VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

COMPILER DESIGN

Submitted by

GAMANA YELURI R (1BM21CS065)

Under the Guidance of
Prof. M Lakshmi Neelima
Assistant Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
November-2023 to March-2024

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**COMPILER DESIGN**" carried out by **GAMANA YELURI R (1BM21CS065),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester November-2023 to March-2024. The Lab report has been approved as it satisfies the academic requirements in respect of a COMPILER DESIGN **(22CS5PCCPD)** work prescribed for the said degree.

Prof. M Lakshmi Neelima	Dr. Jyothi S Nayak
Associate Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, GAMANA YELURI R (1BM21CS065), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, hereby declare that, this lab report entitled " **Compiler Design**" has been carried out by me under the guidance of Prof. M Lakshmi Neelima, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester November-2023-February-2024.

I also declare that to the best of my knowledge and belief, the development reported here is not from part of any other report by any other students.

Index Sheet

Course Outcome

CO1	Apply the fundamental concepts for the various phases of compiler design.
CO2	Analyze the syntax and semantic concepts of a compiler.
CO3	Design various types of parsers and Address code generation
CO4	Implement compiler principles, methodologies using lex, yacc tools

Q1) Write a program to design Lexical Analyzer in C/C++/Java/Python Language (to recognize any five keywords, identifiers, numbers, operators and punctuations)

CODE :

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
bool isDelimiter(char ch)
{
if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
ch == '[' || ch == ']' || ch == '{' || ch == '}')
return (true);
return (false);
}
bool isOperator(char ch)
{
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);
```

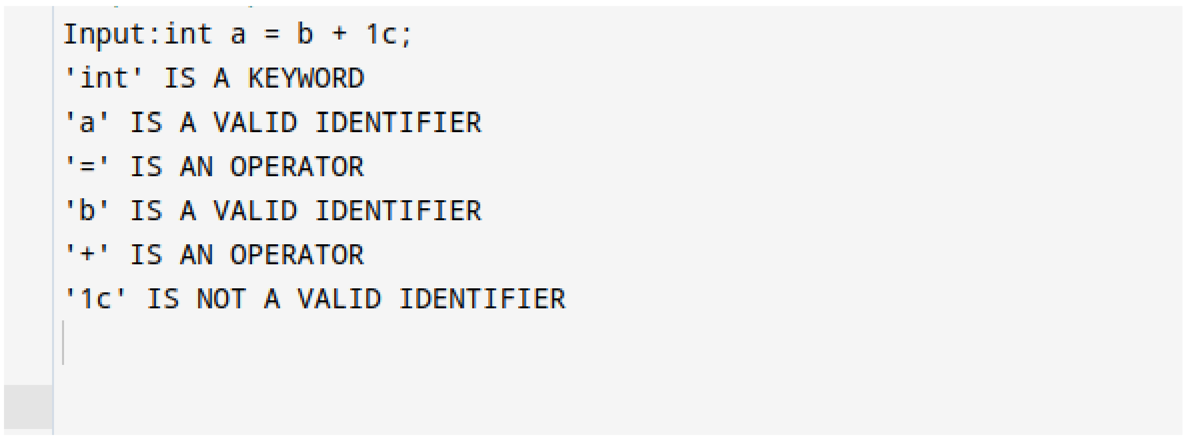
```
for (i = 0; i < len; i++) {
if (str[i] != '0' && str[i] != '1' && str[i] != '2'
&& str[i] != '3' && str[i] != '4' && str[i] != '5'
&& str[i] != '6' && str[i] != '7' && str[i] != '8'
&& str[i] != '9' || (str[i] == '-' && i > 0))
return (false);
}
return (true);
}
bool isRealNumber(char* str)
{
int 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
char str[100] = "int a = b + 1c; ";

parse(str); // calling the parse function
```

return (0);

}

OUTPUT:



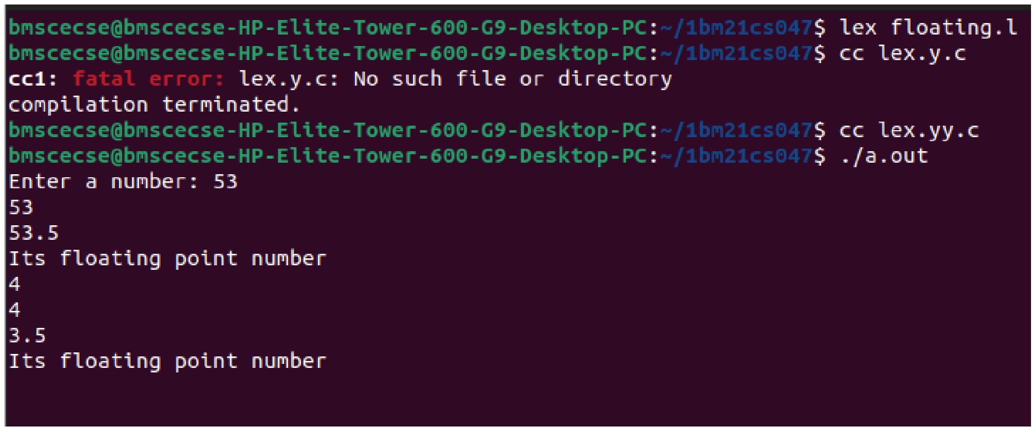
```
Input:int a = b + 1c;
'int' IS A KEYWORD
'a' IS A VALID IDENTIFIER
'=' IS AN OPERATOR
'b' IS A VALID IDENTIFIER
'+' IS AN OPERATOR
'1c' IS NOT A VALID IDENTIFIER
```

Q2)Write a program in LEX to recognize Floating Point Numbers.

CODE:

