CS214 ASSIGNMENT 0

Patrick Nogaj, Alborz Jelvani

October 4th, 2020

Abstract

This document is project assignment 0: Tokenizer for Systems Programming (01:198:214) at Rutgers University - Fall 2020.

1 Setup your Environment

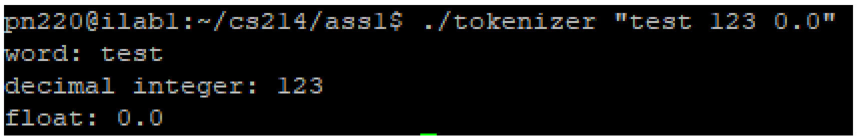
Within the Asst0.tgz file, we have two files that are utilized to run the Tokenizer. We have our source code file, tokenizer.c, which contains the source code required to run the program, and Makefile which allows us to compile and remove the program if needed.

To begin: place Makefile and tokenizer.c in the same directory on the iLab machine. Within terminal, type "*make*" which will generate an executable called 'Tokenizer'.

To clean the document, you can type "*make clean*" in the terminal and this will remove the executable Tokenizer from the directory that the files reside in.

2 Running the Tokenizer

To run the tokenizer, we can type "*./tokenizer arg1*" where arg1 is our input. Any additional arguments will not be tokenized, as per instruction this program is only allowed to tokenize what is passed through argument one. To make an argument contain multiple items in arg1, we can wrap it in quotation marks such as the shown in Figure 1.



```
pn220@ilab1:~/cs214/ass1$ ./tokenizer "test 123 0.0"
word: test
decimal integer: 123
float: 0.0
```

Figure 1: Multiple arguments in arg1

3 Features

The tokenizer is able to figure out the following token types:

word, decimal integer, octal, hex, float, left paranthesis, right paranthesis, left bracket, right bracket, struct member, struct pointer, sizeof, comma, negate, ones compliment, shift right, shift left, bitwise or, bitwise xor, increment, decrement, addition, division, logical or, logical and, conditional true/false, equality, inequality, greater or less than, greater or less than equals to, assignments, plus equals, minus equals, time sequals, divide equals, mode equals, shiftright equals, shiftleft equals, bitiwse and equals, bitwise xor equals, bitwise or equals, address operator, minus operator, multiply operator

In addition, we completed the extra credit therefore, the program will be able to tokenize the additional:

single line comments, multi-line comments, C keywords (such as goto, for, etc)

4 Design Implementation

The design idea behind the Tokenizer was to assess what each token was as we come across it. To accomplish this, we would start at the beginning of our pointer of argv[1]. As we come across something that has not been tokenized yet, we attempt to figure out the possibilities of what it can be. This can be done by setting up if/else statements to decide what criteria must be met to be classified as a token.

For example, if our first unidentified token is an '0', we have mulitple options to explore before classifying it as such. If we see that the index of '0' plus an additional index (one to the right of '0') is an 'x', we know we have a hexadecimal, likewise for any other possible token.

