МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
Высшего образования
**«Нижегородский государственный университет им. Н.И. Лобачевского»**
**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**
**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ**

# «Преобразование арифметических выражений в обратную польскую запись »

**Выполнил:** студент группы 381706-1
Митягина Дарья Сергеевна
_____ Подпись

**Научный руководитель:**
ассистент каф. МОСТ ИИТММ
_____ Лебедев И.Г

Нижний Новгород

2018.

# Оглавление

## 1. Введение

Целью данной работы являлось создание программы, способной преобразовывать арифметические выражения в обратную польскую запись и производить вычисления.

*Обратная польская запись* (англ. Reverse Polish notation, RPN) — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций.

В общем виде запись выглядит следующим образом:

- Запись набора операций состоит из последовательности операндов и знаков операций. Операнды в выражении при письменной записи разделяются пробелами.
- Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.
- Результатом вычисления выражения становится результат последней вычисленной операции.

Например, рассмотрим вычисление выражения 7 2 3 * − (эквивалентное выражение в инфиксной нотации: 7 − 2 * 3).

- Первый по порядку знак операции — «*», поэтому первой выполняется операция умножения над операндами 2 и 3 (они стоят последними перед знаком). Выражение при этом преобразуется к виду 7 6 − (результат умножения — 6, — заменяет тройку «2 3 *»).
- Второй знак операции — «−». Выполняется операция вычитания над операндами 7 и 6.
- Вычисление закончено. Результат последней операции равен 1, это и есть результат вычисления выражения.

## 2. Постановка задачи

Для достижения цели данной лабораторной работы необходимо:

• Описать и реализовать класс строки TString.

• Реализовать ряд функций, служащих для преобразования арифметических выражений и получения конечного результата. Эти функции: GetPriority(const char operation), IsOperation(char symbol), TQueue<char> ToPolish(TString str), GettingRez(TQueue<char> queue);

• Написать тесты для проверки реализованных методов.

• Реализовать класс TException для обработки исключений.

## 3. Руководство пользователя

При запуске программы пользователю представляется простой пример. Сначала показана работа программы с уже составленным выражением. Затем пользователю предлагается написать свой пример.

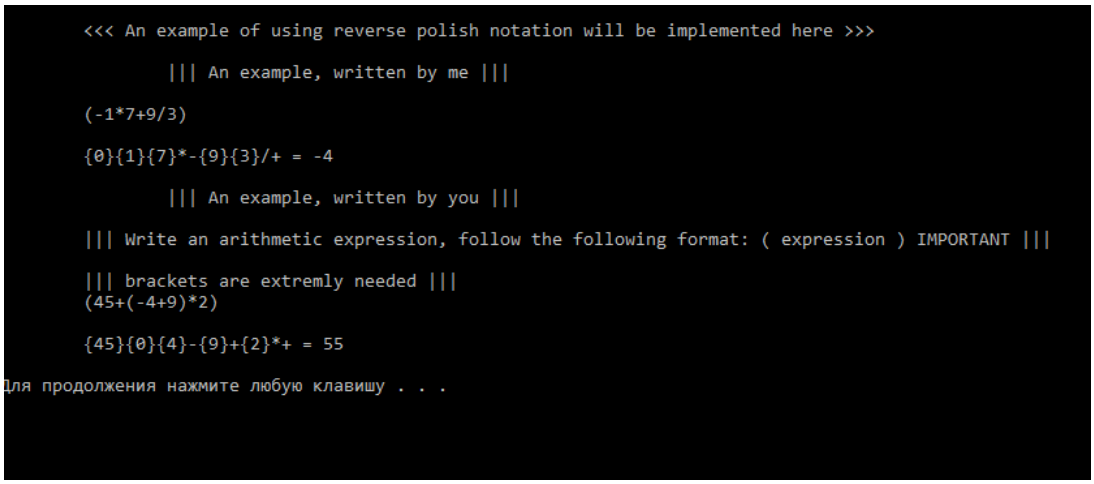


Рис. 1 – пример работы программы.