```
%{
#include<stdio.h>
%}
%%
[+|-]?[0-9]*[.][0-9]* {printf("%s is a floating-point number\n",yytext);}
.* {printf("%s is not a floating-point number\n",yytext);}
%%
int yywrap()
{
}
int main()
{
printf("Enter the string : ");
yylex();
return 0;
}
```

OUTPUT:



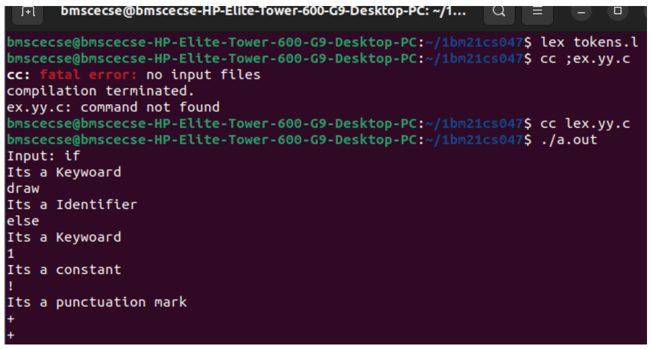
```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ lex floating.l
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ cc lex.y.c
cc1: fatal error: lex.y.c: No such file or directory
compilation terminated.
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ cc lex.yy.c
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ ./a.out
Enter a number: 53
53
53.5
Its floating point number
4
4
3.5
Its floating point number
```

Q3) Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.

CODE:

```
%{
#include<stdio.h>
%}
%%
int|char|float|else|for|void|mainz\while {printf("%s is keyword\n",yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {printf("%s is identifier\n",yytext);}
[0-9]* {printf("%s is a constant\n",yytext);}
[+*^%/<>&=()|]* {printf("%s is operator\n",yytext);}
[?|,.'";:]* {printf("%s is punctuation\n",yytext);}
%%
int yywrap()
{
}
int main()
{
printf("Enter input\n");
yylex();
return 0;
}
```

OUTPUT:



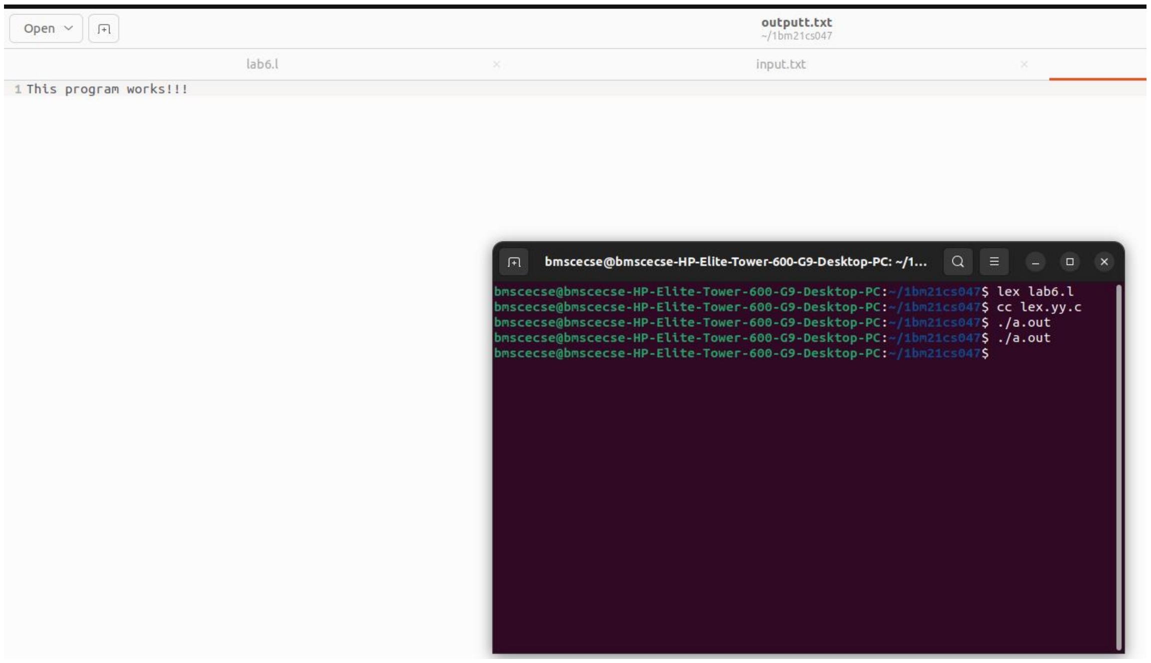
```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ lex tokens.l
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ cc ;ex.yy.c
cc: fatal error: no input files
compilation terminated.
ex.yy.c: command not found
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ cc lex.yy.c
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ ./a.out
Input: if
Its a Keyword
draw
Its a Identifier
else
Its a Keyword
1
Its a constant
!
Its a punctuation mark
+
+
```

Q4) Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.

CODE:

