NAME – SAJAL BRAHMA

REGISTER NUMBER – 21BPS1045

Design a lexical analyser 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 analyser 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.

PROGRAM CODE –

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main() {
    FILE *file;
    char line[100];
    char *token;
    int in_string = 0;

    file = fopen("add.cpp", "r");

    if (file == NULL) {
        printf("File not found.\n");
```

```
    return 1;
}

while (fgets(line, sizeof(line), file))
    {
    token = strtok(line, " \t\n");

    while (token != NULL)
            {
        if (in_string == 1)
                    {
            printf("Type: STRING_LITERAL, Lexeme: %s\n", token);
            if (strchr(token, '"') != NULL)
                        {
                in_string = 0;
            }
        }
                    else if (strcmp(token, "#include") == 0 || strcmp(token, "int") == 0 || strcmp(token, "float") == 0 ||
                strcmp(token, "struct") == 0 || strcmp(token, "double") == 0 || strcmp(token, "char") == 0 ||
                strcmp(token, "if") == 0 || strcmp(token, "else") == 0 || strcmp(token, "for") == 0 ||
```

```
        strcmp(token, "while") == 0 || strcmp(token, "main") == 0)
                {
    printf("Type: KEYWORD, Lexeme: %s\n", token);
}
                else if (strchr("<>{}[](),;/", token[0]) != NULL)
                {
    printf("Type: DELIMETER, Lexeme: %s\n", token);
} else if (strchr("+-=.*%", token[0]) != NULL)
                {
    printf("Type: OPERATOR, Lexeme: %s\n", token);
}
                else if (isdigit(token[0]))
                {
    printf("Type: CONSTANT, Lexeme: %s\n", token);
}
                else if (isalpha(token[0]))
                {
    printf("Type: IDENTIFIER, Lexeme: %s\n", token);
}
                else if (strchr(token, "") != NULL)
```

```
                {
            printf("Type: STRING_LITERAL, Lexeme: %s\n", token);
            if (strchr(token, "") == NULL)
                        {
                in_string = 1;
            }
        }

        token = strtok(NULL, " \t\n");
    }
  }

  fclose(file);
  printf("Type: END_OF_FILE, Lexeme:");
  return 0;
}
```

OUTPUT –

```
Type: KEYWORD, Lexeme: #include
Type: DELIMETER, Lexeme: <
Type: IDENTIFIER, Lexeme: stdio.h
Type: DELIMETER, Lexeme: >
Type: KEYWORD, Lexeme: struct
Type: IDENTIFIER, Lexeme: Point
Type: DELIMETER, Lexeme: {
Type: KEYWORD, Lexeme: int
Type: IDENTIFIER, Lexeme: x
Type: DELIMETER, Lexeme: ;
Type: KEYWORD, Lexeme: int
Type: IDENTIFIER, Lexeme: y
Type: DELIMETER, Lexeme: ;
Type: DELIMETER, Lexeme: }
Type: DELIMETER, Lexeme: ;
Type: KEYWORD, Lexeme: int
Type: KEYWORD, Lexeme: main
Type: DELIMETER, Lexeme: (
Type: DELIMETER, Lexeme: )
Type: DELIMETER, Lexeme: {
Type: KEYWORD, Lexeme: int
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMETER, Lexeme: [
Type: DELIMETER, Lexeme: ]
Type: OPERATOR, Lexeme: =
Type: DELIMETER, Lexeme: {
Type: CONSTANT, Lexeme: 1
Type: DELIMETER, Lexeme: ,
Type: CONSTANT, Lexeme: 2
Type: DELIMETER, Lexeme: ,
Type: CONSTANT, Lexeme: 3
Type: DELIMETER, Lexeme: ,
Type: CONSTANT, Lexeme: 4
Type: DELIMETER, Lexeme: ,
Type: CONSTANT, Lexeme: 5
Type: DELIMETER, Lexeme: }
Type: DELIMETER, Lexeme: ;
Type: KEYWORD, Lexeme: for
Type: DELIMETER, Lexeme: (
Type: KEYWORD, Lexeme: int
Type: IDENTIFIER, Lexeme: i
Type: OPERATOR, Lexeme: =
```

