ICG Documentation

About 2 pages

1. **Problem Statement**

   With the lexical and syntax analyzers constructed, the next step is to construct a symbol table handler and assembly code generator for the simplified version of RAT24S. While the syntax handler handles the grammatical rules, the symbol table and associated error handling handles the semantic rules of the RAT24S grammar. Every identifier declared in the program should be placed in the symbol table and accessed by symbol table handling procedures. Each entry should hold the lexeme and a "memory address" where the identifier is placed. If an identifier is used without being declared, or if an identifier is declared twice, the parser should provide an error message. The parser should also check for type mismatches. Finally, the parser needs to generate assembly code instructions for the simplified RAT24S, which does not allow function definitions or the "real" data type. The assembly instructions should be kept in an array of at least size 1000 and printed out at the end to produce a listing of assembly instructions. The compiler should also produce a listing of all identifiers in the symbol table. A fully functional syntax analyzer with symbol table handling and assembly code generation should be able to parse the entire simplified RAT24S program if it is syntactically and semantically correct.

2. **How to use your program**

   1. Navigate to the project directory.
   2. Open command prompt from the directory.
   3. Type: main3.exe
   4. You will be prompted to enter the name of a source code text file.
      a. Please type the name of the text file in the project directory that holds the RAT24S code (or junk code) that you wish to use.
      b. Example: Type: icg_test_1.txt
   5. You will be prompted to enter another text file name to write the output.
      a. Please either *add a text file yourself* to the project directory for your own test files, *or use the file (jout.txt) provided for you*.
      b. Example: Type: icg_out_1.txt

3. **Design of your program**

   The program will have a main function that initializes the necessary data structures and calls the parser function. The parser function will implement the modified production rules for the simplified RAT24S grammar, excluding function definitions and the "real" type. It will recursively call helper functions corresponding to each production rule.

   Symbol Table Handling:
   Define a global integer variable Memory_Address initialized to 5000.

Use a global 2D array to store the symbol table.
Define void function to insert identifiers into the symbol table using Memory_Address-5000 as the memory location, and increment Memory_Address.
Define a function that returns a string of the memory location if an identifier is in the table and "0" if it isn't.

Generating Assembly Code:
Define a global 2D array for assembly instructions with capacity for at least 1000 instructions.
Define void function to insert instructions into the instruction table using Instruction_Address-1 and incrementing Instruction_Address.
Created an integer vector to use as a stack for the jump0 instructions that need to be backpatched.
Defined a void function to backpatch the jump0 instruction.

Use a global bool variable to determine whether the tables print when called.

4. **Any Limitation**
   There is a limit of size 1000 for the instruction table and a limit of size 100 for the symbol table. Also, if identifiers are longer than 25 characters, it will mess up the printing of the symbol table.

5. **Any shortcomings**
   None