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 ЛАБОРАТОРНОЙ РАБОТЕ №11**

**«Арифметика с использование обратной польской записи»**

**Выполнил:** студент группы 381903-3

Зинкин Кирилл Сергеевич

**Проверил:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

Нижний Новгород

2020.

Содержание

[1.Введение 3](#_Toc2955132)

[2. Цели и задачи 6](#_Toc2955133)

[2.1. Используемые инструменты 6](#_Toc2955134)

[3. Руководство пользователя 7](#_Toc2955135)

[4. Руководство программиста 8](#_Toc2955136)

[4.1. Описание структуры программы 8](#_Toc2955137)

[4.2. Описание функций и процедур, их алгоритмов 8](#_Toc2955138)

[5. Заключение 12](#_Toc2955140)

[6. Литература 13](#_Toc2955141)

# 1.Введение

Как правило арифметические выражения удобно преобразовывать в обратную польскую запись (ОПЗ), чтобы избавиться от скобок, содержащихся в выражении. Выражения, преобразованные в ОПЗ, можно вычислять последовательно, слева направо.

Преобразование выражения в ОПЗ с использованием стека:

Нам понадобится стек для переменных типа char, т.к. исходное выражение мы получаем в виде строки.  
Рассматриваем поочередно каждый символ:  
1. Если этот символ - число (или переменная), то просто помещаем его в выходную строку.  
2. Если символ - знак операции (+, -, \*, / ), то проверяем приоритет данной операции. Операции умножения и деления имеют наивысший приоритет (допустим он равен 3). Операции сложения и вычитания имеют меньший приоритет (равен 2). Наименьший приоритет (равен 1) имеет открывающая скобка.  
Получив один из этих символов, мы должны проверить стек:  
а) Если стек все еще пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка) имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.  
б) Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа, то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к пункту а).  
3. Если текущий символ - открывающая скобка, то помещаем ее в стек.  
4. Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор, пока не встретим в стеке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить. Закрывающая скобка также уничтожается.

Если вся входная строка разобрана, а в стеке еще остаются знаки операций, извлекаем их из стека в выходную строку.

Рассмотрим алгоритм на примере простейшего выражения:  
Дано выражение:  
a + ( b - c ) \* d

Рассмотрим поочередно все символы:

|  |  |  |  |
| --- | --- | --- | --- |
| Символ | Действие | Состояние выходной строки после совершенного действия | Состояние стека после совершенного действия |
| a | 'a' - переменная. Помещаем ее в выходную строку | a | пуст |
| + | '+' - знак операции. Помещаем его в стек (поскольку стек пуст, приоритеты можно не проверять) | a | + |
| ( | '(' - открывающая скобка. Помещаем в стек. | a | + ( |
| b | 'b' - переменная. Помещаем ее в выходную строку | a b | + ( |
| - | '-' - знак операции, который имеет приоритет 2. Проверяем стек: на вершине находится символ '(', приоритет которого равен 1. Следовательно мы должны просто поместить текущий символ '-' в стек. | a b | + ( - |
| c | 'c' - переменная. Помещаем ее в выходную строку | a b c | + ( - |
| ) | ')' - закрывающая скобка. Извлекаем из стека в выходную строку все символы, пока не встретим открывающую скобку. Затем уничтожаем обе скобки. | a b c - | + |
| \* | '\*' - знак операции, который имеет приоритет 3. Проверяем стек: на вершине находится символ '+', приоритет которого равен 2, т.е. меньший, чем приоритет текущего символа '\*'. Следовательно мы должны просто поместить текущий символ '\*' в стек. | a b c - | + \* |
| d | 'd' - переменная. Помещаем ее в выходную строку | a b c - d | + \* |

Теперь вся входная строка разобрана, но в стеке еще остаются знаки операций, которые мы должны просто извлечь в выходную строку. Поскольку стек - это структура, организованная по принципу LIFO, сначала извлекается символ '\*', затем символ '+'.  
Итак, мы получили конечный результат: a b c - d \* +.

# 2. Цели и задачи

В рамках лабораторной работы ставится задача эффективной реализации вычислений с помощью преобразования арифметических выражений в обратную польскую запись.

В процессе выполнения лабораторной работы требуется использовать систему контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Выполнение работы предполагает решение следующих задач:

1. Реализация вычислений с помощью преобразования арифметических выражений в обратную польскую запись, с использованием классов TStack и TQueue.
2. Реализация класса для обработки исключений– TException, которые могут возникнуть при выполнении различных операций, согласно заданному интерфейсу.
3. Обеспечение работоспособности тестов и примера использования.
4. Модификация примера использования в тестовое приложение, позволяющее задавать арифметические выражения с использованием основных операций и получать верный ответ.

## 2.1. Используемые инструменты

* Система контроля версий [Git](https://git-scm.com/book/ru/v2). Рекомендуется использовать один из следующих клиентов на выбор студента:
  + [Git](https://git-scm.com/downloads)
  + [GitHub Desktop](https://desktop.github.com/)
* Фреймворк для написания автоматических тестов [Google Test](https://github.com/google/googletest).
* Среда разработки Microsoft Visual Studio (2008 или старше).
* Опционально. Утилита [CMake](http://www.cmake.org/) для генерации проектов по сборке исходных кодов. Может быть использована для генерации решения для среды разработки, отличной от Microsoft Visual Studio 2008 или 2010.

