# My process description:

## Adding operator functionalities:

When I started the project I was intimidated, because I have no previous experience with C++, and it has been a while since I took the introductory programming course that used C as the language. However, I quickly realized that the best way to start this project was to just look through all the source code provided, since most of the extensions to the program involve adding new operators for the program to handle. After looking through how the operator .h files worked and how subexpression.cpp implemented them I figured that I could just use the already created .h operator files as a template to create the functionality for the new operators and then simply add a call to them and a return from them in subexpression.cpp, just like the other operators do. This worked for all operators except for not and the conditional expression operator, to which the .h is named ternaryop in my program. For all the operators besides the two mentioned above I just used the built in C++ operators in the return statements of their respective .h files.

For the not operator, I originally created a case for it and returned it like the other operators, but I was worried about the requirement that it take in left and right as arguments. In my initial testing it seemed to work fine, but to be safe and prevent undefined behavior I decided to check if the operator was a not operator before the switch statement and if it was only pass in left operand. Furthermore, since Subexpression requires a right operand still, I have not.h pass nullptr as an argument. This ensure when the not operator is found, only left operand is needed and acted upon.

For the conditional expression operator handling, I took a similar approach to not.h. I check if the colon of the conditional operator is the current operation before the switch statement. Since this operator is ternary, I needed to account for this in subexpression.h without changing everything else in the program to save time and keep it clean. So, I added an Expression\* third = nullptr to subexpression.h’s parameter list. This allows for the third operand and setting a default value to nullptr allows all the other operators to remain the same and not pass in nullptr for each operator to satisfy this new requirement of three operands. Next, I updated the pointers and references in subexpression.h and .cpp to contain this new operand. Finally, in the else if handling the conditional expression operator it was as simple as to get the three operands, the ? operator, the closing parenthesis and then returning a new TeranaryOp.h with left, right and third as arguments.

## What logical expressions do for value output:

When logical expressions, except for the conditional expression operator, are used in this program the output value will be 1 for true statements and 0 for false.

## Changes to Module3.cpp for file input

For module3.cpp, which contains the main of this program I made a few modifications to support the requirements of this project. The project states that we are to extend the program to accept input from a file, I take that to mean we are to add the functionality of accepting input from file, without depreciating the programs previous capabilities. So, I made it a choice to either read in from file, or to read in from the standard I/O in console. I also coded this part of the program to keep asking for an input selection, tell user file could not be read and it allows user to press a key to exit, I did this by wrapping it all in a do while statement and checking what selection choice was. If file input is selected, the filename is stored in filename and a ifstream is created with the file. In order for the program to work with an ifstream a input I had to change parse, subexpression and their respective .h files to take in istream& in. this allows either the file input or the standard cin to function properly. The program treats the file input the same as input form the console, except it keeps going through all lines in the file.

## Other thigs to note:

To get int output from to come from expression, I changed its type to int, so that way all the .h files return types are automatically of type int. Since the specification say particularly for expression to return int, I thought to do this in its .h file, instead of leaving it double and having all the files that use it cast their returns to int.

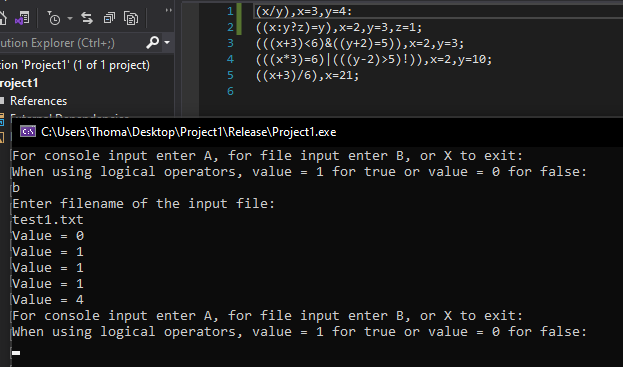
I had to add a few lines of code to handle cases where the user selected something that is not valid. Before these changes, my cout lines would print several times according to how many characters were in the incorrect selection. Using cin.clear an the following line (lines 55 and 56) prevented this from happening.

