PART B
EXPERIMENT NUMBER 1

Aim: Write a program to implement Lexical Analyzer for given language using Finite Automata.

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded at the end of the practical)

Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 05/02/2021	**Date of Submission:** 05/02/2021
Grade:	

B.1 Software Code written by a student:
(Paste your code completed during the 2 hours of practice in the lab here)

- **SPCC-1.c**

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int dfa = 0;
void start(char c)
{
        if (c == 'a')
                dfa = 1;
        else if (c == 'b')
                dfa = 3;
        else
                dfa = -1;
}
void state1(char c)
{
        if (c == 'a')
                dfa = 2;
        else if (c == 'b')
                dfa = 4;
        else
                dfa = -1;
}
void state2(char c)
{
        if (c == 'b')
                dfa = 3;
        else if (c == 'a')
                dfa = 1;
        else
                dfa = -1;
}
void state3(char c)
{
        if (c == 'b')
                dfa = 3;
        else if (c == 'a')
                dfa = 4;
        else
                dfa = -1;
}
void state4()
{
        dfa = -1;
}

int isAccepted(char str[])
{

        int i, len = strlen(str);

        for (i = 0; i < len; i++) {
                if (dfa == 0)
                        start(str[i]);
                else if (dfa == 1)
                        state1(str[i]);
                else if (dfa == 2)
                        state2(str[i]);
                else if (dfa == 3)
                        state3(str[i]);
                else if (dfa == 4)
                        state4(str[i]);
                else
                        return 0;
        }
        if (dfa == 3)
                return 1;
        else
                return 0;
}
int main()
{
        char str[50];

        printf("Kindly enter string in a sequence of a and b: ");
```

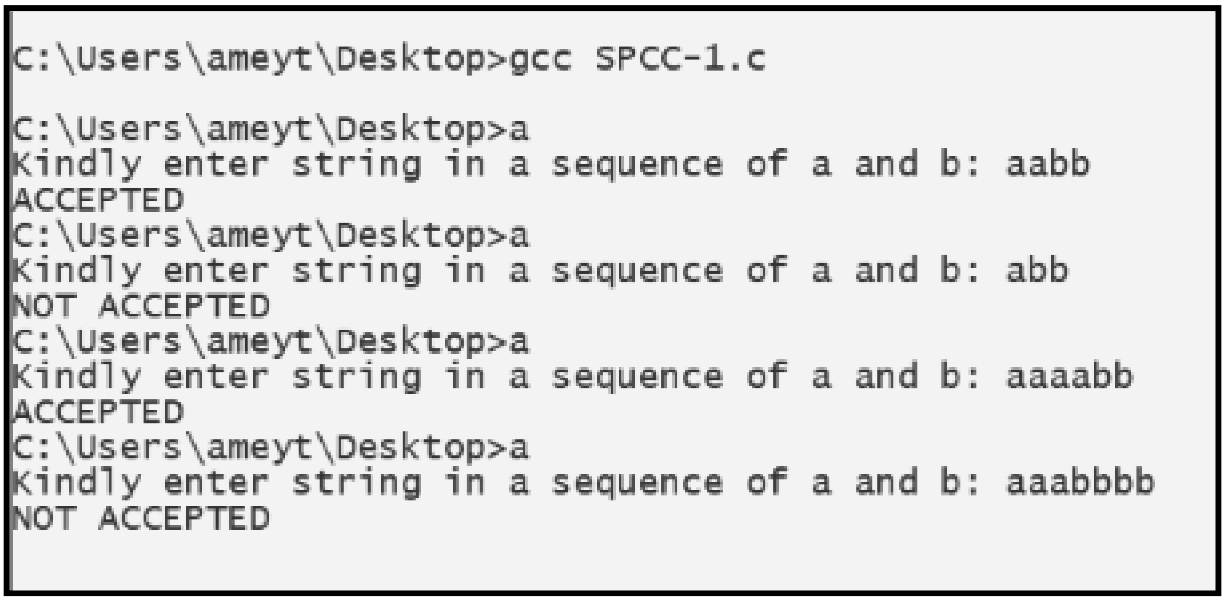
```
	scanf("%s",str);
	if (isAccepted(str))
		printf("ACCEPTED");
	else
		printf("NOT ACCEPTED");
	getch();
	return 0;
}
```

- **Lexical_Analyzer.c**

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

int isKeyword(char buffer[]){
	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, flag = 0;

	for(i = 0; i < 32; ++i){
		if(strcmp(keywords[i], buffer) == 0){
			flag = 1;
			break;
		}
	}

	return flag;
}

int main(){
	char ch, buffer[15], operators[] = "+-*/%=";
	FILE *fp;
	int i,j=0;

	fp = fopen("C:\\Users\\ameyt\\Desktop\\test.txt","r");

	if(fp == NULL){
		printf("error while opening the file\n");
		exit(0);
	}
	while((ch = fgetc(fp)) != EOF){
		for(i = 0; i < 6; ++i){
			if(ch == operators[i])
				printf("%c is operator\n", ch);
		}

		if(isalnum(ch)){
			buffer[j++] = ch;
		}
		else if((ch == ' ' || ch == '\n') && (j != 0)){
				buffer[j] = '\0';
				j = 0;

				if(isKeyword(buffer) == 1)
					printf("%s is keyword\n", buffer);
				else
					printf("%s is indentifier\n", buffer);
		}

	}

	fclose(fp);

	return 0;
}
```

