Lexer

Kit Lao

June 12, 2020

Goal

Convert every symbol within the language into a token. The input should be a string read from either a file or standard input, and the output should be a list of tokens.

The lexer should be able to identify things like operators, brackets, parenthesis, keywords, numbers, identifiers, comments and other symbols in the language.

If the lexer comes across symbols that are not recognized in the language, or illegal identifiers, it should propagate an error.

Important Stuff

So what's a token?

It's essentially just an object that contains the symbol or string from the input text, and the type of token. The values are usually just going to be strings like the identifier name, keywords for initializing a for loop. Tokens for numbers would have the type either integer of float.

So what are the types of tokens?

Each symbol from the language has a unique type. They all start with the abbreviation "Token Type" as TT, followed by an abbreviation of what symbol it is. For example, the token type for the left curly bracket, "{" is "TT_L_C_BRACK", and the token type for the equals sign "=" is TT_EQ.

Some extra stuff

So I tagged a starting and ending position for each token, which is useful for handling errors. I explain more of that in the error handling section.

Implementation

So this is like pretty straight forward. I just wrote an algorithm that traverses through the text and considers each case token type case. What happens if it comes across an addition symbol? Well then create a token for the addition symbol and store it. What about a number token? Same thing, create a token for the number and store it. All this is done in 'generate_tokens'.

For symbols that have multiple characters like identifiers, numbers, keywords, or just certain operators, their tokens are created by using helper function to parse the particular sub-sequence. This is where the methods 'generate_number', 'generate_word', and 'generate_compare comes' in.

A good strategy to keep track of where the pointer for the lexer is to make the pointer global, and use a method to dynamically move the pointer around. This is done using the 'advance' method.