Antonio E. Ramirez

13 October 2019

Project 2

CMSC 330, 6381, University of Maryland University College

# Process & Lessons Learned

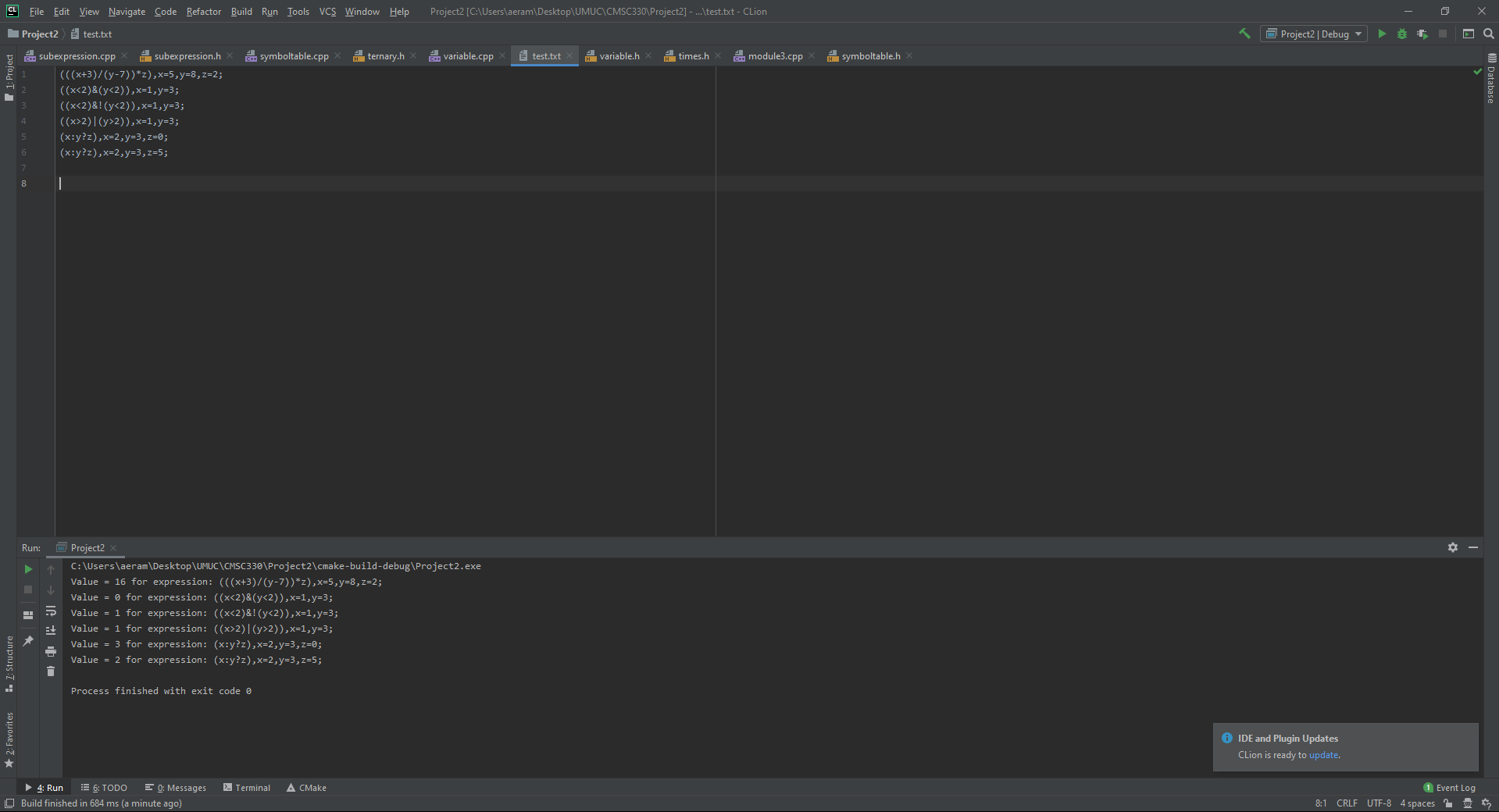
For this project we were required to complete and extend an expression interpreter using the C++ language. This presented some unique challenges. The first challenge was understanding the syntax of the new language. I had no experience with C++ prior to this project. I had to find a new IDE and install a compiler. I ended up using an IDE that was developed by the same people who developed the Java IDE I use, IntelliJ. There a dew nuances to C++ that were interesting to work with. The use of the .h and a separate .cpp file was kind of confusing at first since the .cpp is always not required. One huge help was the provided code in the readings. Most of the program was provided, 7 out of 10 classes, and the last 3 classes need were the rest of the operator classes. We were provided with one of the operators, so it was used as a template for the other 3. Now that we have a functioning basic program, there were a few things that needed to be changed or added. The basic program took input from the console and could only process 4 kinds of operators: addition, subtraction, multiplication, and division. The first step was to add functionality to process comparative and conditional operators. This was simple since we already had templates from the other operators. Ince the new operator classes were created, we just needed to modify the switch statement that processed which operator was being used. Another easy modification was changing the type from double to int. I did not have any issues with this change. The biggest modification was changing the input from the console to a file. The original program used the basic cin to read data from the console. This only allowed for one expression to be evaluated at a time. We wanted to modify the program to read a .txt file and read each line which contained an expression. The output remained to the console. To accomplish this an ifstream was used. This stream would read each line using stringstream. This allowed the string to be tokenized and read. It is important that the string is tokenized so each operand/operator can be evaluated. Some considerations that had to be kept in mind were making sure that the proper #include tags were used with each class. I was having an issue because of this problem. The other consideration being that all methods that used the cin functionality, needed to be modified to accept a parameter of the file input. With a functioning program we now needed to test the program. Because the program accepts multiple inputs per file, we can test the functionality quickly and easily. We want to test each operator and make sure they function as intended. We also want to test incorrect syntax.

