1.The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Develop a lexical Analyzer to identify identifiers, constants, operators using C program

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

#define MAX_IDENTIFIER_LENGTH 50

// Function prototypes
int isOperator(char ch);
int isKeyword(char* str);
int isDelimiter(char ch);

int main() {
    char code[1000];
    char currentLexeme[MAX_IDENTIFIER_LENGTH + 1];
    int i = 0, j = 0;

    printf("Enter C code (end with Ctrl+D or Ctrl+Z):\n");

    // Read input until EOF
    while ((code[i++] = getchar()) != EOF);

    i = 0;
```

```
// Loop through the code
while (code[i] != '\0') {
    // Ignore spaces, tabs, and new lines
    if (isspace(code[i])) {
        i++;
        continue;
    }

    // Ignore single-line comments
    if (code[i] == '/' && code[i + 1] == '/') {
        while (code[i] != '\n')
            i++;
        continue;
    }

    // Ignore multi-line comments
    if (code[i] == '/' && code[i + 1] == '*') {
        i += 2;
        while (code[i] != '*' || code[i + 1] != '/') {
            i++;
            if (code[i] == '\0') {
                printf("Error: Unterminated comment\n");
                return 1;
            }
        }
        i += 2;
        continue;
    }

    // If it's an operator
```

```
if (isOperator(code[i])) {
    printf("Operator: %c\n", code[i]);
    i++;
}
// If it's a delimiter
else if (isDelimiter(code[i])) {
    printf("Delimiter: %c\n", code[i]);
    i++;
}
// If it's a digit (constant)
else if (isdigit(code[i])) {
    printf("Constant: ");
    while (isdigit(code[i])) {
        printf("%c", code[i]);
        i++;
    }
    printf("\n");
}
// If it's an identifier or keyword
else if (isalpha(code[i])) {
    memset(currentLexeme, 0, sizeof(currentLexeme));
    j = 0;
    while (isalnum(code[i]) && j < MAX_IDENTIFIER_LENGTH) {
        currentLexeme[j++] = code[i++];
    }
    currentLexeme[j] = '\0';

    if (isKeyword(currentLexeme))
        printf("Keyword: %s\n", currentLexeme);
    else
        printf("Identifier: %s\n", currentLexeme);
```

```
        }
        // If it's none of the above, it's an invalid character
        else {
            printf("Invalid character: %c\n", code[i]);
            i++;
        }
    }

    return 0;
}

// Function to check if a character is an operator
int isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '=' || ch == '<' || ch == '>' || ch == '&' || ch == '|' || ch == '!');
}

// Function to check if a string is a keyword
int isKeyword(char* str) {
    char keywords[32][10] = {"auto", "break", "case", "char", "const", "continue", "default", "do", "double", "else", "enum", "extern", "float", "for", "goto", "if", "int", "long", "register", "return", "short", "signed", "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned", "void", "volatile", "while"};
    int i;

    for (i = 0; i < 32; ++i) {
        if (strcmp(keywords[i], str) == 0) {
            return 1;
        }
    }

    return 0;
```

```
}


// Function to check if a character is a delimiter
int isDelimiter(char ch) {
    return (ch == ' ' || ch == ',' || ch == ';' || ch == '(' || ch == ')' || ch == '{' || ch == '}');
}
```

```
Enter C code (end with Ctrl+D or Ctrl+Z):
int main()
{
int a;
}
^Z
Keyword: int
Identifier: main
Delimiter: (
Delimiter: )
Delimiter: {
Keyword: int
Identifier: a
Delimiter: ;
Delimiter: }
Invalid character:

--------------------------------
Process exited after 31.34 seconds with return value 0
Press any key to continue . . .
```