VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

COMPILER DESIGN

Submitted by

PALLE PADMAVATHI (1BM21CS125)

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
Oct 2022-Feb 2023

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled **"COMPILER DESIGN"** carried out by **PALLE PADMAVATHI(1BM21CS125)**, 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 year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of **Compiler Design Lab - (22CS5PCCPD)**work prescribed for the said degree.

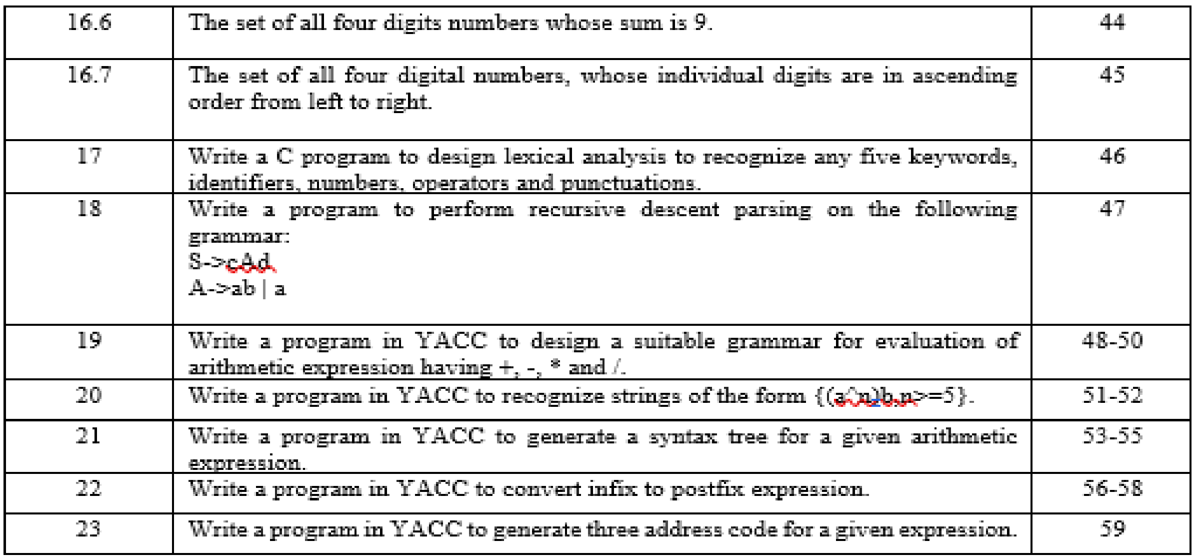
Sonika Sharma D	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

B. M. S. COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, Palle Padmavathi (1BM21CS125), student of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this lab report entitled " **Compiler Design**" has been carried out by me under the guidance of Prof. Prameetha Pai, 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.

TABLE OF CONTENTS



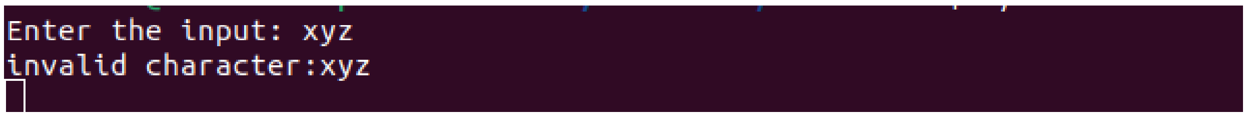
Course outcomes:

CO1	Apply the fundamental concepts for the various phases of compiler design.
CO2	Analyse 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.

1.Write a Lex program to find whether the given input character is valid or invalid.

```
%option noyywrap

%{

#include<stdio.h>

%}

%%

[0-9]+ {printf("number:%s\n",yytext);}

[+-] {printf("operator:%s\n",yytext);}

[ \t\n] {/*ignore whitespaces and newline*/}

[a-zA-Z]* {printf("invalid character:%s\n",yytext);}

%%

int main()

{

printf("Enter the input: ");

yylex();

return 0;

}
```

OUTPUT

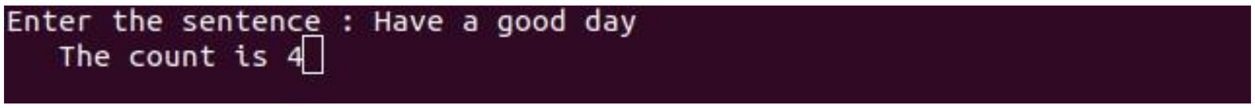


```
Enter the input: xyz
invalid character:xyz
```

2.Write a lex program to count number of characters in given input string.

```
%{
#include<stdio.h>
int c=0;
%}
%%
[a-zA-Z0-9]+ {c++;}
\n {printf("The count is %d",c);}
%%
int yywrap()
{
}
int main()
{
printf("Enter the sentence : ");
yylex();
return 0;
}
```

OUTPUT

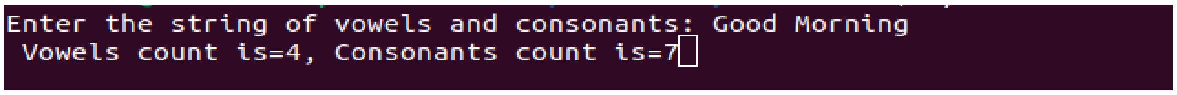


```
Enter the sentence : Have a good day
   The count is 4
```

3.Write a Lex program to count number of vowels and consonants in a sentence.

```
{
#include<stdio.h>
int vow_count=0;
int const_count=0;
%}
%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
\n {printf("Vowels count is=%d, Consonants count is=%d",vow_count,const_count);}
%%
int yywrap()
{
}
int main()
{
printf("Enter the string of vowels and consonants: ");
yylex();
return 0;}
```

OUTPUT

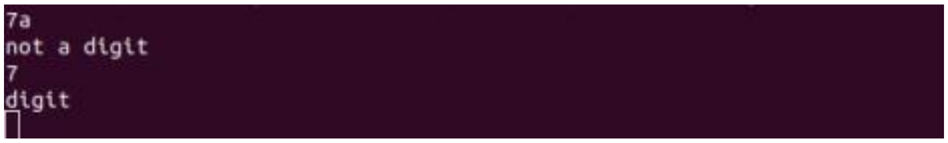


```
Enter the string of vowels and consonants: Good Morning
 Vowels count is=4, Consonants count is=7
```

4.Write a lex program to check whether input is digit or not.

