CS323 Assignment #3

1. **Problem Statement:**

   In this assignment, we had to implement symbol handling as well as create an instructions table filled with assembly-level instructions on how the program was parsed. The approach chosen to complete the assignment was the RDP one mentioned in the partial solution by Dr. James Choi.

2. **How to use the program:**

   The program can have a text file passed in as a command line argument. In order to call the file, the following command must be issued (assumes python3):

   python3 rdp_compiler.py InputFile.txt

   If no command line argument is given, the input defaults to the *SampleInput3.txt* file, which is included with this turn-in. This allows us to call the program without command line arguments the following way:

   python3 rdp_compiler.py

   The program should then parse the entire file, then generate the instructions table and the symbols table. The program is made such that if an error or exception occurs, the instructions and symbol tables are still printed out, even if they're only partially generated.

3. **Design of the Program**

   To complete this assignment, SyntaxAnalyzer.py was not needed, as the production rules were re-programmed here. The way the program was designed was to hold both stacks, instructions table, symbols table, and a few other variables as global variables to be used in all production rules.

   To start off, the program opens the default text file or the one input by command line argument. It then calls the Lexer program from assignment 1, and parses the entire file into tokens. Finally, it has a main loop that makes sure each token is iterated through. At the heart of the loop, the Statement() production rule is called. From here, all other production rules are called in a similar order as the partial solution to achieve the same results as the output sample files.

   Inside the Statement function, the program detects if its handling a declaration, an assignment, and if clause, or a while loop. All of these were completely implemented. All of the following functions are then used by these subroutines:

   Expression()

   ExpressionPrime()

   Term()

   TermPrime()

   Factor()

Etc. There also exists a Condition() function that handles the conditional statements that drive the ‘if’ clause and the ‘while’ loops. There are also a great amount of helper functions that allow the handling of the instructions table and the symbols table. Each table has a dedicated class with values as self variables that get appended to each table respectively. There is also a monolithic function called create_instruction() that receives a value and an instruction name, and handles the appropriate behavior based on the assembly instruction passed in.

4. Limitations

A simple limitation that arises from this implementation is the complication { } block of code has more than one statement in it. Otherwise, the parser works great, and the { } braces become optional for single statement blocks.

5. Shortcomings

The implementation handles the requested instructions on the prompt. All assembly instructions were individually tested and function as intended. No shortcomings.