```
/*Definition Section*/
%{
#include<stdio.h>
%}
%%
[\t" "]+ fprintf(yyout," ");
.|\n fprintf(yyout,"%s",yytext);
%%
int yywrap()
{
  return 1;
}
int main(void)
{
yyin=fopen("input.txt","r");
yyout=fopen("output.txt","w");
yylex();
return 0;
}
```

OUTPUT:



```
This program works!!!
```

```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ lex lab6.l
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ cc lex.yy.c
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ ./a.out
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ ./a.out
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$
```

Q5) Write a LEX program to recognize the following tokens over the alphabets {0,1,..,9}

a) The set of all string ending in 00.

b) The set of all strings with three consecutive 222's.

c) The set of all string such that every block of five consecutive symbols contains at least two 5's.

d) The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.

e) The set of all strings such that the 10th symbol from the right end is 1.

f) The set of all four digits numbers whose sum is 9

g) The set of all four digital numbers, whose individual digits are in ascending order from left to right.

CODE:

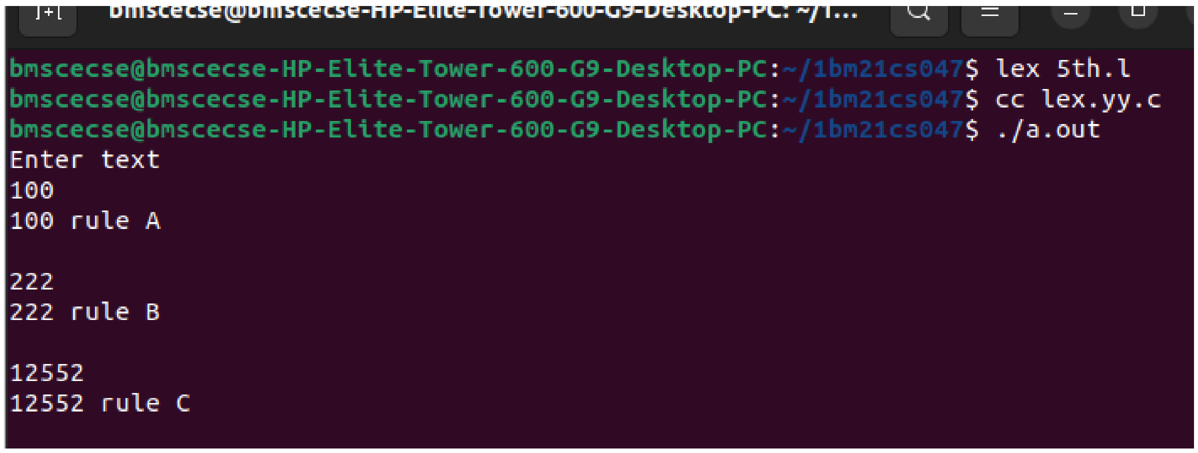
```
{
int c1=0,c2=0,c3=0,c4=0,c5=0,c6=0,c7=0;
%}
d[0-9]
%%
({d})*00 {
c1++; printf("%s rule A\n",yytext);
}
({d})*222({d})* {
 c2++;
  printf("%s rule B \n",yytext);
}

(1(0)*(11|01)(01*01|00*10(0)*(11|1))*0)(1|10(0)*(11|01)(01*01|00*10(0)*(11|1))*10)* {
c4++;
printf("%s rule D \n",yytext);
}
```

```
({d})*1{d}{9} {
c5++;
printf("%s rule E \n",yytext);
}
{d}{4} {
 int sum=0,i;
 for(i=0;i<4;i++) {
  sum=sum+yytext[i]-48; }
 if(sum==9) { c6++; printf("%s rule F \n",yytext);
 }
 else
 {
  sum=1;

  for(i=0;i<3;i++){
   if(yytext[i]>yytext[i+1]) { sum=0;
     break;
   }
  }
  if(sum==1) {
   c7++;
   printf("%s rule G\n",yytext);
  }
  else { printf("%s doesn't match any rule\n",yytext); }
 }
}
({d})* {
int i,c=0;
```

```
if(yyleng<5) { printf("%s doesn't match any rule\n",yytext); }
else
{
for(i=0;i<5;i++) { if(yytext[i]=='5') {
c++; } }
if(c>=2)
{

for(;i<yyleng;i++)

{

if(yytext[i-5]=='5') { c--; }
if(yytext[i]=='5') { c++;
}

if(c<2) { printf("%s doesn't match any rule\n",yytext); break; }
}

if(yyleng==i) { printf("%s rule C\n",yytext); c3++; }
}
else
{
printf("%s doesn't match any rule\n",yytext);
}
}
}
\n {
```

