Student Name	Swanand Garge
PRN No	2280030433
Roll No	39
Program	Computer Engg.
Year	Third Year
Division	D (D2S)
Subject	Systems Programming (BTECCE22504)
Assignment No	6

- Design Lexical analyzer for the subset of "C" Language. Accept input from file.

  Output : Line No, Lexeme , Token , Token_Value.

- Also implement any one error checking. Submit single .pdf file with input C program Token listing and source code in sequence.

INPUT CODE :-

```c
int main() {
    int x = 10;
    float y = 20.5;
    if (x < y) {
        return x;
    } else {
        return y;
    }
}
```

OUTPUT :-

```
Line  Lexeme          Token Type      Token Value
-----------------------------------------------------------
1     int             KEYWORD         1
1     main            IDENTIFIER      1
1     (               PUNCTUATOR      1
1     )               PUNCTUATOR      2
1     {               PUNCTUATOR      3
2     int             KEYWORD         2
2     x               IDENTIFIER      2
2     =               OPERATOR        1
2     20              NUMBER          1
2     ;               PUNCTUATOR      4
2     float           KEYWORD         3
2     y               IDENTIFIER      3
2     =               OPERATOR        2
2     39.20           NUMBER          2
2     ;               PUNCTUATOR      5
3     if              KEYWORD         4
3     (               PUNCTUATOR      6
3     x               IDENTIFIER      4
3     y               IDENTIFIER      5
3     )               PUNCTUATOR      7
3     {               PUNCTUATOR      8
4     return          KEYWORD         5
4     x               IDENTIFIER      6
4     ;               PUNCTUATOR      9
5     }               PUNCTUATOR      10
5     else            KEYWORD         6
5     {               PUNCTUATOR      11
6     return          KEYWORD         7
6     y               IDENTIFIER      7
6     ;               PUNCTUATOR      12
7     }               PUNCTUATOR      13
8     }               PUNCTUATOR      14
PS D:\PROGRAMMING\PYTHON\DWDM\SP>
```

CODE :-

```
import re

# Define token patterns using regular expressions
TOKEN_PATTERNS = [
    ("KEYWORD", r"\b(int|float|if|else|while|return)\b"),
    ("IDENTIFIER", r"\b[a-zA-Z_][a-zA-Z0-9_]*\b"),
    ("NUMBER", r"\b\d+(\.\d+)?\b"),  # Integer or Float
    ("OPERATOR", r"[+\-*/%=]"),
    ("PUNCTUATOR", r"[{}();,]"),
    ("NEWLINE", r"\n"),
    ("SKIP", r"[ \t]+"),  # Spaces and tabs
    ("MISMATCH", r"."),  # Any unrecognized character
]

# Arrays for storing tokens by type
keywords = []
identifiers = []
numbers = []
operators = []
punctuators = []

class Token:
    def __init__(self, line_number, lexeme, type, token_value):
        self.line_number = line_number
        self.lexeme = lexeme
        self.type = type
        self.token_value = token_value

def analyze_file(filename):
    tokens = []
    with open(filename, 'r') as file:
        line_number = 1
        for line in file:
            tokens.extend(tokenize_line(line, line_number))
            line_number += 1
    return tokens

def tokenize_line(line, line_number):
    tokens = []
    matcher = build_matcher(line)

    for match in matcher:
        for token_type, _ in TOKEN_PATTERNS:
            lexeme = match.group(token_type) if token_type in match.groupdict()
else None
            if lexeme is not None:
                if token_type == "NEWLINE" or token_type == "SKIP":
                    # Ignore newlines and spaces/tabs
                    pass
                elif token_type == "MISMATCH":
                    print(f"Error: Unexpected character '{lexeme}' on line
{line_number}")
```

```
                else:
                    token_value = add_to_specific_array(token_type, lexeme)
                    tokens.append(Token(line_number, lexeme, token_type,
token_value))
                break
    return tokens

def add_to_specific_array(token_type, lexeme):
    if token_type == "KEYWORD":
        target_list = keywords
    elif token_type == "IDENTIFIER":
        target_list = identifiers
    elif token_type == "NUMBER":
        target_list = numbers
    elif token_type == "OPERATOR":
        target_list = operators
    elif token_type == "PUNCTUATOR":
        target_list = punctuators
    else:
        raise ValueError(f"Unexpected token type: {token_type}")

    target_list.append(lexeme)
    return len(target_list)  # Return 1-based index within the array

def build_matcher(line):
    regex = "|".join(f"(?P<{type}>{pattern})" for type, pattern in TOKEN_PATTERNS)
    return re.finditer(regex, line)

def print_tokens(tokens):
    print(f"{'Line':<5} {'Lexeme':<15} {'Token Type':<15} {'Token Value':<15}")
    print("-" * 60)
    for token in tokens:
        print(f"{token.line_number:<5} {token.lexeme:<15} {token.type:<15}
{token.token_value:<15}")

if __name__ == "__main__":
    # Specify the input file here
    filename = "input.c"  # Change this to your input file's name

    try:
        tokens = analyze_file(filename)
        print_tokens(tokens)
    except IOError as e:
        print(f"Error reading file: {e}")
```