**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**Compiler Design**

***Submitted by***

**VIBHA HUGAR (1BM21CS255)**

***in partial fulfilment 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**

**Nov -2023 to Feb-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  **VIBHA HUGAR (1BM21CS255),** 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 Nov -2023 to Feb-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.

Dr. Latha N R             Dr. Jyothi S Nayak

Assistant Professor                             Professor and Head

Department of CSE                 Department of CSE

BMSCE, Bengaluru                 BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Lab Program No.** | **PROGRAM** | **Page No.** |
| 1 | WEEK-1 | 4 |
| 2 | WEEK-2 | 8 |
| 3 | WEEK-3 | 14 |
| 4 | WEEK-4 | 20 |
| 5 | WEEK-5 | 24 |
| 6 | WEEK-6 | 27 |
| 7 | WEEK-7 | 31 |
| 8 | WEEK-7 | 38 |
| 9 | WEEK-9 | 41 |

**Course Outcome**

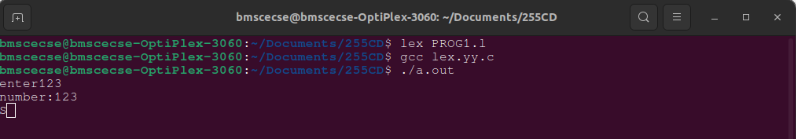
|  |  |
| --- | --- |
| CO1 | Apply the fundamental concepts for the various phases of compiler design. |
| CO2 | CO2 Analyse the syntax and semantic concepts of a compiler. |
| CO3 | Design various types of parsers and Address code generation |
| CO4 | Implement compiler principles, methodologies using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

**WEEK-1**

**1.Write a LEX program to identify numbers and operators from input.**

%option noyywrap  
%{  
#include<stdio.h>  
%}  
%%  
O[+-]  {printf("operator:%s\n",yytext);}  
[ \t\n]  {/\*ignore whitespaces and newline\*/}  
[a-zA-Z]\*  {printf("invalid character:%s\n",yytext);}  
%%  
  
int main()  
{  
printf("enter");  
yylex();  
return 0;  
}

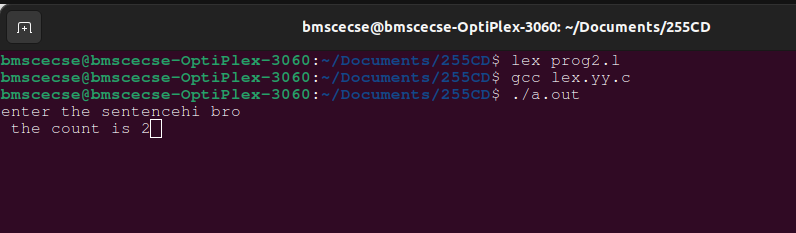
OUTPUT



**2. Write LEX a program to identify the number of words in the sentence.**

%{  
#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;  
}

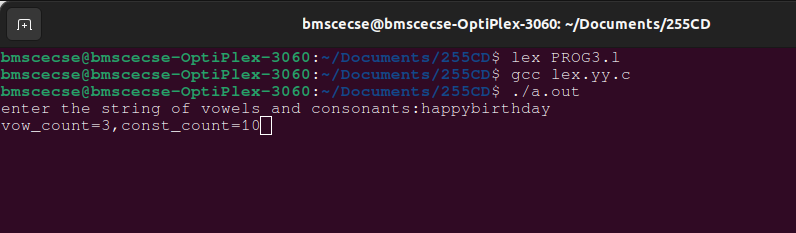
OUTPUT



**3. Write a LEX program to give the 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("vow\_count=%d,const\_count=%d",vow\_count,const\_count);}  
%%  
int yywrap()  
{  
}  
int main()  
{  
printf("enter the string of vowels and consonants:");  
yylex();  
return 0;  
}