When I was coding for file input, I had issues when reusing variables. I found the issue to lie in symboltable.cpp. If the variable were assigned a value on a previous input file line, the program would crash because the program looks up previously assigned variables and did not allow for them to be changed. I did notice that If a variable was already assigned a value on a previous line, the variable could be used in another expression without reassigning the variable. I thought this should be functionality that should be preserved and left lookup alone. So, I coded symboltable.cpp’s insert to search for a variable and if it is found reassign it, otherwise assign a new variable. I thought this to be a pretty cool feature, because now I can set x, y, and z to some value, and then preform many expressions with the same value or I can reuse x, y, z on every expression with new values.

Test 1:

Provided expressions that should process without error provided in Ask the Professor comments.

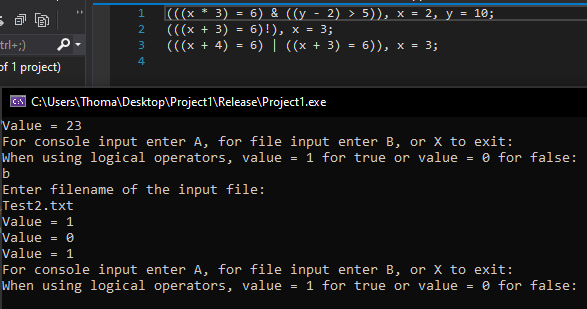
|  |  |  |  |
| --- | --- | --- | --- |
| Inputs: | Expected Outputs | Actual Outputs | Pass/Fail |
| (x/y),x=3,y=4; | 0 | 0 | Pass |
| ((x:y?z)=y),x=2,y=3,z=1; | 1 | 1 | Pass |
| (((x+3)<6)&((y+2)=5)),x=2,y=3; | 1 | 1 | Pass |
| (((x\*3)=6)|(((y-2)>5)!)),x=2,y=10; | 1 | 1 | Pass |
| ((x+3)/6),x=21; | 4 | 4 | Pass |



Test 2:

Single spaced operators and operands, plus more expressions.

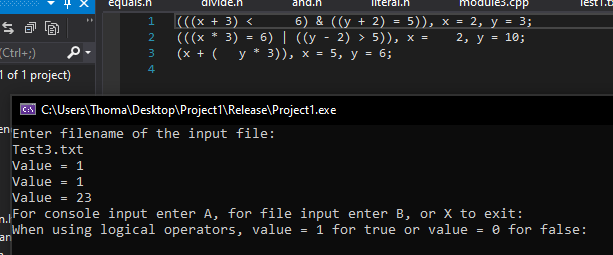
|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Expected Outputs | Actual Outputs | Pass/Fail |
| (((x \* 3) = 6) & ((y - 2) > 5)), x = 2, y = 10; | 1 | 1 | Pass |
| (((x + 3) = 6)!), x = 3; | 0 | 0 | Pass |
| (((x + 4) = 6) | ((x + 3) = 6)), x = 3; | 1 | 1 | Pass |



