Github: https://github.com/pukoarmin/Formal-Langauges-and-Compiler-Design.git

Scanner:
A set of functions that serve the purpose of the scanner's functionalities

isPartOfOperator(char): Boolean => Returns True if the char is part of an operator, False otherwise
isEscapedQuote(line, index): Boolean => Returns True if the string on the line at index is an escaped quote or not
getStringToken(line, index): String => Returns the string and the end index of the token starting from index on line
getOperatorToken(line, index): String => Returns the string and the end index of the operator token starting from index on line
tokenGenerator(line, separators): String => Returns the generated tokens form line in regards to the set separators
isIdentifier(token): Boolean => Returns True if the token is an identifier, False otherwise
isConstant(token): Boolean => Returns True if the token is a constant, False otherwise
isSystemToken(previous_token, token, lineNo) => Returns True if the token is a separators, operator or reserved keyword, throws exception if it is a reserved keyword used as identifier and False otherwise

ProgramInternalForm:
A class that contains the pif content and the methods required to operate
- content: List => Stores the PIF's content

+ add(token, Integer): void => Adds the token with the corresponding id to the content list

In main, we use these methods and class as follows:
1. We read line by line
2. For every identified token on that line we check if it is a system token, an identifier or a constant
Note that the PIF table stores the operators and separators with the id -1. If it so happens that the Hash Table inside the Symbol Table had conflicts and multiple nodes are stored on the same index, the pif contains as the token's id a list of the hashed index and the index of the element inside the node chain
2.1 If it is, then it is added into the symbol table then into the pif
2.2 If it is not, then an Unidentified token exception is raised
3. We print the two tables