https://github.com/cristiifrim/Formal-Languages-and-Compiler-Design/tree/master/Lab3

For the Symbol Table I implemented a Hash Table due to the efficiency of the operations.

I have two symbol tables, one representing the constants and the other representing the identifiers.

The Get method returns the value associated with a key in a O(1) complexity (best case scenario) or O(n),

where n is the length of the list in the hashtable.

The Add method adds a symbol to the table. This is usually doine in O(1)

The GetPair method returns a key-value pair based on the given key in a O(1) complexity (best case

scenario) or O(n), where n is the length of the list in the hashtable.

Program Internal Form (PIF)

The PIF is maintained as a straightforward list containing pairs in the form of (token, position), where the position is represented by another pair indicating the token's position in the Symbol Table. For operators, separators, and reserved words, the position value is consistently set to (-1, -1) since these are not stored in the ST.

Tokenization

The tokenization algorithm iterates through each character on a line, determining whether the current sequence is part of an operator, a separator, the beginning of a string, or is constructing a constant or identifier. Subsequently, the identified tokens are appended to a list, which is then returned.

Scanning

The scanning algorithm divides each line of the program into tokens, and for each token, it follows the specified actions. For constants or identifiers, it looks up their position in the ST; for operators, separators, or reserved words, the position is set to (-1, -1). Additionally, if the token is a constant or identifier, it is added to the PIF with the code "const" or "id," respectively.

In the case where the token does not fall into any of the categories mentioned above, indicating a lexical error in that line, the error is appended to the message.