Aaron Galang
arg206
CS344
Assignment 0: C Tokenizer

Important Features:

- **enum tokenType**: this enum stores all types of tokens, from the common ones to the operators in the reference card. Ex. WORD, HEX, SHIFT_LEFT, ect. It also contains an INVALID type for chars not mentioned in the reference card. The purpose of this enum is to allow for the use of a switch statements(which are faster and neater in large cases) in the code as well as added safety.

- **struct Token**: This struct represents a token in the String. It contains three variables - tokenType type, int start, and int end. Type stores what type of token this is, and the start/end ints describe where in the String the token is. Start is inclusive, and end is inclusive, replicating the substring() method in Java.

- **#define IS macros**: This program makes use of several macros(mainly in getToken()) as a way of cleaning up the code and making the if statements more readable. For instance, the IS_DIGIT(c) macro takes in a char, and returns 1 if it is a digit from 0-9, and returns 0 if not. The IS_WHITE_SPACE macro exists above the main method, but the rest are above the method getToken().

The Program:

- **Token getToken(int start, char *s)**: This method takes in the first index of a token within a String, as well as the String itself, and returns a Token struct with all its variables. From here it initializes a token, setting its start value to start. Then, it checks the first character(from start index), as well as its following and determines the token's tokenType. For chars that are larger than size 1, then we generally use a while loop to iterate the token until we

find its end index. Should the char not be amongst the common types such as WORD and DECIMAL, then it goes into a switch statement containing the operators.
Time Complexity: **O(k)**, where k is the length of the token
Space Complexity: **O(1)**, since it only asks for and runs through the string, rather than making its own. Also, Tokens take in indices rather than its string allowing for minimal space to be taken.

- **void printToken(Token t, char *s)**: This method takes in a Token, as well as its String. First it runs through a switch statement, and stores a string dependent on the tokens tokenType. In then prints out the token type, and runs through the argument from the tokens stored start and end indices.
  Time Complexity: **O(k)**, where k is the length of the token
  Space Complexity: **O(1)**, since the string stored after the switch statement is not based on the number or length of the argument

- **The main method**: This is where the core program happens. First, it checks argc to ensure only two arguments have been entered, the ./tokenizer, and the String. Entering more or fewer will result in the program printing an error warning to add/remove arguments, as well as returning EXIT_FAILURE. Afterwards the actual program starts:
  1. The program uses two variables, int start, and Token tok. Start represents the start of each token as it passes through the program, and tok represents each created token.
  2. It skips over white space characters with a while loop by incrementing start.
  3. A while loop begins, iterating over the entire argument.
     Inside the loop:
     a. Assigns tok to a token made from the method getTok, using start and argv[1] as parameters.
     b. The method printToken() takes in tok and the argument and prints out tok.
     c. It checks whether or not the tok was a structure member, and increments start by one instead of setting it to tok.end, to follow

the reference card on structure members.(the word repeats after structure member)

d. Else, it sets start to tok.end, thereby moving on to the next Token in the string. This works because end is exclusive and start is inclusive.
e. It then removes any whitespaces afterwards.

Overall Time Complexity: **O(n)**, the while loop iterates through the argument for worst case O(n) time(Worst case being when the number of tokens+whitespaces are equal to n, best case being when its a single token for O(1) time), where n is the length of the argument. Since printToken() and getToken() are only the length of each individual token, adding them up will bring a total for O(n) for both of them. 3O(n) = O(n).

Overall Space Complexity: **O(1)**

Instructions:

1. Compile the program with: gcc -o tokenizer tokenizer.c
2. Run the program with: ./tokenizer "Insert argument here"
   Notes:
   a. Ensure to enclose your argument with "" or '', or the program won't take in whitespaces and other stuff, as well as treating the String as several arguments
   b. Invalid characters will show up as invalid tokens. Ex. the input "#" will return the output: INVALID TYPE: "#"