

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 <ctype.h>

int main() {

    char ch;

    printf("Enter input (Ctrl+D to end):\n");

    while ((ch = getchar()) != EOF) {

        if (isspace(ch)) continue;

        if (isalpha(ch))

            printf("Identifier starts with: %c\n", ch);

        else if (isdigit(ch))

            printf("Constant: %c\n", ch);

        else if (ch=='+' || ch=='-' || ch=='*' || ch=='/' || ch=='=') // operator

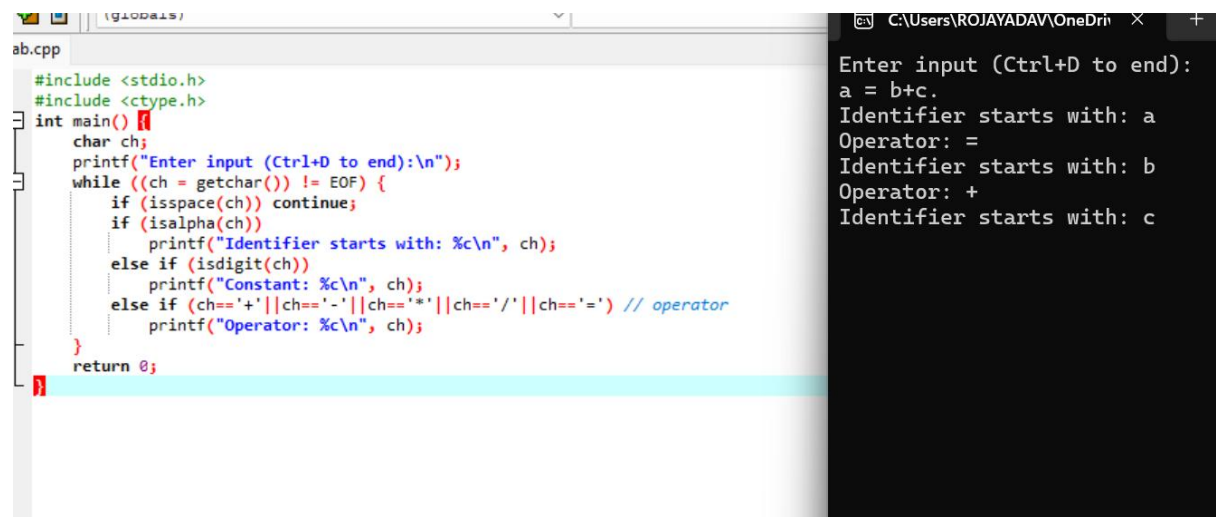
            printf("Operator: %c\n", ch);

    }

    return 0;

}
```

Output:

The image shows a screenshot of a C program in a code editor and its output in a terminal window. The code is a lexical analyzer that processes input character by character. It ignores spaces and identifies identifiers, constants, and operators. The terminal shows the program's execution with the input 'a = b+c.' and the corresponding output for each token.

```
ab.cpp
#include <stdio.h>
#include <ctype.h>
int main() {
    char ch;
    printf("Enter input (Ctrl+D to end):\n");
    while ((ch = getchar()) != EOF) {
        if (isspace(ch)) continue;
        if (isalpha(ch))
            printf("Identifier starts with: %c\n", ch);
        else if (isdigit(ch))
            printf("Constant: %c\n", ch);
        else if (ch=='+' || ch=='-' || ch=='*' || ch=='/' || ch=='=') // operator
            printf("Operator: %c\n", ch);
    }
    return 0;
}
```

```
Enter input (Ctrl+D to end):
a = b+c.
Identifier starts with: a
Operator: =
Identifier starts with: b
Operator: +
Identifier starts with: c
```