```
{
	if (ch == '+' || ch == '-' || ch == '*' ||
		ch == '/' || ch == '>' || ch == '<' ||
		ch == '=')
		return (true);
	return (false);
}

bool validIdentifier(char* str)
{
	if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
		str[0] == '3' || str[0] == '4' || str[0] == '5' ||
		str[0] == '6' || str[0] == '7' || str[0] == '8' ||
		str[0] == '9' || isDelimiter(str[0]) == true)
		return (false);
	return (true);
}
bool isKeyword(char* str)
{
	if (!strcmp(str, "if") || !strcmp(str, "else") ||
		!strcmp(str, "while") || !strcmp(str, "do") ||
		!strcmp(str, "break") ||
		!strcmp(str, "continue") || !strcmp(str, "int")
		|| !strcmp(str, "double") || !strcmp(str, "float")
		|| !strcmp(str, "return") || !strcmp(str, "char")
		|| !strcmp(str, "case") || !strcmp(str, "char")
		|| !strcmp(str, "sizeof") || !strcmp(str, "long")
		|| !strcmp(str, "short") || !strcmp(str, "typedef")
		|| !strcmp(str, "switch") || !strcmp(str, "unsigned")
		|| !strcmp(str, "void") || !strcmp(str, "static")
		|| !strcmp(str, "struct") || !strcmp(str, "goto"))
		return (true);
	return (false);
}
bool isInteger(char* str)
```

```
{
	int i, len = strlen(str);

	if (len == 0)
		return (false);
	for (i = 0; i < len; i++) {
		if (str[i] != '0' && str[i] != '1' && str[i] != '2'
			&& str[i] != '3' && str[i] != '4' && str[i] != '5'
			&& str[i] != '6' && str[i] != '7' && str[i] != '8'
			&& str[i] != '9' || (str[i] == '-' && i > 0))
			return (false);
	}
	return (true);
}
bool isRealNumber(char* str)
{
	int i, len = strlen(str);
	bool hasDecimal = false;

	if (len == 0)
		return (false);
	for (i = 0; i < len; i++) {
		if (str[i] != '0' && str[i] != '1' && str[i] != '2'
			&& str[i] != '3' && str[i] != '4' && str[i] != '5'
			&& str[i] != '6' && str[i] != '7' && str[i] != '8'
			&& str[i] != '9' && str[i] != '.' ||
			(str[i] == '-' && i > 0))
			return (false);
		if (str[i] == '.')
			hasDecimal = true;
	}
	return (hasDecimal);
}
char* subString(char* str, int left, int right)
{
```

```
	int i;
	char* subStr = (char*)malloc(
				sizeof(char) * (right - left + 2));

	for (i = left; i <= right; i++)
		subStr[i - left] = str[i];
	subStr[right - left + 1] = '\0';
	return (subStr);
}
void parse(char* str){
	int left = 0, right = 0;
	int len = strlen(str);

	while (right <= len && left <= right) {
		if (isDelimiter(str[right]) == false)
			right++;

		if (isDelimiter(str[right]) == true && left == right) {
			if (isOperator(str[right]) == true)
				printf("'%c' IS AN OPERATOR\n", str[right]);

			right++;
			left = right;
		} else if (isDelimiter(str[right]) == true && left != right
				|| (right == len && left != right)) {
			char* subStr = subString(str, left, right - 1);

			if (isKeyword(subStr) == true)
				printf("'%s' IS A KEYWORD\n", subStr);

			else if (isInteger(subStr) == true)
				printf("'%s' IS AN INTEGER\n", subStr);

			else if (isRealNumber(subStr) == true)
				printf("'%s' IS A REAL NUMBER\n", subStr);
```

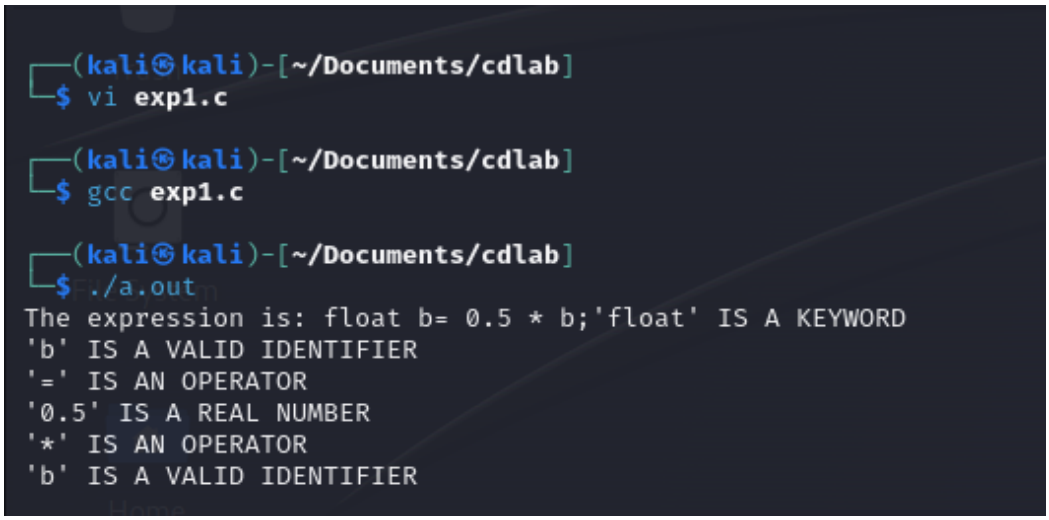
```
                    else if (validIdentifier(subStr) == true
                            && isDelimiter(str[right - 1]) == false)
                        printf("'%s' IS A VALID IDENTIFIER\n", subStr);

                    else if (validIdentifier(subStr) == false
                            && isDelimiter(str[right - 1]) == false)
                        printf("'%s' IS NOT A VALID IDENTIFIER\n", subStr);
                    left = right;}}
        return;}
int main(){
        // maximum length of string is 100 here
        printf("The expression is: float b= 0.5 * b;\n");
        char str[100] = "float b = 0.5 * b; ";

        parse(str); // calling the parse function

        return (0);
}
```

