Assignment 0 - C Tokenizer

Karun Kanda (kk951), Junxian Cai (jc2411)

October 4, 2020

1 Name of Program

tokenizer - tokenizes a string input to individual tokens built in the C programming language.

2 Usage

```
./tokenizer <string input>
```

3 Description of Algorithm

This program takes a single string of an arbitrary length and breaks them up into types of tokens, including word, decimal integer, octal integer, hexadecimal integer, floating point, C operator.

First, we take argv[1] and take the length of argv for when we iterate through it later. Then copy the contents of argv[1] into a char* to also use when we read the token. Then before we start to print the tokens out we initialize a a struct variable that will hold data about the token and the enum of the specific token (if its a word, decimal integer, octal integer, etc).

Next we proceed with the actual tokenization where we pass the input string's length, the input itself that was copied from argv[1] and the Token struct that we made into the function:

```
void tokenScanner(int inputlen, char*input, Token currToken);
```

Which will iterate through the input string character by character determining the token type and start to concatenate the data of the token which will used to print the token in the token printer function that is called in tokenScanner(). Now in the tokenScanner() function we are concatenating the string into the Token struct's data field we call the function:

```
Token tokenPrinter(Token t)
```

to perform the actual printing. tokenPrinter takes the Token struct that we created and determine the type name via the enum type that is held in the Token struct. Then prints out the individual token in the form:

```
<token type>: "<token name>"
```

Additionally, the program can recognize the reserved keywords in C as distinct tokens, skip // and /* */ comments, and recognize strings in quotes as single tokens.

4 Unique Features of our Implementation

This program runs fast designed to run in linear time and reads each char only once in most cases. It's implemented with the idea of state machine using switch cases, hence the decision process of the type of a token is straight forward and won't run into irrelevant types.

5 Efficiency Analysis

With the algorithm we implemented to tokenize the string has a worst-case running time of O(n) because the tokens are printed until the null terminator is found and the string is a unknown size which we can consider as a n input size.