Kevin Abrahams

CMSC330 Project 2

3/9/21

# Introduction and Programming Process

My programming process started with recognizing how to use the previous code I was given to finalize and complete the rest of the solution. I also had to think of ways to read the input file specified by the user. I thought to separate the various tasks my program had to achieve to numerous classes to simplify, expedite, and facilitate the process. Having the previous code enabled me to work effectively off of the blueprint provided to derive a solution to implementing the missing additional functionality to the program. Once I understood how the previous classes of the operators were implemented, it was straightforward and much simpler to implement the rest of the classes for the operators, such as the logical and relation operators and the mathematical operators.

# Test Cases

For each of the following test cases, I compiled the program using the following statement in the command line, which produced an executable file called Project.exe showing the results of running the program: g++ main.cpp parse.cpp symboltable.cpp subexpression.cpp operand.cpp variable.cpp -o Project.

1. My first test case involves the mathematical operators of addition, subtraction, multiplication, and division. It involves a file that contains several statements or expressions to indicate the program’s ability to successfully read and interpret an input file’s expressions line by line.

The input file for the first test case, input.txt, contains the following content:

(x+y), x = 2, y= 6; /\* Expect value = 8 \*/

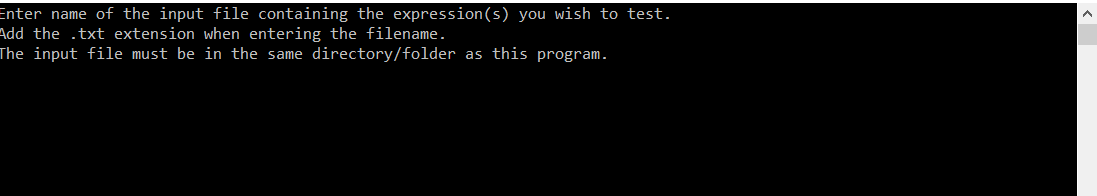
(x-y), x=0, y=0; /\* Expect value = 0 \*/

((x+y)/(x-y)), x=10, y=8; /\* Expect value = 9 \*/

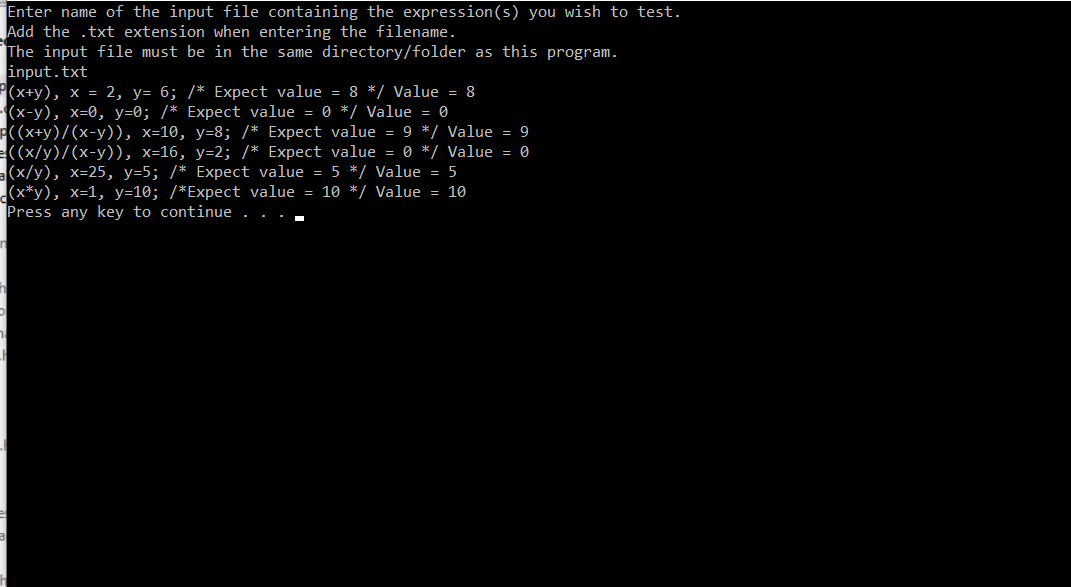
((x/y)/(x-y)), x=16, y=2; /\* Expect value = 0 \*/

(x/y), x=25, y=5; /\* Expect value = 5 \*/

(x\*y), x=1, y=10; /\*Expect value = 10 \*/

When the program starts, the user shall be greeted with the following prompt asking for the input file. 