```
printf("Total number of tokens matching rules are : \n");
printf("Rule A : %d \n",c1);
printf("Rule B : %d \n",c2);
printf("Rule C : %d \n",c3);
printf("Rule D : %d \n",c4);
printf("Rule E : %d \n",c5);
printf("Rule F : %d \n",c6);
printf("Rule G : %d \n",c7);
}
%%
int yywrap()
{
}
int main()
{
printf("Enter text\n");
yylex();
return 0;
}
```

OUTPUT:



```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ lex 5th.l
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ cc lex.yy.c
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/1bm21cs047$ ./a.out
Enter text
100
100 rule A

222
222 rule B

12552
12552 rule C
```

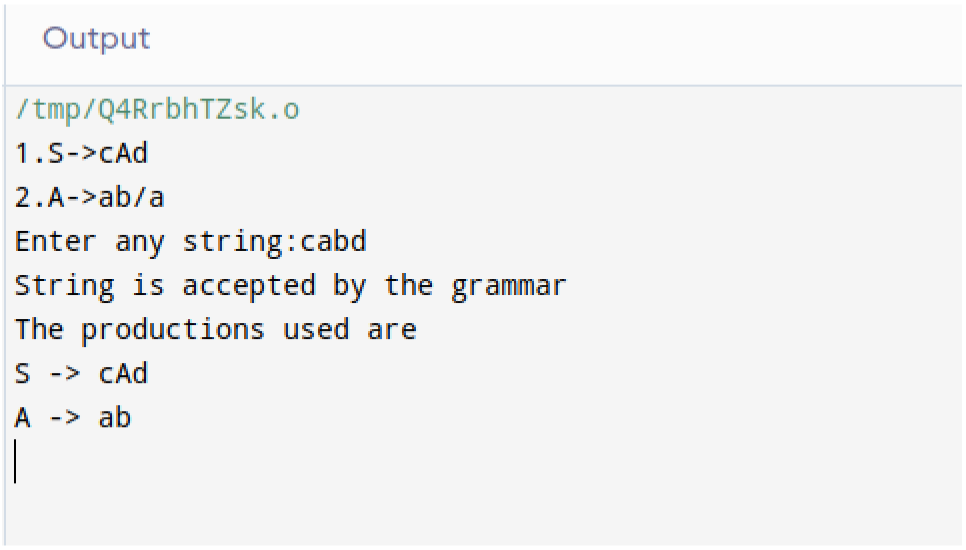
Part-B: Part-B: Implementation of Parsers (Syntax Analyzers) Using C/C++/Java/Python language)

Q1) Write a program to implement (a) Recursive Descent Parsing with back tracking (Brute Force Method). S→ cAd , A →ab /a

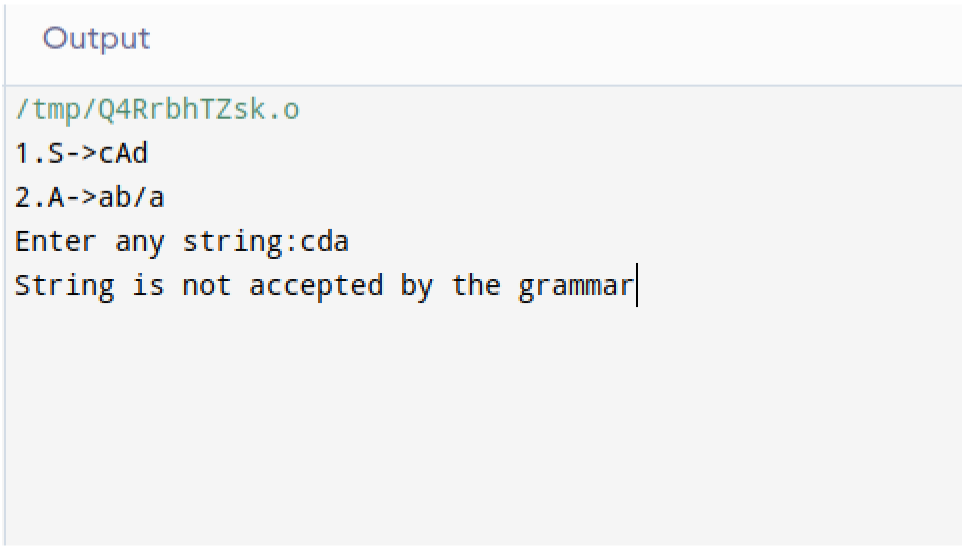
CODE:

```
#include<stdio.h>

#include<string.h>

int A();

void parse();

char str[15];

int isave,curr_ptr=0;

int c=1;

int main(void)

{

    printf("1.S->cAd\n2.A->ab/a\n");

    //printf("this is parser for the above grammar:\n");

    printf("Enter any string:");

    scanf("%s",str);

    while(curr_ptr<strlen(str))

