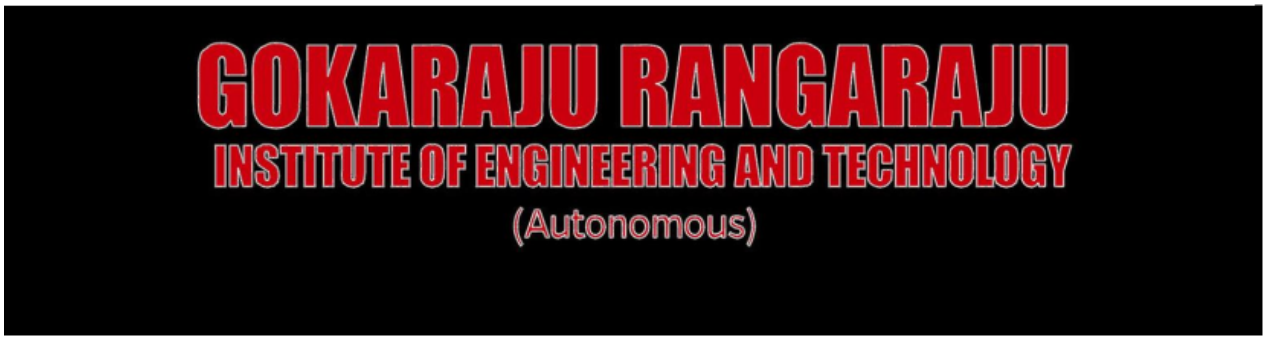
GR20 Regulations

IV B.Tech I Semester

Compiler Design Lab

(GR20A4055)

Department of Computer Science and Engineering



COMPILER DESIGN LAB

SYLLABUS

COURSE CODE: GR20A4055

Marks Int: 30 Extn: 70 **L:0 T:0 P:4 C: 2**

COURSE OBJECTIVES

1. To introduce the major concept areas of language translation and compiler design.
2. To understand practical programming skills necessary for constructing a compiler.
3. To learn parsing techniques and to parse given string.
4. To learn lex& yacc tool to develop a scanner & parser.
5. To provide deeper insights into the concept of code generation.

COURSE OUTCOMES

1. Demonstrate different phases of compiler through programming language.
2. Define the role of lexical analyser and use of regular expressions.
3. Develop program for implementing parsing techniques.
4. Understand the working of lex and yacc compiler and develop simple applications.
5. Design programs that execute faster by using code optimization techniques.

SYLLABUS

1. Design a lexical analyser for given language (ignore redundant spaces, tabs,comments new lines etc..)
2. Write a program to recognize strings under 'a*', 'a*b+', 'abb'.
3. Implement symbol table formation.
4. Write a program to implement predictive parser table.
5. Write a Program To Compute FIRST() and FOLLOW() Functions.
6. Construct operator precedence parser.
7. Write a program to parse a string using Shift Reduce Parser.
8. Solve the given string using LALR parser.
9. Write a program to implement lexical analyzer functionalities using LEX tool.
10. Design a simple arithmetic calculator using LEX .
11. Lex program to count no of characters, words, lines and special characters in a file
12. Implement code optimization technique.

INDEX

TASK 1

TASK-1: Design a lexical analyzer for given grammar

Aim : To design a lexical analyzer for given grammar

Program:

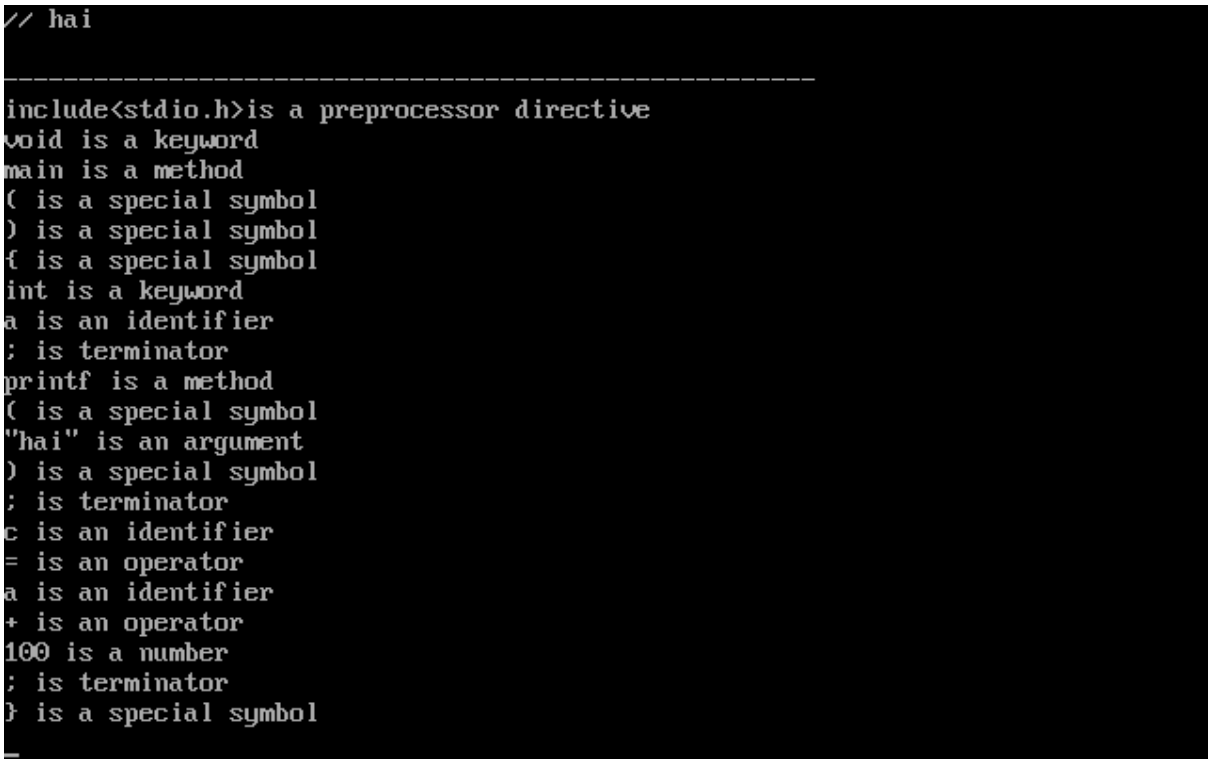
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

void main(void)
{
 char *s=(char *)calloc(sizeof(char),20000);
 char *t=(char *)calloc(sizeof(char),20000);
 char
key[32][12]={"auto","break","case","char","const","continue","default","do","double"
,"else","enum","extern","float","for","goto","if","int","long","register","return","short
","signed","sizeof","static","struct","switch","typedef","union","unsigned","void","vol
atile","while"};
 inti,j,k=0,c;
  char f[20],ch;
  FILE *fp;
 //clrscr();
 printf("enter a file: ");
 scanf("%s",f);
 fp=fopen(f,"r");
 if(fp==NULL)
    printf("file cannot be opened");
 else
 {
  while((ch=fgetc(fp))!=EOF)
  {
    *(s+k)=ch;
    k++;
  }
   *(s+k)='\0';
 }
 printf("%s\n",s);
 printf("---------------------------------------\n");
 i=0;
 x:while(*(s+i)!='\0')
```

```
{
 j=0;c=0;
 if(*(s+i)=='#')
 {
  i++;
  while(*(s+i)!='\n')
  {
   printf("%c",*(s+i));
   i++;
  }
  printf("is a pre processor directive\n");
  i++;
 }
 else if(isalpha(*(s+i))||*(s+i)=='_')
 {
  *(t+j)=*(s+i);
 i++;    j++;
 while((*(s+i)=='_'||isdigit(*(s+i))||isalpha(*(s+i)))&&*(s+i)!='\0')
 {
  *(t+j)=*(s+i);
  i++;
  j++;
 }
 *(t+j)='\0';
 for(k=0;k<32;k++)
 if(strcmp(t,key[k])==0)
 {
  c=1;
  break;
 }
 if(c==1)
  printf("%s is a keyword\n",t);
 else
 {
 if(*(s+i)=='(')
 {
   printf("%s is a method\n",t);
   goto x;
 }
 else
 printf("%s is an identifier\n",t);
 }
 }
 else if(*(s+i)=='')
```

```
{
   printf("\"");
   i++;
   while(*(s+i)!="")
   {
     printf("%c",*(s+i));
     i++;
   }
   printf("\" is an argument \n");
   i++;
}
else if((*(s+i)=='/'&&*(s+i+1)=='/')||(*(s+i)=='/'&&*(s+i+1)=='*'))
{
  i++;
  if(*(s+i)=='/')
  {
    i++;
    while(*(s+i)!='\n')
         {
         printf("%c",*(s+i));
           i++;
          }
   }
  else if(*(s+i)=='*')
  {
    i++;
    while(*(s+i)!='*'&&*(s+i+1)!='/')
         {
          // if(*(s+i)=='\n')
        printf("%c",*(s+i));

           i++;
         }
     i=i+2;
   }
         printf("is a comment line\n");
}
else if(*(s+i)=='['||*(s+i)==']'||*(s+i)=='('||*(s+i)==')'||*(s+i)=='{'||*(s+i)=='}')
{
  printf("%c is a special symbol\n",*(s+i));
  i++;
}
else if(isdigit(*(s+i)))
{
```