Simply type input.txt to select the input file. The file must be in the same folder/directory as the main.cpp file, meaning it is best to not store the input file in an inner folder.



For each computation, the correct result is returned by the program for the expression. As the following screenshot demonstrates, the program correctly performs each mathematical computation and operation. For instance, it correctly returns a value of 8 from 2+6 and a value of 0 from 0 – 0. It also returns the correct values for more complex statements, such as 10 + 8 / (10 – 8), which is 18 / 2. This has a value of 9, as the program correctly shows. It also correctly performs 1 \* 10.

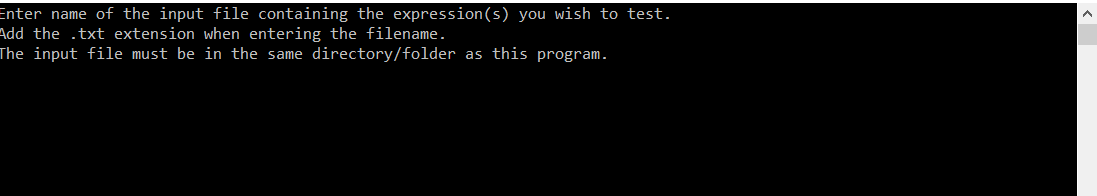
1. The second test case involves reading an input file containing expressions using the less than operator.

The input file, input2.txt, which is stored in the same directory as the program and is read in by the program, contains the following content:

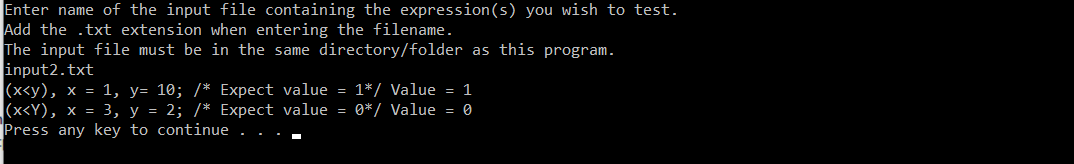
(x<y), x = 1, y= 10; /\* Expect value = 1\*/

(x<Y), x = 3, y = 2; /\* Expect value = 0\*/

When the program starts, the user shall be greeted with the following prompt asking for the input file.



Simply by entering input2.txt, the second input file shall be read.



Upon running the program, the correct result is displayed that x is indeed less than y. The program returns a value of 1 indicating that the condition is true.

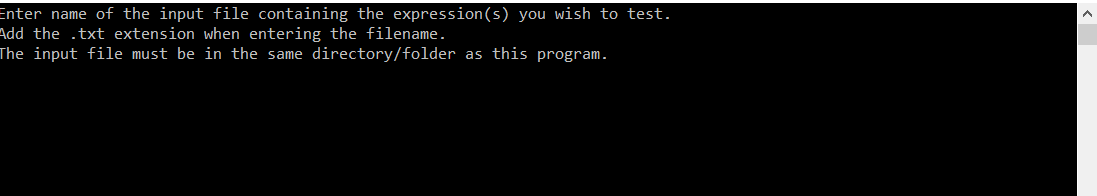
1. The third test case involves comparison using the greater than operator to illustrate that the expressions are correctly interpreted.

