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

Выполнила:

Студентка и-та ИТММ гр. 0823-2

Кабалова А.А.

Проверил:

ассистент каф. МО ЭВМ, ИИТММ

Сиднев А.А.

Нижний Новгород

2015 г.

Содержание

[Введение 3](#_Toc439150770)

[Постановка задачи 4](#_Toc439150771)

[Руководство пользователя 5](#_Toc439150772)

[Руководство программиста 6](#_Toc439150773)

[Используемые инструменты: 6](#_Toc439150774)

[Общая структура проекта: 6](#_Toc439150775)

[Описание структуры программы: 6](#_Toc439150776)

[Описание структуры данных: 7](#_Toc439150777)

[Структура данных список 7](#_Toc439150778)

[Структура данных стек 7](#_Toc439150779)

[Описание алгоритмов: 8](#_Toc439150780)

[Алгоритм перевода в постфиксную форму записи 8](#_Toc439150781)

[Алгоритм подсчета выражения в постфиксной форме записи 9](#_Toc439150782)

[Описание тестов: 9](#_Toc439150783)

[Тесты для алгоритма перевода в постфиксную форму записи 9](#_Toc439150784)

[Тесты для списка 9](#_Toc439150785)

[Тесты для стека 10](#_Toc439150786)

[Заключение 12](#_Toc439150787)

[Литература 13](#_Toc439150788)

[Приложения 14](#_Toc439150789)

[Пример работы консольного приложения sample.exe 14](#_Toc439150790)

[Пример работы консольного приложения Sample\_list.exe 14](#_Toc439150791)

[Пример работы консольного приложения Sample\_stack.exe 15](#_Toc439150792)

# Введение

В мире существует огромное количество задач, которые исключают использование статичных структур данных и требуют введения динамических структур, способных увеличиваться в размерах в процессе работы программы. Если до начала работы с данными невозможно определить, сколько памяти потребуется для их хранения, то память должна выделяться по мере необходимости. Именно такие динамические структуры, как списки, стеки и очереди помогают программисту оптимально использовать память.

Говоря о задаче, которая поставлена в данной лабораторной работе, а именно вычисление арифметического выражения в инфиксной форме записи, то для ее реализации удобнее использовать такую структуру данных, как стек. С помощью стеков можно перевести инфиксную запись в постфиксную, которая более удобна для работы вычислительной машины.

# Постановка задачи

Необходимо разработать статическую библиотеку, которая реализует динамическую структуру данных - стек на основе другой динамической структуры данных - список. Для каждой такой структуры написать тестирующую программу с помощью Google C++ Testing Framework. Так же для каждой такой структуры написать консольные приложения, демонстрирующие работу стека и списка. С помощью стека реализовать алгоритм перевода из инфиксной формы записи арифметических выражений в постфиксную. Реализовать консольное приложение, демонстрирующее работу данного алгоритма.

Входные данные для перевода из инфиксной формы в постфиксную:

1. Арифметическое выражение в инфиксном виде.

2. Значение каждого параметра.

Выходные данные:

1. Арифметическое выражение в постфиксном виде.

2. Значение введенного выражения.

# Руководство пользователя

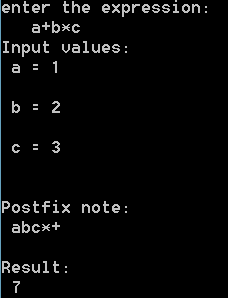
Данная программа предназначена для перевода символьного арифметического выражения из инфиксной записи в постфиксную. В дальнейшем производится вычисление введенного выражения на основе тех данных, которые вводит пользователь о каждой символьной переменной.

Для запуска программы необходимо открыть исполняемый файл sample.exe и далее следовать инструкциям программы.

Первым делом нужно ввести арифметическое выражение.

1.PNG

Далее необходимо ввести значение каждой буквы. И далее программа выведет постфиксную форму записи и вычисленное значение выражения.



Для завершения работы программы необходимо нажать на любую клавишу.

# 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 арифметических операций.

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 - статическая библиотека, содержащая объявление и реализацию шаблонных классов node, list и stack.

a) node - описывает узел списка. Узел хранит в себе значение val и указатель на следующий узел.

б) list - класс список.

в) stack - класс стек.

2. arithmetic - статическая библиотека, использующая функционал класса stack, содержащая класс arithmetic со статическими методами перевода арифметического выражения в постфиксную форму и вычисления введенного выражения.

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 - метод поиска элемента с заданным ключом, возвращающий указатель на элемент.

5. del - метод удаления элемента с заданным ключом, возвращающий указатель на элемент, который удаляется.

6. insertF - метод создания элемента с заданным ключом и вставки его в начало списка.

7. insertL - метод создания элемента с заданным ключом и вставки его в конец списка.

8. insertA - метод вставки элемента, на который передан указатель, после элемента с заданным ключом.

