

Министерство образования и науки Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования «Национальный исследовательский Нижегородский  
государственный университет им. Н.И. Лобачевского»

Институт Информационных технологий, математики и механики

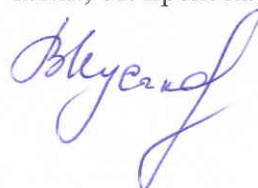
Отчёт по лабораторной работе

## Вычисление арифметических выражений

Выполнил:  
студент гр. 381606-3

Пигин А.П.

Проверил:  
к.т.н., ст. преп. каф. МОСТ ИИТММ



Кустикова В.Д.

Нижний Новгород  
2017 г.

# РЕАЛИЗАЦИЯ СТЕКА НА ЛИНЕЙНОМ ОДНОСВЯЗНОМ СПИСКЕ

## Содержание:

- Постановка задачи
- Руководство пользователя
- Руководство программиста  
Общая структура проекта  
Описание структуры программы  
Описание структур данных  
Структура данных "список"  
Структура данных "стек"  
Описание алгоритмов  
Алгоритм перевода в постфиксную запись  
Алгоритм подсчета выражения в постфиксной записи
- Заключение

## ПОСТАНОВКА ЗАДАЧИ

**Цель данной лабораторной работы** — разработать на языке программирования C++ статическую библиотеку, реализующую динамическую структуру данных — стек, основанный на динамической структуре — список.

В качестве примера реализации стеков, разработать алгоритм преобразования инфиксной записи арифметических выражений в постфиксную. 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 консольные приложения для демонстрации работы списков и стеков.

## РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

### ЗАПУСК ПРИЛОЖЕНИЯ И ВВОД ДАННЫХ

Данная программа предназначена для перевода символьного арифметического выражения из инфиксной записи в постфиксную и последующего вычисления результата на основе данных, введенных пользователем. Запустите программу и следуйте указаниям.

Пример:

1. Введите арифметическое выражение и нажмите клавишу "Enter" и увидите постфиксную форму выражения
2. Введите значение каждой из символьных переменных, нажимая клавишу "Enter" после каждого введенного значения.
3. Получите численный результат.

Для завершения работы нажмите любую клавишу.

## РУКОВОДСТВО ПРОГРАММИСТА

### ОБЩАЯ СТРУКТУРА ПРОЕКТА

Структура проекта:

- include — директория для размещения заголовочных файлов.
- samples — директория для размещения тестового приложения.
- sln — директория с файлами решений и проектов для Visual Studio 2010
- src — директория для размещения исходных кодов (cpp-файлы).

## ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

- `Obj` — описывает узел списка. Узел хранит в себе значение ключа и указатель на следующий узел, то есть на объект такого же класса.
- `List` — класс "список", агрегирующий в себе класс `Unit`.
- `Stack` — класс "стек", агрегирующий в себе класс `List`.
- `postfix_lib` — статическая библиотека, использующая функционал класса `Stack`, содержащая класс `Postfix` со статическими методами перевода арифметического выражения из инфиксной формы в постфиксную и вычисления полученного выражения.
- `Sample_list` — консольное приложение, содержащее функцию `main`, которая запрашивает у пользователя выражение в инфиксной записи и выводит выражение в постфиксной форме и результат, полученные от функций библиотеки `postfix_lib`.

## ОПИСАНИЕ СТРУКТУР 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 (`root`).

Принципиальным преимуществом перед линейным 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.

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

- Конструктор по умолчанию.
- Конструктор копирования списков.
- Деструктор.
- `erase` — перегруженный метод удаления элемента с заданным ключом или по указателю на элемент.
- `push_up` — метод создания элемента с заданным ключом и вставки его в начало списка.
- `push_back` — метод создания элемента с заданным ключом и вставки его в конец списка.
- `push_under` — метод вставки элемента, на который передан указатель, до элемента с заданным ключом.
- `pop_back` — метод удаления элемента из конца списка, возвращает ключ элемента.
- `pop_up` — метод удаления из начала списка, возвращает ключ.
- `get_size` — метод возвращает количество элементов списка
- Оператор равенства перегружен для правильного присваивания.

Класс `List` реализован с использованием шаблонов для покрытия его использования с различными типами данных.

### 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 в виде 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 агрегирует в себя объект класса *List* и содержит следующие методы:

- Конструктор по умолчанию, который явно вызывает конструктор класса *List*.
- Конструктор копирования.
- Деструктор.
- *empty* — метод проверки стека на пустоту
- *full* — метод проверки стека на полноту. По факту, проверяется наличие доступной памяти в виртуальном адресном пространстве программы для создания нового узла списка.
- *push* — метод добавления элемента с заданным значением на вершину стека.
- *pop* — метод изъятия элемента с вершины стека. Метод возвращает значение элемента.

Класс *Stack* реализован с использованием шаблонов для покрытия его использования с различными типами данных.

### ОПИСАНИЕ АЛГОРИТМОВ

#### АЛГОРИТМ ПЕРЕВОДА В ПОСТФИКСНУЮ ЗАПИСЬ

Описание алгоритма перевода из инфиксной записи в постфиксную:

4. Каждой операции ставится приоритет. Операциям умножения  $*$  и деления  $/$  — наивысший приоритет, равный 3. Операциям сложения  $+$  и вычитания  $-$  — приоритет 2. Операции открывающей скобки  $($  — приоритет 1. Операции равенства  $=$  — приоритет 0.
5. Создается 2 стека: стек для хранения текущей постфиксной формы *Stack\_one* и стек для хранения операций *Stack\_two*.
6. Выражение просматривается слева-направо, при этом возможно 4 случая: Встретился операнд. Тогда он добавляется в стек *Stack\_one*. Встретилась операция, приоритет которой выше, чем приоритет операции, лежащей на вершине стека *Stack\_two*, или стек *Stack\_two* пуст. В этом случае операция добавляется в стек для хранения операций *Stack\_two*. Встретилась операция, приоритет которой равен или ниже приоритета операции, лежащей на вершине стека *Stack\_two*. В этом случае все операции, приоритет которых больше данной, перекадываются в стек *Stack\_one* до тех пор, пока на вершине стека *Stack\_two* не появится операция с меньшим приоритетом или *Stack\_two* не станет пустым. Новая же операция добавляется в стек хранения операций. Встретилась операция закрывающей скобки. В этом случае из стека *Stack\_two* перекадываются все операции в *Stack\_one* до первого вхождения операции открывающей скобки. Операция открывающей скобки удаляется из стека операций.
7. Если выражение закончилось, то все операции из стека операций *Stack\_two* перекадываются в стек хранения текущей постфиксной формы *Stack\_one*.

#### АЛГОРИТМ ПОДСЧЕТА ВЫРАЖЕНИЯ В ПОСТФИКСНОЙ ЗАПИСИ

Описание алгоритма вычисления арифметического выражения в постфиксной форме:

8. Создается первый стек с вещественным типом данных *double\_list*, для сохранения введенных значений пользователя.

9. Выражение просматривается справа на лево.
10. Если встречается операнд, то запрашивается число, которое будет соответствовать этому операнду и добавляется в `double_list`
11. Создается второй стек `end_list`
12. Встретился операнд. В этом случае на вершину стека `end_list` добавляется элемент из вершины `double_list`. Встретилась операция. Тогда из стека `end_list` изымаются 2 операнда, над ними совершается операция, результат операции снова добавляется в стек.
13. При достижении конца арифметического выражения, в стеке будет находиться единственный элемент — численный результат выражения.

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