The input file, input3.txt, contains the following content:

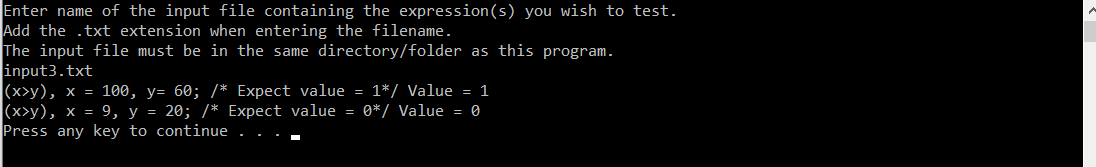
(x>y), x = 100, y= 60; /\* Expect value = 1\*/

(x>y), x = 9, y = 20; /\* Expect value = 0\*/

When the program starts, the user shall be greeted with the following prompt asking for the input file.



Simply by entering input3.txt, the third input file shall be read.



For each expression, the correct result is produced and shown by the program. The first expression, that 100 is greater than 60, is true, so the program should return a value of 1 which it does. The second expression, that 9 is greater than 20, should have a value of 0 to indicate that it is false. The program successfully does this, as the above screenshot shows.

1. In the fourth test case, an expression comparing two operands for equality is tested.

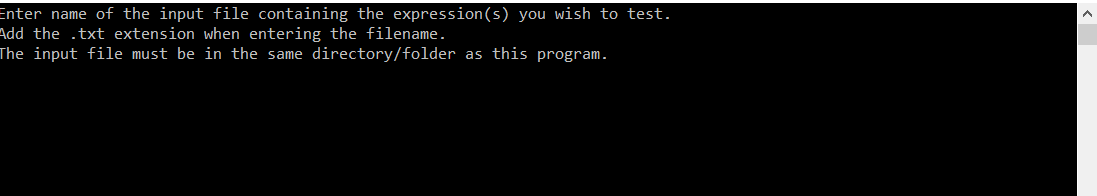
The input file, input4.txt, contains the following three expressions testing the equality operator:

(x=y), x = 8, y= 8; /\* Expect value = 1\*/

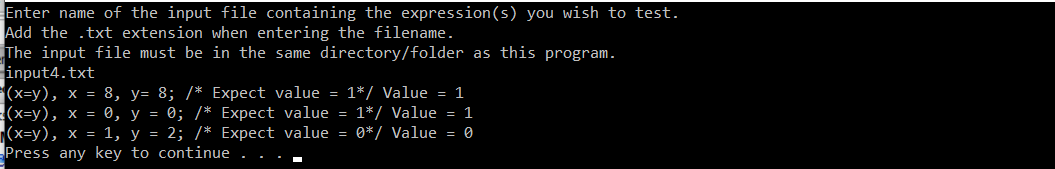
(x=y), x = 0, y = 0; /\* Expect value = 1\*/

(x=y), x = 1, y = 2; /\* Expect value = 0\*/

When the program starts, the user shall be greeted with the following prompt asking for the input file.



Simply by entering input4.txt, the fourth input file shall be read.



For the first expression, the program should return a value of 1, which it does, to indicate that the expression is true, as 8 is indeed equal to 8. For the second expression, the program should return a value of 1 once again to indicate that it is true, as 0 is in fact the same as 0. For the third expression, since it is false that 1 is equal to 2, the program should return a value of 0. The program does each of these successfully.

1. The fifth test case involves verifying that the and logical operator functions properly and as intended.

The input file, input5.txt, contains the four following expressions:

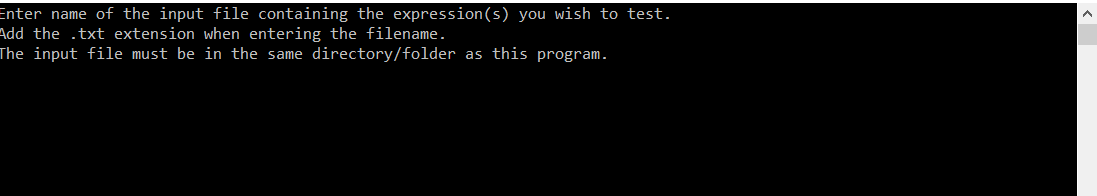
((x>2)&(y>2)), x = 8, y= 8; /\* Expect value = 1\*/

