Exp No: 3

Date:

DEVELOP A LEXICAL ANALYZER TO RECOGNIZE TOKENS USING LEX TOOL

AIM:

To implement the program to identify C keywords, identifiers, operators, end statements like [], {} using LEX tool.

ALGORITHM:

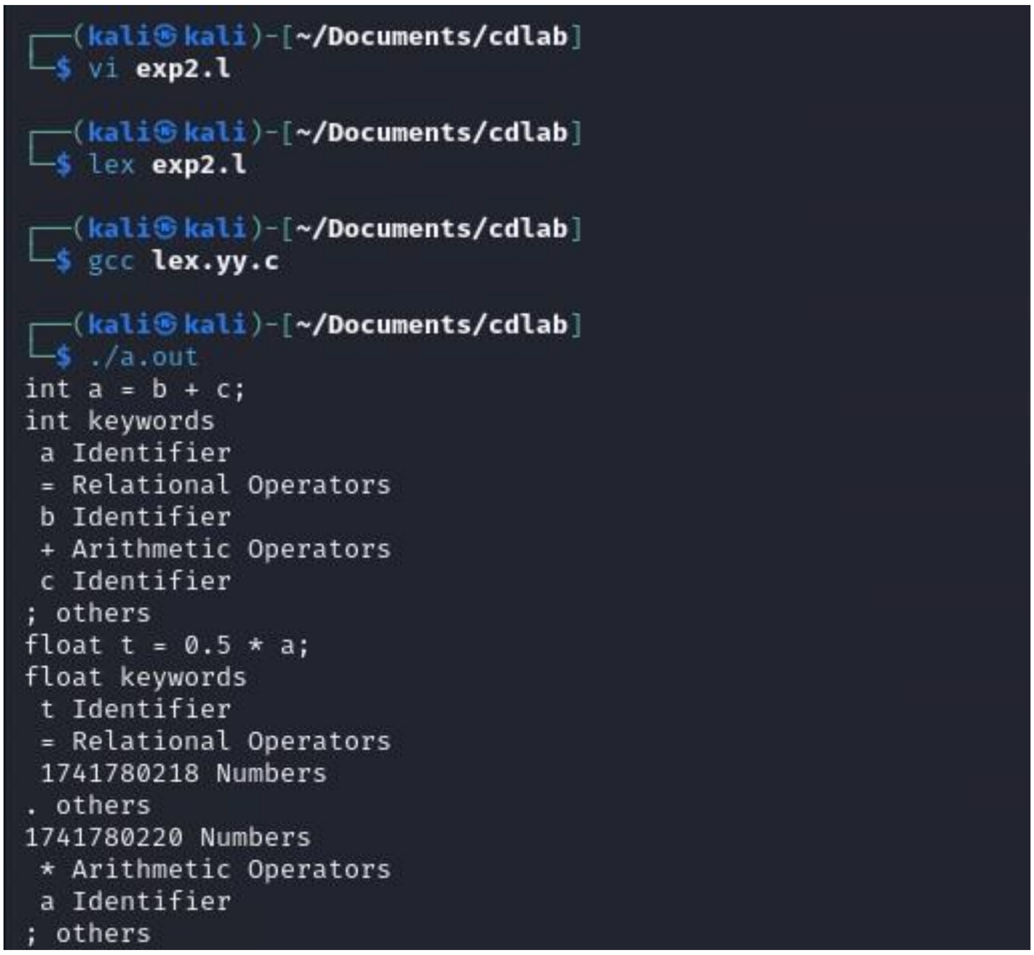
1. Initialize a variable n to count the number of lines.
2. Define patterns for letters, digits, identifiers, arithmetic operators (AO), relational operators (RO), preprocessor directives (pp), and other symbols.
3. Define actions to perform when a pattern is matched and display the corresponding pattern type.
4. Open the file "sample.c" for reading and invoke lexical analysis with yylex().
5. Count the number of newline characters encountered and store it in n.
6. Display the number of lines, n.

PROGRAM:

```
%option noyywrap
letter [a-zA-Z]
digit [0-9]
id [_|a-zA-Z]
AO [+|-|/|%|*]
RO  [<|>|<=|>=|==]
pp [#]
%{
int n=0;
%}
%%
"void"                          printf("%s return type\n",yytext);
{letter}*[(][)]                 printf("%s Function\n",yytext);
"int"|"float"|"if"|"else"  printf("%s keywords\n",yytext);
"printf"                                printf("%s keywords\n",yytext);
{id}({id}|{digit})*             printf("%s Identifier\n",yytext);
{digit}{digit}*                 printf("%d Numbers\n",yytext);
```

```
{AO}                                    printf("%s Arithmetic Operators\n",yytext);
{RO}                                    printf("%s Relational Operators\n",yytext);
{pp}{letter}*[<]{letter}*[.]{letter}[>] printf("%s processor
Directive\n",yytext);
[\n]                                    n++;
"."|","|"}"|"{"|";"             printf("%s others\n",yytext);
%%
int main(){
        yyin=fopen("sample.c","r");
        yylex();
        printf("No of Lines %d\n",n);}
```

OUTPUT:



```
┌──(kali㉿kali)-[~/Documents/cdlab]
└─$ vi exp2.l

┌──(kali㉿kali)-[~/Documents/cdlab]
└─$ lex exp2.l

┌──(kali㉿kali)-[~/Documents/cdlab]
└─$ gcc lex.yy.c

┌──(kali㉿kali)-[~/Documents/cdlab]
└─$ ./a.out
int a = b + c;
int keywords
 a Identifier
 = Relational Operators
 b Identifier
 + Arithmetic Operators
 c Identifier
; others
float t = 0.5 * a;
float keywords
 t Identifier
 = Relational Operators
 1741780218 Numbers
. others
1741780220 Numbers
 * Arithmetic Operators
 a Identifier
; others
```

RESULT:

Thus, a c program is implemented to identify C keywords, identifiers, operators, end statements like [], {} using LEX tool.

Exp No: 4

Date:

DESIGN A DESK CALCULATOR USING LEX TOOL

AIM:

To create a calculator that performs addition, subtraction, multiplication and division using lex tool.

ALGORITHM:

1. Initialize variables and declare a function prototype.
2. Define patterns for digits, arithmetic operations, and line breaks.
3. Implement lexical rules to perform actions based on matched patterns.
4. Define a function to convert tokens to floats and perform arithmetic operations.
5. Invoke lexical analysis in the main function.
6. Indicate the end of input with the yywrap() function.

PROGRAM:

```
%{
int op = 0,i;
float a, b;
int digi();
%}

dig [0-9]+|([0-9]*)"."([0-9]+)
add "+"
sub  "-"
mul "*"
div "/"
pow "^"
ln \n

%%

{dig} {digi();}
{add} {op=1;}
{sub} {op=2;}
{mul} {op=3;}
{div} {op=4;}
{pow} {op=5;}
{ln} {printf("\n The Answer :%f\n\n",a);}

%%

int digi() {
if(op==0)
/* atof() is used to convert
        - the ASCII input to float */
a=atof(yytext);
else{
```