CPT-281 Team Project 2: Infix Expression Parser

Contributors: Joe Simon, Eric Vaughn, Jordan Pham,

Project Summary:

This project is an infix expression parser that takes and evaluates the resulting values of any infix expression.

Technical Requirements:

▪ The parser should parse an infix expression that supports the following arithmetic and logical operators with their appropriate precedencies.

1) Power (Precedence 7)
2) Arithmetic such as multiplication, division, modulo (Precedence 6)
3) Arithmetic such as addition and subtraction (Precedence 5)
4) Comparison operators (Precedence 4)
5) Equality comparisons (Precedence 3)
6) Logical and "&&" (Precedence 2)
7) Logical or "| |" (Precedence 1)

- Users should not have to worry about the format, as each expression must be parsed from the way the user inputs it. For example: "1 + 2" should be the same as "1+2."
- The main() should read expressions from an input file, then output them to the console afterwards.

Functionality:

- Read input from the text file.
- Fix any formatting errors the user may have made.
- Parse the given expressions accordingly and output the results to the console.

System Design:

- **Helper functions:**

  The helper functions in this program are designed with three key functions in mind. “Get_precedence” function takes a given operand and returns its corresponding precedence. The “infix_to_postfix” function converts an infix expression to a postfix expression returning the successfully converted postfix expression. Lastly, the “add_spaces_between_terms” function adds spaces appropriately so that the results of the expression are consistent no matter how the user inputs their expression.

- **Evaluator**

  Evaluator.h has two stacks and two class member functions. Evaluator.h depends on helper_functions.h. The Main program creates a Evaluator object to evaluate the infix expression. We first call eval_infix, with a given string infix expression, which calls multiple functions within helper_functions. First, it calls the function add_spaces_between_terms, which sets the right amount of spaces for the string. Second, it calls evaluate_specific_terms, which takes the top two elements from the stack<int> (holding the operands) and evaluates the operator connecting the two operands. Third, it calls get_precedence, which makes sure that the operands get evaluated in the proper order for infix_expressions.

Data Structures:

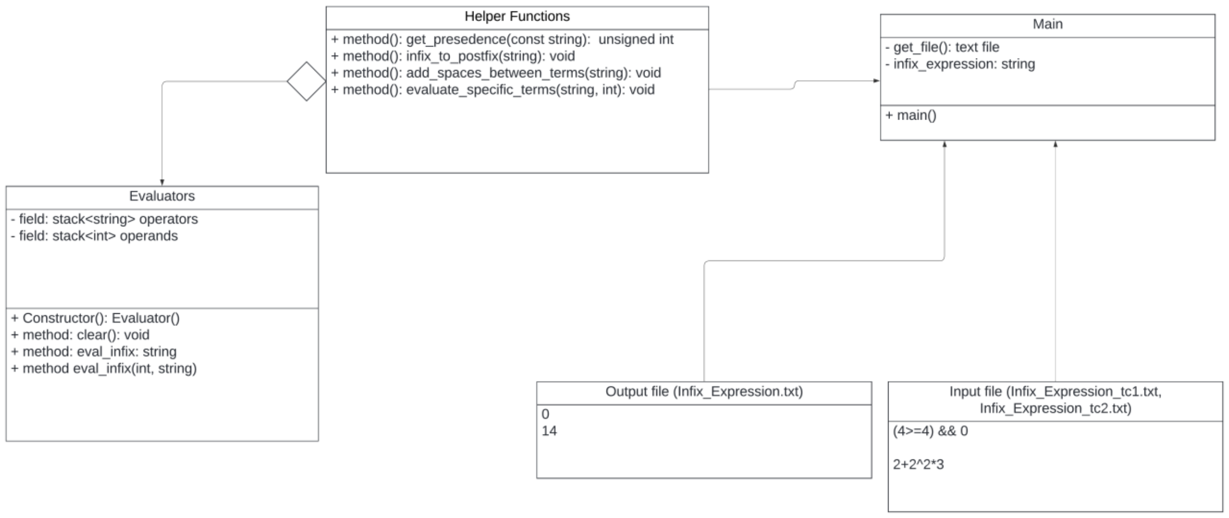
- **Stacks**

  The system uses the stack data structure to efficiently store each operator and operand of an expression so that they can efficiently be pushed and popped as needed.

