# Project 1 Documentation:

# Expression Evaluator

Vincent Testagrossa

University of Maryland Global Campus

CMSC 330: Advanced Programming Languages

Dr. Yuhua Jiang

October 2nd, 2022

# Contents

## [Problem Description](#_Problem_Description_3)

## [Testing](#_Expression_Input_File:)

## [Reflection](#_Reflection_1)

## Problem Description

This program evaluates expressions provided to it in the following format:

<program> -> <exp> , <assigns> ;

<exp> -> ( <operand> <op> <operand> ) |

( <operand ':' <operand> '?' <operand> ) |

( <operand> '!' )

<op> -> '+' | '-' | '\*' | '/' | '>' | '<' | '=' | '&' | '|'

<operand> -> <literal> | <variable> | <exp>

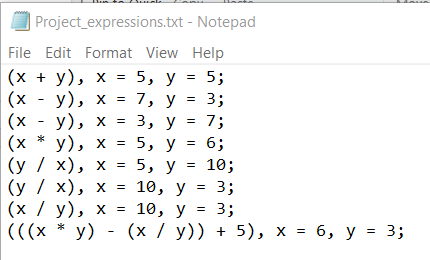
<assigns> -> <assigns> , <assign> | <assign>

<assign> -> <variable> = <literal>

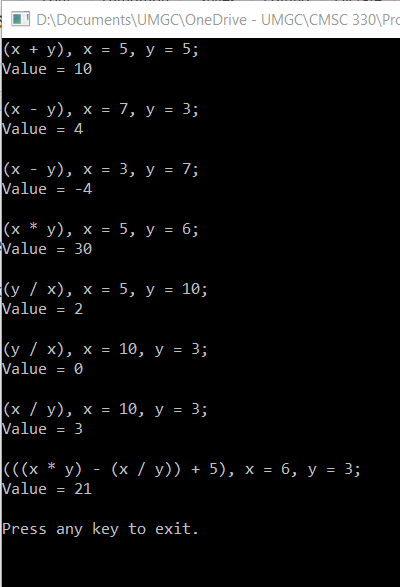
It accepts user input from the command line and reads expressions from one or more files, with one expression per line. The evaluation will display the result back to the user as an integer.Testing

## Expression Input File:

Input:

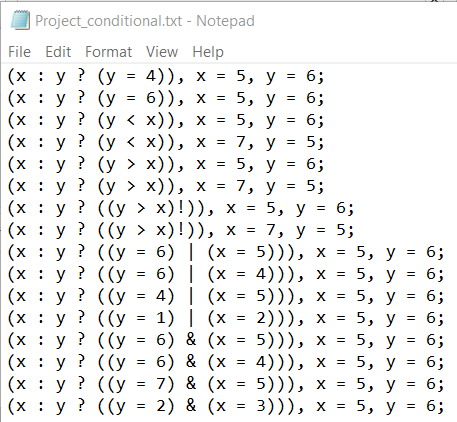


Output:

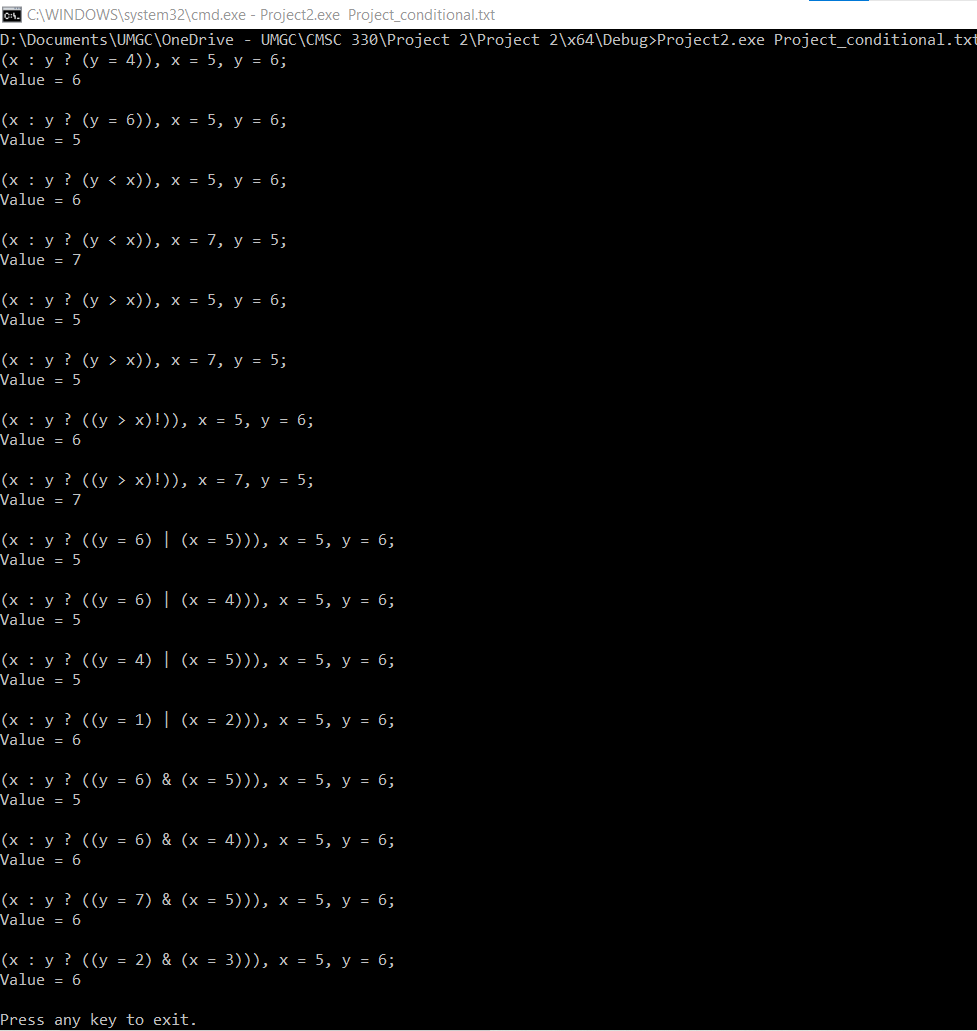


## Conditional/Logical Operators Input File:

Input:

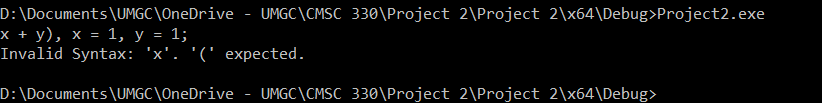
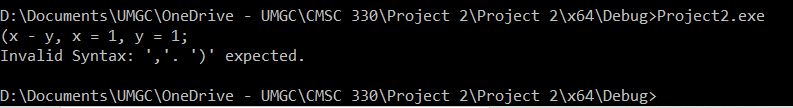


Output:

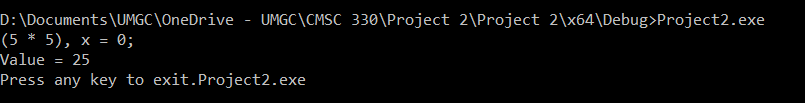


## Testing Syntax Errors and No Args

Input/Output:



## Testing Literals



# Reflection

During the design of this program, developing the minus, divide, and times operations was relatively simple due to the example of the add operation. A big issue from the start was multifile compilation with GCC on Windows 10. Initially, the IDE used was Visual Studio Code, and the instructions for installation don’t cover a lot of the setup required for accomplishing this task using GCC via mingw/msys2 from within VSCode. It wasn’t a cohesive experience, so the IDE was swapped to Visual Studio Community Edition with MSVC as the compiler, which didn’t require nearly as much setup to accomplish a valid minimal build of the program. Once the basic functionality of the program was able to be tested and debugging was made possible, speed of development was only limited by inexperience in C++, namely with the use of streams, instead of the typical paradigm that other languages use (like a print statement). Ultimately, the benefit of using input and output streams were a net benefit due to how simple they are to modify, and redirect to another stream, such as input from a file, which saved development time for the more challenging operations that needed to be included like negate and the conditional operation.

Testing was greatly simplified by the addition of file input. The test cases were categorized to keep the screenshots more manageable and to ensure logical grouping of functions. Conditionals and logical operators work together, so they were tested at the same time. Each conditional operator was tested in a truth-table style to validate that each possible case was tested. The expression operators were tested in a separate file and each operator was tested in an unnested expression, with a nested expression that includes all the operations, variables, and a literal. Lastly, the syntax errors and operator errors were tested using the program with no arguments, mainly because each of the errors stop the program execution and multiple errors can’t be checked in a single file, which was also a logical time to test input from the command line.