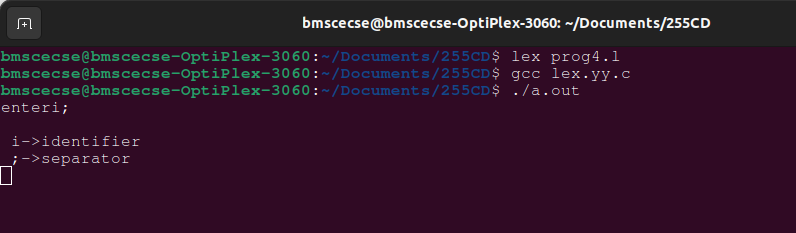
OUTPUT



**4. Write a LEX program to identify keywords, separator and identifiers.**

%option noyywrap  
%{  
#include<stdio.h>  
%}  
%%  
int|char|float {printf("\n%s->keyword",yytext);}  
,|; {printf("\n %s->separator",yytext);}  
[a-zA-Z0-9]\* {printf("\n %s->identifier",yytext);}  
%%  
int wrap()  
{  
}  
int main()  
{  
printf("enter");  
yylex();  
return 0;  
}

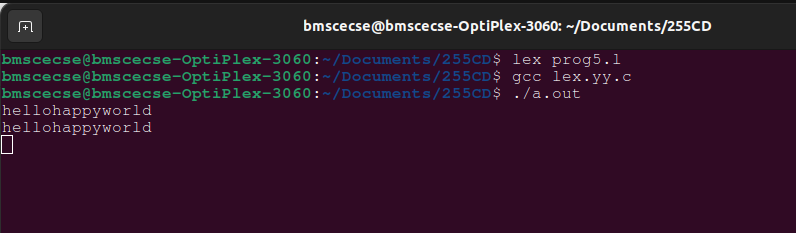
OUTPUT



**5. Write a LEX program to print the input given.**

%%  
. ECHO;  
%%  
int yywrap(void)  
{  
}  
int main(void)  
{  
yylex();  
return 0;  
}