# Test Cases

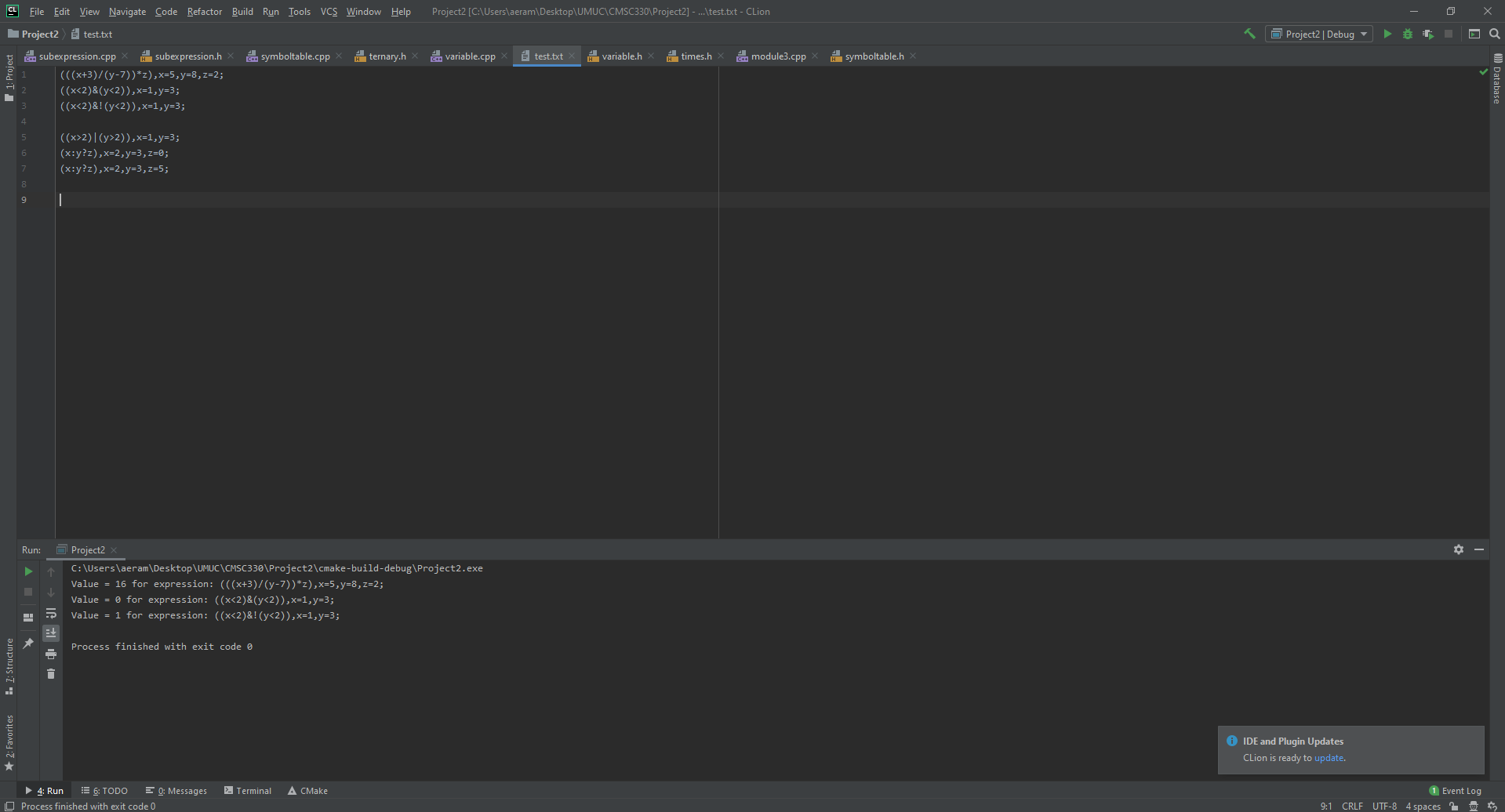
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Test Scenario | Input | Expected Output | Pass/Fail | Screenshot Number |
| 1 | Testing functionality of all operators and file input | (((x+3)/(y-7))\*z),x=5,y=8,z=2; ((x<2)&(y<2)),x=1,y=3; ((x<2)&!(y<2)),x=1,y=3; ((x>2)|(y>2)),x=1,y=3; (x:y?z),x=2,y=3,z=0; (x:y?z),x=2,y=3,z=5; | Value = 16 for expression: (((x+3)/(y-7))\*z),x=5,y=8,z=2;  Value = 0 for expression: ((x<2)&(y<2)),x=1,y=3;  Value = 1 for expression: ((x<2)&!(y<2)),x=1,y=3;  Value = 1 for expression: ((x>2)|(y>2)),x=1,y=3;  Value = 3 for expression: (x:y?z),x=2,y=3,z=0;  Value = 2 for expression: (x:y?z),x=2,y=3,z=5; | Pass | 1 |
| 2 | Testing functionality with a blank line between expressions | (((x+3)/(y-7))\*z),x=5,y=8,z=2; ((x<2)&(y<2)),x=1,y=3; ((x<2)&!(y<2)),x=1,y=3;  ((x>2)|(y>2)),x=1,y=3; (x:y?z),x=2,y=3,z=0; (x:y?z),x=2,y=3,z=5; | First 3 expressions to output to the console and skipping the expressions after the space | Pass | 2 |
| 3 | Testing functionality with back to back operators | (((x+-3)/(y-+7))\*/z),x=5,y=8,z=2; | Expression should not work and nothing should output | Pass | 3 |

# Screenshots

Screenshot 1



Screenshot 2



Screenshot 3

