CS4100/5100 COMPILER PROJECT
Part 1, 3: Interpreter Construction

Objectives:
This part of the assignment will utilize the SymbolTable, QuadTable, and ReserveTable ADTs from Part 1:1 and 1:2 to actually build a Quad interpreter. The main driver program will need the capability to:

1) Create a QuadTable instance and fill it with the appropriate Quad codes developed to calculate 10-summation and 10-factorial (discussed in class, and in posted example);

2) Create and populate a SymbolTable to accommodate the data elements needed to execute the summation and factorial Quad codes correctly;

3) Run/interpret the Quad codes using the interpreter algorithm given, and print the final answers for the problems. The interpreter itself must create a ReserveTable and populate it with the given opcode mnemonics and op codes so that the TRACE mode shown below will display the mnemonics.

The Interpreter Class
The interpreter must have a constructor:

public Interpreter()

which initializes the Interpreter's ReserveTable, adding its opcode mappings and any other initialization needed.

In addition, ***for the use of this test program,*** the interpreter must offer two public initialization methods:

boolean initializeFactorialTest(SymbolTable stable, QuadTable qtable)

and

boolean initializeSummationTest(SymbolTable stable, QuadTable qtable)

These methods each accept constructed/initialized SymbolTable and QuadTable objects, and add the necessary variable and opcode data to them in order to accomplish the assigned Factorial and Summation functions.

In order to execute, the interpreter also requires the public method:

public void InterpretQuads(Quadtable Q, SymbolTable S, boolean TraceOn, String filename);

with the initialized and filled-in Quad table and Symbol table as inputs to the interpreter. Your program must implement the full 'Interpreter' functionality from the ***Intro to Quad Codes*** document, assuming for now that *all numeric operations will use operands which are INTEGER data-types*. Translate the interpreter guidelines from the pseudo-code located there into equivalent Java code. Within the WHILE loop, the interpreter must be able to produce an echo 'trace mode' printout when ***TraceOn*** is true, both to a text file full pathname called ***filename***, and also to the console. The tracing simply indicates the current PC (program counter) and quad data and operands (with SymbolTable name data in <>) that will be executed next, like:

```
PC = 0025: ADD 5 <count>, 8 <2>, 2 <sum>
PC = 0026: BNZ 12
```

Note that this will ultimately be used for debugging and executing the compiler-generated code at the end of the entire project. Code will be provided which implements the above formatting.

Structure of the code

In order to receive full points for this program, it must utilize good Software Engineering practices, have in-code comments as documentation, and be logically structured. Meaningful variable names and other good practices, such as clear comments within the code, and **virtually no 'magic number' literal numeric constants** *(instead of '20', use a symbolic constant,* ***static maxVarLength = 20)*** , are expected to be used throughout.

Turn-In Requirements

Along with the SymbolTable, ReserveTable, and QuadTable Java source files, **on Canvas** turn in the Java ***Interpreter*** class source, implementing the Interpreter interface described above. All source code must be neatly documented and logically arranged. High quality coding standards are implicit in a senior/graduate level course. A sample main Java program will be provided by the instructor, and the output from a the test run will be turned in as .txt files.