FLCD - Scanner Documentation

Source code repository: https://github.com/914-marcu-paul/FLCD-LAB3

The scanner class has several array fields:
- a separators field;
- an operators field;
- and, finally, a reserved words field.

These cannot be modified after the class object's initialization and they contain the respective particles found in the token.in file.

The methods included in the scanner class are get_string_token(line, index), is_part_of_operator(char), get_operator_token(line, index), tokenize(line), is_identifier(token) and is_constant(token).

The get_string_token(line, index) method is utilized in the tokenization method, when an apostrophe (string indicator) is found. At that point, the method is called and iterates through the line's characters from the given index until it finds a second apostrophe, marking the end of the ASCII string.

The is_part_of_operator(char) method is also utilized in the tokenization method to discriminate against non-operator-related characters and those that are part of operators. In the case wherein it's found that the current character conforms to the conditions, the get_operator_token(line, index) method will be called, which acts much in the same way as get_string_token(line, index), and iterates through the line until it finds a character that isn't part of the operator character pool, at which point it isolates the token.

The tokenize(line) method is the main dynamo utilized to separate the groups of characters into tokens; it iterates through a line's characters one by one, slicing them if a character from the separators field is found. Additionally, it has branches for the special cases outlined above.

The is_identifier(token) and is_constant(token) methods are both quite similar, using simple regex checks to verify if a token can be categorized into either an identifier or a constant (integer or string) based on its morphology.

The PIF (program internal form) contains the specific token (if it is a reserved token) or const/id (if it is a constant or identifier) and, in the former case, a (-1, -1) position marker, while in the latter case it contains the address at which it is found in the symbol table.

At runtime, the program scans through each of the code file's lines, one by one, calling the tokenizer, after which it attempts to categorize the tokens using the methods detailed in the scanner. If it is at any point unsuccessful, then we know that we've encountered a lexical error, and it is flagged as such in the scanner's console output.