Kennesaw State University

College of Computing and Software Engineering

Department of Computer Science

4308 Concepts of Programming Languages

Scanner Report #1

Steven Kemp: [skemp13@students.kennesaw.edu](mailto:skemp13@students.kennesaw.edu)

Cody Ziegler: [cziegle4@students.kennesaw.edu](mailto:cziegle4@students.kennesaw.edu)

October 18, 2019

**Initial Problem Statement**

The initial problem of this project is to create a scanner module for an interpreter for a limited version of the Julia language. The scanner needed to intake a text file and scan through it to find each lexeme and assign a proper token to it. Once it had gone through the file it needed to output the lexemes and their respective tokens to another file to be read by the parser module.

**Summary and purpose of the report/assignment**

The purpose of the assignment is to create the first part of the interpreter. The scanner will perform a lexical analysis of the input file and create an output file that contains all the lexemes and tokens needed for the parser to perform its operations. The purpose of the report is to have the students thoroughly examine the work performed as well as reflect upon how it could have been done differently or better.

**Detailed Description of the Solution Used in the Project**

The scanner module consists of several programs that all work in unison to decode the input text file. The ScannerDriver starts the whole program by taking in the input file and sending it to the Scanner class. Currently, the Scanner class only calls the LexicalAnalyzer class and sends the input file as a parameter. The LexicalAnalyzer class will trim the excess white space off of the input and then begins to analyze the file lexeme by lexeme through the Token class. The LexicalAnalyzer will create a Token object that then calls the fromString method which will get the next lexeme in the input using the TokenType class. The TokenType class will determine what token the lexeme is and then return that type to the Token class that then returns the Token object to the LexicalAnalyzer. The LexicalAnalyzer then outputs the lexeme and token to the output file. This repeats until the entire file has been analyzed.

**List of Input Data**

b = 3

a = b + 2

if a < 4

then

print( a + b )

else

print( a )

while a > 3

a = a - 2

end

end

**Results**

Next token is: 5012 Next lexeme is: b

Next token is: 5003 Next lexeme is: =

Next token is: 5001 Next lexeme is: 3

Next token is: 5012 Next lexeme is: a

Next token is: 5003 Next lexeme is: =

Next token is: 5012 Next lexeme is: b

Next token is: 5006 Next lexeme is: +

Next token is: 5001 Next lexeme is: 2

Next token is: 5004 Next lexeme is: if

Next token is: 5012 Next lexeme is: a

Next token is: 5015 Next lexeme is: <

Next token is: 5001 Next lexeme is: 4

Next token is: 5012 Next lexeme is: then

Next token is: 5012 Next lexeme is: print

Next token is: 5013 Next lexeme is: (

Next token is: 5012 Next lexeme is: a

Next token is: 5006 Next lexeme is: +

Next token is: 5012 Next lexeme is: b

Next token is: 5014 Next lexeme is: )

Next token is: 5005 Next lexeme is: else

Next token is: 5012 Next lexeme is: print

Next token is: 5013 Next lexeme is: (

Next token is: 5012 Next lexeme is: a

Next token is: 5014 Next lexeme is: )

Next token is: 5012 Next lexeme is: while

Next token is: 5012 Next lexeme is: a

Next token is: 5016 Next lexeme is: >

Next token is: 5001 Next lexeme is: 3

Next token is: 5012 Next lexeme is: a

Next token is: 5003 Next lexeme is: =

Next token is: 5012 Next lexeme is: a

Next token is: 5008 Next lexeme is: -

Next token is: 5001 Next lexeme is: 2

Next token is: 5012 Next lexeme is: end

Next token is: 5012 Next lexeme is: end

Lexical analysis complete!

**Limitations of the Design of the System**

The above implementation could run into buffer overflow issues if the input file is too large due to the system reading in the entire file all at once. There is also the limitation of using an enum for the token types in that they must be carefully placed in order to avoid misreads.

**How the System Could be Improved or Extended**

The system could be improved by reading only sections of the input file at a time in order to avoid the overflow. The system could also be extended by adding additional variable types or more functions from the original language.

**Conclusion**

Overall we believe the implementation of the scanner module has been a successful one as it performs the task at hand for the provided examples in the discussion post. The implementation of the module also allows for easy integration of the parser portion of the overall project.