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

Чапыгин С. А.

Проверил:

ассистент каф. МО ЭВМ, ИИТММ

Сиднев А.А.

Нижний Новгород

2015 г.

Содержание

[Введение 3](#_Toc439150614)

[Постановка задачи 4](#_Toc439150615)

[Руководство пользователя 5](#_Toc439150616)

[Руководство Программиста 6](#_Toc439150617)

[Используемые инструменты 6](#_Toc439150618)

[Общая структура проекта 6](#_Toc439150619)

[Описание структуры программы 6](#_Toc439150620)

[Описание структуры данных 7](#_Toc439150621)

[Описание алгоритмов 8](#_Toc439150622)

[Описание тестов 10](#_Toc439150623)

[Тесты для класса stack: 10](#_Toc439150624)

[Тесты для функции перевода и вычисления выражения: 10](#_Toc439150625)

[Тесты для класса list: 10](#_Toc439150626)

[Заключение 12](#_Toc439150627)

[Литература 13](#_Toc439150628)

[Приложения 14](#_Toc439150629)

[Приложение 1. Пример работы sample.exe 14](#_Toc439150630)

[Приложение 2. Пример работы sample\_list.exe 14](#_Toc439150631)

[Приложение 3. Пример работы sample\_stack.exe 15](#_Toc439150632)

# Введение

Польская нотация, также известна как префиксная нотация, это форма записи логических, арифметических и алгебраических выражений. Характерная черта такой записи – оператор располагается слева от операндов. В моей лабораторной работе реализуется возможность вычислять арифметические выражения, используя польскую нотацию. Кроме того, эта лабораторная работа демонстрирует использование таких структур данных как стек и список.

# Постановка задачи

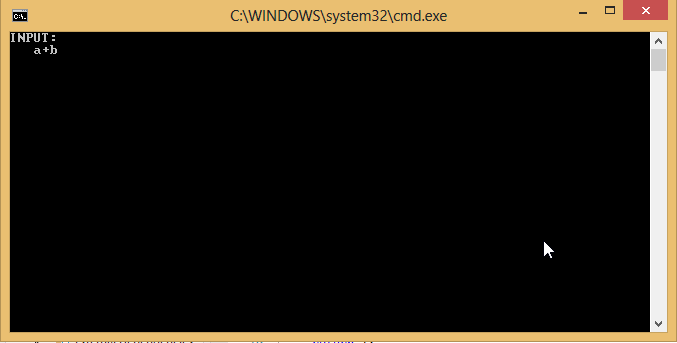
Необходимо создать динамическую структуру данных стек, используя динамическую структуру список. С помощью стека реализовать алгоритм перевода инфиксной записи арифметического выражения в постфиксную. Также программа должна выполнять вычисление арифметического выражения. Создать консольное приложение, 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

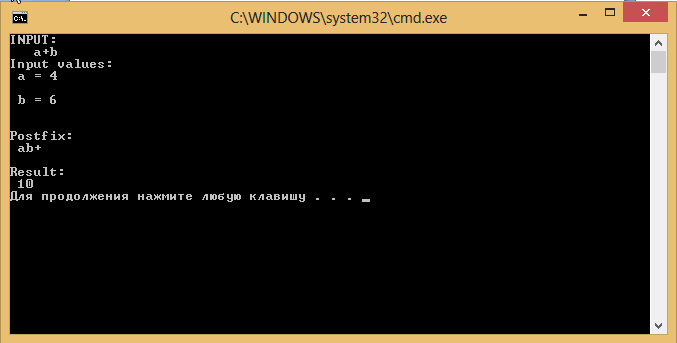
С помощью данной программы вы сможете вычислять арифметические выражения.

Чтобы приступить к работе, загрузите приложение sample.exe.

В окне приложения будет написано «Input». Это означает, что программа ожидает от вас ввода выражения. Возможны операции: +, -, \*, /, допустимо использование скобок и символьного обозначения переменных.



Если вы решились на использование символов, то программа попросит вас присвоить им значения.



Если вы допустили оплошность при вводе, то программа сообщит вам об этом и капитулирует.

Но если ввод был совершен безошибочно, то программа выполнит свою работу: на экране отобразится польская запись введенного вами примера, а также результат вычисления.

# Руководство Программиста

## Используемые инструменты

Для лабораторной работы использовались:

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.

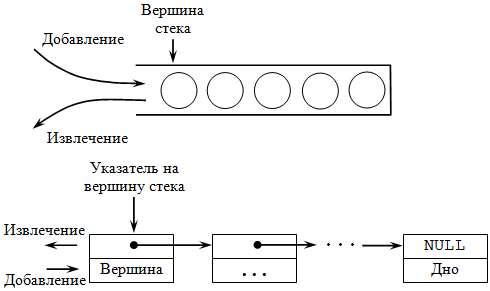
10. print - печать списка.

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.

## Описание тестов

### Тесты для класса stack:

1. TEST(Stack, can\_create\_stack) – проверка корректной работы конструктора класса stack.
2. TEST(Stack, can\_create\_copy) – проверка корректной работы конструктора копирования.
3. TEST(Stack, copy\_is\_equal) – проверка на то, равен ли скопированный объект исходному.
4. TEST(Stack, copy\_has\_its\_own\_memory) – проверка наличия собственной области памяти у копии.
5. TEST(Stack, IsEmpty) – проверка метода IsEmpty.
6. TEST(Stack, IsFull) – проверка метода IsFull.
7. TEST(Stack, can\_push) – проверка метода push.
8. TEST(Stack, throws\_when\_empty\_pop) – проверка исключительной ситуации метода pop.
9. TEST(Stack, can\_pop) – проверка метода pop.