```
    if(isdigit(*(s+i))||*(s+i)=='.')
    {
    printf("%c",*(s+i));
     i++;
    while(isdigit(*(s+i))||*(s+i)=='.')
    {
            printf("%c",*(s+i));
            i++;
    }
    printf(" is a number\n");
    }
    else if(isalpha(*(s+i)))
   {
        while(isalpha(*(s+i))||*(s+i)=='_')
     {
        printf("%c",*(s+i));
        i++;
     }
        printf(" is an invalid token\n");
    }
   }
  else if(*(s+i)=='='||*(s+i)=='+'||*(s+i)=='-
'||*(s+i)=='*'||*(s+i)=='/'||*(s+i)=='%'||*(s+i)=='<'||*(s+i)=='>'||*(s+i)=='&'||*(s+i)=='|')
  {
    printf("%c is an operator\n",*(s+i));
     i++;
  }
  else if(*(s+i)==';')
  {
     printf("%c is terminator\n",*(s+i));
    i++;
  }
   else
    i++;
 }
 fclose(fp);
}
```

OUTPUT


// hai
include<stdio.h>is a preprocessor directive
void is a keyword
main is a method
(is a special symbol
) is a special symbol
{ is a special symbol
int is a keyword
a is an identifier
; is terminator
printf is a method
(is a special symbol
"hai" is an argument
) is a special symbol
; is terminator
c is an identifier
= is an operator
a is an identifier
+ is an operator
100 is a number
; is terminator
} is a special symbol

TASK 2

TASK 2: Write a program to recognize strings under 'a' , 'a*b+' , 'abb'

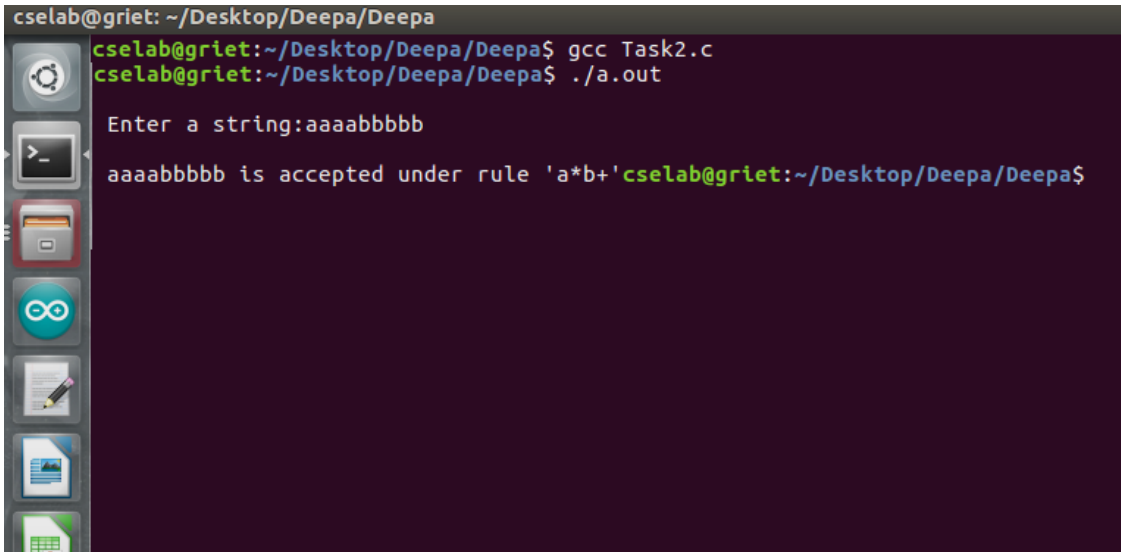
Aim: To write a program to recognize strings under 'a*' , 'a*b+' , 'abb'

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main()
{
char s[20],c;
int state=0,i=0;
printf("\n Enter a string:");
gets(s);
while(s[i]!='\0')
 {
  switch(state)
  {
   case 0:c=s[i++];
          if(c=='a')
          state=1;
          else if(c=='b')
          state=2;
          else
          state=6;
          break;
   case 1: c=s[i++];
          if(c=='a')
          state=3;
          else if(c=='b')
          state=4;
          else
          state=6;
          break;
   case 2: c=s[i++];
          if(c=='a')
          state=6;
          else if(c=='b')
          state=2;
          else
          state=6;
```

```
            break;
   case 3: c=s[i++];
            if(c=='a')
            state=3;
            else if(c=='b')
            state=2;
            else
            state=6;
            break;
   case 4: c=s[i++];
            if(c=='a')
            state=6;
            else if(c=='b')
            state=5;
            else
            state=6;
            break;
   case 5: c=s[i++];
            if(c=='a')
            state=6;
            else if(c=='b')
            state=2;
            else
            state=6;
            break;
   case 6: printf("\n %s is not recognised.",s);
            exit(0);
     }
  }
 if(state==1)
     printf("\n %s is accepted under rule 'a'",s);
 else if((state==2)||(state==4))
     printf("\n %s is accepted under rule 'a*b+'",s);
 else if(state==5)
      printf("\n %s is accepted under rule 'abb'",s);
}
```