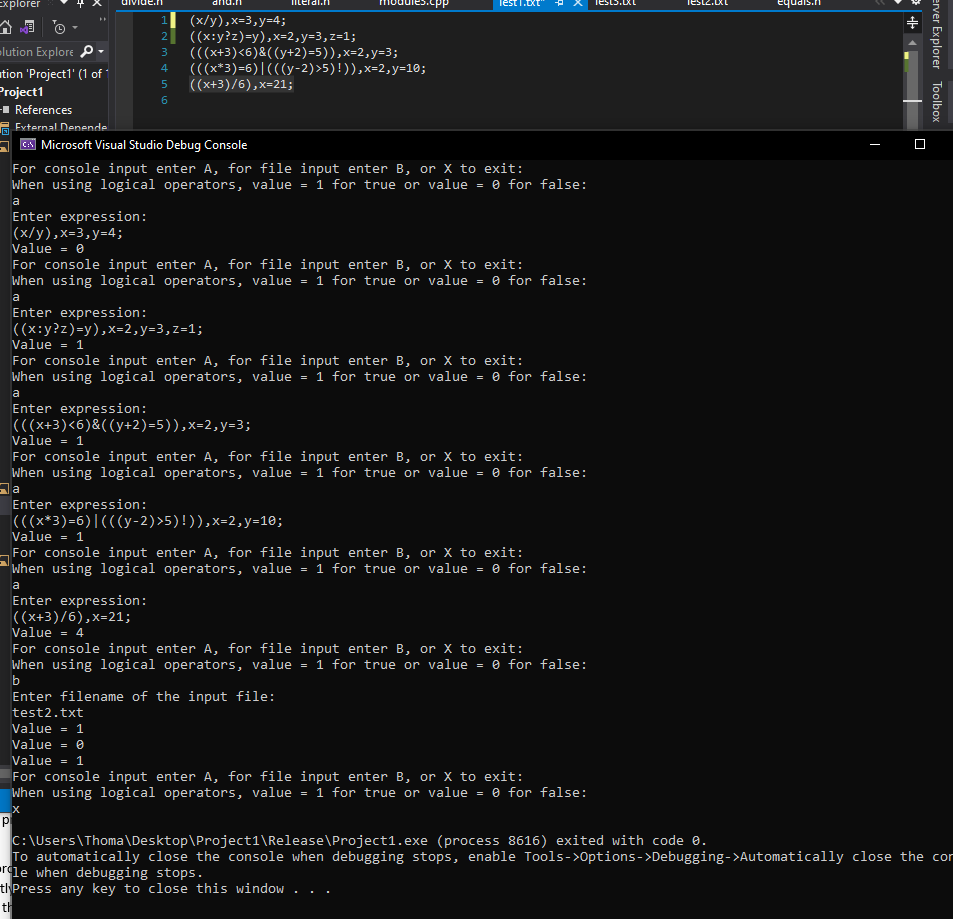
Test 3:

Mixed spacing.

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Expected Outputs | Actual Outputs | Pass/Fail |
| (((x + 3) < 6) & ((y + 2) = 5)), x = 2, y = 3; | 1 | 1 | Pass |
| (((x \* 3) = 6) | ((y - 2) > 5)), x = 2, y = 10; | 1 | 1 | Pass |
| (x + ( y \* 3)), x = 5, y = 6; | 1 | 23 | Pass |



Test 4:

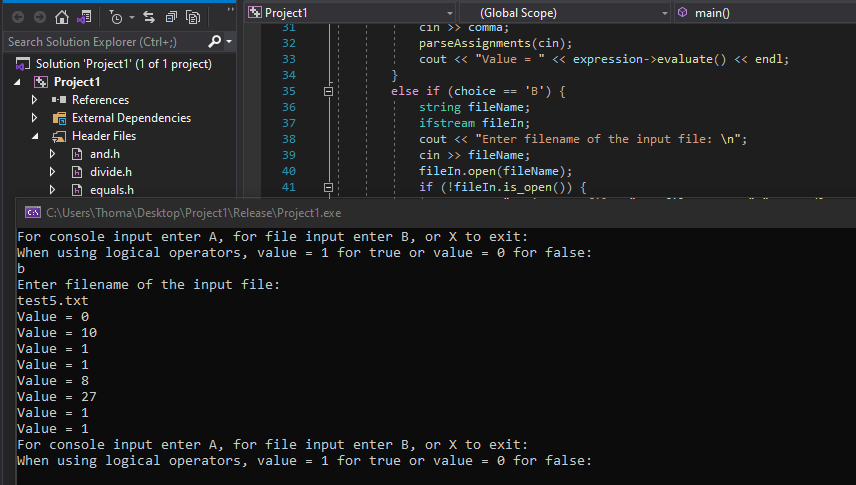
Testing Console input, file input and close selection together

