Github link:

→https://github.com/Badetto/FLCD_Lab2

Scanner

Description: Is a lexical scanner that iterated through the given code and identifies the “PIF” and “ST” tables if the program is lexically corect, otherwise it will print the line where there are lexical errors and also the first lexical error encountered.

→**Private variables:** The names are pretty suggestive, we have a symbol table, all the reserved words, the operators, the separators and the line we are currently at inside the program’s code.

→**IsIdentifier(token: string): bool**: It returns true if the current “token” is an identifier, otherwise it will return false.

→**IsNumberConstant (token: string): bool**: It returns true if the current “token” is a number, otherwise it will return false.

→**ProcessStringIdentifier (combinedToken: string)**: **string**: Places a ‘ ” ’ at the end of the string constant and returns the newly created string.

→**ProcessStringToken (combinedToken: string)**: Checks if the string constant is already inside the symbol table, if not it places it inside the symbol table and also “adds” it to the PIF table.

→**ProcessIdentifierToken(combinedToken: string)**: Checks if the identifier is already inside the symbol table, if not it places it inside the symbol table and also “adds” it to the PIF table.

→ **ProcessNumberToken(combinedToken: string)**: Checks if the number constant is already inside the symbol table, if not it places it inside the symbol table and also “adds” it to the PIF table.

→**TokenizeLine(line: string):** Breaks each code line in multiple tokens based on the private parameters we initialised inside the constructor and tries to clasify them. If the token is either a number constant/ a string constant/ an identifier it adds them inside the symbol table, if the token represents either a reserved work/ separator/ operator it just "adds" them to the PIF table and if it finds a lexical error (either a variable was declared wrongly or if the quotes are opened but never closed) it throws a ScannerException with the line where we are currently at and the invalid token.

→ **Tokenize(sourceCode: string):** Breaks the initial code (sourceCode) in lines in iterates through them keeping count of the line number where we are currently at.