CS323 Project 3 Documentation

1. Problem statement

- Part 1: Symbol table handling

  The goal of this part is to implement a symbol table for the simplified Rat23S language. The symbol table should be able to store the following information for each identifier.
  - the lexeme of the identifier
  - the type of the identifier
  - the memory address of the identifier

  The symbol table should be implemented as a hash table. When an identifier is declared, it should be inserted into the symbol table. When an identifier is used, its information should be looked up in the symbol table. If an identifier is not found in the symbol table, an error should be reported.
- Part 2: Generating assembly code

  The goal of this part is to modify the parser to generate assembly code for the simplified Rat23S language. The assembly code should be generated in the following format:
  - each instruction should be stored in an array
  - the instructions should be numbered starting from 1
  - the instructions should be printed out to produce a listing of assembly code
  - the listing should include an array index for each entry so that it serves as a label to JMP to
  - the compiler should also produce a listing of all the identifiers

  The assembly code should be generated for the following simplified Rat23S statements:
  - assignment statements
  - arithmetic statements
  - boolean expressions
  - if statements
  - while statements
  - function calls
  - return statements

  The assembly code should be generated in a way that is efficient and easy to read.

2. **How to use our program**
    1. Goto command line
    2. Type ‘java -jar ICG.jar’
        - **You may need to input the file pathway to the ICG.jar if just the filename does not work.**
        - **Example: ‘java -jar C:\user\downloads\compilerProjectFinal\ICG.jar’**
    3. Once it runs, you will be prompted to input the file destination of the output file.
        - **Make sure the end of the pathway ends with a \.**
        - **Example: C:\user\downloads\compilerProjectFinal**
    4. Then, you will be prompted to input the test case file name. The output will be shown on the command line.
        - **For the file name you may need to include the file pathway if just the filename alone does not work.**
        - **Example: ‘input your file: C:\user\downloads\testCase1.txt’**

3. **Design of our program**

    The design of the program mostly involves editing our existing syntax analyzer. We had to create a symbol table and instruction table where the instruction table can refer to the symbol table in order to create the needed assembly code. To do this we created two classes that can hold the parameters of the symbol table and instruction table. Afterwards, we can create the table using an arraylist that holds the classes as elements. Next, just involves creating a method for adding to the instruction table to be placed in certain methods (rules) so that it can properly generate correct assembly code. Also, we created a couple separate methods that can handle adding symbols to the symbol table for certain Identifiers.

4. **Limitations**

    None.

5. **Shortcomings**

    None.