OUTPUT



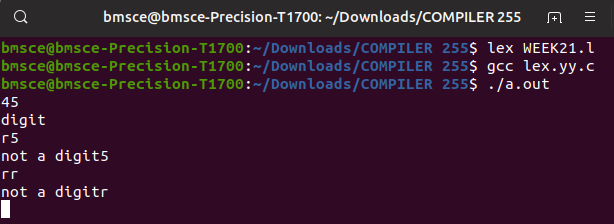
**WEEK-2**

**1. 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

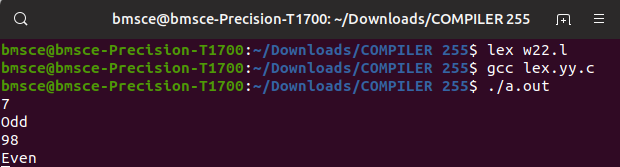


**2. Write a lex program to check whether the given number is even or odd.**

%{  
#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

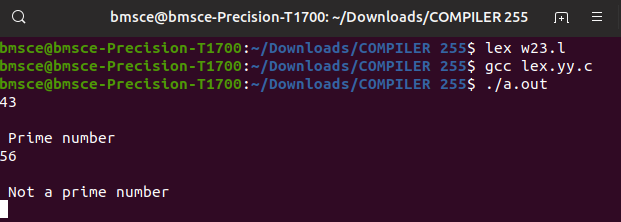


**3. Write a lex program to check whether a number is Prime or not.**

%{  
     
   #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



**4. Write a lex program to recognize a) identifiers**

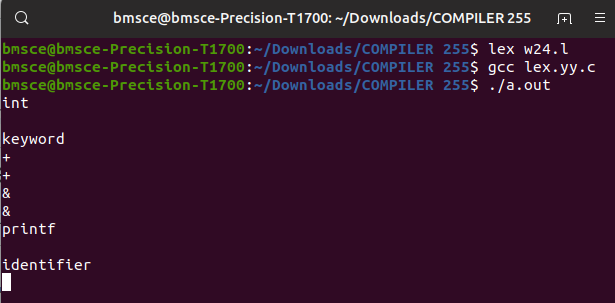
**b) keyword-int and float**

**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



**5. Write a lex program to identify a) identifiers**

**b) keyword-int and float**

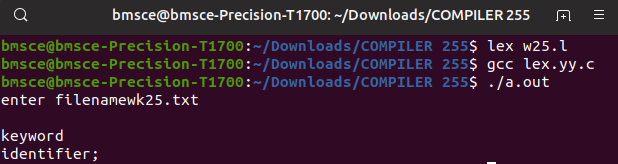
**c) anything else as invalid tokens**

**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



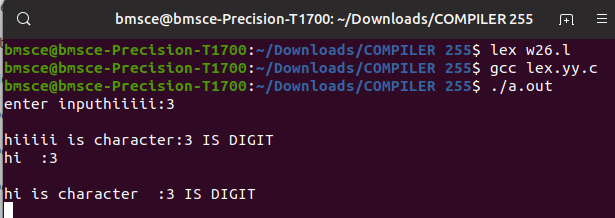
**6)Write a Program to print invalid string if a Alpha-Numeric string is entered as input.**

%{

#include<stdio.h>  
%}  
alpha [a-zA-Z0-9]\*  
%%  
[0-9]\* {printf("%s IS DIGIT",yytext);}  
[a-zA-Z]\* {printf("\n%s is character",yytext);}  
{alpha} {printf("invalid string");}  
%%  
int yywrap()  
{  
}  
int main()  
{  
printf("enter input");  
yylex();  
return 0;

}

OUTPUT



**WEEK-3**

**1.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;

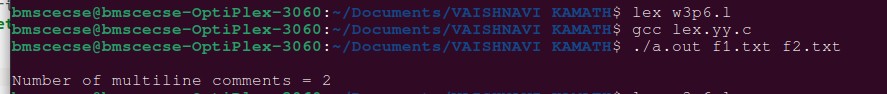
}

yyin=fopen(argv[1],"r");

yyout=fopen(argv[2],"w"); yylex();

printf("\nNumber of multiline comments = %d\n",cc); return 0;

}OUTPUT



**2.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){

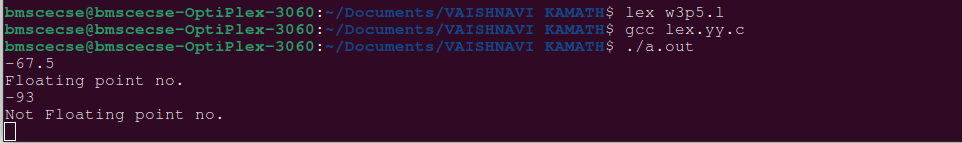
printf("Not ﬂoating pnt no.");

}

return 0;

}

OUTPUT



**3.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){

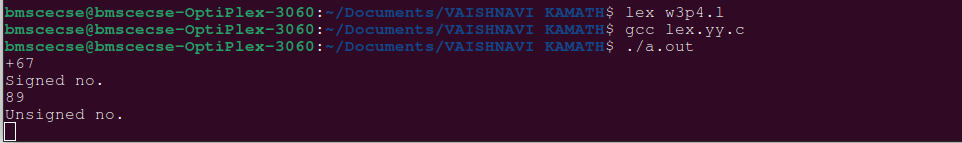
printf("Not ﬂoating pnt no.");

}

return 0;

}

OUTPUT



**4.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()

{

}

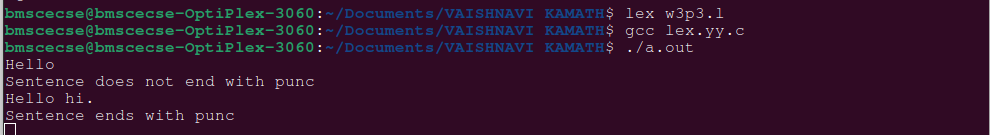
int main()

{

yylex(); return 0;

}

OUTPUT



**5. Write a program to read an input sentence and to check if the sentence begins with**

**English articles (A, a,AN,An,THE and The). If the sentence starts with the article**

**appropriate message should be printed. If the sentence does not start with the article**

**appropriate message should be printed.**

%{

#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;}

**WEEK-4**

**1.Write a program in LEX to recognize different tokes:Keywords, Identifiers, Constants, Operators and Punctuations.**

CODE

%{

   #include<stdio.h>

%}

alpha[a-zA-Z]

digit[0-9]

%%

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

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

[+|-|\*|/] {printf("\n operator");}

[0-9]+ {printf("\n constants");}

[?|.|!] {printf("\n punctuation");}

%%

int yywrap()

