**CPT-281 Team Project 2: Infix Expression Parser**

Contributors: Athul Jaishankar, Timothy Huffman, Kathleen Dunn

Project Summary:

This project is an Infix expression parser System that helps parse an infix expression that supports arithmetic and logical operators with specified precedencies. The system utilizes stacks for efficient management of expression data.

Technical Requirements:

▪ The Infix expression parser system will support:

Operator Precedence Example

1) Power ( ‘^’ ) 7 2 ^ 8

2) Arithmetic ( ‘\*’, ‘/’, ‘%’ ) 6 6 \* 2

3) Arithmetic ( ‘+’, ‘-’ ) 5 6 - 2

4) Comparison ( ‘>’, ‘>=’, ‘<’, ‘<=’ ) 4 6 > 5

5) Equality Comparison ( ‘==’, ‘!=’ ) 3 6 != 5

6) Logical And ( ‘&&’ ) 2 6 > 5 && 4 > 5

7) Logical Or ( ‘| |’ ) 1 1 | | 0

▪ The infix expression parser is flexible with the given expressions. The user don’t need to worry about writing the spaces between operands and operators

▪ The file that keeps track of the infix expression is a plain text file. An original file input format is made based on this example:

((2 + 3) \* 4) - (5 \* (6 - 7))

(1 | | (0 && 1)) && (1^ ( 1 && 0 ))

(( 2 \*3) ^ 2 ) + ( 4\* 5) % 3

In the example above, each line stores a valid infix expression with appropriate suitable operators and operands.

**System Design:**

**Data Structures:**

**UML:**

A diagram of a function

Description automatically generated

**Test Cases:**

**Team Member Contributions:**

* **Athul Jaishankar**
* **Expression\_Parser.h:** Implemented the Expression\_Parser class, responsible for parsing infix expressions and evaluating the result. Defined method for parsing and evaluating infix expressions, handling operator precedence, converting infix expressions to postfix, and evaluating postfix expressions. Also, created the handle\_error method for handling exceptions.
* **Expression\_Parser.cpp:** Implemented all methods of the Expression\_Parser class. Developed algorithms for parsing infix expressions while maintaining their format for efficiency, converting infix to postfix, and evaluating. Ensured error handling by throwing exceptions and handling them.
* **Operator\_Precedence.h**: Defined an enum class called Operator\_Precedence, specifying the precedence levels for different operators.
* **Main.cpp:** Implemented the logic to read infix expressions from an input file, parser and evaluate each expression using the Expression\_Parser class, and display the result to the console. Integrated file I/O operations for input file handling and collaborated with team members to create and execute test cases. Addressed questions regarding program design and functionality.
* **Bug Fixes:** Addressed issues related to incorrect output for expressions like ‘2 ^ 3 ^ 2’ by fixing the power function to calculate the exponents correctly. Ensured that the program generates the expected output for all test cases.
* **Project Management:** Took the initiative to lead the project by designing the overall structure and goals of the infix expression parser system. Scheduled and organized team meetings to facilitate communication and collaboration among team members, ensuring smooth progress throughout the project.
* **Task Division:** Effectively divided tasks among team members, assigning responsibilities for coding, testing and documentation.
* **Testing:** Collaborated with team members to create test cases covering various expressions and scenarios. Verified the correctness of the program by comparing the actual output with the expected output.
* **Quality Assurance:** Ensured code quality by writing clean, well-commented code with meaningful variable names and function names. Maintained an organized repository structure and adhered to coding standards to facilitate code review and future maintenance. Effectively divided tasks among team members, assigning responsibilities for coding, testing and documentation.

**Future Improvements:**