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.

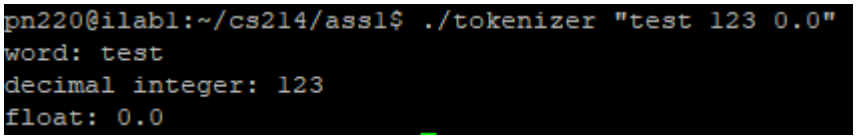

Figure 1: Multiple arguments in arg1

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 our char at index 0, we attempt to figure out the possibilities of what it can be. This is done by passing the current index and argv[1] pointer to a series of functions that are each designed to return the length of the detected token.

For example, if our first character is an '0', we have multiple options to explore before classifying it as 1 token. 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. To figure this out, we pass the argv[1] pointer and index to a series of functions, such as, *findFloat*, *findHex*, *findDecimal*, and even some functions that make no sense, *findCop*. The idea behind this is that, we know some general cases, such as a word must start with an alpha character, but some tokens that can have many forms, such as a C op or C keyword, we just pass the character pointer and index to the function, and it will simply return -1 or 0, indicating token not detected or a token of length 0 detected. Then, by calling every function after each pass of tokenizing, we can choose the results of the function that returns the longest length, hence our usage of a *greedy algorithm*.

After we have determined our longest token type and length, we can print out the type of token, and the token itself, since we have the current index, and the length of the token. Finally, we can increment the current index by the length, and continue our while loop from the top in the *tokenize* function. The while loop keeps doing this task, until it detects the current index is greater than the input string length in argv[1].

This design means that our program does not need to store any tokens in memory. It just compares the length of the token returned by each function, and chooses the longest one. Then, it prints out the token and the string, which is already in memory from the inputted argument.

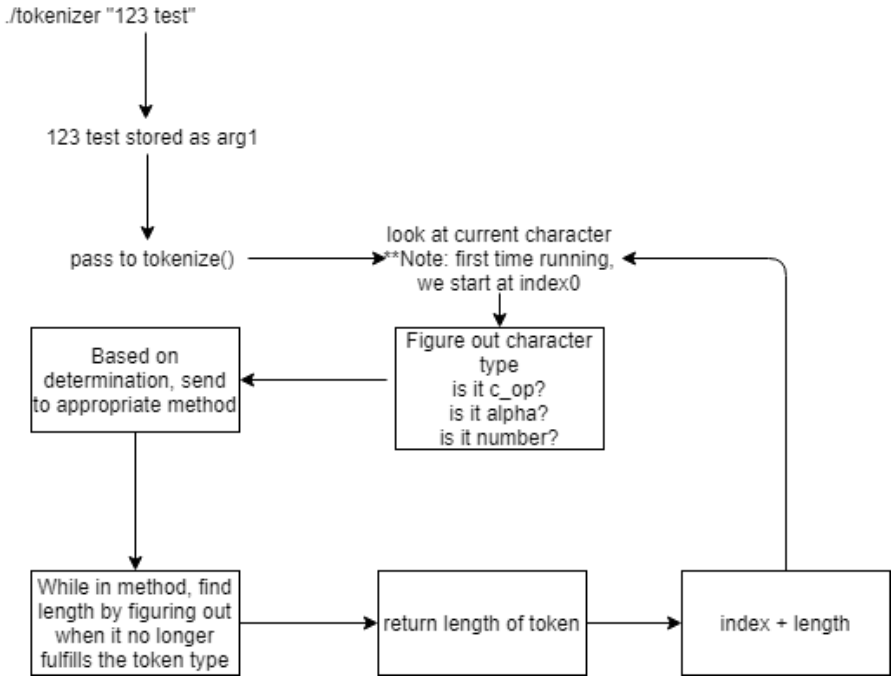

Figure 2: Basic concept of flow of program