{

}

int main()

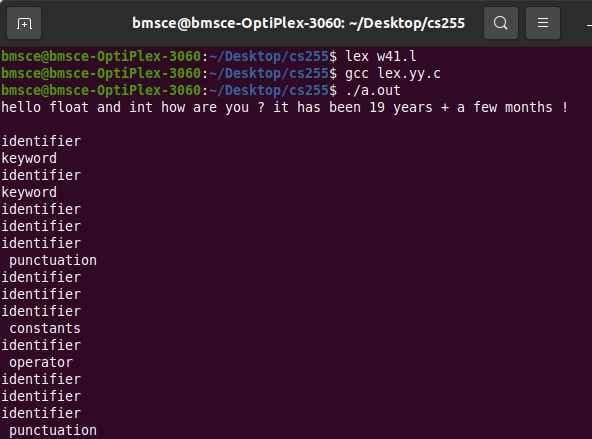
{

yylex();

return 0;

}

OUTPUT



**2. 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.**

CODE

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);

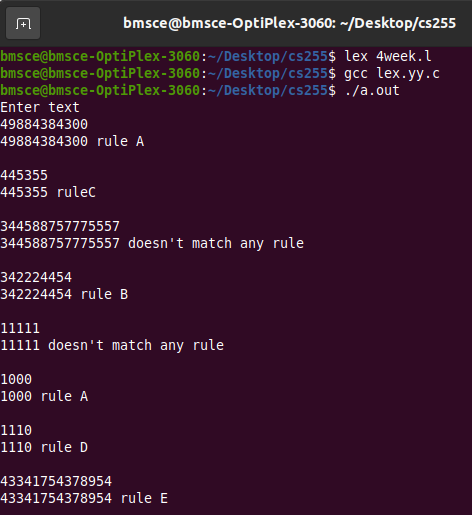
printf("Rule D : %d \n",c4);

printf("Rule E : %d \n",c5);

return 0;

}

OUTPUT



**WEEK-5**

**1.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):

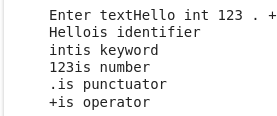
print(token + "is punctuator") elif(token.isnumeric()):

print(token + "is number") elif(not token[0].isnumeric()):

print(token + "is identifier") else:

print(token + "is not valid identifier")

func()



**2.Write a Lex Program that copies a file,replacing each nonempty sequence of white spaces by a single blank.**

%{

#include<stdio.h>

%}

%%

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

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

%%

int yywrap()

{

return 1;

}

int main(void)

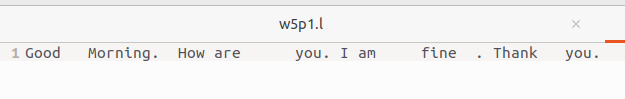
{

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

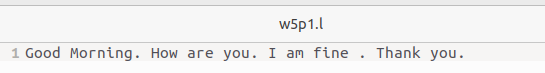
yylex();

return 0;

Input.txt



Output.txt



**WEEK-6**

**1.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**

**CODE**

**LEX**

%{

#include "y.tab.h"

%}

%%

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

[\t] ;

\n return 0;

. return yytext[0];

%%

int yywrap()

{

}

**YACC**

%{

#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("\Enter an arithmetic expression\n");

yyparse();

return 0;

}

int yyerror()

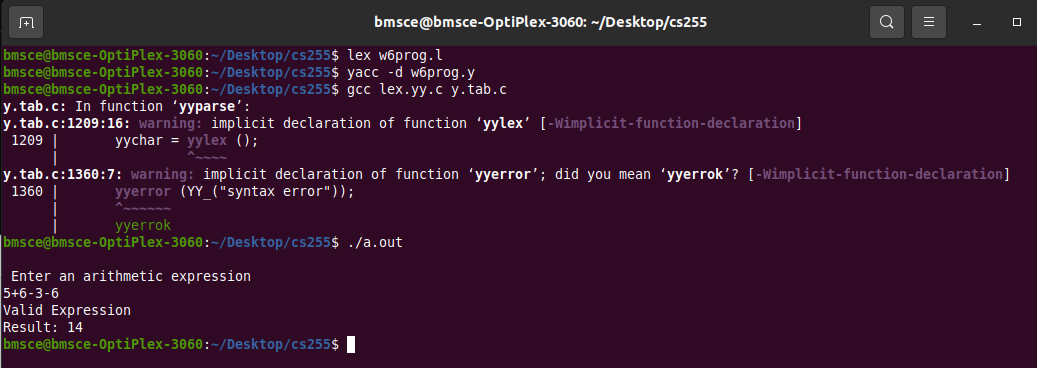
{

printf("\nInvalid expression\n");

return 0;