- **String:**

  The string data type is the main way an infix expression is inputted and outputted.

UML:

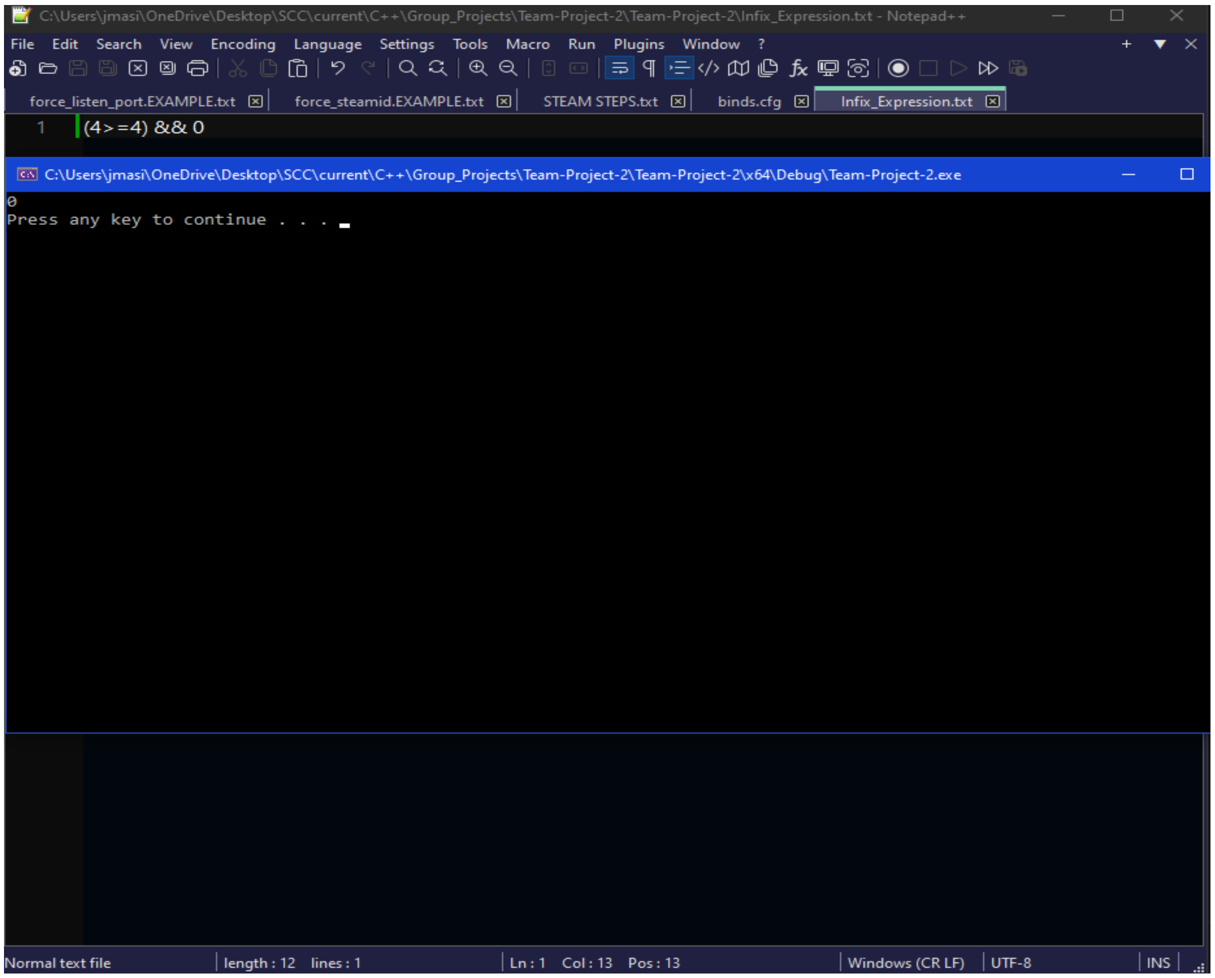


Helper Functions
+ method(): get_presedence(const string): unsigned int
+ method(): infix_to_postfix(string): void
+ method(): add_spaces_between_terms(string): void
+ method(): evaluate_specific_terms(string, int): void
Main
- get_file(): text file
- infix_expression: string
+ main()
Evaluators
- field: stack<string> operators