```
Type: DELIMETER, Lexeme: (
Type: KEYWORD, Lexeme: int
Type: IDENTIFIER, Lexeme: i
Type: OPERATOR, Lexeme: =
Type: CONSTANT, Lexeme: 0
Type: DELIMETER, Lexeme: ;
Type: IDENTIFIER, Lexeme: i
Type: DELIMETER, Lexeme: <
Type: CONSTANT, Lexeme: 5
Type: DELIMETER, Lexeme: ;
Type: IDENTIFIER, Lexeme: i
Type: OPERATOR, Lexeme: ++
Type: DELIMETER, Lexeme: )
Type: DELIMETER, Lexeme: {
Type: KEYWORD, Lexeme: if
Type: DELIMETER, Lexeme: (
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMETER, Lexeme: [
Type: IDENTIFIER, Lexeme: i
Type: DELIMETER, Lexeme: ]
Type: OPERATOR, Lexeme: %
Type: CONSTANT, Lexeme: 2
Type: OPERATOR, Lexeme: ==
Type: CONSTANT, Lexeme: 0)
Type: DELIMETER, Lexeme: {
Type: IDENTIFIER, Lexeme: printf
Type: DELIMETER, Lexeme: (
Type: STRING_LITERAL, Lexeme: "Even:%d\n"
Type: DELIMETER, Lexeme: ,
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMETER, Lexeme: [
Type: IDENTIFIER, Lexeme: i
Type: DELIMETER, Lexeme: ]
Type: DELIMETER, Lexeme: )
Type: DELIMETER, Lexeme: ;
Type: DELIMETER, Lexeme: }
Type: KEYWORD, Lexeme: else
Type: DELIMETER, Lexeme: {
Type: IDENTIFIER, Lexeme: printf
Type: DELIMETER, Lexeme: ("Odd:%d\n"
Type: DELIMETER, Lexeme: ,
Type: IDENTIFIER, Lexeme: numbers
```

```
Type: DELIMETER, Lexeme: (
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMETER, Lexeme: [
Type: IDENTIFIER, Lexeme: i
Type: DELIMETER, Lexeme: ]
Type: OPERATOR, Lexeme: %
Type: CONSTANT, Lexeme: 2
Type: OPERATOR, Lexeme: ==
Type: CONSTANT, Lexeme: 0)
Type: DELIMETER, Lexeme: {
Type: IDENTIFIER, Lexeme: printf
Type: DELIMETER, Lexeme: (
Type: STRING_LITERAL, Lexeme: "Even:%d\n"
Type: DELIMETER, Lexeme: ,
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMETER, Lexeme: [
Type: IDENTIFIER, Lexeme: i
Type: DELIMETER, Lexeme: ]
Type: DELIMETER, Lexeme: )
Type: DELIMETER, Lexeme: ;
Type: DELIMETER, Lexeme: }
Type: KEYWORD, Lexeme: else
Type: DELIMETER, Lexeme: {
Type: IDENTIFIER, Lexeme: printf
Type: DELIMETER, Lexeme: ("Odd:%d\n"
Type: DELIMETER, Lexeme: ,
Type: IDENTIFIER, Lexeme: numbers
Type: DELIMETER, Lexeme: [i]
Type: DELIMETER, Lexeme: )
Type: DELIMETER, Lexeme: ;
Type: DELIMETER, Lexeme: }
Type: DELIMETER, Lexeme: }
Type: IDENTIFIER, Lexeme: return
Type: CONSTANT, Lexeme: 0
Type: DELIMETER, Lexeme: ;
Type: DELIMETER, Lexeme: }
Type: END_OF_FILE, Lexeme:
```