    {

        //S has only one immediate derivation which is cAd

        //match with c

        if (str[curr_ptr]=='c')

        {

            curr_ptr++;

            //call function to match A

            if (A()) //checking the productions of A->ab/a

            {

                curr_ptr++;
                //match d
                if (str[curr_ptr]=='d' && str[curr_ptr+1]=='\0')
                {
                    //success
                    printf("String is accepted by the grammar\n");
                    parse();
                    return 1;
                }
                else break;
            }
            else break;
        }
        else break;
    }
    //incase any of them fail to match return negatively.
    printf("String is not accepted by the grammar");
    return 0;
}

int A()
//sub function A()
{
    //this function matches all terminal strings generated by the variable

    isave=curr_ptr;
    //match with a and advance and match with b. If successful return

    if (str[curr_ptr]=='a')
    {
        curr_ptr++;
        if(str[curr_ptr]=='b')
        {
            c=1;
            return 1;
        }

    }
    curr_ptr=isave; //return to start
    //check if a is matched and return accordingly.
    if(str[curr_ptr]=='a')
    {
        c=2;
        return 1;
    }
    else
        return 0;
}
void parse(){
    printf("The productions used are \n");
    printf("S -> cAd\n");
    if(c==1)
        printf("A -> ab\n");
    else
        printf("A -> a\n");

}
```

OUTPUT:



```
/tmp/Q4RrbhTZsk.o
1.S->cAd
2.A->ab/a
Enter any string:cabd
String is accepted by the grammar
The productions used are
S -> cAd
A -> ab
```



```
/tmp/Q4RrbhTZsk.o
1.S->cAd
2.A->ab/a
Enter any string:cda
String is not accepted by the grammar
```

PART-C :Syntax Directed Translation using YACC tool

Q1) Design a suitable grammar for evaluation of arithmetic expression having + and – operators. + has least priority and it is left associative - has higher priority and is right associative

prog.l

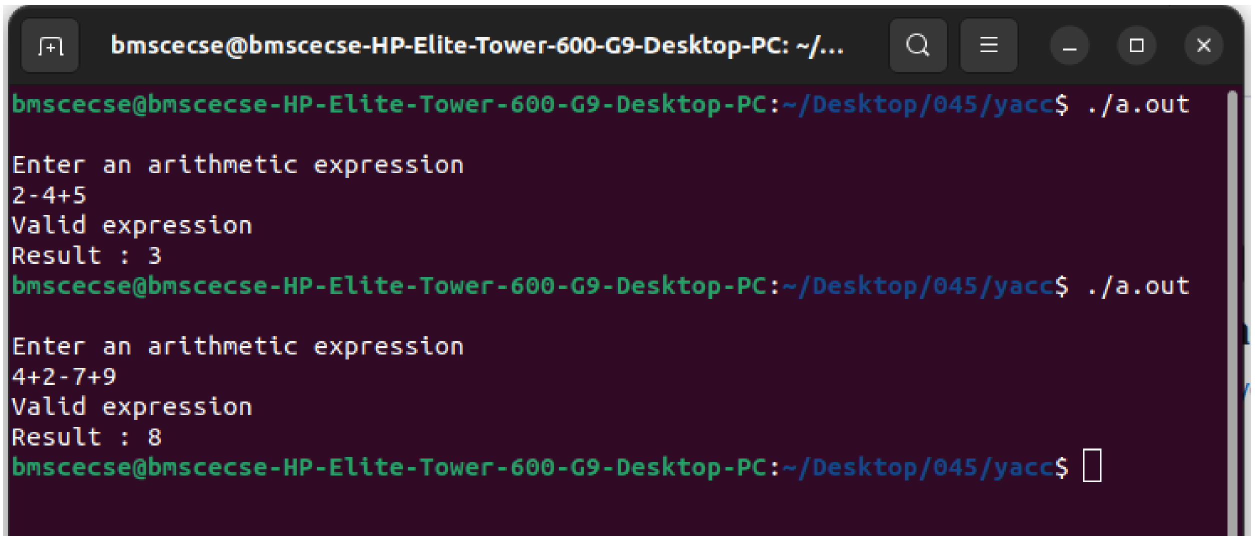
```
%{
#include "y.tab.h"
%}
%%
[0-9]+ {yylval = atoi(yytext);
return NUM;}
[\t] ;
\n return 0;
. return yytext[0];
%%
int yywrap()
{
}
```

prog.y

```
%{
#include <stdio.h>
%}
%token NUM
%left '+'
%right '-'
%%
expr:e {printf("Valid expression\n");
printf("Result : %d\n",$$);
return 0;}
```

```
e: e'+'e {$$=$1+$3;}
| e'-'e {$$=$1-$3;}
| NUM {$$=$1;}
;
%%
int main(){
printf("\nEnter an arithmetic expression\n");
        yyparse();
        return 0;
}
int yyerror(){
        printf("\nInvalid expression\n");
        return 0;
}
```

OUTPUT:



```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045/yacc$ ./a.out

Enter an arithmetic expression
2-4+5
Valid expression
Result : 3
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045/yacc$ ./a.out