((x>2)&(y>2)), x = 1, y= 8; /\* Expect value = 0\*/

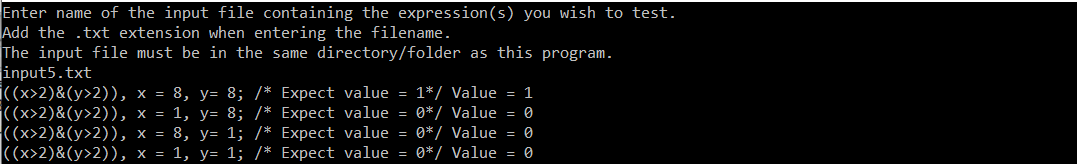
((x>2)&(y>2)), x = 8, y= 1; /\* Expect value = 0\*/

((x>2)&(y>2)), x = 1, y= 1; /\* Expect value = 0\*/

When the program starts, the user shall be greeted with the following prompt asking for the input file.



Simply by entering input5.txt, the fifth input file shall be read.



All four expressions use the and operator to verify whether both conditions are true in the composite statement. The condition being tested for in each of the four expressions is whether both x and y are greater than 2. However, this is only true in the first expression, and so the program should return a value of 1 for the first expression and a value of 0 for the rest. As the program does so, it works properly. Thus, all the tested expressions in this fifth test case display the correct, expected value and result.

1. The sixth test case involves the or logical operator.

The input file, input6.txt, contains the following content:

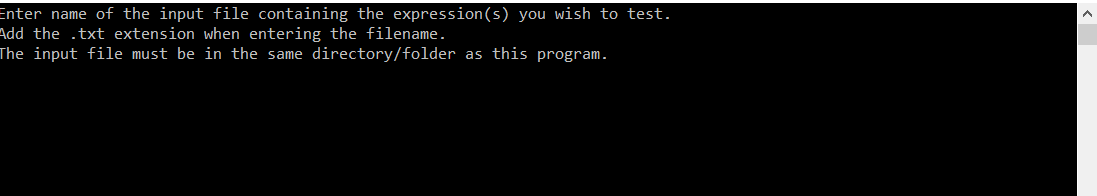
((x>10)|(y>10)), x = 2, y= 20; /\* Expect value = 1\*/

((x>10)|(y>10)), x = 20, y= 20; /\* Expect value = 1\*/

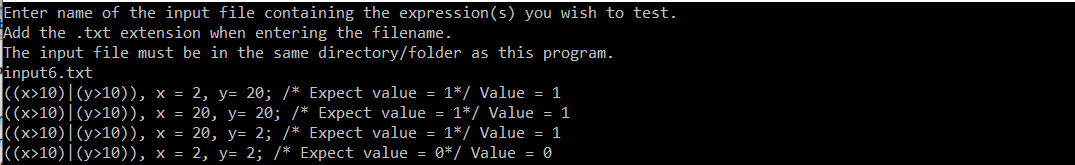
((x>10)|(y>10)), x = 20, y= 2; /\* Expect value = 1\*/

((x>10)|(y>10)), x = 2, y= 2; /\* Expect value = 0\*/

When the program starts, the user shall be greeted with the following prompt asking for the input file.



Simply by entering input6.txt, the sixth input file shall be read.



