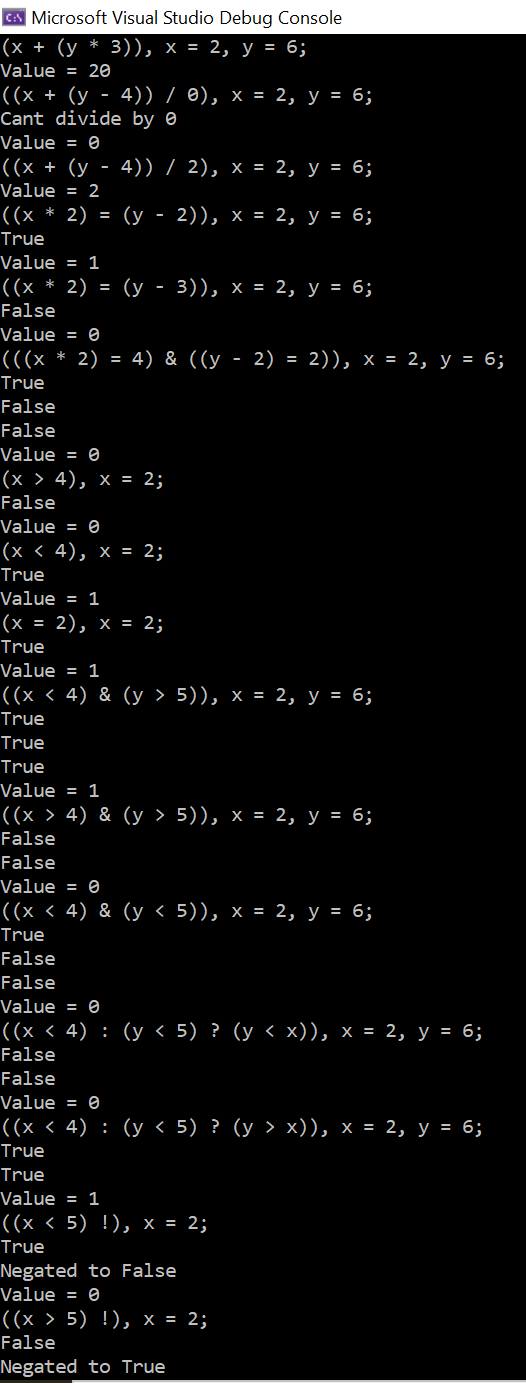
Kresimir Tokic  
CMSC330 Project 2  
Test Plan, Lessons Learned

**Test Plan**

The following test plan should cover all the bases illustrating the successful input and output of all methods in the program.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| (x + (y \* 3)), x = 2, y = 6; | Value = 20 | Value = 20 | Pass |
| ((x + (y - 4)) / 0), x = 2, y = 6; | Cant Divide By 0 | Cant Divide By 0 | Pass |
| ((x + (y - 4)) / 2), x = 2, y = 6; | Value = 2 | Value = 2 | Pass |
| ((x \* 2) = (y - 2)), x = 2, y = 6; | TRUE | TRUE | Pass |
| ((x \* 2) = (y - 3)), x = 2, y = 6; | FALSE | FALSE | Pass |
| (((x \* 2) = 4) & ((y - 2) = 2)), x = 2, y = 6; | FALSE | FALSE | Pass |
| (x > 4), x = 2; | FALSE | FALSE | Pass |
| (x < 4), x = 2; | TRUE | TRUE | Pass |
| (x = 2), x = 2; | TRUE | TRUE | Pass |
| ((x < 4) & (y > 5)), x = 2, y = 6; | TRUE | TRUE | Pass |
| ((x > 4) & (y > 5)), x = 2, y = 6; | FALSE | FALSE | Pass |
| ((x < 4) & (y < 5)), x = 2, y = 6; | FALSE | FALSE | Pass |
| ((x < 4) : (y < 5) ? (y < x)), x = 2, y = 6; | FALSE | FALSE | Pass |
| ((x < 4) : (y < 5) ? (y > x)), x = 2, y = 6; | TRUE | TRUE | Pass |
| ((x < 5) !), x = 2; | Negated to False | Negated to False | Pass |
| ((x > 5) !), x = 2; | Negated to True | Negated to True | Pass |

See screen shots below showing successful execution of the program.



**Lessons Learned**

Thankfully half of this program was already written for me, otherwise there is no way I would’ve been able to complete it in the given time period. First, I learned that I still dislike C++ and prefer Java and that is largely because of comfort with the latter. Through the class discussion over the weeks I did gain some appreciation for the language. I still think the language is unforgiving, syntactically difficult and ugly making it hard for me to read. I don’t mind the verbosity that most people dislike about Java. I may be wrong in this, but I view header files in C++ as somewhat analogous to interfaces in Java. That helped me get through this project adding the minus, divide and logic operators. I believe this what touches on separating the specification from the implementation. I originally was against this, but as long as the spec and implantation are easily accessible, I don’t think it’s a bad thing. I had to look up ternary operators to understand that request. I found this site broke it down very simply for me. <https://www.freecodecamp.org/news/c-ternary-operator/> I also found an example on Stack Overflow that puts my strings back into “cin” so that I didn’t have to make as many modifications. <https://stackoverflow.com/questions/3797280/injecting-string-to-cin> As far as I understand, this isn’t considered good practice but I’m not sure why. I still prefer to pass parameters by value like Java does. I don’t really see a need for this program to leverage concurrency or any mutual exclusion as it only reads, parses, and evaluates line by line instead of attempting the whole file at once, furthermore the input data set isn’t large enough to concern me. It took me a little while to understand the flow of this program. I read through the Module 3: Imperative Languages—Control Flow a couple times and starting copying/pasting the code into Visual Studio helped me make some sense. Once I realized the header files are all essentially the same, I felt a lot more confident tackling the missing arithmetic operators and simple logic operators.