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs(from file): | Expected Outputs | Actual Outputs | Pass/Fail |
| (((x \* 3) = 6) & ((y - 2) > 5)), x = 2, y = 10; | 1 | 1 | Pass |
| (((x + 3) = 6)!), x = 3; | 0 | 0 | Pass |
| (((x + 4) = 6) | ((x + 3) = 6)), x = 3; | 1 | 1 | Pass |
| **Inputs from console promt** |  |  |  |
| (x/y),x=3,y=4; | 0 | 0 | Pass |
| ((x:y?z)=y),x=2,y=3,z=1; | 1 | 1 | Pass |
| (((x+3)<6)&((y+2)=5)),x=2,y=3; | 1 | 1 | Pass |
| (((x\*3)=6)|(((y-2)>5)!)),x=2,y=10; | 1 | 1 | Pass |
| ((x+3)/6),x=21; | 4 | 4 | Pass |

Test 5:

I wanted to add more examples of the program using expressions that do not just evaluate to true or false, and I have altered previous logical inputs to evaluate to opposite to what they were before to show logical operators functioning properly.

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Expected Outputs | Actual outputs | Pass/Fail |
| ((x:y?z)>y),x=2,y=8,z=1; | 0 | 0 | Pass |
| ((x\*3)+((y-5)+z)),x=2,y=8,z=1; | 10 | 10 | Pass |
| (((a+3)<6)&((b+2)=5)),a=2,b=3; | 1 | 1 | Pass |
| (((x\*3)=6)|(((y-2)>5)!)),x=7,y=5; | 1 | 1 | Pass |
| ((((x+3)/6)\*4)-8),x=21; | 8 | 8 | Pass |
| ((x + y) + z), x=3, y=15, z=9; | 27 | 27 | Pass |
| ((x < y)|(y < z)), x = 4, y = 15, z=3; | 1 | 1 | Pass |
| ((x < y) & (z < y)), x = 4, y = 15, z=3; | 1 | 1 | Pass |



NOTE: I found here that reusing variables preciously assigned does not work. I thought it did in my initial testing but realize now it only works if there are only two expressions in the file. Then the second can use the first expressions variables. If any expressions are after this, they will not be evaluated. I have a feeling to solve this would require changes outside of symboltable.cpp and sense this was not a requirement of the project I will leave it as is working perfectly fine when variables are assigned for each expression. Also, I do not want to go deeper into this and change things, because I worry I may break the whole program or just run into more and more problems.

# Lessons learned:

It was so much easier to test and show the program working for this project after I coded the selection of input file or console input to keep running on a loop, even if input were incorrect or file could not be opened. To me it shows that coding a program to handle unexpected input and other problems with out crashing can streamline the testing process and at the same time have a more robust project to be proud of.

I learned that coding for a partially complete project can be fun and not too difficult, at least in this case were the already done code was done correctly and we just needed to add extra functionality to it. It was fun because it required adding some extra things to the program, seeing something doesn’t work with the previously done code and then having to figure out what things I have to add to make it work

I also learned that, at least for me, sometimes the hardest part of a project is figuring out where to start. Since this project already had a good chunk of the code done it was easy to read through and figure out the way the code started to do everything required and then continue in the same manner to finish the project.

Finally, I learned that learning one programming language does help when trying to do something in another language. Up until now all my major projects have been in Java, but that seemed to make me more comfortable when diving into a new language like C++. Of course, the readings describing C++ and its attributes helped, but I also think being proficient in one language makes learning or coding in a new one easier.

# Note:

I hope that the extra functionality I added to the program is acceptable even though some of it is not explicitly stated in the project specification. Since the project was to extend this program I wanted to be absolutely sure I did not take away any functionality the program already had.