OUTPUT:



RESULT:

Thus, a C program is implemented to identify C keywords, identifiers, operators and end statements.

Roll Number: 210701325
Name: Varssha Balasundaram

Exp No: 2

Date:

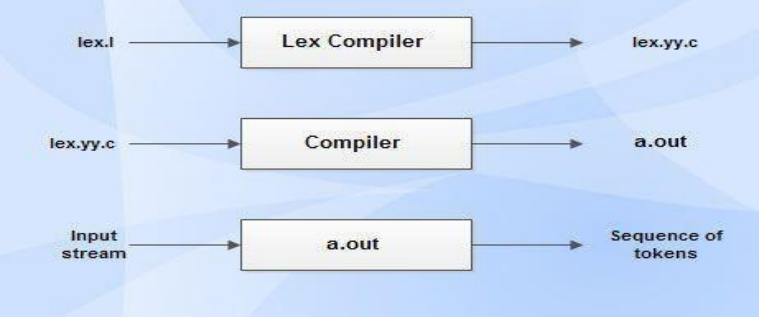
IMPLEMENT A LEXICAL ANALYZER TO COUNT THE NUMBER OF WORDS USING LEX TOOL

AIM:

To implement the program to count the number of words in a string using LEX tool.

STUDY:

Lex is a tool in lexical analysis phase to recognize tokens using regular expression. Lex tool itself is a lex compiler.



- lex.l is an a input file written in a language which describes the generation of lexical analyzer. The lex compiler transforms lex.l to a C program known as lex.yy.c.
- lex.yy.c is compiled by the C compiler to a file called a.out.
- The output of C compiler is the working lexical analyzer which takes stream of input characters and produces a stream of tokens.
- yylval is a global variable which is shared by lexical analyzer and parser to return the name and an attribute value of token.
- The attribute value can be numeric code, pointer to symbol table or nothing.
- Another tool for lexical analyzer generation is Flex.

STRUCTURE OF LEX PROGRAMS:

Lex program will be in following form declarations

%%

translation rules

%%

auxiliary functions

Roll Number: 210701325
Name: Varssha Balasundaram

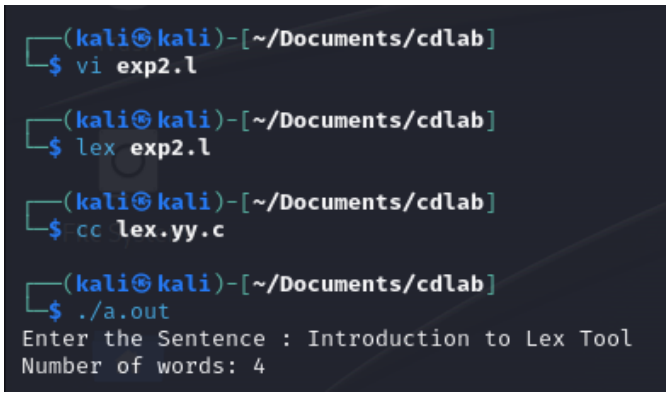
ALGORITHM:

1. Initialize counters for line count (lc), space count (sc), tab count (tc), character count (ch), and word count (wc).
2. Define rules to match newline, space, tab, and non-space/tab/newline characters. Increment corresponding counters based on matches.
3. Prompt the user to enter a sentence.
4. Invoke lexical analysis using yylex().
5. Signal the end of input.
6. Display the total word count.

PROGRAM:

```
%{
#include<stdio.h>
int lc=0,sc=0,tc=0,ch=0,wc=0;
%}
%%
[\n] { lc++; ch+=yyleng;}
[ \t] { sc++; ch+=yyleng;}
[^\t] { tc++; ch+=yyleng;}
[^\t\n ]+ { wc++; ch+=yyleng;}
%%
int yywrap(){ return 1; }
int main(){
        printf("Enter the Sentence : ");
        yylex();
        printf("Number of words: %d\n",wc);
        return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to count the number of words in a string using LEX tool has been implemented.

Roll Number: 210701325
Name: Varssha Balasundaram