CMSC 330 Project 2

Jade Pearl

Section 6381

Professor Yuhua Jiang

**Questions From the Professor**

1. What IDE or Editor you used for the building and running of your C++ code.

I used Visual Studio Code as my editor for running C++ since I am most familiar with it.

1. How to run your code with the location of your input files.

To run the code, open the Windows PowerShell command prompt and redirect the terminal’s current working directory to the folder where the .cpp and .h files of the project are located using the cd command. Make sure the input file is in the same directory as the other files or it WILL NOT work. You must compile all of the files at once rather than one file at a time otherwise, the program will not compile or work. Compile the all the .cpp files with the command “g++ \*.cpp” and then run the program with the command “start a.exe.”. This is because when a .cpp file is compiled in PowerShell without a specific output file name determined, Windows by default chooses the name of “a.exe” as the output file.

1. What problems you experienced in building and running your code.

The problems I ran into when building and running the code is the inability to compile all of the files separately. I.e. if I compiled project2.cpp, it would output errors that do not actually exist. This shows that in order to work properly, all .cpp files must be compiled simultaneously. I was not sure if this was an intentional requirement or if this was dependent on the IDE or editor used. Another issue I experience is whenever I run the code and there is an exception thrown due to issues such as uninitialized variables or variables initialized multiple times, the output shows in the PowerShell very briefly before terminating on its own and closing the PowerShell terminal. To make sure that the output remained visible, I clicked and held the border of the window of the terminal that opened as if I was about to move it, this made the terminal remain visible. This can be slightly tedious since it may take a few tries to hold the window before it terminates and exits automatically.

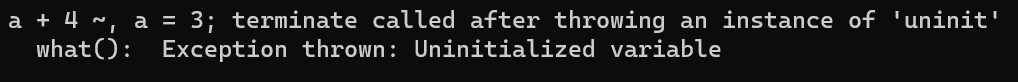
**Approach**

I approached this project in a step-by-step manner. After looking through and understanding the codes provided in the skeleton .zip file, I went through each task. First, I created new .h files and programmed the classes for each operator already not included, starting with the remaining binary operators not yet implemented and ending with the quaternary operator. This was done while also editing the .cpp and .h files already provided to adjust to the new classes being added. I then edited the code to allow for underscores and unsigned floating numbers to be parsed. After that, I completed the third and last task which required initialization of the symbol table and exception handling of uninitialized and reinitialized variables.

**Test Plan**

There are five sample input files that were used in the test plan for this project. These tests were meant to verify if all operators worked and if the exception handling was accurate:

Test 1: Single line expression with no surrounding parentheses 🡪 syntactically incorrect



Result: This shows that there must be at least one set of parentheses around an expression for the program to work.

Test 2: Sample input 🡪 syntactically correct

A math equations and symbols

Description automatically generated with medium confidence  
A black screen with white text

Description automatically generated

Result: Exception thrown at the second to last line and the last line of the input is not read due to program termination. This shows that the exception handling for an uninitialized variable works properly. The lines before the second to last line all exercise the use of all operators and the underscores. The calculations are correct, so all operators and the underscore works properly.

Test 3: Randomized input 1

A number and equation with black text

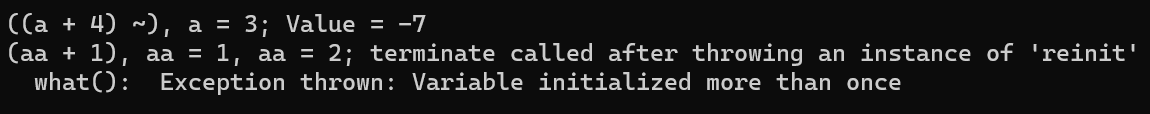
Description automatically generated with medium confidence  
A black background with white text

Description automatically generated

Result: This result ensured proper calculations and resetting of symbol table after each line so that each line of the input is calculated correctly. Also shows that the unary ~ negation expression functionality works.

Test 4: Randomized input 2

A number and symbols of a mathematical equation

Description automatically generated with medium confidence  


Result: Shows that the exception handling for a variable initialized more than once works properly.

**Lessons Learned & Possible Improvements**

I have learned how to implement new parts of grammar such as the binary operators beyond addition, subtraction, multiplication, and division. The unary, ternary, and quaternary operations were also a new learning experience. This project has taught me how to configure parsing of new operations within expressions and how to interact between many different classes and .cpp files to get the desired results from input. Even making a calculator for analysis of simple expressions still requires a lot of practice and knowledge in interacting classes and of the functionality of a parser. The possible improvements I could make for next time is learning to better understand exception handling under certain conditions. Exception handling was already a difficult process to code in C++ for me, so the exceptions were a challenge for me to program in this project. I have higher experience with exception handling, however this project left me wanting to practice it even further in order to master the concept.