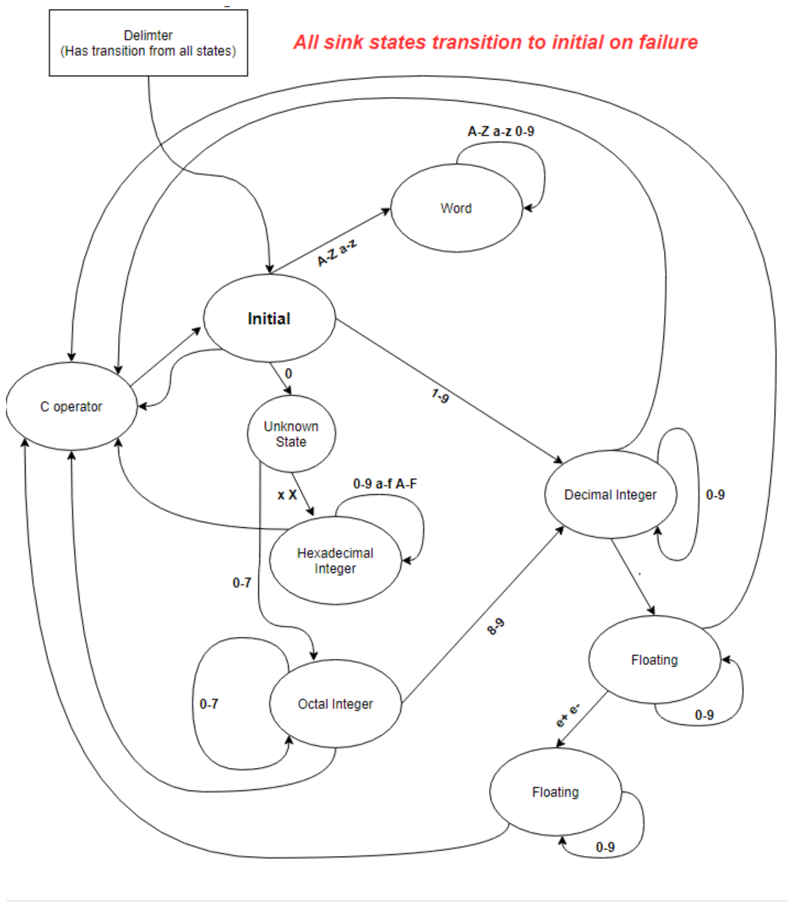


Figure 2: FSM diagram

Once we have figured out the type of token it can be, we send it to a specific method designated for that token type. Within these methods we are finding the length of the token, and returning the length back to our tokenize method. This allows us to figure out when a token begins and ends, thus useful for printing and not going back to any previous characters that were already identified. Returning the length from those methods like *findOctal, findDecimal, etc* allows us to advance our tokenize method by taking the current index in addition with the length of the token to begin classifying the next type of token. This is essential because one requirement to the assignment was not to re-evaluate tokens in the pointer that we already addressed.

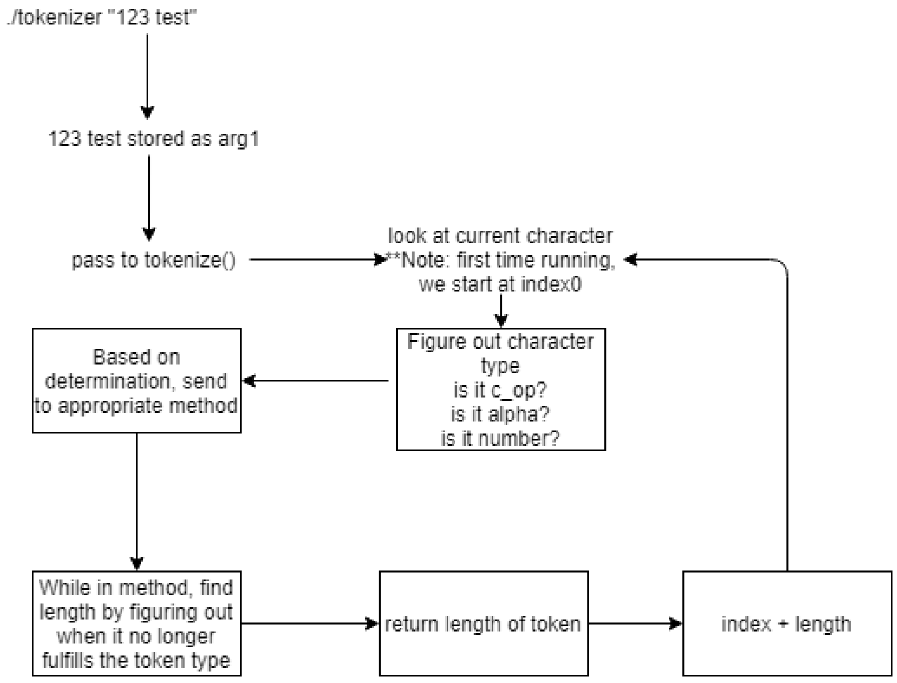

./tokenizer "123 test"
123 test stored as arg1
pass to tokenize()
look at current character **Note: first time running, we start at index0
Figure out character type is it c_op? is it alpha? is it number?
Based on determination, send to appropriate method
While in method, find length by figuring out when it no longer fulfills the token type
return length of token
index + length

Figure 3: Basic concept of flow of program