**CS323 Documentation**

About 2 pages

1. **Problem Statement**

*Write a parsing program that is able to read a file containing simplified Rat23S source code and output a file that has two tables: the symbol table containing the identifier and the corresponding memory address, and the instruction table containing the pseudo assembly code for the Rat23S source code.*

1. **How to use your program**

Program can be run without double click lexer.exe

*Instructions for* ***Powershell*** *(Windows)*

Step 1) Open Powershell

Step 2) Navigate to directory containing the lexer.exe

Step 3) Ensure that input files and lexer.exe are in same directory.

Step 4) Type the following command.

./lexer.exe <name of input file>  >  <name of output file>

*Note: Important to include the “>” symbol in between input and output file*

*Instructions for* ***Command Prompt*** *(Windows)*

Step 1) Open Command Prompt

Step 2) Navigate to directory containing the lexer.exe

Step 3) Ensure that input files and lexer.exe are in same directory

Step 4) Type the following command

lexer.exe <name of input file>  >  <name of output file>

*Note: Main difference from Powershell is the lack of the “./” prefix before lexer.exe*

1. **Design of your program**

***Main()***

1. *Get name of input file via command line arguments*
2. *Begin reading from input file*
3. *Remove any UTF8-BOM headers from start of file stream via* ***removebom()***
4. *Call* ***Rat23S()***
5. *Continue reading until EoF*
6. *Print out Symbol Table*
7. *Print out Instruction Table*
8. *Close file stream*

*Instruction and Symbol Table Struct*

*We create a single struct for both the instruction table and the symbol table, however, certain struct members will only be used by each table.*

1. *Initialize current memory address variable*
   1. *Memory address begins at 1*
   2. *Every subsequent instruction will add one to this value*
2. *Initialize symbol address variable*
   1. *Symbol address begins at 5000*
   2. *Every new identifier will add one to this value*
3. *Addr member is a vector of integers representing memory address where instructions are stored*
4. *‘Op’ member is a vector of strings representing operation codes for each instruction*
5. *‘Oprd’ member is array of integers representing operands for each instruction*
6. *‘Ids’ member is vector of strings representing identifiers associated with each instruction*
7. *‘Type’ member is a vector of strings representing the type of identifier*

***Gen\_instr()***

1. *Set address, opcode, and operand for current instruction in Instruction Table*
2. *Increment current address in Instruction Table struct*

***Back\_patch()***

1. *Function takes an integer input ‘jmpAddr’*
2. *Retrieves an address from top of jump stack*
3. *Sets the operand for the instruction at that address to ‘jmpAddr’*
4. *Removes the address from jump stack*

***Get\_addr()***

1. *Function searches for identifier ‘var’ in ‘ids’ member of symbol table struct*
2. *If identifier found, return memory address associated with that identifier.*
3. *If identifier NOT found, print error message.*

***Init\_sym()***

1. *Search for identifier ‘var’ in ‘ids’ member of the symbol table*
2. *If identifier is NOT found, add identifier to ‘ids’ member*
3. *Set associated memory address and type in ‘addr’ and ‘type’ members*
4. *If identifier if already present, do nothing*

***Check\_sym()***

1. *Search for identifier ‘var’ in ‘ids’ member of symbol table*
2. *If identifier is found, print error message that identifier is already declared*
3. *If identifier is not found, do nothing*

***Type\_check()***

1. *Retrieve/pop top two elements off type stack*
2. *Check if types are equal*
   1. *If not, print error message*
3. *If types are booleans, check if operation is allowed*
   1. *If not, print error message*
4. *Clear type stack*
5. **Any Limitation**

*None*

1. **Any shortcomings**

*None*