}

**OUTPUT**



**2.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**

**CODE**

**LEX**

%{

#include "y.tab.h"

%}

%%

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

[\t] ;

\n return 0;

. return yytext[0];

%%

int yywrap()

{

}

**YACC**

%{

#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()

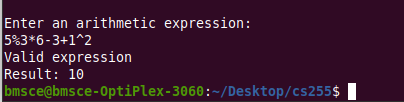
{

    printf("\nInvalid expression\n");

    return 0;

}

**OUTPUT**



**WEEK-7**

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

**Hint :S → aaaaaEb**

**E →a E| €**

**CODE:**

**LEX**

%{

#include "y.tab.h"

%}

%%

[aA] {return A;}

[bB] {return B;}

\n {return NL;}

.   {return yytext[0];}

%%

  int yywrap()

{

   return 1;

}

**YACC**

%{

#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()

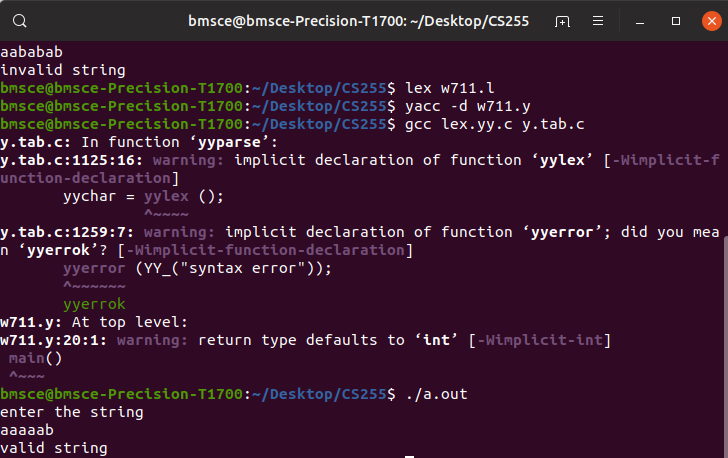
 {

  printf("enter the string\n");

  yyparse();

 }

**OUTPUT**

****

**1b)Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using  the  grammar  (anbn, n>= 0).**

**Hint : S → aSb | €**

**CODE:**

**LEX**

%{

#include "y.tab.h"

%}

%%

[aA] {return A;}

[bB] {return B;}

\n {return NL;}

.   {return yytext[0];}

%%

  int yywrap()

{

  return 1;

}

**YACC**

%{

#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()

