Lexical Scanner

Hutupasu Stefana - 933/2

Link to git repo

Implementation:

- treats a program as a set of characters, and parses it, attempting to detect and classify tokens
- for any token identifying attempt, the string considered is the program code as a raw string, but at an index which is updated accordingly as the program processes tokens (the part of the program which has already been split into tokens is no longer considered, except for previously detected tokens in special cases)
- special cases in which remembering the last found token include, for example, the moment when a string might look and have the structure of an identifier and the scanner might want to just insert it into the Symbol Table, but it could actually just be random characters floating in the program, so the scanner has to check for the last token to have been one representing a type (num, char etc.) in order to be able to insert it in the ST for the first time

Operations:

- skipWhiteSpaces()
  - skips blank characters and updates the index accordingly
  - returns: true if such a skip happened, false otherwise

- skipComment()
  - skips the current line if a comment-marking character is found (in our case, “$”) and updates the index accordingly
  - returns: true if such a skip happened, false otherwise

- stringConstant()
  - checks if current string matches the structure of a string constant, delimited by “”
  - returns true if a match is found, false otherwise (if invalid characters are found or if the quotes are not closed, a lexical error is raised

- intConstant()
  - checks if current string matches the structure of a int constant; also checks that last token was one from the token list, otherwise it would be just a floating number
  - returns true if a match is found, false otherwise

- tokenFromList()
    - checks if current string matches any token from tokens.in (read and print only count if next character is “(“; if last token is “fun”, for function declaration, the current token won’t be considered as one from the list even though it might match, because an identifier is expected)
    - returns true if a match is found, false otherwise

- isFun(string S)
    - checks if given string is “fun”

- isIO(string S)
    - checks if given string is defining an IO token (read or print)

- identifier()
    - checks if current string matches the structure of an identifier (even if it does but it isn’t in the ST yet, if last token was not a type token, then current string won’t be considered for an identifier, as it was not previously declared)
    - returns true if a match is found, false otherwise

- isTypeToken(string S)
    - checks if the given string matches a token representing a type (int, num, char, fun etc.)

- nextToken()
    - skips whitespaces and comments and tests current string that it either matches a constant, an identifier or a token from the list