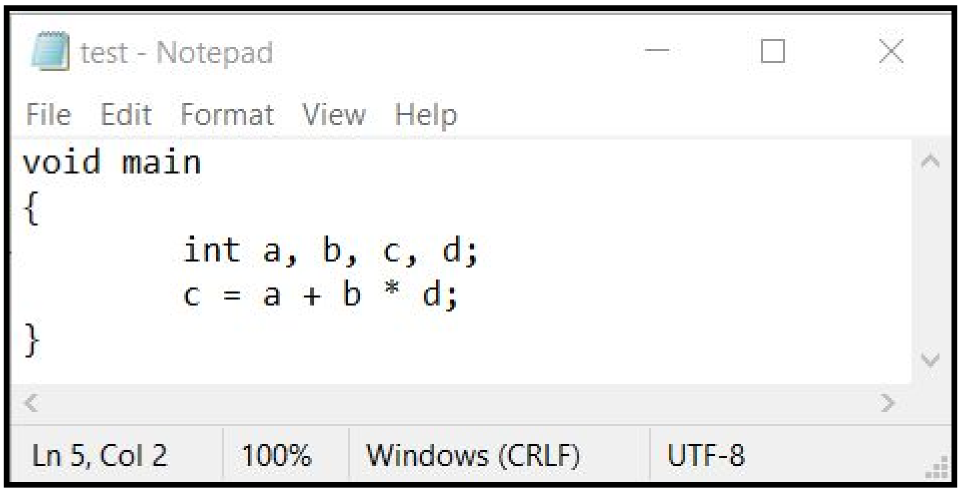
B.2 Input and Output:

- **SPCC-1.c**

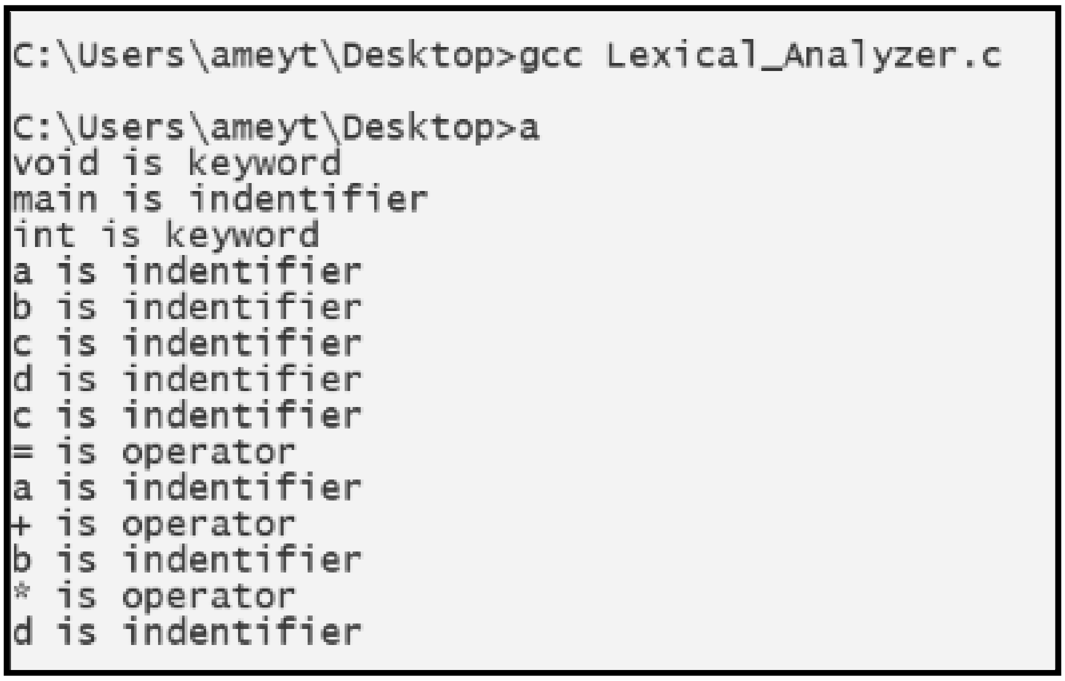


```
C:\Users\ameyt\Desktop>gcc SPCC-1.c

C:\Users\ameyt\Desktop>a
Kindly enter string in a sequence of a and b: aabb
ACCEPTED
C:\Users\ameyt\Desktop>a
Kindly enter string in a sequence of a and b: abb
NOT ACCEPTED
C:\Users\ameyt\Desktop>a
Kindly enter string in a sequence of a and b: aaaabb
ACCEPTED
C:\Users\ameyt\Desktop>a
Kindly enter string in a sequence of a and b: aaabbbb
NOT ACCEPTED
```