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%%
^[0-9]* printf("digit");
^[^0-9]|[0-9]*[a-zA-Z] printf("not a digit");
.;
%%
int yywrap()
 {
}
int main()
 {
yylex();
return 0;
}
```

OUTPUT

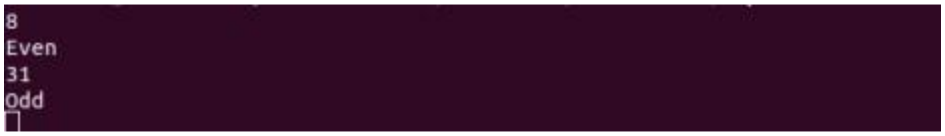


```
7a
not a digit
7
digit
```

5.Write a lex program to check whether the given number is even or odd.

```
 %{
#include<stdio.h>

int i;

%}

%%

[0-9]+ {i=atoi(yytext);

          if(i%2==0)

               printf("Even");

         else

         printf("Odd");}

%%

int yywrap(){}

int main()

 {

    yylex();

    return 0;

 }
```

OUTPUT:



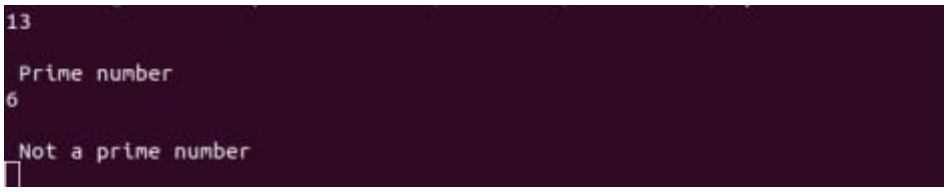
```
8
Even
31
Odd
```

6. Write a lex program to check whether a number is Prime or not.

```
%{
   #include<stdio.h>
   #include<stdlib.h>
   int flag,c,j;
%}
%%
[0-9]+ {c=atoi(yytext);
         if(c==2)
          {
           printf("\n Prime number");
          }
         else if(c==0 || c==1)
          {
           printf("\n Not a Prime number");
          }
         else
          {
           for(j=2;j<c;j++)
```

```
            {
           if(c%j==0)
             flag=1;
            }
           if(flag==1)
             printf("\n Not a prime number");
           else if(flag==0)
             printf("\n Prime number");
            }
          }
%%
int yywrap()
 {
 }
int main()
  {
  yylex();
  return 0;
  }
```

OUTPUT

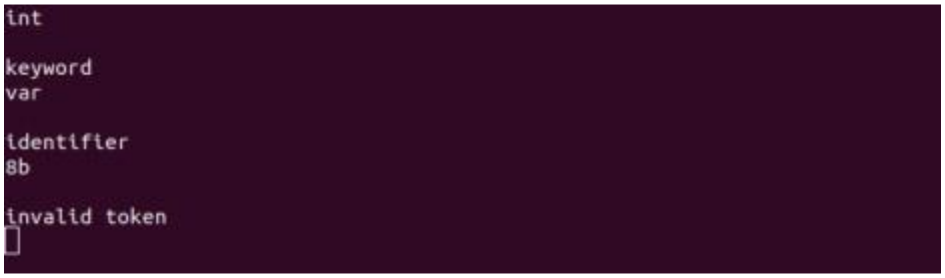


```
13

 Prime number
6

 Not a prime number
```

7 .i)Write a lex program to recognize a) identifier
b)keyword-int and float
c)anything else as invalid tokens

```
%{

   #include<stdio.h>

 %}

 alpha[a-zA-Z]

 digit[0-9]

 %%

 (float|int) {printf("\nkeyword");}

  {alpha}({digit}|{alpha})* {printf("\nidentifier");}

  {digit}({digit}|{alpha})* {printf("\ninvalid token");}

 %%

 int yywrap()

  {

  }

 int main()

  {

 yylex();

 return 0;

  }
```

OUTPUT:



```
int

keyword
var

identifier
8b

invalid token
```

7. **ii)Write a lex program to identify a) identifiers**

b) **keyword-int and float**

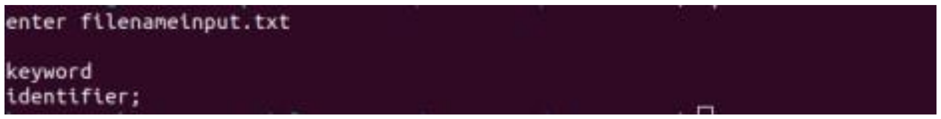
c) **anything else as invalid tokens**

Read these from a text file.

```
%{
  #include<stdio.h>
  char fname[25];
%}
alpha[a-zA-Z]
digit[0-9]
%%
(float|int) {printf("\nkeyword");}
 {alpha}({digit}|{alpha})* {printf("\nidentifier");}
 {digit}({digit}|{alpha})* {printf("\ninvalid token");}
%%
int yywrap()

}
int main()
 {
```

```
printf("enter filename");
scanf("%s",fname);
yyin=fopen(fname,"r");
yylex();
return 0;
fclose(yyin);
}
```

OUTPUT



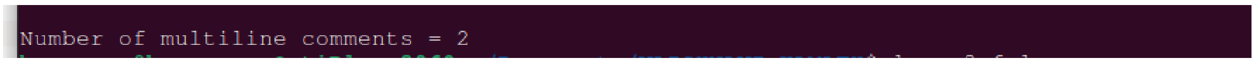
```
enter filenameinput.txt

keyword
identifier;
```

8. Lex program to count the number of comment lines (multi line comments or single line) in a program. Read the input from a file called input.txt and print the count in a file called output.txt

```
%{
#include <stdio.h>
int cc=0;
%}
%x CMNT
%%
"/*" {BEGIN CMNT;}
<CMNT>. ;
```

