**EX 1 : Develop a lexical Analyzer to identify identifiers, constants, operators using C program.**

**Aim**

To design and implement a lexical analyzer in C that can identify identifiers, constants, and operators while ignoring redundant spaces, tabs, newlines, and comments.

**Algorithm**

1. **Start**
2. Read the input code snippet.
3. Initialize counters and buffers.
4. While the end of the input is not reached:
   * Ignore spaces, tabs, and newlines.
   * Ignore comments:
     + Single-line (//)
     + Multi-line (/\* \*/)
   * Detect identifiers and keywords:
     + If a letter or underscore is found, read alphanumeric characters to form identifiers.
     + Check if the identifier is a keyword.
   * Detect numeric constants by reading consecutive digits.
   * Detect and print operators.
   * Skip unrecognized characters.
5. Print detected tokens.
6. **Stop**

**Code**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#include <stdbool.h>

#define MAX\_IDENTIFIER\_LENGTH 31

// Function to check if a character is an operator

bool isOperator(char ch) {

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

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

if (ch == operators[i]) {

return true;

}

}

return false;

}

// Function to check if a string is a keyword

bool isKeyword(const char \*str) {

const char \*keywords[] = {

"int", "float", "char", "if", "else", "for", "while", "return", "void", "main"

};

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

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

if (strcmp(str, keywords[i]) == 0) {

return true;

}

}

return false;

}

// Function to recognize identifiers and constants

void lexicalAnalysis(const char \*input) {

int i = 0;

while (input[i] != '\0') {

// Ignore spaces, tabs, and newlines

if (isspace(input[i])) {

i++;

continue;

}

// Ignore comments (single-line // and multi-line /\* \*/)

if (input[i] == '/' && input[i + 1] == '/') {

while (input[i] != '\0' && input[i] != '\n') i++;

continue;

} else if (input[i] == '/' && input[i + 1] == '\*') {

i += 2;

while (input[i] != '\0' && !(input[i] == '\*' && input[i + 1] == '/')) i++;

i += 2;

continue;

}

// Check for identifiers and keywords

if (isalpha(input[i]) || input[i] == '\_') {

char buffer[MAX\_IDENTIFIER\_LENGTH + 1] = {0};

int j = 0;

while ((isalnum(input[i]) || input[i] == '\_') && j < MAX\_IDENTIFIER\_LENGTH) {

buffer[j++] = input[i++];

}

buffer[j] = '\0';

if (isKeyword(buffer)) {

printf("Keyword: %s\n", buffer);

} else {

printf("Identifier: %s\n", buffer);

}

continue;

}

// Check for numeric constants

if (isdigit(input[i])) {

char buffer[32] = {0};

int j = 0;

while (isdigit(input[i])) {

buffer[j++] = input[i++];

}

printf("Constant: %s\n", buffer);

continue;

}

// Check for operators

if (isOperator(input[i])) {

printf("Operator: %c\n", input[i]);

i++;

continue;

}

// Skip unrecognized characters

i++;

}

}

int main() {

char input[1024];

printf("Enter the code snippet: \n");

fgets(input, sizeof(input), stdin);

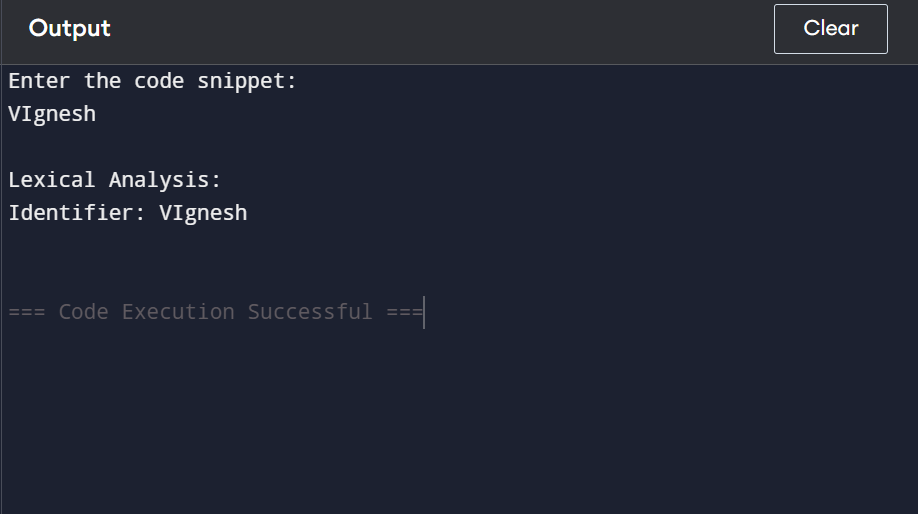
printf("\nLexical Analysis:\n");

lexicalAnalysis(input);

return 0;

}

**OUTPUT**

****