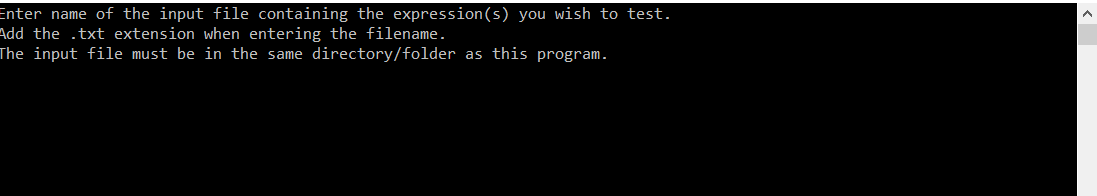
The program displays each correct, expected value. All four expressions use the or logical operator to verify if at least one of the conditions in the composite statement is true. For all four expressions, x or y must be greater than 10. All four expressions are true except for the fourth one, and so the program should show a value of 1 for all of them except for the last expression, which should show a zero. For instance, in the first expression, y is greater than 10 while x is not. For the second expression, both are greater than 10, while in the third only x is. In the fourth, neither is greater than 10.

1. The seventh test case involves the conditional operator.

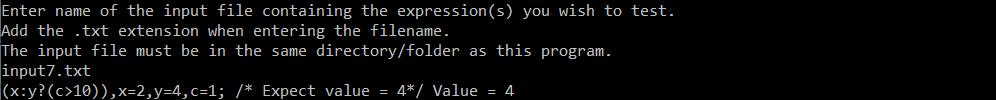
The input file, input7.txt, contains the following content:

(x:y?(c>10)),x=2,y=4,c=1; /\* Expect value = 4\*/

When the program starts, the user shall be greeted with the following prompt asking for the input file.



Simply by entering input7.txt, the seventh input file shall be read.



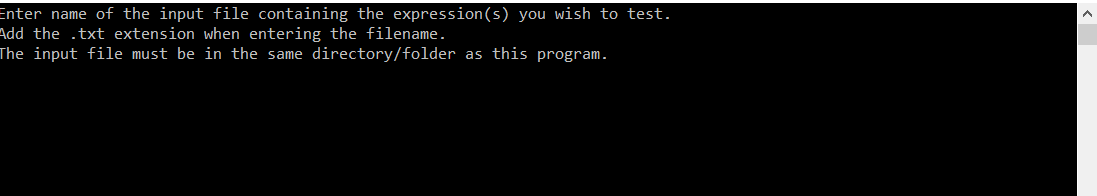
The expression is false, because c, which is equal to 1, is not less than 10, and so the program should return a value of 4, which it does, illustrating that the conditional operator functions properly.

1. The eighth test case involves the unary not operator.

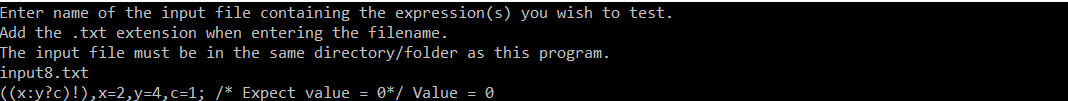
The input file, input8.txt, contains the following content:

((x:y?c)!),x=2,y=4,c=1; /\* Expect value = 0\*/

When the program starts, the user shall be greeted with the following prompt asking for the input file.



Simply by entering input8.txt, the eighth input file shall be read.



The part of the statement stating (x:y?c) evaluates to true, but the unary not operator causes it to switch to false. Thus, the program should display a value of 0 to indicate the statement is false, which it does, and so the unary not operator works properly and as expected.

# Conclusion and Lessons Learned

This assignment taught me much by increasing my familiarity with C++, as I’ve mainly programmed in Java up until this point. I learned more about how compilation differs in C++ from Java. I also learned about other different nuances from C++ and Java, such as how classes are developed with the use of header files. I also learned about namespaces and defining the scope of variables. I also understand how a linker is used in a language such as C++ to ensure an object file has the proper information needed to successfully configure and run the program. I also learned about input files and how to read their contents using C++. I learned a lot of vital things in this program, such as writing classes and methods and creating variables in C++. This program was invaluable for teaching me more about how to read expressions and grammars and how to better understand parsers.