CMSC 330 - Advanced Programming Languages
Programming Project 2

The second project involves completing and extending the C++ program that evaluates statements of an expression language contained in the module 3 case study.

The statements of that expression language consist of an arithmetic expression followed by a list of assignments. Assignments are separated from the expression and each other by commas. A semicolon terminates the expression. The arithmetic expressions are fully parenthesized infix expressions containing integer literals and variables. The valid arithmetic operators are +, –, *, /. Tokens can be separated by any number of spaces. Variable names begin with an alphabetic character, followed by any number of alphanumeric characters. Variable names are case sensitive. This syntax is described by BNF and regular expressions in the case study.

The program reads in the arithmetic expression and encodes the expression as a binary tree. After the expression has been read in, the variable assignments are read in and the variables and their values of the variables are placed into the symbol table. Finally the expression is evaluated recursively.

Your first task is to complete the program provided by providing the three missing classes, `Minus`, `Times` and `Divide`.

Next, you should extend the program so that it supports relational, logical and conditional expression operators as defined by the following extension to the grammar:

```
<exp> -> '(' <operand> <op> <operand> ')' |
  '(' <operand> ':' <operand> '?' <operand>  ')' |
  '(' <operand> '!' ')'
<op> -> '+' | '-' | '*' | '/' | '>' | '<' | '=' | '&' | '|'
```

Note that there are a few differences in the use of these operators compared to their customary use in the C family of languages. There differences are

- In the conditional expression operator the symbols are reversed and the third operand represents the condition. The first operand is the value when true and the second the value when false
- The logical operators use single symbols not double, for example the *and* operator is `&` not `&&`
- The negation operator `!` is a postfix operator, not a prefix one
- There are only three relational operators not the usual six and the operator for equality is `=` not `==`

Like C and C++, any arithmetic expression can be interpreted as a logical value, taking 0 as false and anything else as true

Your final task is to make the following two modifications to the program:

- The program should accept input from a file, allowing for multiple expressions arranged one per line. Some hints for accomplishing this transformation will be provided in the conference
- All results should be changed from `double` to `int`. In particular the `evaluate` function should return an `int`.

You may assume that all input to the program is syntactically correct.

You are to submit the source code for the entire program in a .zip file. Your program must compile with Microsoft Visual C++.