# 3. Руководство пользователя

1.Запускаем программу файла sample\_polish.cpp.

2.Пользователю предлагается ввести арифметическое выражение с использованием операций “+”, “-”, “\*”, “/” и скобочек “(”, “)”.

3.Программа выведет результат.

# 4. Руководство программиста

Разработка системы вычисления проводились в среде “Microsoft Visual Studio 2010”.

В данной работе будет использовано 4 класса:

* Класс «Стек» (TStack), реализованный с использованием массива.
* Класс «Очередь» (TQueue), построенного на основе класса TStack.
* Класс исключения (TExсeption).
* Базовый класс Строки.

А также:

* пространство имен Polish.

## 4.1. Описание структуры программы

Модульная структура программы:

* tstack.h– модуль с классом TStack, в котором определен интерфейс шаблонного класса Стек и реализация его методов.
* exсeption.h – модуль с классом исключения TExсeption.
* tqueue.h– модуль с классом TQueue, в котором определен интерфейс шаблонного класса Очередь и реализация его методов.
* polish.h – модуль с пространством имен Polish.
* polish\_sample.cpp , sample\_performance\_check.cpp– модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.
* test\_main.cpp, test\_polish.cpp – модуль с функциями тестирования для созданной программы. Содержит 9 тестов для функций пространства имен Polish.

## 4.2. Описание функций и процедур, их алгоритмов

Рассмотрим реализацию методов шаблонного класса TStack:

template <typename StackType>

class TStack

1. TStack(int n = 0) - конструктор класса, принимающий размер стека. По умолчанию создается стек размера 0 с позицией вершины стека 0.

2. TStack(TStack<StackType> &S) - конструктор копирования.

Принимает ссылку на объект класса TStack.

3. ~TStack() - деструктор. Освобождает выделенную под вектор память.

4. TStack& operator=(const TStack<StackType>& stack) – перегрузка оператора присваивания одного стека другому.

Присваивает полям первого объекта класса поля второго объекта класса.

5. int GetSize() - возвращает размер стека.

6. StackType Get() – метод изъятия элемента из вершины стека с удалением.

7. StackType ShowLast() – метод изъятия элемента из вершины стека без удаления.

8. void Put(StackType A) - метод, позволяющий добавить новый элемент в стек.

9. bool IsFull() – метод проверки стека на полноту.

10. bool IsEmpty() – метод проверки стека на пустоту.

11. void PrintStack() - метод отображения текущих элементов стека.

12. bool operator==(const TStack<StackType>& stack) const – перегрузка оператора сравнения.

Выполняется проверка стеков на равенство. Возвращает true, если равенство выполняется, false в противном случае.

13. bool operator!=(const TStack<StackType>& stack) const - перегрузка оператора сравнения.

Выполняется проверка стеков на неравенство. Возвращает true, если неравенство выполняется, false в противном случае.

Рассмотрим реализацию методов шаблонного класса TQueue:

template <typename QueueType>

class TQueue : public TStack <QueueType>

1. TQueue(int n = 0) - конструктор класса с одним параметром.

2. TQueue(TQueue <QueueType> &Q) - конструктор копирования.

Принимает ссылку на объект класса TQueue.

3. TQueue& operator=(const TQueue<QueueType>& queue) - перегрузка оператора присваивания одной очереди другому.

Присваивает полям первого объекта класса поля второго объекта класса.

4. QueueType Get() - метод изъятия элемента из начала очереди с удалением.

5. void Put(QueueType Q) - метод, позволяющий добавить новый элемент в конец очереди.

6. bool IsFull() - метод проверки очереди на полноту.

7. bool IsEmpty() - метод проверки очереди на пустоту.

8. void PrintQueue() - метод отображения текущих элементов очереди.

9. bool operator!=(const TQueue<QueueType>& queue) const - перегрузка оператора сравнения.

Выполняется проверка очередей на равенство. Возвращает true, если равенство выполняется, false в противном случае.

10. bool operator==(const TQueue<QueueType>& queue) const - перегрузка оператора сравнения.

Выполняется проверка очередей на неравенство. Возвращает true, если неравенство выполняется, false в противном случае.

Рассмотрим реализацию функций пространства имен Polish:

namespace Polish

1. inline int GetPriority( const char operation ) – функция, определяющая приоритет записи различных символов.

2. inline int Convert( const char symbol ) – функция, которая переводит символ в число.

3. TQueue<char> ToPolish( string str ) – функция, принимающая строку. Преобразует арифметическое выражение в вид обратной польской записи.

4. double Calculate( TQueue<char> polish ) – функция, принимающая очередь. Производит вычисления и возвращает конечный результат арифметического выражения.

# 5. Заключение

В результате лабораторной работы была разработана реализация преобразования арифметических выражений в обратную польскую запись, а также освоены такие инструменты разработки программного обеспечения, как система контроля версий [Git](https://git-scm.com/book/ru/v2) и фрэймворк для разработки автоматических тестов.

Созданная программа была протестирована с использованием google tests, а также были проведены эксперименты для сравнения теоретической и практической сложности выполнения операций на функции пространства имен.

# 6. Литература

1. Лабораторный практикум: Учебно-методическое пособие / Мееров И.Б. [и др.] – Нижний Новгород: Нижегородский госуниверситет , 2017. – 105с.
2. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
3. 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