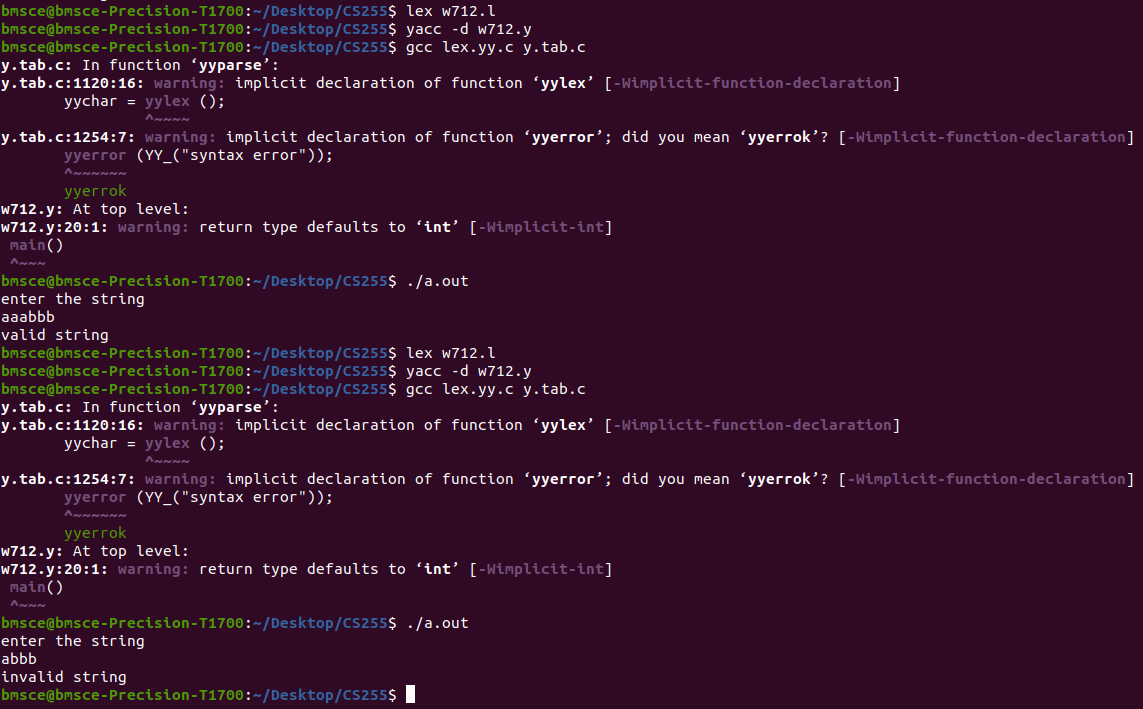
 {

  printf("enter the string\n");

  yyparse();

 }

**OUTPUT**

****

**2) Recursive Descent Parsing with back tracking(Brute Force Method).S->cAd,A->ab/a**

**CODE:**

#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() {

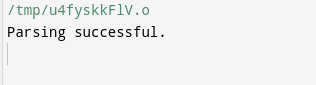
    char input\_string[] = "cabd";

    recursive\_descent\_parser(input\_string);

    return 0;

}

**OUTPUT**

****

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

**CODE:**

**LEX**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

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

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

}

**YACC**

%{

#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;

}

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

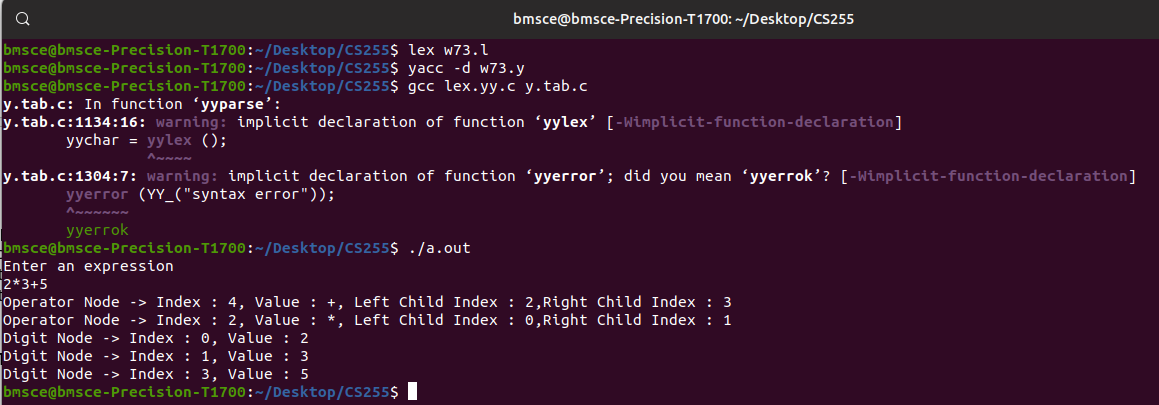
printf("Operator Node -> Index : %d, Value : %s, Left Child Index : %d,Right Child Index : %d\n",cur\_ind,syn\_tree[cur\_ind].val, syn\_tree[cur\_ind].lc,syn\_tree[cur\_ind].rc);

my\_print\_tree(syn\_tree[cur\_ind].lc);

my\_print\_tree(syn\_tree[cur\_ind].rc);