- field: stack<int> operands
+ Constructor(): Evaluator()
+ method: clear(): void
+ method: eval_infix: string
+ method eval_infix(int, string)
Output file (Infix_Expression.txt)
0
14
Input file (Infix_Expression_tc1.txt, Infix_Expression_tc2.txt)
(4>=4) && 0
2+2^2*3

Test Cases

Test Case #1:

The results of the first input and output are shown below:

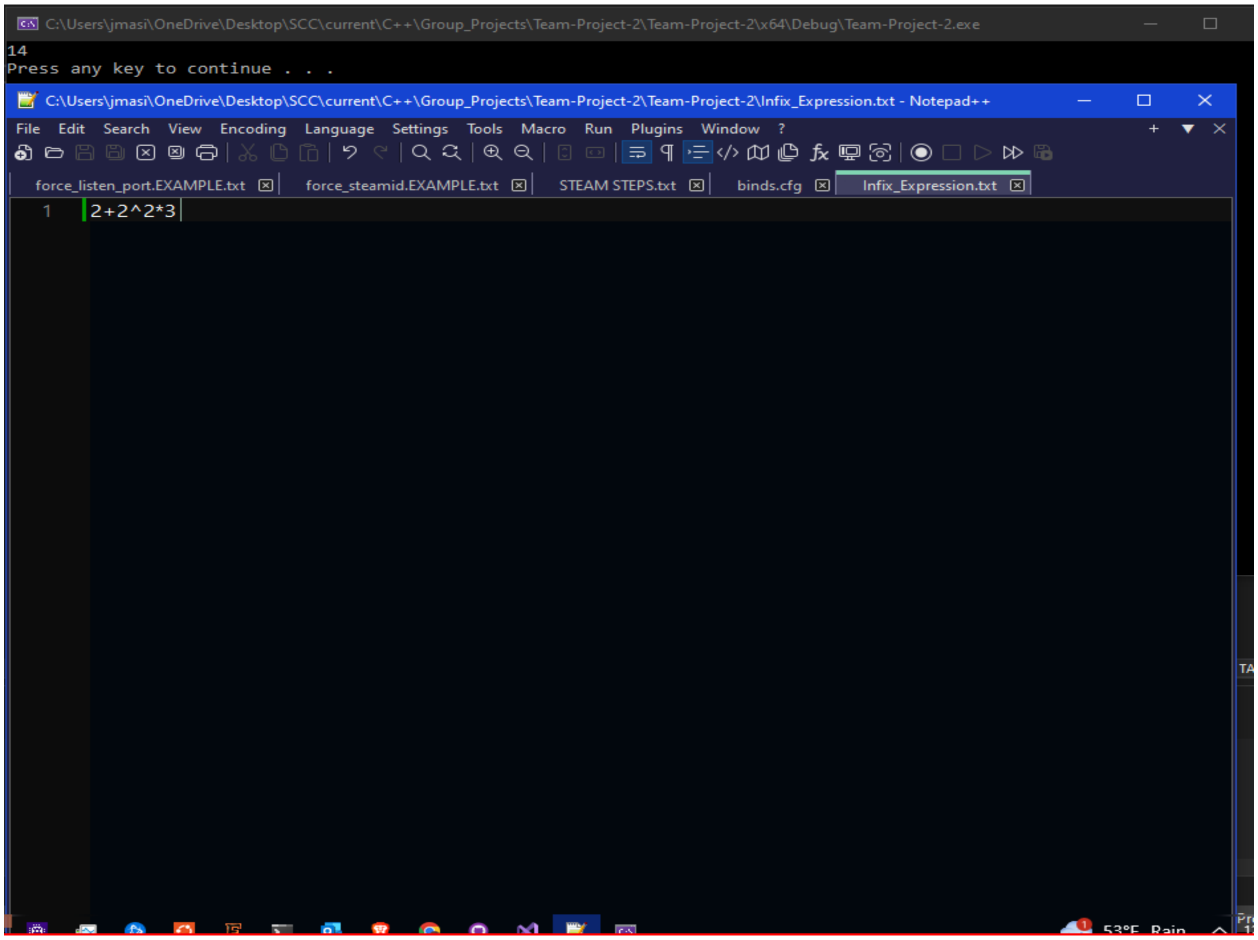

```
(4>=4) && 0
```

```
0
Press any key to continue . . .
```

