Assignment 0: tokenizer README

Peter Basily

RUID: 169006568

October 4, 2020

About:

This program takes in a single string and tokenizes it based on terms defined as C data types, structure members, keywords, or operators. The program saves the tokens in a linked list and then prints them out once the end of the string is reached.

Performance:

The program iterates through a string of size n. For every iteration through this string, the program performs a constant number of iterations. In the worst, the performance of this program is O(n). I chose to store the tokens in a linked list instead of automatically printing each one as it is created for two reasons. The first is that it would have added one extra step for every token made (the performance would have still stayed O(n)), the second reason is that I think the overall usefulness and modularity of this program benefits from storing the tokens.

Extra features:

I made a few auxiliary functions to increase modularity and save on time writing this program:

The first function is void **addtok(tnode **head, char *dtype, char *instr, int ststr, int len)** where head is the reference to the tokens list, dtype is the data type of the token, instr is the input string (from argv[1]), ststr is the starting point of the token, and len is the length of the token. When called, the function iterates through our token list and copies the token from instr+ststr to the length of the token using strncpy. This works because adding the starting point to the address of the input string gives you the address of start of the token by moving the pointer that amount of bites forward. This also means that I store the data type of the token separately. This is to further help modularity (we can for example create a function that changes the data type of the token from char to its respective data type).

The next function I'd like to point out is **int keywordcheck(char *cstr, char *inpt, int strt)** where cstr is the comparison string, inpt is the input string from argv[1], and strt is the start of the check. It returns 0 if the string created from input+start to the length of the compare string are equal, or !0 otherwise. I use this function to check for keywords and the "sizeof" operator. This function makes it easier to add checks for other words in the future.

Last but not least, we have **void printok(tnode *head)** which creates a temporary point point to head, iterates through the list, and prints the token's data type follow by its data.

This function can be modified to change the output formatting if needed. It also makes it possible to save data types as a single word (for example “word” instead of “word: “).