}

**OUTPUT**

****

**WEEK-8**

**1. Use YACC to convert: Infix expression to Postfix expression.**

**CODE**

**LEX**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

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

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

}

**YACC**

%{

#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();

}

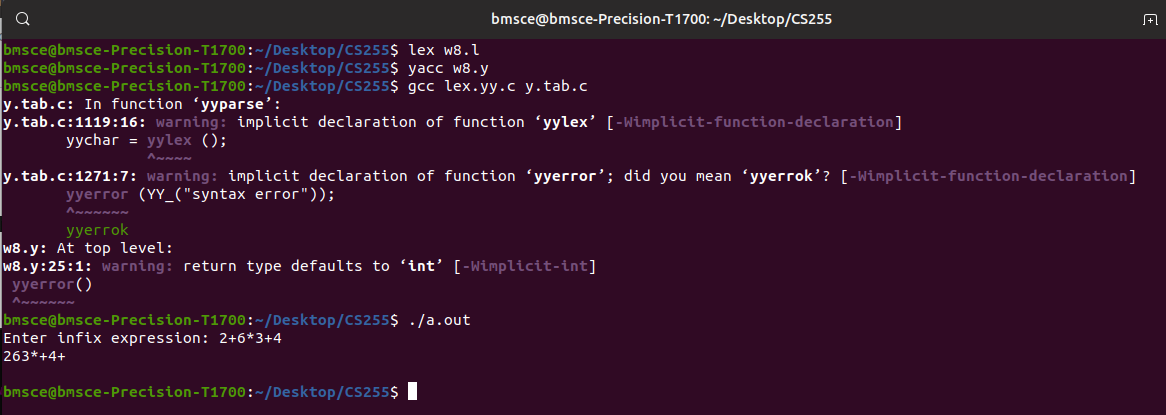
yyerror()

{

printf("Error");

}

**OUTPUT**

****

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

**CODE**

**LEX**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

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

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

}

**YACC**

%{

#include <ctype.h>

#include<stdio.h>

#include<stdlib.h>

%}

%token digit

%%

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

;

E:  E '+'  T { printf ("+");}

|  E '-'  T { printf ("-");}

|  T

;

T:  T '\*'  P { printf("\*");}

|  T '/'  P { printf("/");}

|  P

;

P:  F '^'  P { printf ("^");}

|  F

;

F: '(' E ')'| digit {printf("%d", $1);}

;

%%

int main()

{

printf("Enter infix expression: ");

yyparse();

}

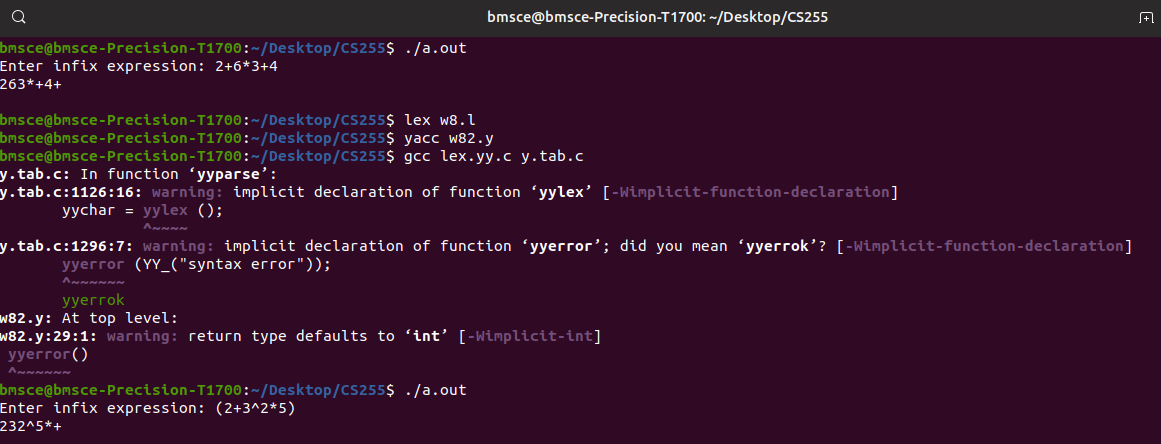
yyerror()

{

printf("Error");

