Rushit Sanghrajka

Ethan Berroa

CS 214: Programming Assignment 1

Tokenizer in C

Theoretically, a tokenizer is a core systems tool that accepts text, or blocks of code as it's input, and breaks the text up into meaningful elements for the computer to understand, called tokens. The tokens that are generated by the tokenizer are further used for processing and parsing by the compiler.

The tokenizer that we have implemented is a basic version of the tokenizers present in current day compilers, however, we have implemented the core features of a basic tokenizer.

Structure of our Tokenizer:

The tokenizer basically takes in the text, and creates a dynamic Tokenizer struct. This tokenizer struct stores the text in it. We then pass this Tokenizer instance to the getNextToken method. In the getNextToken, we scan through the string to recognize and separate a token from another. The getNextToken returns the token that is found. In order to find tokens that are not separated to each other by a whitespace, we make the following assumptions:

- If a string starts with special symbols but is followed by alphabets or numbers, we break the special symbols into a separate token.
- A sequence of numbers followed by alphabets/ special symbols is still considered as one token.
- Multiple special character symbols adjacent to each other but not mentioned in the C reference card are assumed as malicious. For example, "?>)" would be recognized as malicious, but "+=" would be recognized as "plus equals".
- If a string that started with alphabets encounters a special symbol, it will consider the alphabets as a separate token. For example, in "Bread, butter and cheese" we would split the "Bread" and the comma sign.
- However, in case of alphanumeric tokens with an adjacent special symbol, it would consider everything as one token.

In order to separate these cases, we created an internal FSM structure to separate these tokens before trying to find their type.

Once the token has been returned by getNextToken, we pass that token to getTokenType. getTokenType breaks the token down character by character, and using a finite state machine, it traverses through all the states and returns the final token type of the token.

The main function in the tokenizer.c file simply creates a tokenizer, calls getNextToken to find each token in the input, and calls getTokenType to find out what type of token it is.

Dealing with escape characters:

- As mentioned in the given assignment, the tokenizer that we created gives an error message for the various escape sequences passed to it.
- We check for the presence of escape characters in a string within the getNextToken itself, and print an error incase the given string contains an escape character.

Various test cases and their outputs:

- INPUT: "Hello world! This is CS214"
  OUTPUT:
  Tokenizer created: Hello world! This is CS214
  Token is Hello and token type is WORD
  Token is world and token type is WORD
  Token is ! and token type is exclamation
  Token is This and token type is WORD
  Token is is and token type is WORD
  Token is CS214 and token type is WORD

- INPUT: "34.5 - 12 = ? (we are playing with numbers)"
  OUTPUT:
  Creating Tokenizer..
  Tokenizer created: 34.5 - 12 = ? (we are playing with numbers)
  Token is 34.5 and token type is float
  Token is - and token type is dash
  Token is 12 and token type is digit
  Token is = and token type is equals
  Token is ? and token type is question mark
  Token is (and token type is left round brace
  Token is we and token type is WORD
  Token is are and token type is WORD
  Token is playing and token type is WORD
  Token is with and token type is WORD
  Token is numbers and token type is WORD
  Token is) and token type is right round brace

- INPUT: “0x78 is valid but 0xQW isnt. 0.5e34 is valid but 51.e is not.”
  OUTPUT:
  Token is 0x78 and token type is hexadecimal
  Token is is and token type is WORD
  Token is valid and token type is WORD
  Token is but and token type is WORD
  Token is 0xQW and token type is malicious
  Token is isnt and token type is WORD
  Token is . and token type is period
  Token is 0.5e34 and token type is float
  Token is is and token type is WORD
  Token is valid and token type is WORD
  Token is but and token type is WORD
  Token is 51.e and token type is malicious
  Token is is and token type is WORD
  Token is not and token type is WORD
  Token is . and token type is period

- INPUT: “\n is an escape character”

OUTPUT: error [0x0A]

- INPUT: i >> 2
  OUTPUT:
  Creating Tokenizer..
  Tokenizer created:i >> 2
  Token is i and token type is WORD
  Token is >> and token type is right shift
  Token is 2 and token type is digit