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

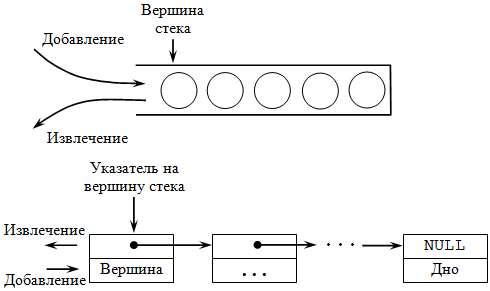
10. print - метод вывода элементов списка.

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 - метод добавления элемента с заданным значением на вершину стека.

7. pop - метод изъятия элемента с вершины стека. Метод возвращает значение элемента.

8. GetFirst - метод просмотра элемента на вершине стека.

9. operator == - перегруженный оператор сравнения. Для проверки стеков на равенство.

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.

## Описание тестов:

### Тесты для алгоритма перевода в постфиксную форму записи

1. TEST(arithmetic, can\_convert\_in\_a\_postfix\_notation) – тест, проверяющий верность перевода в постфиксную форму записи.
2. TEST(arithmetic, can\_calculate\_expresion\_in\_a\_postfix\_notation) – тест, который проверяет, правильно ли работает метод вычисления значения выражения.
3. TEST(arithmetic, throws\_when\_input\_invalid) – тест, показывающий, что при введении неправильного выражения программа его не обработает и оповестит пользователя о том, что выражение введено неверно.
4. TEST(arithmetic, throws\_when\_try\_convert\_empty\_writing) – тест, который показывает, что если пользователь не введет выражение, то программа его не обработает и оповестит пользователя о том, что введено пустое выражение.

### Тесты для списка

1. TEST(List, can\_create\_list) – тест, проверяющий, можно ли создать список.
2. TEST(List, can\_create\_copied\_list) – тест, проверяющий на работоспособность конструктор копирования.
3. TEST(List, copied\_list\_is\_equal\_to\_sourse\_one) – тест, проверяющий правильность работы конструктора копирования, равен ли скопированный список исходному.
4. TEST(List, copied\_list\_has\_its\_own\_memory) – тест, проверяющий, есть ли у скопированного списка своя память.
5. TEST(List, can\_print\_empty\_list) – тест, проверяющий, выводится ли информация о пустом списке.
6. TEST(List, can\_search\_element\_with\_actual\_value) – тест, который проверяет метод поиска узла с заданным ключом.
7. TEST(List, return\_null\_when\_cant\_search\_element) – тест, показывающий, что если производится поиск узла с заданным ключом, а такого узла не существует, то программа оповестит об этом.
8. TEST(List, throws\_when\_search\_in\_empty\_list) – тест, показывающий, что если происходит поиск узла с заданным ключом в пустом списке, то программа оповестит о том, что это невозможно.
9. TEST(List, can\_delete\_element\_with\_actual\_value) – тест, проверяющий, что можно удалить существующий узел с заданным ключем в списке.
10. TEST(List, return\_null\_when\_delete\_not\_actual\_value) – тест, показывающий, что невозможно удалить узел с заданным ключом, которого не существует в списке.
11. TEST(List, can\_delete\_first\_element) – тест, показывающий, что можно в непустом списке удалить первый узел.
12. TEST(List, can\_not\_find\_delete\_element) – тест, показывающий, что удаление работает верно. Т.е. при вызове поиска узла списка с заданным ключом, который ранее был удален, программа его не найдет.
13. TEST(List, throws\_when\_try\_delete\_in\_empty\_list) – тест, показывающий, что невозможно удалить узел в пустом списке.
14. TEST(List, can\_insert\_first\_element) – тест, показывающий, что можно вставить узел в начало списка.
15. TEST(List, can\_insert\_last\_element) – тест, показывающий, что можно вставить узел в конец списка.
16. TEST(List, can\_insert\_last\_element\_when\_list\_is\_empty) – тест, показывающий, что можно вставить элемент в пустой список.
17. TEST(List, can\_insert\_after) – тест, показывающий, что можно вставить в список новый узел после узла с заданным ключом.
18. TEST(List, can\_insert\_after\_when\_it\_last) – тест, показывающий, что метод вставки узла после работает и в том случае, если нужно вставить узел после последнего звена.
19. TEST(List, throws\_when\_cant\_find\_place\_to\_insert\_after) – тест, показывающий, что невозможно использовать метод вставки после звена с заданным ключом, если такого не существует в списке.
20. TEST(List, throws\_when\_insert\_after\_in\_empty\_list) – тест, показывающий, что невозможно использовать метод вставки после звена с заданным ключом, если список пуст.
21. TEST(List, can\_get\_first) – тест, показывающий, что можно получить значение ключа первого звена списка.