```
<CMNT>"*/" {BEGIN 0; cc++;}
%%
int yywrap() { }
int main(int argc, char *argv[])
{
if(argc!=3)
{
printf("Usage : %s <scr_file> <dest_file>\n",argv[0]);
return 0;
}
yyin=fopen(argv[1],"r");
yyout=fopen(argv[2],"w");
yylex();
printf("\nNumber of multiline comments = %d\n",cc);
return 0;
}
```

OUTPUT

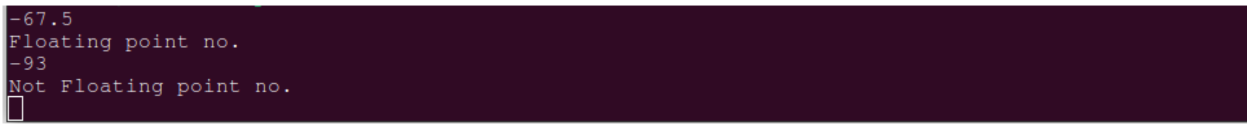


```
Number of multiline comments = 2
```

9.Write a program in LEX to recognize Floating Point Numbers. Check for all the following input cases

```
%{
#include<stdio.h>
int cnt=0;
%}
sign [+|-]
num [0-9]
dot [.]
%%
{sign}?{num}*{dot}{num}* {printf("Floating point no.");cnt=1;}
{sign}?{num}* {printf("Not Floating point no.");cnt=1;}
%%
int yywrap()
{ }
int main()
{
yylex();
if(cnt==0){
printf("Not floating pnt no.");
}
return 0;
}
```

OUTPUT

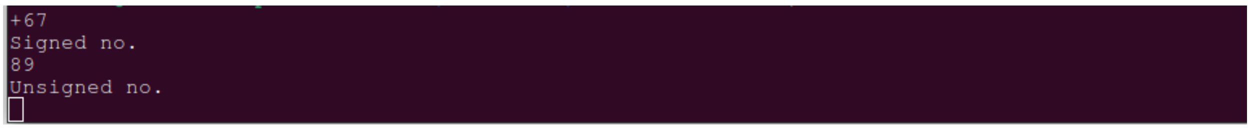


```
-67.5
Floating point no.
-93
Not Floating point no.
```

10.Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character

```
%{
#include<stdio.h>
int cnt=0;
%}
sign [+|-]
num [0-9]
dot [.]
%%
{sign}{num}*{dot}*{num}* {printf("Signed no.");cnt=1;}
{num}*{dot}*{num}* {printf("Unsigned no.");cnt=1;}
%%
int yywrap()
{
}
int main()
{
yylex();
if(cnt==0){
printf("Not floating pnt no.");
}
return 0;
}
```

OUTPUT



```
+67
Signed no.
89
Unsigned no.
```

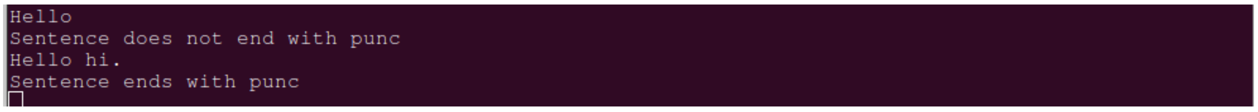
11.Write a program to check if the input sentence ends with any of the following

punctuation marks (? , fullstop , !)

```
%{
#include<stdio.h>
int cnt=0;
%}
punc [?|,|.|!]
chars [a-z|A-Z|0-9|" "|\t]
%%
{chars}*{punc} {printf("Sentence ends with punc");}
{chars}* {printf("Sentence does not end with punc");}
%%
int yywrap()
{
}
int main()
{
```

```
yylex();

return 0;

}
```

OUTPUT



```
Hello
Sentence does not end with punc
Hello hi.
Sentence ends with punc
```

12.a)Write a Lex program to find an article(a,an,the).

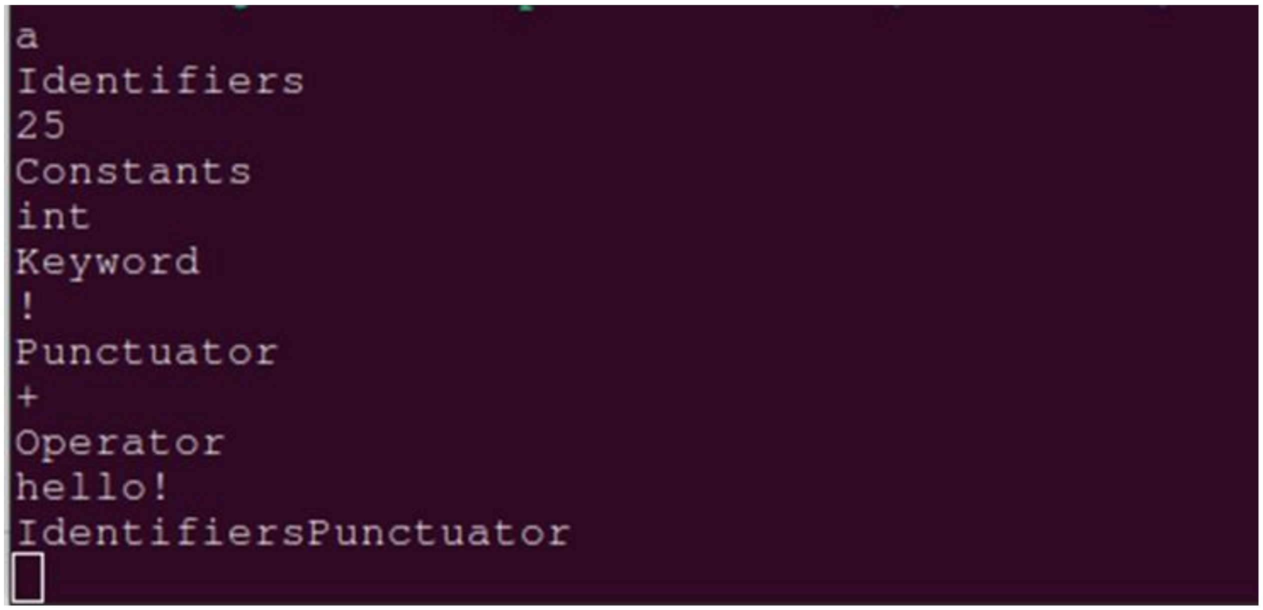
```
%{

#include<stdio.h>

int cnt=0;

%}

chars [a-z|A-Z|0-9]

check [A|a|AN|An|THE|The]

%%

{check}+{chars}* {printf("Begins with %s",yytext);}{chars}* {printf("Invalid");}

%%

int yywrap()

{

}

int main()

{

yylex();

return 0;
}
```

12. b).Write a program in LEX to recognize different tokes:Keywords, Identifiers, Constants, Operators and Punctuations?