# 4. Руководство программиста

## 4.1. Описание структуры программы.

Программа состоит из модулей:

1. PolandLib – содержит описание и реализацию класса TString, а также реализацию ряда необходимых для поставленной задачи функций;

2. QueueLib - содержит описание и реализацию класса TQueue;

3. StackLib - содержит описание и реализацию класса TStack;

4. Poland - содержит пример использования программы;

5. PolandTest - содержит набор тестов для класса TString, а также для остальных функций;

6. Exception - содержит реализацию класса исключений TException.

## 4.2. Описание структур данных.

**Класс TString**

Рассмотрим *protected-часть*:

1. int size – длина строки;

2. char *mas – массив элементов строки.

Рассмотрим *public-часть*:

1. TString() – конструктор по умолчанию;

2. TString(TString &A) – конструктор копирования;

3. TString(char *str) - конструктор;

4. ~TString() - деструктор;

5. TString operator + (TString &A) – перегрузка оператора сложения;

6. TString & operator = (TString &A) - перегрузка оператора присваивания;

7. char & operator [] (int n) - перегрузка оператора индксации;

8. void Print() – вывод строки на экран;

9. int GetSize() – получение дляны строки;

10. friend ostream & operator << (ostream &out, TString &A);

11. friend istream& operator >> (istream &in, TString &A);

Дополнительные функции, не входящие в класс TString:

1. int GetPriority(const char operation) – получение приоритета операции;

2. bool IsOperation(char symbol) – проверка символа;

3. TQueue<char> ToPolish(TString str) – преобразование в обратную польскую запись;

4. double GettingRez(TQueue<char> queue) – получние конечного результата;

### 4.3. Описание алгоритмов

В данной части не будут рассматриваться тривиальные методы, внимание уделим лишь некоторым.

*1. Преобразование выражения*

( в стеке St хранятся операции и открывающиеся скобки, в очереди Qu – операнды по мере их появления при прохождении строки)

Алгоритм:

- Когда появляется закрывающаяся скобка, **ПОВТОРЯТЬ:** в Qu переносятся элементы, находившиеся до этого в St. **ПОКА:** St не пуст и не появилась открывающаяся скобка.

- Приоритет операций op распределен следующим образом:
  для **‘(‘** и **‘)’** приоритет = 1
  для **‘+’** и **‘-‘** приоритет = 2
  для **‘*’** и **‘/‘** приоритет = 3
  для остальных символов: 0.
- **ЕСЛИ** приоритет текущей операции op больше приоритета элемента, находящегося на вершине St, **ТО** кладем op в St.
  **ИНАЧЕ**: **ПОВТОРЯТЬ:** в Qu переносятся элементы, находившиеся до этого в St. **ПОКА:** St не пуст и приоритет операции, находящейся на вершине St, меньше приоритета текущей операции op.
- После завершения этих действий текущая операция отправляется в St.

Отдельно рассматриваются следующие случаи:

**- a …** , т.е. когда выражение начинается со знака минус.

**…(-b…)…** , т.е. когда выражение в скобках, являющееся частью большего, начинается со знака минус.

*2. Получение результата*

Алгоритм:

- **ЕСЛИ** пришедшая очередь начинается с символа, не являющегося операцией, **ТО ПРОДОЛЖАТЬ:**
  Взять первый элемент очереди, проверить, не является ли он операцией. Затем считываем символы. Если необходимо (т.е. рядом расположены несколько цифр), вычисляем значение числа. Записываем его в стек.
  **ИНАЧЕ** (т.е. символ является операцией) производим вычисления в соответствии с полученным символом (+, -, *, /).

В стек кладем результат вычисления.

Переменной r, предназначенной для хранения ответа, присваиваем значение вершины стека.

- **ЕСЛИ** стек пуст, **ТО** действие функции останавливается, возвращается r.

## 5. Заключение

В процессе работы над данной лабораторной работой мне удалось реализовать программу, способную преобразовывать арифметические выражения в обратную польскую запись и производить вычисления.

Кроме того было достигнуто более глубокое понимание принципов работы с данной структурой, принципов тестирования программы с помощью GT.

## 6. Список литературы

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», 2015.

2. Динамические структуры данных: очередь и стек, официальный сайт Интуит[https://www.intuit.ru/studies/courses/648/504/lecture/11457?page=2] 3

3. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.: Пер. с англ. М.: Издат. дом «Вильямс», 2000. С. 58–76 5.

4. Статья о стеке в Викиконспектах

[https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D1%82%D0%B5%D0%BA]

5. Статья, посвященная теме обратной польской записи [https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C]