Enter an arithmetic expression
4+2-7+9
Valid expression
Result : 8
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045/yacc$ 
```

Q2) Use YACC to implement, evaluator for arithmetic expressions (Desktop calculator) .

prog.l

```
%{
/* Definition section */
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
/* Rule Section */
%%
[0-9]+ {
                yylval=atoi(yytext);
                return NUMBER;


        }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

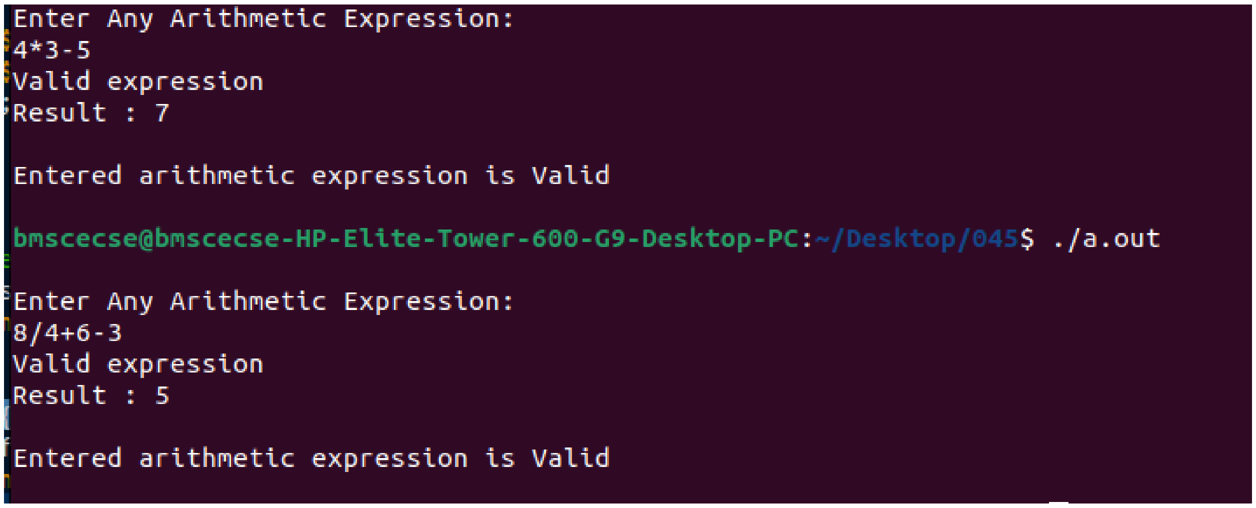
prog.y

```
%{
/* Definition section */
#include<stdio.h>
int flag=0;
```

```
%}
%token NUMBER
%right '^'
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
/* Rule Section */
%%
expr:E {printf("Valid expression\n");
printf("Result : %d\n",$$);
return 0;}
E:E'+'E {$$=$1+$3;}
|E'-'E {$$=$1-$3;}
|E'*'E {$$=$1*$3;}
|E'/'E {$$=$1/$3;}
|E'%'E {$$=$1%$3;}
|E'^'E {$$=$1^$3;}
|'('E')' {$$=$2;}
| NUMBER {$$=$1;}
;
%%
//driver code
void main()
{
printf("\nEnter Any Arithmetic Expression:\n");
yyparse();
if(flag==0)
printf("\nEntered arithmetic expression is Valid\n\n");
```

```
}

void yyerror()
{
printf("\nEntered arithmetic expression is Invalid\n\n");
flag=1;
}
```

OUTPUT:



```
Enter Any Arithmetic Expression:
4*3-5
Valid expression
Result : 7

Entered arithmetic expression is Valid

bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045$ ./a.out

Enter Any Arithmetic Expression:
8/4+6-3
Valid expression
Result : 5

Entered arithmetic expression is Valid
```

Q3) Use YACC to generate Syntax tree for a given expression.

prog.l

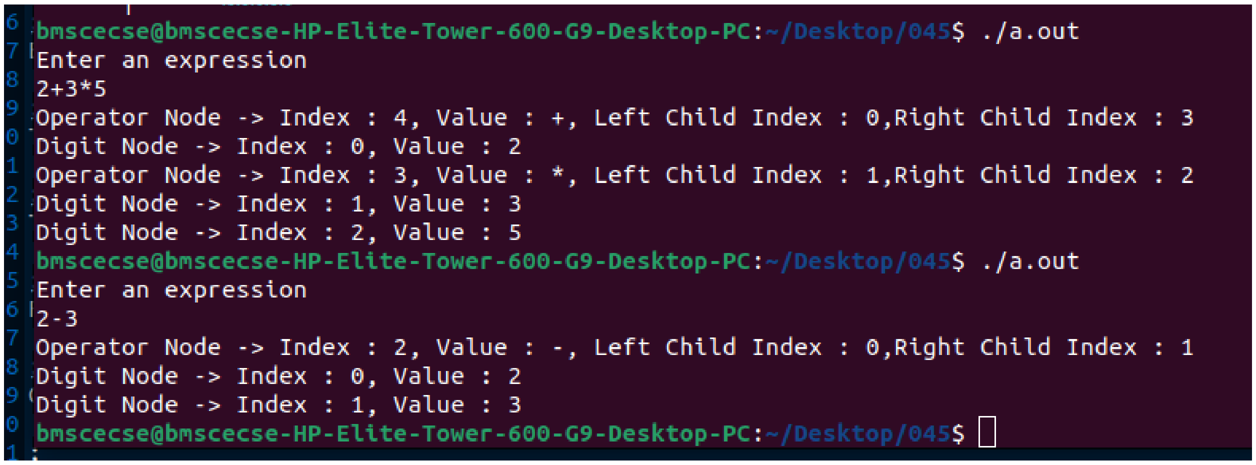
```
%{
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ { yylval = atoi(yytext);
return digit; }

[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap(){
}
```