### Тесты для стека

1. TEST(Stack, can\_create\_stack) – тест, показывающий, что возможно создать стек.
2. TEST(Stack, can\_create\_copied\_stack) – тест, проверяющий на работоспособность конструктор копирования.
3. TEST(Stack, copied\_stack\_is\_equal\_to\_sourse\_one) – тест, проверяющий правильность работы конструктора копирования, равен ли скопированный стек исходному.
4. TEST(Stack, copied\_stack\_has\_its\_own\_memory) – тест, проверяющий, есть ли у скопированного стека своя память.
5. TEST(Stack, IsEmpty\_returns\_1\_when\_stack\_is\_empty) – тест, показывающий, что если стек пуст, то метод проверки стека на пустоту вернет 1.
6. TEST(Stack, IsEmpty\_returns\_0\_when\_stack\_isnt\_empty) – тест, показывающий, что если стек не пуст, то метод проверки стека на пустоту вернет 0.
7. TEST(Stack, IsFull\_returns\_0\_when\_stack\_isnt\_full) – тест, показывающий, что если стек не полон, то метод проверки стека на полноту вернет 0.
8. TEST(Stack, can\_push\_new\_element) – тест, показывающий, что можно добавить новый элемент в стек.
9. TEST(Stack, throws\_when\_try\_pop\_element\_when\_stack\_is\_empty) – тест, показывающий, что при вызове метода изъятия элемента для пустого списка, программа оповестит о том, что данная операция невозможна.
10. TEST(Stack, can\_pop\_element\_from\_no\_empty\_stack) – тест, показывающий, что можно изъять элемент из непустого списка.
11. TEST(Stack, poped\_element\_is\_equal\_to\_pushed\_one) – тест, показывающий, что первый изъятый элемент равен последнему добавленному.

# 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 36 тестов, которые покрывают всевозможные ситуации использования методов класса. Все тесты успешно пройдены.

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

# Литература

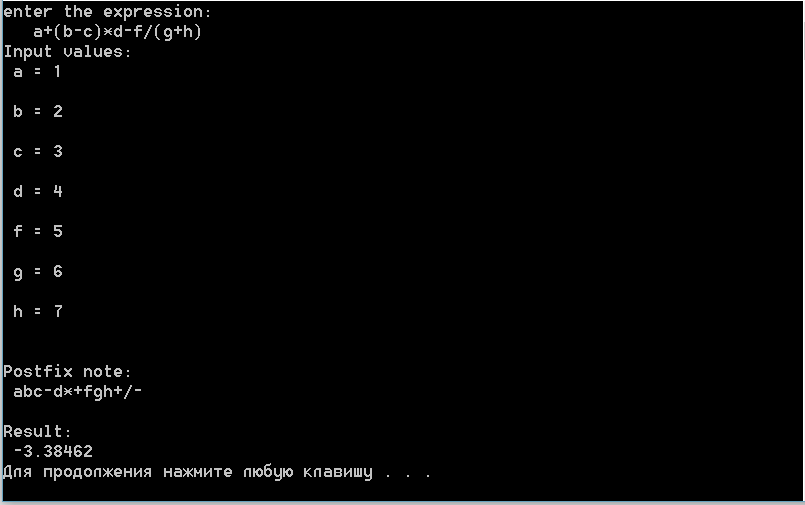
1. Обратная польская запись – [http://habrahabr.ru/post/100869/].

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

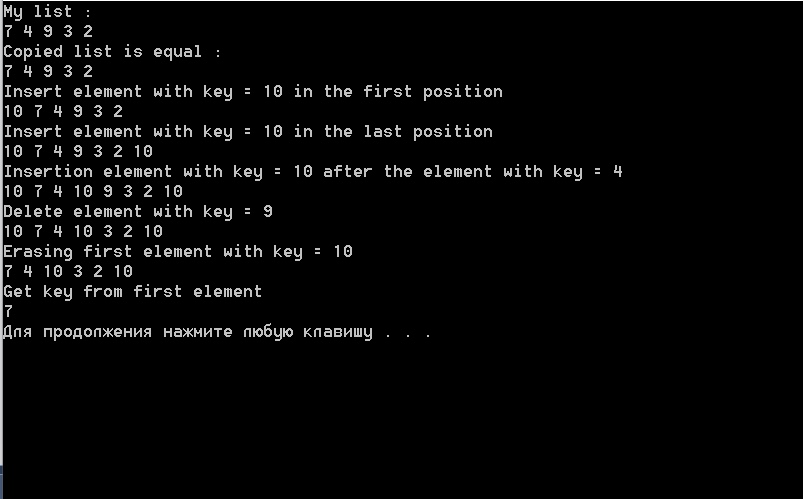
3. Майкл Мейн, Уолтер Савитч. Структуры данных и другие объекты в C++ = Data Structures and Other Objects Using C++. — 2-е изд. — М.: Вильямс, 2002. — 832 с

# Приложения

## Пример работы консольного приложения sample.exe



## Пример работы консольного приложения Sample\_list.exe



## Пример работы консольного приложения Sample\_stack.exe

