](#_Toc439150668)

[Постановка задачи 4](#_Toc439150669)

[Руководство пользователя 5](#_Toc439150670)

[Руководство Программиста 6](#_Toc439150671)

[Используемые инструменты 6](#_Toc439150672)

[Общая структура проекта 6](#_Toc439150673)

[Описание структуры программы 6](#_Toc439150674)

[Описание структуры данных 7](#_Toc439150675)

[Описание алгоритмов 8](#_Toc439150676)

[Тесты 10](#_Toc439150677)

[Тесты для методов класса list 10](#_Toc439150678)

[Тесты для методов класса stack 10](#_Toc439150679)

[Тесты для методов класса arithmetic 11](#_Toc439150680)

[Заключение 12](#_Toc439150681)

[Литература 13](#_Toc439150682)

[Приложения 14](#_Toc439150683)

[Приложение 1. Пример работы sample.exe 14](#_Toc439150684)

[Приложение 2. Пример работы sample\_list.exe 14](#_Toc439150685)

[Приложение 3. Пример работы sample\_stack.exe 15](#_Toc439150686)

# 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.

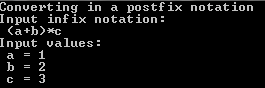
# Руководство пользователя

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

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

C:\Users\Vip\Desktop\раз.PNG

Когда арифметическое выражение будет введено, программа 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.

## Общая структура проекта

1. gtest - библиотека Google Test.

2. include - директория для размещения заголовочных файлов.

3. sample - директория для размещения исходного кода (cpp-файлы) тестовых приложения для стека, списка и функций для перевода и вычисления арифметических выражений.

4. sln - директория с файлами решений и проектов для Visual Studio.

5. src - директория для размещения исходных кодов (cpp-файлы).

6. test - директория с модульными тестами.

7. doc - отчет о выполненной лабораторной работе.

8. README.md - краткая постановка задачи.

## Описание структуры программы

Программа состоит из 7 проектов:

1. stack - проект, содержащий объявление и реализацию шаблонных классов node, list, stack.

a) list.h - заголовочный файл, содержащий объявление классов node, list.

б) stack.h - заголовочный файл, содержащий объявление класса stack.

в) list.cpp - файл, содержащий исходный код реализации методов класса list.

г) stack.cpp - файл, содержащий исходный код реализации методов класса stack.

2. arithmetic - проект, содержащий реализацию функций перевода выражений в постфиксную форму и вычисления выражений в постфиксной форме.

3. sample - консольное приложение, демонстрирующее работу алгоритма перевода из инфиксной формы в постфиксную форму, а так же вычисление значения выражения.

4. Sample\_list - консольное приложение, демонстрирующее работу методов класса list.

5. Sample\_stack - консольное приложение, демонстрирующее работу методов класса stack.

6. test - консольное приложение, использующее библиотеку Google Test, проверяющее корректность реализации классов list и stack.

7. 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 (pFirst).



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. poisk - поиск элемента списка с заданным ключом

5. del - удаление элемента списка с заданным ключом.

6. insertF - вставка нового элемента в начало списка.

7. insertL - вставка нового элемента в конец списка.

8. insertA - вставка нового элемента после элемента с заданным ключом.

9. GetFirst - метод, возвращающий указатель на первый элемент списка.

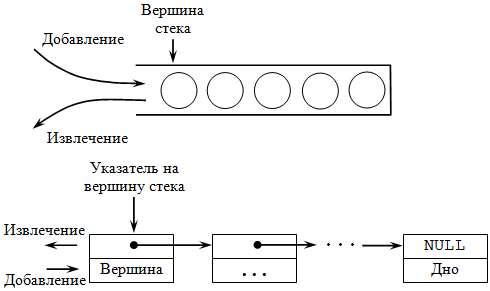
10. print - печать списка.

Пример использования данной структуры данных содержится в приложении Sample\_list.exe.

Структура данных стек

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

8. GetFirstVal - метод просмотра элемента на вершине стека.

9. Оператор == - перегрузка оператора сравнения.

Пример использования данной структуры данных содержится в приложении Sample\_ stack.exe.

## Описание алгоритмов

Алгоритм перевода в постфиксную форму записи

1. У каждой операции есть свой приоритет. Для определения этого приоритета используется функция prior, которая возвращает номер приоритета операции:

a) Операциям умножения \* и деления / наивысший приоритет, равный 3.

б) Операциям сложения + и вычитания - приоритет 2.

в) Операции открывающей скобки ( приоритет 1.

г) Операции равенства = приоритет 0.

2. Используется два стека: стек операций operation и стек операндов operand.

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

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

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

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

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

4. Если выражение закончилось, то все операции из стека operation перекладываются в стек operand.