```
%{

#include<stdio.h>

int cnt=0;

%}

letter [a-zA-Z]

digit [0-9]

punc [!|,|.]

oper [+|*|-|/|%]

boole [true|false]

%%

{digit}+|{digit}*.{digit}+ {printf("Constants");}

int|float {printf("Keyword");}

{letter}({digit}|{letter})* {printf("Identifiers");}

{oper} {printf("Operator");}

{punc} {printf("Punctuator");}
```

```
%%

int yywrap()
{
}

int main()
{
yylex();
return 0;
}
```

OUTPUT



```
a
Identifiers
25
Constants
int
Keyword
!
Punctuator
+
Operator
hello!
IdentifiersPunctuator
```

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

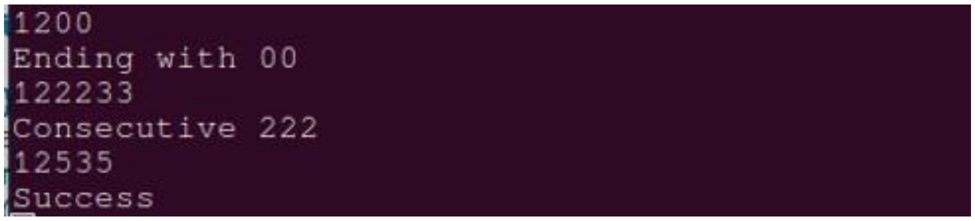
a) **The set of all string ending in 00.**

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.**

```
%{
#include<stdio.h>
int flag=0,i;
%}


letter [a-zA-Z]
digit [0-9]
A [0-9]
punc [!|,|.]
oper [+|*|-|/|%]
boole [true|false]
%%
{digit}*00 {printf("Ending with 00");}
{digit}*222{digit}* {printf("Consecutive 222");}
{A}{A}{A}{A}{A}  {
flag=0;
for(i=0;i<yyleng;i++){
if(yytext[i]=='5'){
flag=flag+1;
}
}
if(flag>=2){
printf("Success");
}
else{
printf("Failure");
}
}
%%
int yywrap()
{
}
int main()
{
yylex();
return 0;
}
```

OUTPUT



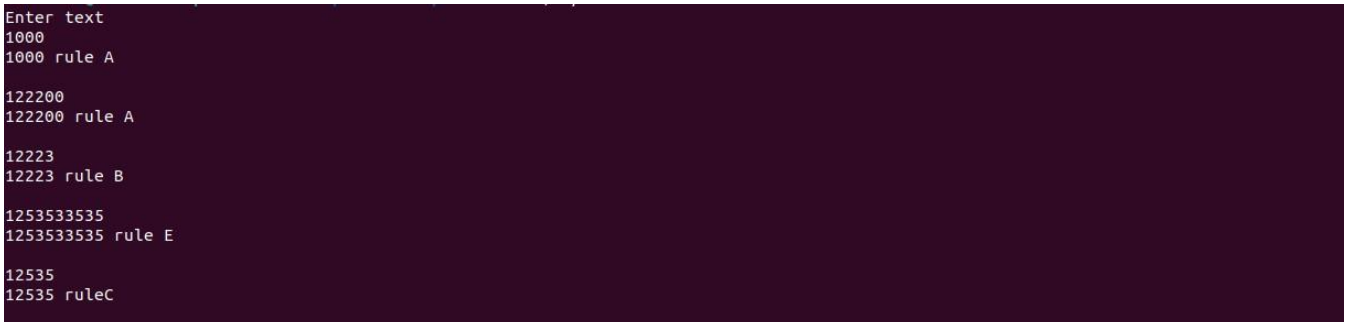
```
1200
Ending with 00
122233
Consecutive 222
12535
Success
```

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

```
 d[0-9]
%{
/* d is for recognising digits */
int c1=0,c2=0,c3=0,c4=0,c5=0,c6=0,c7=0;
/* c1 to c7 are counters for rules a1 to a7 */
%}
%%
({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})* {
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 ruleC\n",yytext); c3++; }
}
else
{
printf("%s doesn't match any rule\n",yytext);
}

}

}

%%

int yywrap()

{

}

int main()

{

printf("Enter text\n");

yylex();

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);

return 0;

}
```

OUTPUT:

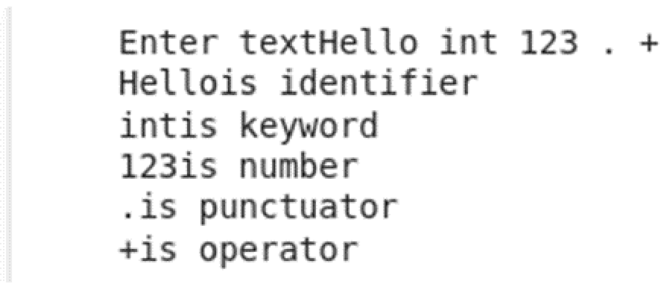


```
Enter text
1000
1000 rule A

122200
122200 rule A

12223
12223 rule B

1253533535
1253533535 rule E

12535
12535 ruleC
```

13.Write a Program to design Lexical Analyzer in C/C++/Java/python language(to recognize any five keywords,identifiers,numbers,operators and punctuation)

```
kwd=['int','float','char','if','else']

oper=['+','-','*','/','%']

punct=['.',',','!']