prog.y

```
%{
#include <math.h>
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct tree_node
{
char val[10];
int lc;
int rc;
};
int ind;
struct tree_node syn_tree[100];
void my_print_tree(int cur_ind);
int mknode(int lc,int rc,char val[10]);
%}
%token digit
%right '^'
%left '+' '-'
%left '*' '/' '%'
%%
S:E { my_print_tree($1); }
;
E:E'+'T { $$= mknode($1,$3,"+"); ; }
|T { $$=$1; }
;
E:E'-'T { $$= mknode($1,$3,"-"); ; }
|T { $$=$1; }
;
T:T'*'F { $$= mknode($1,$3,"*"); ; }
|F {$$=$1 ; }
;
T:T'/'F { $$= mknode($1,$3,"/"); ; }
|F {$$=$1 ; }
;
F:'('E')' { $$=$2; }
|digit {char buf[10]; sprintf(buf,"%d", yylval); $$ = mknode(-1,-1,buf);}
```

```
%%
int main()
{
ind=0;
printf("Enter an expression\n");
yyparse();
return 0;
}
int yyerror()
{
printf("NITW Error\n");
}
int mknode(int lc,int rc,char val[10])
{
strcpy(syn_tree[ind].val,val);
syn_tree[ind].lc = lc;
syn_tree[ind].rc = rc;
ind++;
return ind-1;
}
/*my_print_tree function to print the syntax tree in DLR fashion*/
void my_print_tree(int cur_ind)
{
if(cur_ind==-1) return;
if(syn_tree[cur_ind].lc==-1&&syn_tree[cur_ind].rc==-1)
printf("Digit Node -> Index : %d, Value : %s\n",cur_ind,syn_tree[cur_ind].val);
else
printf("Operator Node -> Index : %d, Value : %s, Left Child Index : %d,Right Child Index :
%d\n",cur_ind,syn_tree[cur_ind].val, syn_tree[cur_ind].lc,syn_tree[cur_ind].rc);
my_print_tree(syn_tree[cur_ind].lc);

my_print_tree(syn_tree[cur_ind].rc);

}
```

OUTPUT:



```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045$ ./a.out
Enter an expression
2+3*5
Operator Node -> Index : 4, Value : +, Left Child Index : 0,Right Child Index : 3
Digit Node -> Index : 0, Value : 2
Operator Node -> Index : 3, Value : *, Left Child Index : 1,Right Child Index : 2
Digit Node -> Index : 1, Value : 3
Digit Node -> Index : 2, Value : 5
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045$ ./a.out
Enter an expression
2-3
Operator Node -> Index : 2, Value : -, Left Child Index : 0,Right Child Index : 1
Digit Node -> Index : 0, Value : 2
Digit Node -> Index : 1, Value : 3
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045$
```

Q4) Use YACC to convert: Infix expression to Postfix expression.

prog.l

```
%{
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ { yylval=atoi(yytext); return digit;}
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
}
```

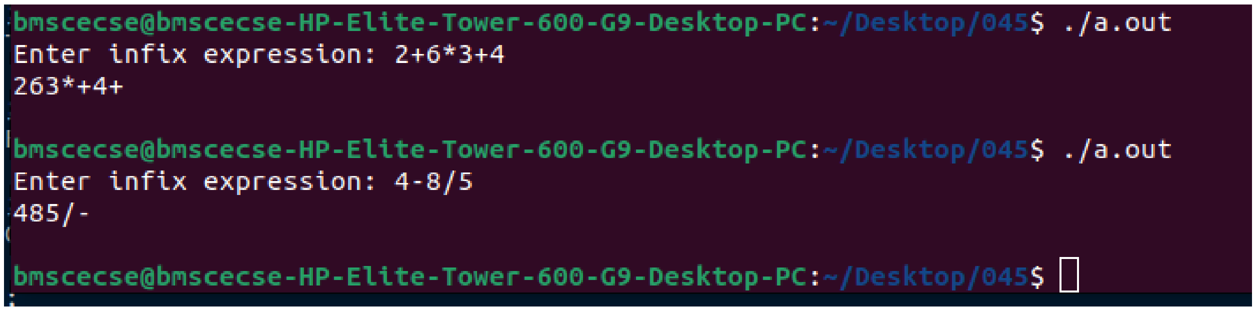
prog.y

```
%{
#include <ctype.h>
#include<stdio.h>
#include<stdlib.h>
%}
%token digit
%right '^'
%left '+' '-'
%left '*' '/'

%%
S: E {printf("\n\n");}
```

```
;
E: E '+' T { printf ("+");}
| T
;
E: E '-' T { printf ("-");}
| T
;
T: T '*' F { printf("*");}
| F
;
T: T '/' F { printf("/");}
| F
;
F: F '^' G { printf("^");}
| G
;
G: '(' E ')'
| digit {printf("%d", $1);}
;
%%
int main()
{
printf("Enter infix expression: ");
yyparse();
}
yyerror()
{
printf("Error");
```

