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 института ИТММ гр. 0823-2

Петрова М.С.

Проверил:

ассистент каф. МО ЭВМ, ИИТММ

Сиднев А.А.

Нижний Новгород

2015 г.

Содержание

[Постановка задачи. 3](#_Toc439151567)

[Структура проекта. 4](#_Toc439151568)

[Классы и их описание. 4](#_Toc439151569)

[Описание алгоритмов. 5](#_Toc439151570)

[Описание тестов. 6](#_Toc439151571)

[Руководство пользователя. 9](#_Toc439151572)

[Список литературы. 11](#_Toc439151573)

# Постановка задачи

Реализация проекта включает в себя построение польской обратной записи. Польская обратная запись – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Преимущества записи состоят в том, что арифметическое выражение не содержит скобок и вычисляется последовательно, слева направо. Например, для выражения

3/(2 \* x \* y – 7 \* z)

польская обратная запись будет выглядеть следующим образом:

32x\*y\*7z\*-/.

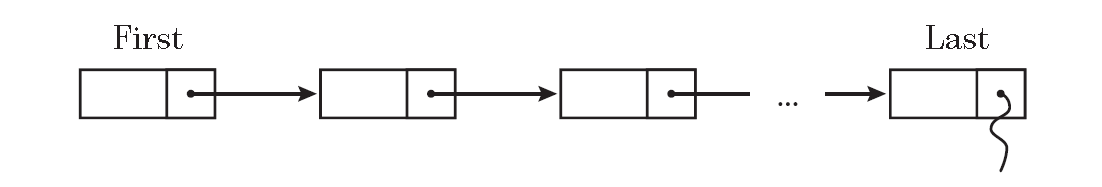
В проекте необходимо учесть использование стека – одной из важнейших динамических структур данных.

# Структура проекта

Для реализации проекта была разработана система классов: класс *Node*, описывающий «узел» списка, класс *List* и класс *Stack*.

# Классы и их описание

**Класс *Node*** содержит значение «val» и указатель на следующий элемент Next.

**Класс *List***. Объект данного класса представляет собой звено линейного списка, поэтому сам класс содержит в себе одно поле: указатель на первый элемент списка. Линейный список – это динамическая структура данных, каждый элемент которой посредством указателя связывается со следующим элементом. Схематически изобразить линейный список можно следующим образом:  
В данном классе имеются следующие методы:

* Конструктор по умолчанию.
* Конструктор копирования.
* Деструктор.
* Поиск элемента с заданным значением «val» *(Search)*.
* Удаление элемента с заданным значением «val» *(del)*.
* Вставка элемента с заданным значением «val» в начало списка *(insertF).*
* Вставка элемента с заданным значением «val» в конец списка *(insertL)*.
* Вставка элемента с заданным значением «val» после заданного элемента *(insertA).*
* Доступ к первому элементу *(GetFirst)*.

**Класс *Stack***. Стек – это динамическая структура данных, представляющая собой упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека.  
Хранение стека осуществляется с помощью линейного списка, то есть каждый элемент содержит помимо хранимой информации в стеке указатель на следующий элемент стека.  
В классе имеются конструктор по умолчанию, конструктор копирования, деструктор. Методы, содержащиеся в классе:  
Проверка стека на полноту *(IsFull)*;Проверка стека на пустоту *(IsEmpty)*;  
Положить в стек *(Push)*;  
Убрать из стека *(Pop)*;  
Доступ к значению элемента *(GetValue)*;

# 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.

Правило вычисления в польской обратной записи состоит в следующем. Польская обратная запись рассматривается слева направо. Если рассматриваемый элемент является операндом, то он кладется в стек, после чего рассматривается следующий элемент. Если рассматриваемый элемент – знак операции, то выполняется эта операция над последними двумя операндами в стеке. Результат операции записывается в стек вместо операндов, участвующих в операции. Просмотр продолжается до тех пор, пока в польской обратной записи не кончатся элементы. В результате последовательного выполнения этого правила будут выполнены все операции, имеющиеся в выражении, и в стеке останется один элемент – результат вычисления арифметического выражения.

# Описание тестов

**test\_list.cpp**

* Тест «can\_create\_list» проверяет программу на создание списка.
* Тест «can\_create\_copied\_list» создание копии списка при использовании конструктора копирования.
* Тест «copied\_list\_is\_equal\_to\_source\_one» проверка на равенство двух списков: исходного и скопированного.
* Тест «copied\_list\_has\_its\_own\_memory» проверка, что у скопированного списка правильно выделяется память.
* Тест «can\_print\_empty\_list» отсутствие исключений при печати пустого списка.
* Тест «can\_search\_element\_with\_actual\_value» проверка на поиск элемента в исходном списке с заданным значением.
* Тест «return\_null\_when\_cant\_search\_element» выдаётся ошибка при поиске отсутствующего элемента.
* Тест «throws\_when\_search\_in\_empty\_list» исключение при попытке поиска элемента в пустом списке.
* Тест «can\_delete\_element\_with\_actual\_value» проверка на удаление элемента с заданным значением.
* Тест «return\_null\_when\_delete\_not\_actual\_value» выдаётся ошибка при попытке удаления отсутствующего элемента.
* Тест «can\_delete\_first\_element» проверка на удаление первого элемента в списке.
* Тест «can\_not\_find\_delete\_element» ошибка при попытке поиска ранее удалённого элемента.
* Тест «throws\_when\_try\_delete\_in\_empty\_list» исключение при попытке удаления элемента из пустого списка.
* Тест «can\_insert\_first\_element» проверка функции вставки элемента в начало списка.
* Тест «can\_insert\_last\_element» проверка функции вставки элемента в конец списка.
* Тест «can\_insert\_last\_element\_when\_list\_is\_empty» проверка на отсутствие исключений при попытке вставки элемента в пустой список.
* Тест «can\_insert\_after» проверка функции вставки после элемента с заданным ключом.
* Тест «can\_insert\_after\_when\_it\_last» проверка на отсутствие исключений функции вставки после элемента с заданным ключом в конец списка.
* Тест «throws\_when\_cant\_find\_place\_to\_insert\_after» ошибка при попытке вставки элемента после значения, не имеющегося в списке.
* Тест «throws\_when\_insert\_after\_in\_empty\_list» ошибка при попытке вставки элемента после значения в пустой список.
* Тест «can\_get\_first» проверка функции получения значения первого элемента списка.