def func():

 txt=input("Enter text")

 txt=txt.split()

 for token in txt:

   if token in kwd:

      print(token + "is keyword")

   elif (token in oper):

      print(token + "is operator")

      elif(token in punct): print(token + "is punctuator")
      elif(token.isnumeric()): print(token + "is number")
      elif(not token[0].isnumeric()): print(token + "is identifier")
      else:
      print(token + "is not valid identifier") func()
```

OUTPUT

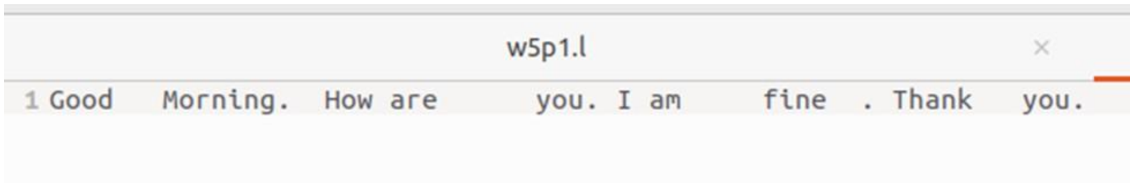


```
Enter textHello int 123 . +
Hellois identifier
intis keyword
123is number
.is punctuator
+is operator
```

14.Write a Lex Program that copies a file, replacing each nonempty sequence of whitespaces by a single blank.

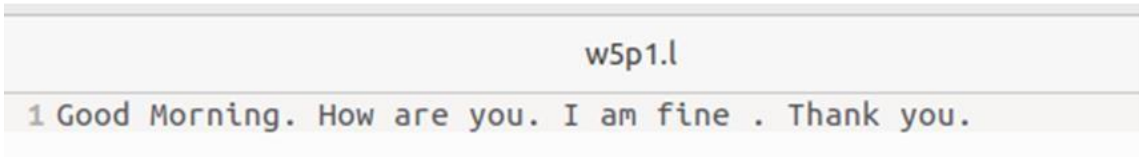
```
%{

#include<stdio.h>

%}

%%
[\t" "]+ fprintf(yyout," ");

.|\n fprintf(yyout,"%s",yytext);

%%

 int yywrap()

{

return 1;

}

int main(void)

{

yyin=fopen("input1.txt","r");

yyout=fopen("output.txt","w");

yylex();

return 0;

}
```

Input.txt



```
Good   Morning.  How are       you. I am      fine  . Thank   you.
```

Output.txt



```
Good Morning. How are you. I am fine . Thank you.
```

15 a) 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

<u>lex</u>

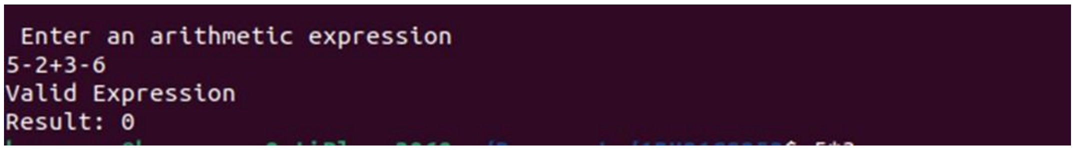
```
%{
#include "y.tab.h"
%}
%%
[0-9]+ {yylval=atoi(yytext); return NUM;}
[\t]    ;
\n      return 0;
.       return yytext[0];
%%
int yywrap()
{
}
```

<u>yacc</u>

```
%{
#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("\n Enter an arithmetic expression\n");
yyparse();
return 0;
}
int yyerror()
{
printf("\nInvalid expression\n");
return 0;
}
```

OUTPUT



```
 Enter an arithmetic expression
5-2+3-6
Valid Expression
Result: 0
```

15 b)Design a suitable grammar for evaluation of arithmetic expression having + , – , * , / , %, ^ operators.
^ having highest priority and right associative

% having second highest priority and left associative

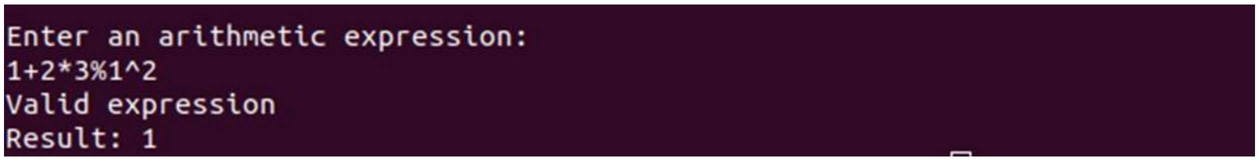
*** , / have third highest priority and left associative**

+ , - having least priority and left associative

```
%{

#include "y.tab.h"

%}

%%

[0-9]+  {yylval=atoi(yytext); return NUM;}

[\t]     ;

\n       return 0;

.        return yytext[0];

%%

int yywrap()

{

}

%{

#include<stdio.h>

%}

%token NUM

%left '+' '-'

%left '*' '/' '%'
```

```
%right '^'

%%

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     {

                    int result = 1;

                    for (int i = 0; i < $3; i++) {

                    result *= $1;

                  }

                      $$ = result;

                 }

      | NUM          {$$ = $1;}

        ;

       %%

    int main()

   {

      printf("\nEnter an arithmetic expression:\n");

       yyparse();

        return 0;
}

int yyerror()
{
    printf("\nInvalid expression\n");
     return 0;
 }
```

OUTPUT



```
Enter an arithmetic expression:
1+2*3%1^2
Valid expression
Result: 1
```

16.a)Program to recognize the grammar (anb, n>= 5).

Hint :S → aaaaaEb

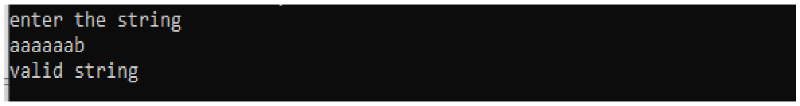
E →a E| €

p2.l

```
%{  #include "y.tab.h"  %}
%%
[aA] {return A;}
[bB] {return B;}
```

```
\n {return NL;}
.   {return yytext[0];}
%%
  int yywrap()
{
   return 1;
 }
p2.y
%{
#include<stdio.h>
  #include<stdlib.h>
%}
  %token A B NL
  %%
stmt: A A A A A S B NL {printf("valid string\n"); exit(0);}
;
S: S A
|  ;
%%
  int yyerror(char *msg)
 {
  printf("invalid string\n");
  exit(0);
 }
main()
 {
  printf("enter the string\n");
  yyparse();
 }
```

OUTPUT:



```
enter the string
aaaaaab
valid string
```

16 .b)Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using the grammar (anbn, n>= 0).

Hint : S → aSb | €

P3.l

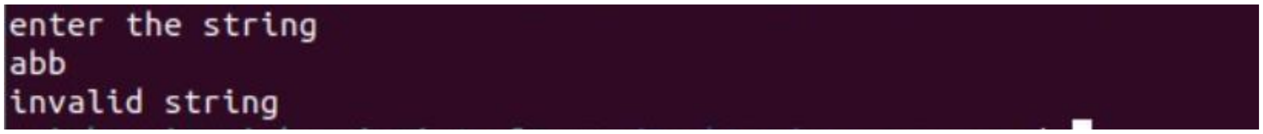
```
%{  #include "y.tab.h"  %}
%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
.   {return yytext[0];}
%%
  int yywrap()  {
   return 1;   }
```

P3.y

```
%{
```

```
#include<stdio.h>

#include<stdlib.h>

%}


%token A B NL


%%

stmt: S NL {printf("valid string\n"); exit(0);}

;

S: A S B

|  ;

%%

int yyerror(char *msg)

 {

  printf("invalid string\n");

  exit(0);

 }

main()

 {

  printf("enter the string\n");

  yyparse();

 }
```

OUTPUT



```
enter the string
abb
invalid string
```

3 **16 c) Write a YACC program to accept strings with exactly one a where Σ={a,b}**

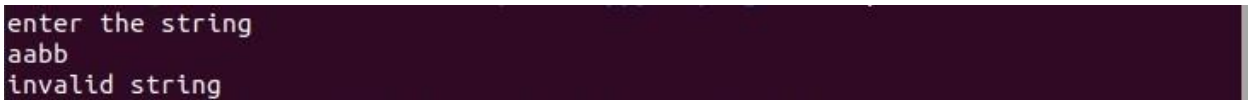
P4.l

```
%{  #include "y.tab.h"  %}
%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
.   {return yytext[0];}
%%
  int yywrap()  {
   return 1;   }
```

P4.y

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token A B NL
%%
stmt: S NL {printf("valid string\n"); exit(0);}
;
S: B S
| A X ;

X : B X |
;
%%
  int yyerror(char *msg)
 {
  printf("invalid string\n");
  exit(0);
 }
main()
 {
  printf("enter the string\n");
  yyparse();
 }
```

OUTPUT



```
enter the string
aabb
invalid string
```

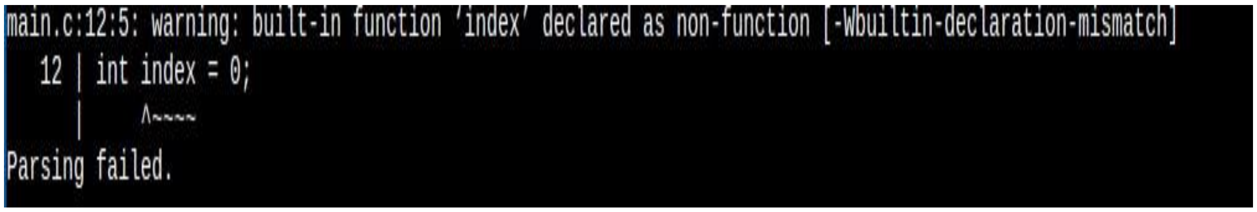
17.Recursive Descent Parsing with back tracking (Brute Force Method).

S->cAd,A>ab/a

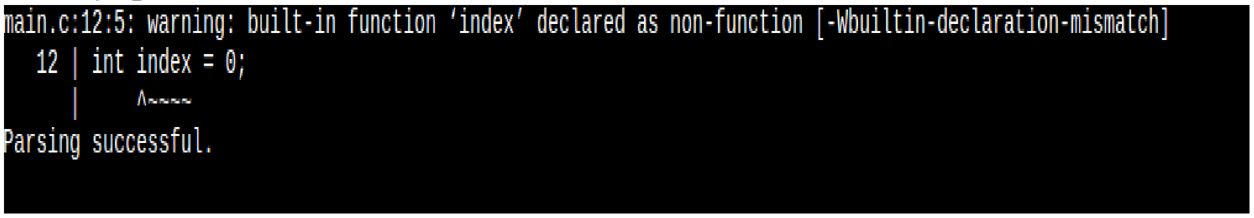
```
 #include <stdio.h>

int index = 0;

int parse_A(char input_str[]) {

    int current_index = index;

    if (input_str[index] == 'a') {

        index++;

        if (input_str[index] == 'b') {

            index++;

            return 1;

        } else {

            // Backtrack

            index = current_index;

            return 0;

        }

    } else if (input_str[index] == 'a') {

        index++;

        return 1;

    }

    return 0;
}

int parse_S(char input_str[]) {
    if (input_str[index] == 'c') {
        index++;
        if (parse_A(input_str)) {
            if (input_str[index] == 'd') {
                index++;
                return 1;
            }
        }
    }
    return 0;
}

void recursive_descent_parser(char input_str[]) {
    index = 0;
    if (parse_S(input_str) && input_str[index] == '\0') {
        printf("Parsing successful.\n");
    } else {
        printf("Parsing failed.\n");
    }
}
```

```
int main() {

    char input_string[] = "cabdc";

    recursive_descent_parser(input_string);



    return 0;

}
```

OUTPUT



```
main.c:12:5: warning: built-in function 'index' declared as non-function [-Wbuiltin-declaration-mismatch]
   12 | int index = 0;
      |     ^~~~~
Parsing failed.
```



```
main.c:12:5: warning: built-in function 'index' declared as non-function [-Wbuiltin-declaration-mismatch]
   12 | int index = 0;
      |     ^~~~~
Parsing successful.
```