}

OUTPUT:



```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045$ ./a.out
Enter infix expression: 2+6*3+4
263*+4+

bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045$ ./a.out
Enter infix expression: 4-8/5
485/-

bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/045$
```

Q5) Use YACC to generate 3-Address code for a given expression.

prog.l

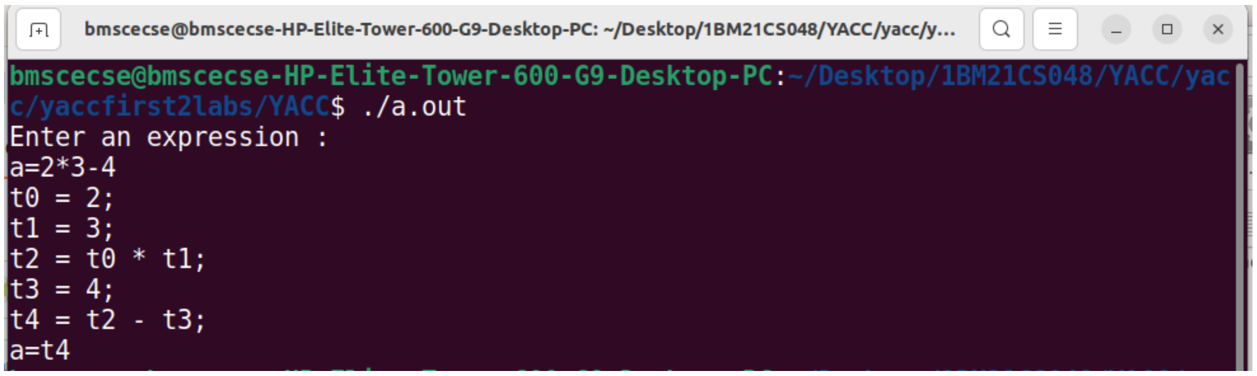
```
d [0-9]+
a [a-zA-Z]+
%{
#include<stdio.h>
#include<stdlib.h>
#include"y.tab.h"
extern int yylval;
extern char iden[20];
%}
%%
{d} { yylval=atoi(yytext); return digit; }
{a} { strcpy(iden,yytext); yylval=1; return id;}
[ \t] {;}
\n return 0;
. return yytext[0];
%%
int yywrap()
{
}
```

prog.y

```
%{
#include <math.h>
#include<ctype.h>
#include<stdio.h>
int var_cnt=0;
char iden[20];
%}
%token id
%token digit
%%
S:id '=' E { printf("%s=t%d\n",iden,var_cnt-1); }
E:E '+' T { $$=var_cnt; var_cnt++; printf("t%d = t%d + t%d;\n", $$, $1, $3 );
}
|E '-' T { $$=var_cnt; var_cnt++; printf("t%d = t%d - t%d;\n", $$, $1, $3 );
}
|T { $$=$1; }
;
T:T '*' F { $$=var_cnt; var_cnt++; printf("t%d = t%d * t%d;\n", $$, $1, $3 ); }
|T '/' F { $$=var_cnt; var_cnt++; printf("t%d = t%d / t%d;\n", $$, $1, $3 ); }
|F {$$=$1 ; }
F:P '^' F { $$=var_cnt; var_cnt++; printf("t%d = t%d ^ t%d;\n", $$, $1, $3 );}
| P { $$ = $1;}
;
P: '(' E ')' { $$=$2; }
|digit { $$=var_cnt; var_cnt++; printf("t%d = %d;\n",$$,$1); }
;
%%
int main()
{
var_cnt=0;
printf("Enter an expression : \n");
yyparse();
return 0;
}
```

```
yyerror()
{
printf("error");
}
```

OUTPUT:


bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC: ~/Desktop/1BM21CS048/YACC/yacc/y...

```
bmscecse@bmscecse-HP-Elite-Tower-600-G9-Desktop-PC:~/Desktop/1BM21CS048/YACC/yac
c/yaccfirst2labs/YACC$ ./a.out
Enter an expression :
a=2*3-4
t0 = 2;
t1 = 3;
t2 = t0 * t1;
t3 = 4;
t4 = t2 - t3;
a=t4
```