Assignment CS.211

Name: Abdelkailk Haggag Sayed ID:200049114

Line 1: Includes the standard I\O library

Line 2: Includes the character type library

Line 3: comment

Line 4: comment

Line 5: Stores the character

Line 6: A character array to store the current lexeme

Line 7: Stores the next character read from input

Line 8: Holds the length of the current lexeme

Line 9: Stores the token value assigned to the current lexeme

Line 10: Stores the next token to be processed

Line 11: in\_fp is a file pointer for input \ fopen() is used to open a file

Declarations functions

Line1: Function prototype for adding a character to lexeme.

Line2: Function prototype for reading the next character from input.

Line 3: Function prototype to skip whitespace characters.

Line4: Function prototype for the lexical analyzer function that processes input and returns tokens.

Character classes

LETTER (O): Represents alphabetic characters (A-Z or a-z)

DIGIT (1): Represents numeric characters (0-9)

UNKNOWN (99): Represents characters that do not fall into LETTER or DIGIT

categories.

Token codes

Line 1: Integer literals

Line2: Identifiers

Line3: Assignment operator (=)

line4: Addition operator (+)

Line 5: Subtraction operator (-)

Line 6: Multiplication operator (\*)

Line 7: Division operator (/)

Line 8: Left parenthesis ( ( )

Line 9: Right parenthesis ( ) )

Main

Line1: If condition (try to open file (front.in)\ If condition (fopen) fails return 0

Line2:if file don’t open that is the message print

Line3:if file open successfully call function getchar();

Line 4: that repeatedly calls lex();\\The loop runs until nextToken equals EOF (End of File)

Line5:comment

Line6: Function declaration\ returns an int and takes a char parameter.

Line7: A **switch statement** to check what (ch) is.

Line8: case 1 If ch is '('

Line9: addChar(); (probably adds the character to a list or buffer).

Line10: nextToken = LEFT\_PAREN; (assigns a token that represents a left parenthesis).

Line11: break; exits the switch.

Line12:case2 If ch is ')'

Line 13: addChar(); (probably adds the character to a list or buffer).

Line14: nextToken = Right\_PAREN; (assigns a token that represents a right parenthesis).

Line15: break; exits the switch

Line 15: If ch is '+', assign the ADD\_OP token.

Line 16: If ch is '-', assign the SUB\_OP token.

Line17: If ch is '\*', assign the MULT\_OP token.

Line18: If ch is '/', assign the DIV\_OP token.

Line 19: If ch **is not recognized**, it still calls addChar(); but sets nextToken = EOF (End of File).

Line20: The function **returns** the assigned token.

Comment (function to add nextChar to lexeme)

1. Declares a function named addChar() that **does not return anything (void)**.
2. **Checks if lexLen is within limits** (max 98 characters).
3. Adds nextChar to lexeme at position lexLen, then increases lexLen by 1
4. Sets the next position to 0, likely marking the end of the string.
5. If lexLen > 98, prints an error message.
6. **Comment**: Describes getChar()—it reads the next character and determines if it's a letter, digit, or unknown.
7. Declares getChar() function (returns nothing).
8. Reads the next character from the file in\_fp and stores it in nextChar\**Reads the** next character from the file in\_fp and stores it in nextChar.
9. If nextChar is a letter, assign LETTER to charClass.
10. If nextChar is a **digit**, assign DIGIT to charClass.
11. Otherwise, mark it as UNKNOWN.
12. If EOFisreached, set charClass = EOF.
13. **Comment**: This function skips spaces and gets the next meaningful character.
14. Declares getNonBlank() (returns nothing).
15. If nextChar is **whitespace (' ', '\t', '\n')**, call getChar() to skip it.
16. **Comment**: lex() is a simple lexical analyzer for arithmetic expressions.
17. Declares lex() function that **returns an integer** (likely a token).
18. ResetslexLento0 (start fresh for a new lexeme).
19. Calls getNonBlank() to skip spaces.
20. A switch statement based on charClass (determined by getChar() earlier).
21. If charClass is LETTER, we found an **identifier** (variable name).
22. Adds the first letter to lexeme.
23. Reads the next character.
24. Continues reading **letters and digits**
25. Adds each character to lexeme and **reads the next one**.
26. **Sets nextToken to IDENT** (indicating that an identifier, like a variable name, was found).
27. **break; exits the switch statement**.
28. If charClass is DIGIT, the input starts with a **number**.
29. Adds the **first digit** to lexeme.
30. Reads the **next character**.
31. **Continues adding digits** as long as the input contains numbers (e.g., 47, 1234).
32. Reads **each digit one by one**.
33. Marks nextToken as INT\_LIT (integer literal).
34. Breaksoutoftheswitchstatement.
35. If charClass is UNKNOWN, the character is **not a letter or a digit**.
36. It could be an **operator (+, -, \*, /) or a parenthesis ((, ))**.
37. Calls lookup(nextChar) **to classify the symbol** (+, -, \*, /, (, )).
38. Reads the **next character**
39. If charClass is EOF, the end of the file is reached.
40. Marks nextToken as EOF to signal the end of input.
41. Stores "EOF" as the lexeme (used for debugging or display).
42. Exits the switch statement.
43. Prints the detected token and lexeme.
44. Returns nextToken to the main program.
45. Closes the lex() function.

Example for lexical analysis

1. The lexical analyzer breaks it into tokens and lexemes:
2. 10 represents an integer literal **(INT\_LIT)**. The number 47 was detected.
3. 26 represents a right parenthesis (RIGHT\_PAREN**)**. The ) was detected.
4. 24 represents a division operator (DIV\_OP). The / was detected.
5. 11 represents an identifier **(**IDENT**)**, meaning a variable name. The word total was detected.
6. (-1 represents the End of File (EOF), meaning no more input is left.)