18.Write a Yacc program to generate syntax tree for a given arithmetic expression

p1.l

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

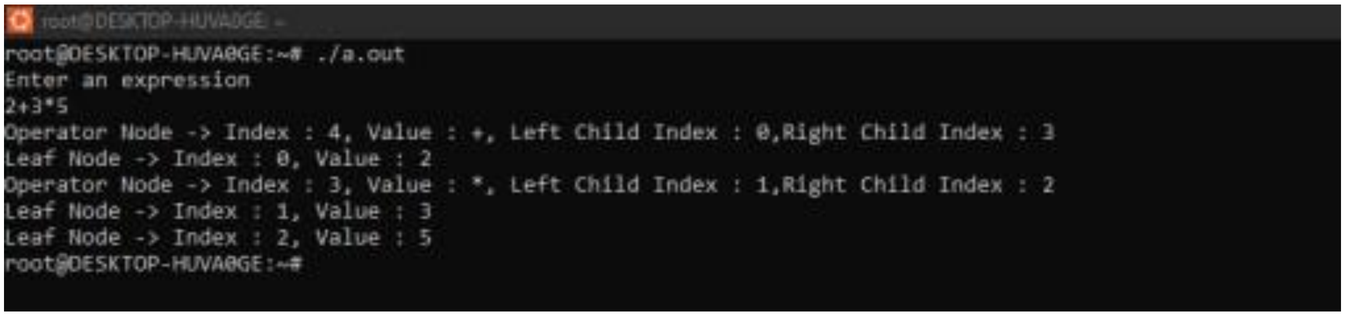
```
{
}
```

p1.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
%%
S:E { my_print_tree($1); }
 ;
E:E'+'T { $$= mknode($1,$3,"+"); ; }
|T { $$=$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



```
root@DESKTOP-HUVA0GE: ~
root@DESKTOP-HUVA0GE:~# ./a.out
Enter an expression
2+3*5
Operator Node -> Index : 4, Value : +, Left Child Index : 0,Right Child Index : 3
Leaf Node -> Index : 0, Value : 2
Operator Node -> Index : 3, Value : *, Left Child Index : 1,Right Child Index : 2
Leaf Node -> Index : 1, Value : 3
Leaf Node -> Index : 2, Value : 5
root@DESKTOP-HUVA0GE:~#
```

19.Use YACC to convert: Infix expression to Postfix expression.

p4.l

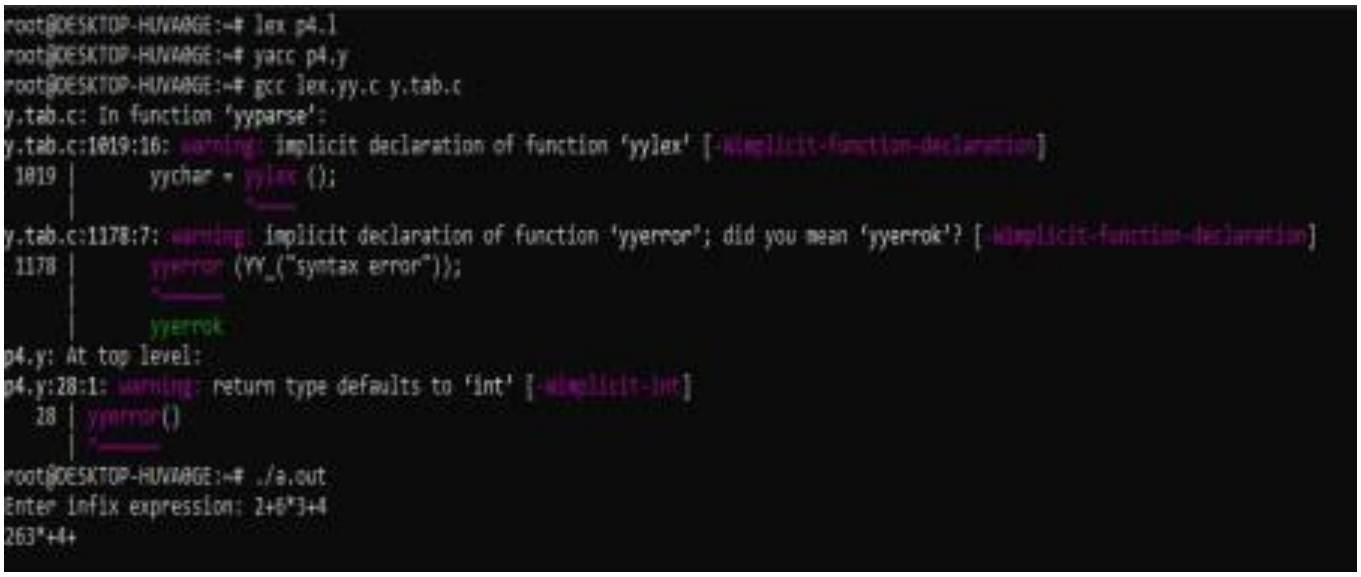
```
%{
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ { yylval=atoi(yytext); return digit;}
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
}
```

p4.y

```
%{
#include <ctype.h>
#include<stdio.h>
#include<stdlib.h>
%}
%token digit
%%
S: E {printf("\n\n");}
;
E: E '+' T { printf ("+");}
| T
;
T: T '*' F { printf("*");}
| F
```

```
;
F: '(' E ')'
| digit {printf("%d", $1);}
;
%%
int main()
{
printf("Enter infix expression: ");
yyparse();
}
yyerror()
{
printf("Error");
}
```