The expected output would be to have the console output 0 as the correct postfix expression. As shown, as our first infix expression is inputted, the program correctly outputs 0 to the console.

Test Case #2:

The input and output for the second test case is shown below:



```
C:\Users\jmasi\OneDrive\Desktop\SCC\current\C++\Group_Projects\Team-Project-2\Team-Project-2\x64\Debug\Team-Project-2.exe
14
Press any key to continue . . .
```

```
C:\Users\jmasi\OneDrive\Desktop\SCC\current\C++\Group_Projects\Team-Project-2\Team-Project-2\Infix_Expression.txt - Notepad++
2+2^2*3
```

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
force_listen_port.EXAMPLE.txt
force_steamid.EXAMPLE.txt
STEAM STEPS.txt
binds.cfg
Infix_Expression.txt
53°F Rain

The expected output of this infix expression is 14. As shown, the program correctly outputs 14 to the console after evaluating this input.

Team Member Contributions:

- **Joe Simon**

  - **Programming:** Designed and thought of the “evaluator” functions and header. Made many logical and design-based contributions as well as providing tips to help teammates with their own work.

- **Team meetings:** Attended team meetings, actively engaging in discussions and decision-making process.
- **Debugging:** Worked in collaboration with team members to identify and fix bugs.
- **UML Diagram:** Helped with the design and descriptions in the UML diagram.

- **Eric Vaughn:**

  - **Team Meetings:** Attended team meetings, actively engaging in discussions and decision-making process.
  - **Programming:** Made contributions to the overall program and was the head contributor for the helper functions.
  - **Debugging:** Worked in collaboration with team members to identify and fix bugs.

- **Jordan Pham**

  - **Documentation:** Documentation, identifying completed tasks, bug fixes, and outlining remaining tasks.
  - **UML Diagram and Cover Page:** Created the UML diagram, providing a visual representation of the project's structure. Designed the cover page, ensuring a professional and cohesive project presentation.
  - **Team Meetings:** Attended team meetings, actively engaging in discussions and decision-making process.
  - **Programming:** Made contributions to the overall program as well as a very early beta for the outline of the program.
  - **Debugging:** Worked in collaboration with team members to identify and fix bugs.
  - **Moral support:** Frequently checked in on members to ensure that team members got the support they needed to finish their tasks.

Future Improvements:

- **GUI Interface:**
  - Develop a graphical user interface for a more user-friendly experience.
- **Clean up:**
  - As it is now, our program is functional, but for anyone looking into the source code it does seem very messy. Potentially condensing the header files and their respective functions could be a future improvement.
- **More team meetings:**
  - As a team we could've met more often to discuss the project than we already did.
- **Error handling:**
  - A system to properly output any potential fatal input errors or handling errors could've been added as an improvement.