Exercise : No -1

Design a lexical analyzer for a simplified programming language that includes, arrays , strings , conditional statements (if-else) with comparison operators , loops (while and for) , user defined types (structs) , ignore single-line and multi-line comments . Your task is to implement a lexical analyzer that reads the input source code from the file then identifies and categorizes tokens (Keywords, Identifiers, Constants, Operators, Delimiters), and handles nested constructs like nested functions, loops, and conditional statements.

INPUT FORMAT :

```
#include <stdio.h>
struct Point {
    int x;
    int y;
};
int main() {
    int numbers[] = {1, 2, 3, 4, 5};
    for (int i = 0; i < 5; i++) {
        if (numbers[i] % 2 == 0) {
            printf("Even: %d\n", numbers[i]);
        } else {
            printf("Odd: %d\n", numbers[i]);
        }
    }
    return 0;
}
```

Output :

```
Type: KEYWORD, Lexeme: #include
Type: DELIMITER, Lexeme: <
Type: IDENTIFIER, Lexeme: stdio.h
Type: DELIMITER, Lexeme: >
Type: KEYWORD, Lexeme: struct
Type: IDENTIFIER, Lexeme: Point
Type: DELIMITER, Lexeme: {
Type: KEYWORD, Lexeme: int
Type: IDENTIFIER, Lexeme: x
Type: DELIMITER, Lexeme: ;
Type: KEYWORD, Lexeme: int
Type: IDENTIFIER, Lexeme: y
Type: DELIMITER, Lexeme: ;
Type: DELIMITER, Lexeme: };
Type: KEYWORD, Lexeme: int
Type: KEYWORD, Lexeme: main
Type: DELIMITER, Lexeme: (
Type: DELIMITER, Lexeme: )
Type: DELIMITER, Lexeme: {
Type: KEYWORD, Lexeme: int
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMITER, Lexeme: [
Type: IDENTIFIER, Lexeme: i
Type: OPERATOR, Lexeme: <
Type: CONSTANT, Lexeme: 5
Type: DELIMITER, Lexeme: ;
Type: IDENTIFIER, Lexeme: i
Type: OPERATOR, Lexeme: ++
Type: DELIMITER, Lexeme: )
Type: DELIMITER, Lexeme: {
Type: KEYWORD, Lexeme: if
Type: DELIMITER, Lexeme: (
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMITER, Lexeme: [
Type: IDENTIFIER, Lexeme: i
Type: DELIMITER, Lexeme: ]
Type: OPERATOR, Lexeme: %
Type: CONSTANT, Lexeme: 2
Type: OPERATOR, Lexeme: ==
Type: CONSTANT, Lexeme: 0
Type: DELIMITER, Lexeme: )
Type: DELIMITER, Lexeme: {
Type: IDENTIFIER, Lexeme: printf
Type: DELIMITER, Lexeme: (
Type: STRING_LITERAL, Lexeme: "Even: %d\n"
Type: DELIMITER, Lexeme: ,
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMITER, Lexeme: [
Type: IDENTIFIER, Lexeme: i
Type: DELIMITER, Lexeme: ]
Type: DELIMITER, Lexeme: )
Type: DELIMITER, Lexeme: ;
Type: DELIMITER, Lexeme: }
Type: KEYWORD, Lexeme: else
Type: DELIMITER, Lexeme: {
Type: CONSTANT, Lexeme: 5
Type: DELIMITER, Lexeme: ]
Type: OPERATOR, Lexeme: =
Type: DELIMITER, Lexeme: {
Type: CONSTANT, Lexeme: 1
Type: DELIMITER, Lexeme: ,
Type: CONSTANT, Lexeme: 2
Type: DELIMITER, Lexeme: ,
Type: CONSTANT, Lexeme: 3
Type: DELIMITER, Lexeme: ,
Type: CONSTANT, Lexeme: 4
Type: DELIMITER, Lexeme: ,
Type: CONSTANT, Lexeme: 5
Type: DELIMITER, Lexeme: }
Type: DELIMITER, Lexeme: ;
Type: KEYWORD, Lexeme: for
Type: DELIMITER, Lexeme: (
Type: KEYWORD, Lexeme: int
Type: IDENTIFIER, Lexeme: i
Type: OPERATOR, Lexeme: =
Type: CONSTANT, Lexeme: 0
Type: DELIMITER, Lexeme: ;
Type: DELIMITER, Lexeme: (
Type: STRING_LITERAL, Lexeme: "Odd: %d\n"
Type: DELIMITER, Lexeme: ,
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMITER, Lexeme: [
Type: IDENTIFIER, Lexeme: i
Type: DELIMITER, Lexeme: ]
Type: DELIMITER, Lexeme: )
Type: DELIMITER, Lexeme: ;
Type: DELIMITER, Lexeme: }
Type: DELIMITER, Lexeme: }
Type: KEYWORD, Lexeme: return
Type: CONSTANT, Lexeme: 0
Type: DELIMITER, Lexeme: ;
Type: DELIMITER, Lexeme: }
Type: IDENTIFIER, Lexeme: printf
Type: END_OF_FILE, Lexeme:
```