Алгоритм подсчета выражения в постфиксной форме записи

1. Создается один стек с вещественным типом данных 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\_print\_empty\_list) – тест, проверяющий вызов вывода на экран для пустого списка.
6. TEST(List, can\_find\_element) – тест, проверяющий метод поиска элемента списка с заданным ключом.
7. TEST(List, throws\_when\_cant\_find\_element) – тест, проверяющий возвращает ли метод поиска элемента исключение, если заданный ключ не был найден.
8. TEST(List, throws\_when\_search\_in\_empty\_list) - тест, проверяющий возвращает ли метод поиска элемента исключение, если поиск происходит в пустом списке.
9. TEST(List, can\_delete\_element) – тест, проверяющий возможно ли удалить элемент списка с заданным ключом.
10. TEST(List, return\_null\_when\_delete\_not\_actual\_value) – тест, проверяющий возвращает ли ноль метод удаления элемента, если элемент не был найден.
11. TEST(List, can\_delete\_first\_element) – тест, проверяющий возможно ли удалить первый элемент списка.
12. TEST(List, throws\_when\_try\_delete\_in\_empty\_list) - тест, проверяющий возвращает ли метод удаления элемента исключение, если удаление происходит в пустом списке.
13. TEST(List, can\_insert\_first\_element) – тест, проверяющий возможно ли вставить элемент в список на первую позицию.
14. TEST(List, can\_insert\_last\_element) – тест, проверяющий возможно ли вставить элемент в список на последнюю позицию.
15. TEST(List, can\_insert\_after) – тест, проверяющий возможно ли вставить элемент в список после элемента с заданным ключом.
16. TEST(List, can\_insert\_after\_when\_it\_last) – тест, проверяющий возможно ли вставить элемент в список после элемента с заданным ключом, если этот элемент последний.
17. TEST(List, throws\_when\_cant\_find\_place\_to\_insert\_after) – тест, проверяющий возвращает ли метод класса вставки элемента в список после элемента с заданным ключом исключение, если этот элемент не был найден.
18. TEST(List, throws\_when\_insert\_after\_in\_empty\_list) - тест, проверяющий возвращает ли метод класса вставки элемента в список после элемента с заданным ключом исключение, если вставка происходит в пустом списке.
19. 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 arithmetic

1. TEST(arithmetic, can\_convert\_in\_a\_postfix\_notation) - тест, проверяющий возможно ли перевести арифметическое выражение в постфиксную форму.
2. TEST(arithmetic, can\_calculate\_expresion\_in\_a\_postfix\_notation) - тест, проверяющий возможно ли вычислить арифметическое выражение в постфиксной форме.
3. TEST(arithmetic, throws\_when\_input\_invalid) - тест, проверяющий, что метод перевода арифметического выражения в постфиксную форму возвращает исключение, если выражение было введено неверно.
4. TEST(arithmetic, 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 35 тестов. Все тесты успешно пройдены.

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

# Литература

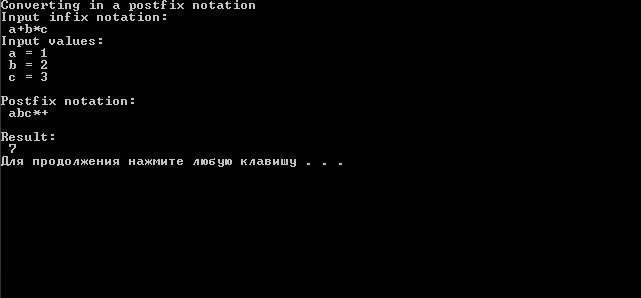
1. Обратная польская запись –[http://habrahabr.ru/post/100869/].

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

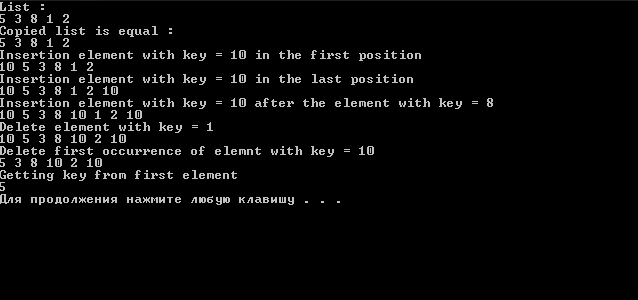
3. Майкл Мейн, Уолтер Савитч. Структуры данных и другие объекты в C++ = Data Structures and Other Objects Using C++. — 2-е изд. — М.: Вильямс, 2002. — 832 с.

# Приложения

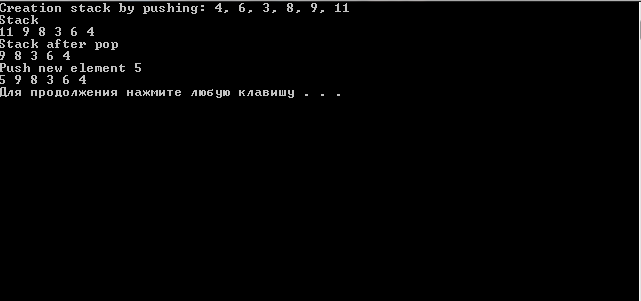
## Приложение 1. Пример работы sample.exe

******

## Приложение 2. Пример работы sample\_list.exe

******

## Приложение 3. Пример работы sample\_stack.exe

******