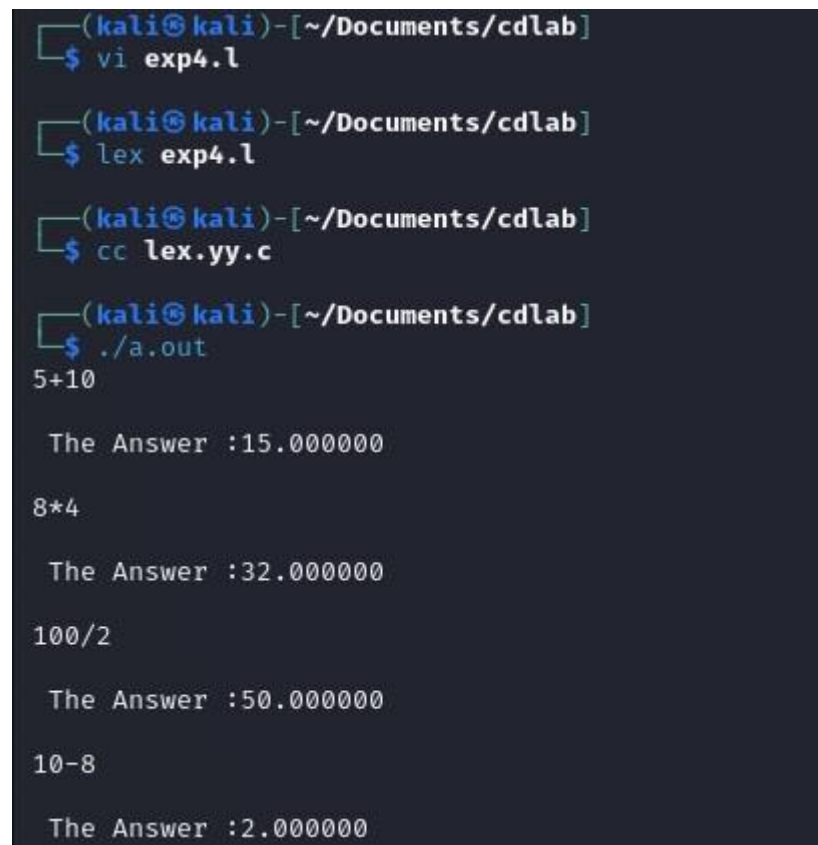


```

b=atof(yytext);
switch(op) {
case 1:a=a+b;
        break;
case 2:a=a-b;
        break;
case 3:a=a*b;
        break;
case 4:a=a/b;
        break;
case 5:for(i=a;b>1;b--)
a=a*i;
        break; }
op=0; } }
int main(int argv,char *argc[]) {
yylex();
}
int yywrap() {
return 1;
}

```

OUTPUT:



```

(kali㉿kali)-[~/Documents/cdlab]
$ vi exp4.l

(kali㉿kali)-[~/Documents/cdlab]
$ lex exp4.l

(kali㉿kali)-[~/Documents/cdlab]
$ cc lex.yy.c

(kali㉿kali)-[~/Documents/cdlab]
$ ./a.out
5+10

The Answer :15.000000

8*4

The Answer :32.000000

100/2

The Answer :50.000000

10-8

The Answer :2.000000

```

RESULT:

Thus, a calculator that performs addition, subtraction, multiplication and division using lex tool is implemented.

Exp No: 5

Date:

DESIGN A DESK CALCULATOR USING LEX TOOL

AIM:

To check whether the arithmetic expression using lex and yacc tool.

ALGORITHM:

- Using the flex tool, create lex and yacc files.
- In the C include section define the header files required.
- In the rules section define the REGEX expressions along with proper definitions.
- In the user defined section define yywrap() function.
- Declare the yacc file inside it in the C definitions section declare the header files required along with an integer variable valid with value assigned as 1.
- In the Yacc declarations declare the format token num id op.
- In the grammar rules section if the starting string is followed by assigning operator or identifier or number or operator followed by a number or open parenthesis followed by an identifier. The x could be an operator followed by an identifier or operator or no operator then declare that as valid expressions by making the valid stay in 1 itself.
- In the user definition section if the valid is 0 print as Invalid expression in yyerror() and define the main function.

LEX AND YACC WORKING:

Parser generator:

- Takes a specification for a context-free grammar.
- Produces code for a parser.