**test\_stack.cpp**

* Тест «can\_create\_stack» проверяет программу на создание стека
* Тест «can\_create\_copied\_stack» создание копии стека при использовании конструктора копирования.
* Тест «copied\_stack\_is\_equal\_to\_source\_one» проверка на равенство двух стеков: исходного и скопированного.
* Тест «copied\_stack\_has\_its\_own\_memory» проверка, что у скопированного стека правильно выделяется память.
* Тест «IsEmpty\_returns\_1\_when\_stack\_is\_empty» проверка стека на пустоту. Стек пуст.
* Тест «IsEmpty\_returns\_0\_when\_stack\_isnt\_empty» проверка стека на пустоту. Стек не пуст.
* Тест «IsFull\_returns\_0\_when\_stack\_isnt\_full» проверка стека на полноту. Стек не полон.
* Тест «can\_push\_new\_element» возможность добавления нового элемента.
* Тест «throws\_when\_try\_pop\_element\_when\_stack\_is\_empty» исключение при попытке удаления элемента из пустого стека.
* Тест «can\_pop\_element\_from\_no\_empty\_stack» успешное удаление элемента из непустого стека.
* Тест «poped\_element\_is\_equal\_to\_pushed\_one» проверка на равенство помещенного в стек элемента, а затем удаленного.

**test\_arithmetic.cpp**

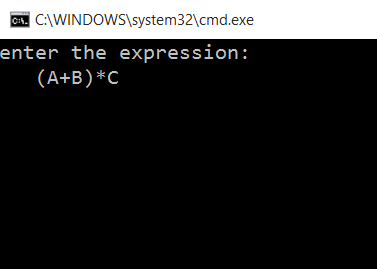
* Тест «can\_convert\_in\_a\_postfix\_notation» проверка функции перевода из обычной записи арифметического выражения в польскую.
* Тест « can\_calculate\_expresion\_in\_a\_postfix\_notation» проверка функции вычисления арифметического выражения в польской записи.
* Тест « throws\_when\_input\_invalid» исключение при некорректном введении данных.
* Тест «throws\_when\_try\_convert\_empty\_string» исключение при вводе пустого арифметического выражения.

# Руководство пользователя

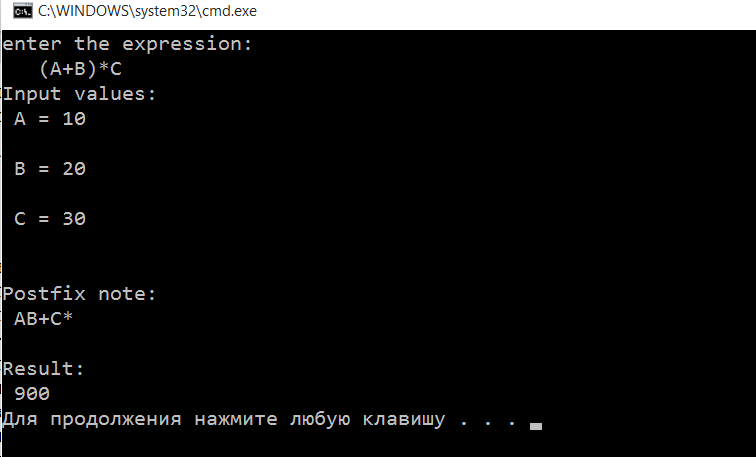
Чтобы запустить программу необходимо открыть исполняемый файл sample.exe.

Далее следуйте инструкциям.

Введите арифметическое выражение, которое вы хотите вычислить.



Затем введите значения операндов.



Далее появится польская запись выражения и его значение.

**Заключение**

В ходе выполнения данного проекта было достигнуто несколько результатов. Выделим и перечислим главные из них.

Во-первых, было получено понимание того, как вычислять арифметические выражения с помощью польской обратной записи. Для этого используется бесскобочная запись арифметического выражения.

Во-вторых, мы узнали новые динамические структуры данных: стек и односвязный линейный список. Были получены знания о том, как устроены эти структуры, их методы реализации. В частности, линейные списки используются повсеместно, так как позволяют эффективно расходовать память компьютера.

# Список литературы

1. Б.Страуструп. Язык программирования C++ -- The C++ Programming Language / Пер. с англ – 3-е изд. – СПб.; М.: Невский диалект – Бином, 1999.
2. Д.Донован. Системное программирование – М.: Мир, 1975.
3. Обратная польская нотация <http://algolist.manual.ru/syntax/revpn.php>.
4. Обратная польская запись <https://ru.wikipedia.org/wiki/Обратная_польская_запись> .