Design a lexical Analyzer for given language should ignore the redundant spaces, tabs and new lines and ignore comments using C

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>


#define MAX_SIZE 1000


// Function to check if a character is an operator

int isOperator(char ch) {

   char operators[] = "+-*/%=<>!&|";

   for (int i = 0; i < strlen(operators); i++) {

      if (ch == operators[i]) return 1;

   }

   return 0;

}


// Function to check if a given word is a keyword

int isKeyword(char *word) {

   char *keywords[] = {"int", "float", "char", "if", "else", "while", "for", "return", "void", "do",
"switch", "case"};

   int numKeywords = sizeof(keywords) / sizeof(keywords[0]);

   for (int i = 0; i < numKeywords; i++) {

      if (strcmp(word, keywords[i]) == 0) return 1;

   }

   return 0;

}


// Function to check if a word is a number

int isNumber(char *word) {

   for (int i = 0; i < strlen(word); i++) {

      if (!isdigit(word[i]) && word[i] != '.') return 0;
```

```
    }
    return 1;
}

// Function to remove redundant spaces, tabs, newlines, and comments
void lexicalAnalyzer(char *input) {
    int len = strlen(input);
    int i = 0, inComment = 0;

    printf("Processed Tokens:\n");

    while (i < len) {
        // Skip whitespace
        if (isspace(input[i])) {
            i++;
            continue;
        }

        // Handle comments
        if (input[i] == '/' && input[i + 1] == '/') {
            while (input[i] != '\n' && input[i] != '\0') i++; // Skip single-line comment
            continue;
        }
        if (input[i] == '/' && input[i + 1] == '*') {
            inComment = 1;
            i += 2;
            while (inComment) {
                if (input[i] == '*' && input[i + 1] == '/') {
                    inComment = 0;
                    i += 2;
                } else if (input[i] == '\0') {
```

```
            break;
        } else {
            i++;
        }
    }
    continue;
}

// Identifiers & Keywords
if (isalpha(input[i])) {
    char word[50];
    int j = 0;
    while (isalnum(input[i])) {
        word[j++] = input[i++];
    }
    word[j] = '\0';

    if (isKeyword(word)) {
        printf("Keyword: %s\n", word);
    } else {
        printf("Identifier: %s\n", word);
    }
}

// Numbers (Constants)
else if (isdigit(input[i])) {
    char num[50];
    int j = 0;
    while (isdigit(input[i]) || input[i] == '.') {
        num[j++] = input[i++];
    }
```

```
            num[j] = '\0';
            printf("Constant: %s\n", num);
        }

        // Operators
        else if (isOperator(input[i])) {
            char op[3] = {input[i], '\0', '\0'};

            // Handle multi-character operators (==, !=, <=, >=, &&, ||)
            if ((input[i] == '=' || input[i] == '!' || input[i] == '<' || input[i] == '>') && input[i + 1] == '=') {
                op[1] = '=';
                i++;
            } else if ((input[i] == '&' || input[i] == '|') && input[i + 1] == input[i]) {
                op[1] = input[i];
                i++;
            }

            printf("Operator: %s\n", op);
            i++;
        }

        // Special characters
        else {
            printf("Symbol: %c\n", input[i]);
            i++;
        }
    }
}

// Main function
int main() {
    char input[MAX_SIZE];


    printf("Enter the code snippet:\n");
    fgets(input, sizeof(input), stdin);


    lexicalAnalyzer(input);


    return 0;
}
```

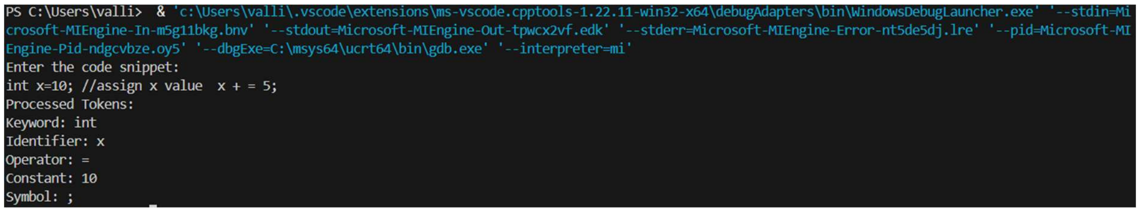
Input:

Int x=10;

//assign x value

X+=5;

Output:



```
PS C:\Users\valli>  & 'c:\Users\valli\.vscode\extensions\ms-vscode.cpptools-1.22.11-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Mi
crosoft-MIEngine-In-m5g11bkg.bnv' '--stdout=Microsoft-MIEngine-Out-tpwcx2vf.edk' '--stderr=Microsoft-MIEngine-Error-nt5de5dj.lre' '--pid=Microsoft-MI
Engine-Pid-ndgcvbze.oy5' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Enter the code snippet:
int x=10; //assign x value  x + = 5;
Processed Tokens:
Keyword: int
Identifier: x
Operator: =
Constant: 10
Symbol: ;
```