- **Lexical_Analyzer.c**



```
void main
{
        int a, b, c, d;
        c = a + b * d;
}
```



```
C:\Users\ameyt\Desktop>gcc Lexical_Analyzer.c

C:\Users\ameyt\Desktop>a
void is keyword
main is indentifier
int is keyword
a is indentifier
b is indentifier
c is indentifier
d is indentifier
c is indentifier
= is operator
a is indentifier
+ is operator
b is indentifier
* is operator
d is indentifier
```

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ subpart assigned)

From the diagram that we drew concerning our DFA, we implemented DFA functions to accept a string of characters from the user and tell whether the string is accepted or not. Hence, we learnt a program to implement Lexical Analyzer for a given language using Finite Automata.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

We learnt how to implement DFA in c using functions.

B.5 Question of Curiosity

(To be answered by a student based on the practical performed and learning/ observations)

1. What is a token?

Ans:

- ➔ A token is the smallest element(character) of a computer language program that is meaningful to the compiler. The parser has to recognize these as tokens: identifiers, keywords, literals, operators, punctuators, and other separators.
- ➔ A stream of these tokens makes up a translation to ASM or in some cases a Low-Level Language as C.
- ➔ Sample of C++ Tokens below.

KeyWords.

1. for,if,else,while,etc.

Source Token Characters For Reserved Words or Operators)

1. a b c d e f g h i j k l m n o p q r s t u v w x y z
2. A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3. 0 1 2 3 4 5 6 7 8 9
4. _ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '

Identifiers —(Variable Names).

1. _ a b c d e f g h i j k l m
2. n o p q r s t u v w x y z
3. A B C D E F G H I J K L M
4. N O P Q R S T U V W X Y Z
5. 1,2,3,4,5,6,7,8,9   ZERO-"0"- IS NOT ALLOWED IN IDENTIFIERS BUT SOME LANGUAGES CONVERT THESE TO A LETTER OR NUMBER IN THE COMPILER
6. C++ Language recognizes all character sets in Japanese.

Unicode. (Examples)

1. 00A8, 00AA, 00AD, 00AF, 00B2-00B5, 00B7-00BA, 00BC-00BE, 00C0-00D6, 00D8-00F6, 00F8-00FF, 0100-02FF, 0370-167F, 1681-180D, 180F-1DBF, 1E00-1FFF, 200B-200D, 202A-202E, 203F-2040, 2054, 2060-206F, 2070-20CF, 2100-218F, 2460-24FF, 2776-2793, 2C00-2DFF, 2E80-2FFF, 3004-3007, 3021-302F, 3031-303F, 3040-D7FF, F900-FD3D, FD40-FDCF, FDF0-FE1F, FE30-FE44, FE47-FFFD, 10000-1FFFD, 20000-2FFFD, 30000-3FFFD, 40000-4FFFD, 50000-5FFFD, 60000-6FFFD, 70000-7FFFD, 80000-8FFFD, 90000-9FFFD, A0000-AFFFD, B0000-BFFFD, C0000-CFFFD, D0000-DFFFD, E0000-EFFFD

Separators.

1. \s, \n, \t, \r,etc.

Comments.

1. /* comment */
2. or
3. // comment

2. What is the role of lexical analyzer?

Ans:
The lexical analyzer performs below given tasks:

➔ Helps to identify token into the symbol table
➔ Removes white spaces and comments from the source program
➔ Correlates error messages with the source program
➔ Helps you to expands the macros if it is found in the source program
➔ Read input characters from the source program

3. What is the output of Lexical analyzer?

Ans:
The lexical analysis produces a stream of tokens as output, which consists of identifier, keywords, separator, operator, and literals.