OUTPUT

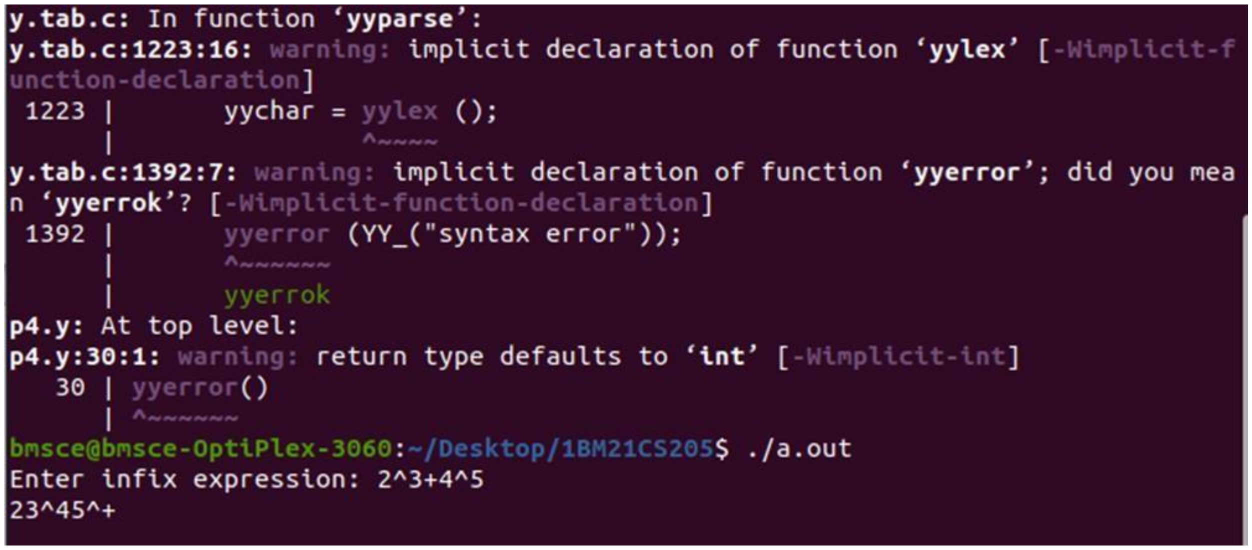


```
root@DESKTOP-HUVA0GE:~# lex p4.l
root@DESKTOP-HUVA0GE:~# yacc p4.y
root@DESKTOP-HUVA0GE:~# gcc lex.yy.c y.tab.c
y.tab.c: In function 'yyparse':
y.tab.c:1019:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
 1019 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1178:7: warning: implicit declaration of function 'yyerror'; did you mean 'yyerrok'? [-Wimplicit-function-declaration]
 1178 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
p4.y: At top level:
p4.y:28:1: warning: return type defaults to 'int' [-Wimplicit-int]
   28 | yyerror()
      | ^~~~~~~
root@DESKTOP-HUVA0GE:~# ./a.out
Enter infix expression: 2+6*3+4
263*+4+
```

20.Modify the program so as to include operators such as / , - , ^ as per their arithmetic associativity and precedence

```
%{
#include <ctype.h>
#include<stdio.h>
#include<stdlib.h>
%}
%token digit
%left '+' '-'
%left '*' '/'
%right '^'
%%
S: E {printf("\n\n");}
;
E: E '+' T { printf ("+");}
|E '-' T { printf ("-");}
| T
;
T: T '*' G { printf("*");}
|T '/' G{ printf("/");}
| G
;
G: G'^'F  { printf("^");}
|F
;
F: '(' E ')'
| digit {printf("%d", $1);}
;
%%
int main()
{
printf("Enter infix expression: ");
yyparse();
}
yyerror()
{
printf("Error");
}
}
```

OUTPUT



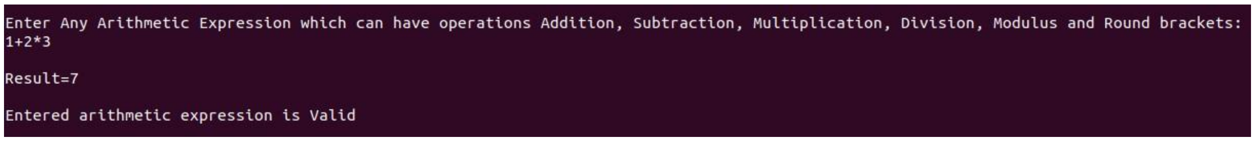
```
y.tab.c: In function ‘yyparse’:
y.tab.c:1223:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-f
unction-declaration]
 1223 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1392:7: warning: implicit declaration of function ‘yyerror’; did you mea
n ‘yyerrok’? [-Wimplicit-function-declaration]
 1392 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
p4.y: At top level:
p4.y:30:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   30 | yyerror()
      | ^~~~~~~
bmsce@bmsce-OptiPlex-3060:~/Desktop/1BM21CS205$ ./a.out
Enter infix expression: 2^3+4^5
23^45^+
```

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

```
%{
  /* 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;
}

token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
/* Rule Section */
%%
ArithmeticExpression: E{
            printf("\nResult=%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')' {$$=$2;}

 | NUMBER {$$=$1;}

 ;

%%

//driver code

void main()

{

   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction,
Multiplication, Division, Modulus and Round brackets:\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 which can have operations Addition, Subtraction, Multiplication, Division, Modulus and Round brackets:
1+2*3

Result=7

Entered arithmetic expression is Valid
```

22.YACC to generate 3-Adress code for given

expression.

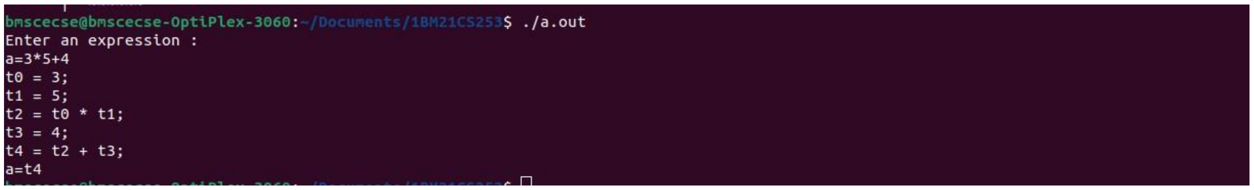
```
 p.l
%{
#include<stdio.h>
#include<stdlib.h>
#include"y.tab.h"
extern int yylval;
extern char iden[20];
%}
d [0-9]+
a [a-zA-Z]+
%%
{d} { yylval=atoi(yytext); return digit; }
{a} { strcpy(iden,yytext); yylval=1; return id;}
[ \t] {;}
\n return 0;
. return yytext[0];
%%
int yywrap()
{
}
```

P.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-OptiPlex-3060:~/Documents/1BM21CS253$ ./a.out
Enter an expression :
a=3*5+4
t0 = 3;
t1 = 5;
t2 = t0 * t1;
t3 = 4;
t4 = t2 + t3;
a=t4
```