Project Title- Asst0: Tokenizer Rutgers CS214 Assignment 0

Introduction:

In this assignment, you will practice programming with C pointers. Much of the pointer manip- ulation will come in the form of operating on C strings. A program that is at the base of every terminal, shell and interpreter is a tokenizer. For this assignment, you will write a tokenizer. A tokenizer is a program that takes a single string of characters of an arbitrary length and breaks them up into types of inputs. For instance, the single string "blah 8 stuff 1.3" contains two 'word'-type inputs, an 'integer'-type input and a 'float'-type input. Your tokenizer's job will be to to take that string and break it up into each type of token, printing out each as it identifies them. You will not know the length of the input string beforehand, so you will not know how much memory to allocate in your code to copy the input string. Since you do not know the length of the input string, you will not know the index of the string's last character. These are two fundamental issues you must consider.

How to run tests:

1. compile the tokenizer.c file in terminal with: gcc -o tokenizer tokenizer.c
2. After compiling, input ./tokenizer "any string you want"
3. The code would tokenize the string, and show how the tokens of the inputted string

Mentionables & How it was designed:

If no string was inputted or more than one string was inputted, then the code would return error. For this code, instead of using switch to categorize the string, the if...else statements were used to tokenize and categorize the string. There were multiple different situations (represented as testcases in the testcases.txt) in which using if...else statements would be more straightforward and specific to conditions very likely to cause bugs, so allowing us read the code from left to right without the need to create substrings from the inputted string for integers and words. Though this code is named tokenizer, we did not actually tokenize the string, instead we identified which substring should be grouped together and printed out with a header. Our code did this by applying the if...else statements, thus we only had to read the inputted string once, and the code would identify tokens and categorize them without the need to tokenize and use for loops/while loops to check if these tokens are correct. The only time in which we formed substrings was with operator substrings (operators from the RefCard.pdf). The operator substrings, unlike words and decimals have a general header, had specific header names, making it more efficient if we form a substring with malloc and check the operator composition of that substring, instead of checking the entire string.