### Тесты для функции перевода и вычисления выражения:

1. TEST(arithmetic, can\_convert\_to\_postfix) – проверка алгоритма перевода записи выражения в постфиксную форму.
2. TEST(arithmetic, can\_calculate\_postfix) – проверка алгоритма вычисления выражения в постфиксной форме.
3. TEST(arithmetic, throws\_when\_input\_invalid) – проверка работы алгоритма с некорректным выражением.
4. TEST(arithmetic, throws\_when\_empty\_writing) – проверка работы алгоритма с пустым выражением.

### Тесты для класса list:

1. TEST(List, can\_create) – проверка конструктора класса list.
2. TEST(List, can\_create\_copy) – проверка конструктора копирования.
3. TEST(List, copy\_is\_equal) – проверка на равенство копии исходному объекту.
4. TEST(List, copy\_has\_its\_own\_memory) – проверка наличия собственной области памяти у копии.
5. TEST(List, can\_print) – проверка метода print.
6. TEST(List, can\_search) – проверка метода poisk.
7. TEST(List, return\_null\_when\_cant\_search) – проверка метода poisk, когда искомого элемента нет.
8. TEST(List, throws\_when\_search\_in\_empty) – проверка метода poisk в пустом списке
9. TEST(List, can\_delete) – проверка метода del.
10. TEST(List, return\_null\_when\_delete\_not\_actual) – проверка метода del при попытке применить его к несуществующему элементу.
11. TEST(List, can\_delete\_first) – проверка возможности удаления первого элемента.
12. TEST(List, throws\_when\_try\_delete\_in\_empty\_list) – проверка работы метода del в пустом списке.
13. TEST(List, can\_inserrt\_first) – проверка метода insertF.
14. TEST(List, can\_insert\_last) – проверка метода insertL.
15. TEST(List, can\_insert\_last\_in\_empty) – проверка метода insertL в пустом списке.
16. TEST(List, can\_insert\_after) – проверка метода insertA.
17. TEST(List, can\_insert\_after\_last) – проверка метода insertA после последнего элемента.
18. TEST(List, throws\_when\_cant\_insert\_after) – проверка исключительной ситуации для метода insertA.
19. TEST(List, throws\_when\_insert\_after\_empty) – проверка работы метода insertA в пустом списке.
20. TEST(List, can\_get\_first) – проверка метода GetFirst.

# Заключение

Итогом лабораторной работы стала программа, работающая по описанному алгоритму. На ее примере были показаны преимущества использования таких структур данных как стек и список. Кроме самой программы присутствуют приложения, наглядно демонстрирующие работу классов стека и списка.

Для проверки программы написано 35 тестов. Ошибок тесты не выявили.

Программа, принимая на вход арифметическое выражение, сначала преобразует его в постфиксную форму, а затем вычисляет результат.

# Литература

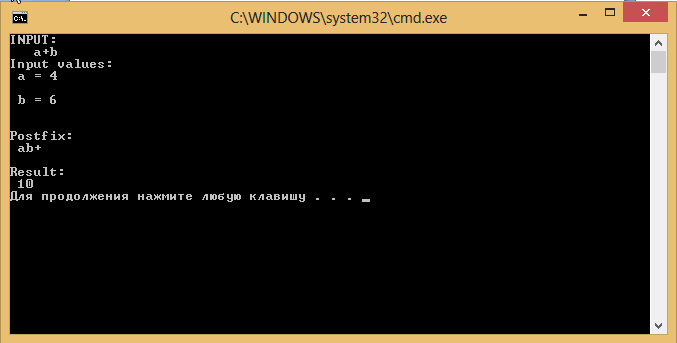
1. Обратная польская запись - http://algolist.manual.ru/syntax/revpn.php

2. Никлаус Вирт. Алгоритмы и структуры данных (+ CD-ROM) — ДМК Пресс, 2011. — 100 с.

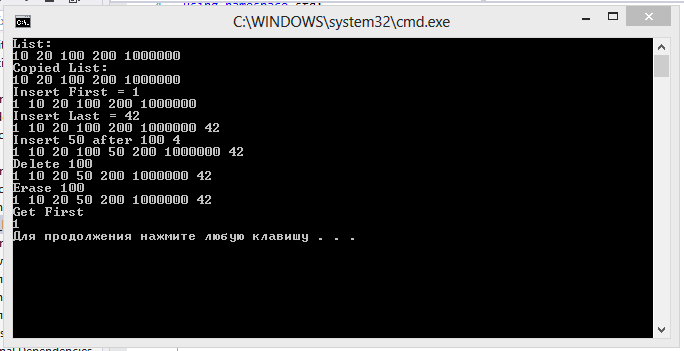
3. Майкл Мейн, Уолтер Савитч. Структуры данных и другие объекты в C++ = Data Structures and Other Objects Using C++. — 2-е изд. — М.: Вильямс, 2002. — 832 с.

# Приложения

## Приложение 1. Пример работы sample.exe



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

******

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

