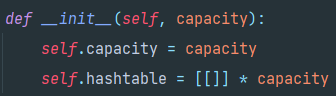
<https://github.com/VieruTudor/Formal-Languages-and-Compiler-Design>

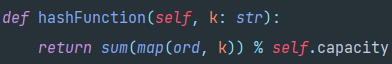
Symbol Table

a. unique for identifiers and constants (create one instance of ST)

The symbol table was implemented using a Hash Table in the form of a list of lists.



At a given position determined by the hash function (represented below), the list will contain all the identifiers/constants that have the same value within the hash function.

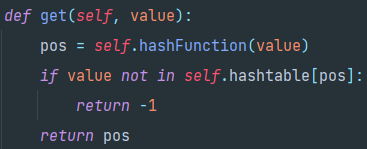


The hash function will compute the sum of ASCII characters modulo the capacity (so it stays within the range of the hash table). The hash function isn’t perfect, since an example such as :   
“ab” -> 97 + 98 = 185

“ba” -> 98 + 97 = 185

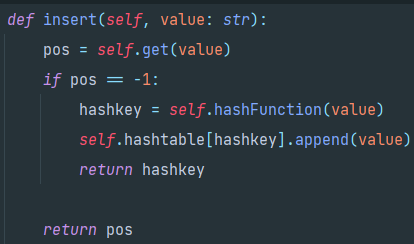
would represent a collision, but it is good enough to provide a quite decent lookup time.

**Lookup(search):**



We compute the position with the hash function. If the value is not in the list at the “pos” position, it will return -1, otherwise, it will return the position.

**Insert:**



First, we try to get the position of the value in the hash table. If the value cannot be found, we insert it at the position computed by the hash function and return the position. Otherwise, we return -1.

By using a hash table as the representation of the symbol table, we get the following complexities :

* Lookup :
  + Average : O(1)
  + Worst case : O(n)
* Insert :
  + Average : O(1)
  + Worst case : O(n)
  + The insert complexity is influenced mostly by the lookup operation

Scanner

The scanner has 3 lists of reserved tokens (separators, reserved words and operators). The last two are parsed at the initialization time.



The tokenizer method reads character by character from a line and has multiple cases that it takes into account :

* If the character is a quote(‘), then it looks for the second quote and the token between them, meaning that it may have found a char constant
* If the character is a doublequote(“), then it looks for the second doublequote and the token between them, meaning that it may have found a string constant
* If the current character may be in a compound operator, it looks for the rest of it and gets the token (i.e. ->, >=, <=, etc.)
* If the current character is a separator, then it moves on
* Otherwise, it has found an identifier and or a numerical constant and adds it to the token list

PIF

The Program Internal Form is built as a list of tuples and when scanning the program, line by line

and is composed as followed:

* If the token is a keyword, operator or separator (except whitespace), add to PIF the tuple (token, -1)
* If the token is an identifier, we add the token to the symbol table and add to PIF the tuple(“identifier”, pos)
* If the token is a constant, we add the token to the symbol table and add to PIF the tuple(“constant”, pos)
* Otherwise, print that it has an error and the line