}

**OUTPUT**

****

**WEEK-9**

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

**CODE**

**LEX**

%{

#include "y.tab.h"

#include <stdlib.h>

extern int yylval;

%}

%%

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

[\t] ;

[\n] return 0;

. return yytext[0];

%%

**YACC**

%{

 #include <stdio.h>

 #include <ctype.h>

 int x[5],y[5],k,j[5],a[5][10],e,w;

%}

%token digit

%%

S : E { printf("\nAnswer : %d\n",$1); }

 ;

E : T { x[e]=$1; } E1 { $$=x[e]; }

 ;

E1 : '+' T { w=x[e]; x[e]=x[e]+$2; printf("Addition Operation %d and %d : %d\n",w,$2,x[e]); } E1 { $$=x[e]; }

 | '-' T { w=x[e]; x[e]=x[e]-$2; printf("Subtraction Operation %d and %d : %d\n",w,$2,x[e]); } E1 { $$=x[e]; }

 | { $$=x[e]; }

 ;

T : Z { y[e]=$1; } T1 { $$=y[e]; }

 ;

T1 : '\*' Z { w=y[e]; y[e]=y[e]\*$2; printf("Multiplication Operation of %d and %d : %d\n",w,$2,y[e]); } T1 { $$=y[e]; }

 | { $$=y[e]; }

 ;

Z : F { a[e][j[e]++]=$1; } Z1 { $$=$3; }

 ;

Z1 : '^' Z { $$=$2; }

 | { for(k=j[e]-1;k>0;k--) { w=a[e][k-1]; a[e][k]=powr(a[e][k-1],a[e][k]); printf("Power Operation %d ^ %d :

%d\n",w,a[e][k],a[e][k-1]); } $$=a[e][0]; j[e]=0; }

 ;

F : digit { $$=$1; printf("Digit : %d\n",$1); }

 | '(' { e++; } E { e--; } ')' { $$=$3; }

2

 ;

%%

int main()

{

//initializing all the variables to zero

 for(e=0;e<5;e++) { x[e]=y[e]=0; j[e]=0; }

 e=0;

// takes input as a expression

 printf("Enter an expression\n");

 yyparse();

 return 0;

}

// if any error yyerror will be called

yyerror()

{

 printf("NITW Error");

}

// when the input is finished yywrap is called to exit the code

int yywrap()

{

 return 1;

}

// power function to calculate m ^ n

int powr(int m,int n)

{

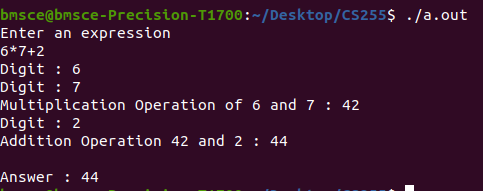
 int ans=1;

 while(n) { ans=ans\*m; n--; }

 return ans;

}

**OUTPUT**

****

**2.YACC to generate 3-Address code for given expression.**

**CODE**

**LEX**

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()

{

}

**YACC**

%{

#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;

}

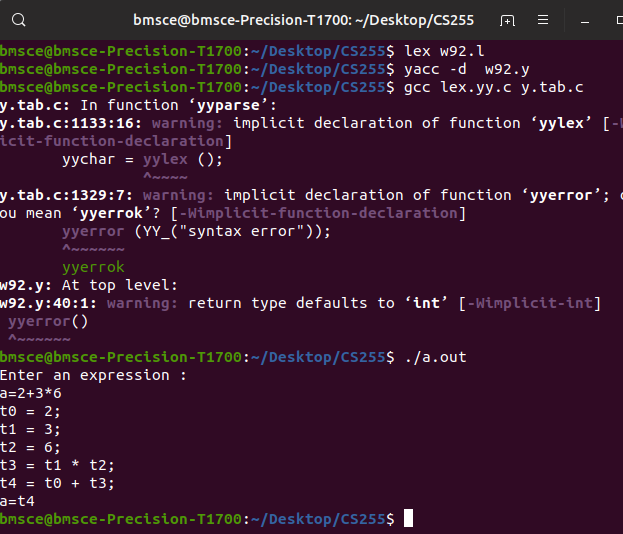
yyerror()

{

printf("error");

}

**OUTPUT**

****