**Infix Expression Parser/Evaluator**

*Data Structures, Project 2A*

*7/17/2016*

# team members: Gulnoza KhakimovA, Rebecca Peabody



**Summary:** This project parses infix expressions with integer operands, opening and closing parentheses, unary operators (!, ++, ==, - -), and binary operators (^, \*, /, %, +, -, >, >=, <, <=, ==, !=, &&, ||). As each operator or operand is parsed, it is passed to an evaluator class which uses two stacks to perform evaluation simultaneous with the parsing function. Parsing and evaluating at the same time increases the efficiency of this solution. Errors in the expression are detected by the parsing class as parsing takes place, also utilizing resources and maximizing efficiency.

Coding coordinated through GitHub, repository URL: <https://github.com/Gnkhakimova/Project_2A.git>

Assumptions

All operands are assumed to be positive integers. Expression such as ‘-5’ are assumed to be the unary negative operator followed by the integer five. A negative value is obtained by applying the action of the negative operator, i.e. multiplying five by negative one.

Boolean value handling . . .

No reuse of stacks, so clearing them isn’t necessary anywhere in the program.

Design/Implementation

The project is designed around three main classes: Tokenizer, Token, and Evaluator.

The Tokenizer class . . .

The Token class . . .

The Evaluator class . . .

Other classes used include:

Discussion of role of main & no user interface . . .

**UML DIAGRAM HERE:**

Performance

General discussion of efficiency and use of resources . . .

**BIG O ANALYSIS HERE:**

Error Checking

Discuss syntax error class and where/how most error checking was done.

References

Stack class from standard library

cplusplus.com – a C++ reference website

StackOverflow.com

Professor Hare’s Tegrity Lectures spring 2016 for help with

Postfix Evaluator code from BB