OUTPUT


cselab@griet: ~/Desktop/Deepa/Deepa
cselab@griet:~/Desktop/Deepa/Deepa$ gcc Task2.c
cselab@griet:~/Desktop/Deepa/Deepa$ ./a.out
Enter a string:aaaabbbbb
aaaabbbbb is accepted under rule 'a*b+'cselab@griet:~/Desktop/Deepa/Deepa$

TASK 3

TASK 3: Implementation of symbol table

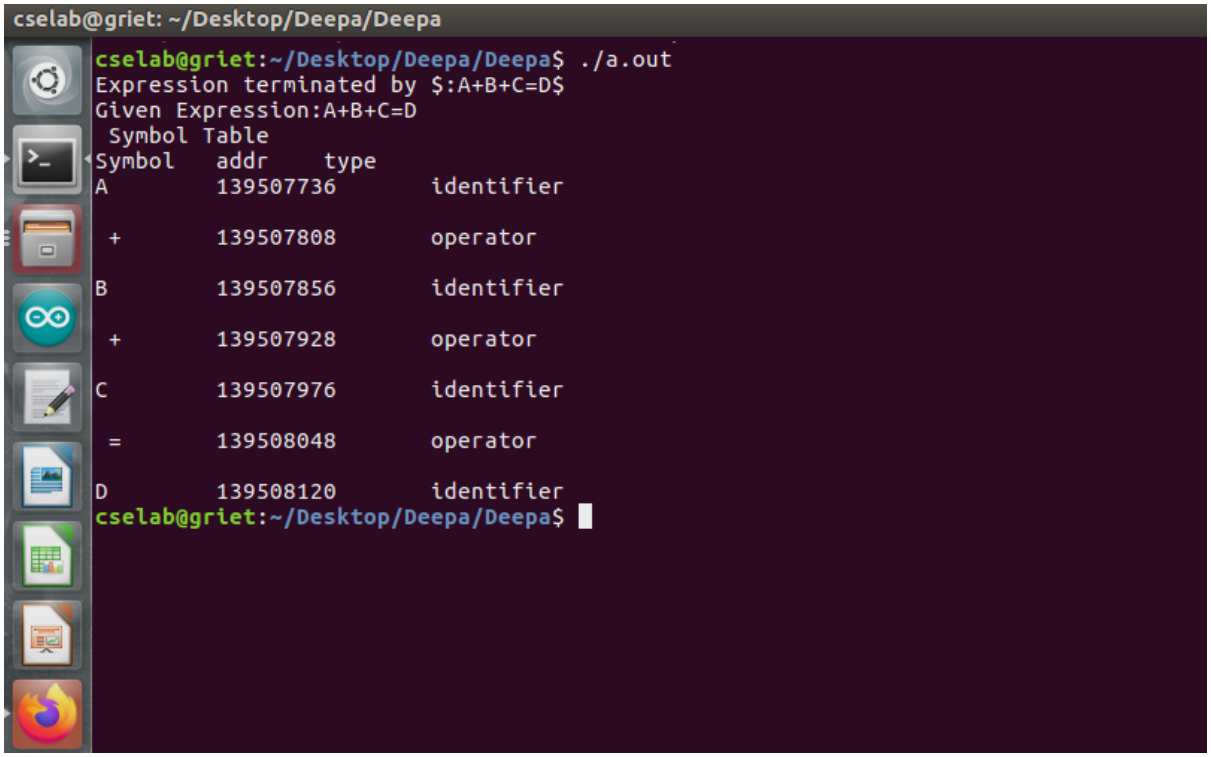
Aim: To implementation of symbol table

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
void main()
{
 int i=0,j=0,x=0,n;
            void *p,*add[5];
            char ch,srch,b[15],d[15],c;
            printf("Expression terminated by $:");
 while((c=getchar())!='$')
 {
            b[i]=c;
             i++;
 }
 n=i-1;
 printf("Given Expression:");
 i=0;
 while(i<=n)
 {
            printf("%c",b[i]);
             i++;
 }
 printf("\n Symbol Table\n");
 printf("Symbol \t addr \t type");
 while(j<=n)
 {
            c=b[j];
  if(isalpha(toascii(c)))
  {
              p=malloc(c);
              add[x]=p;
              d[x]=c;
              printf("\n%c \t %d \t identifier\n",c,p);
```

```
            x++;
            j++;
  }
  else
  {
          ch=c;
            if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
   {
             p=malloc(ch);
             add[x]=p;
             d[x]=ch;
             printf("\n %c \t %d \t operator\n",ch,p);
             x++;
             j++;
   }
}}}
```

OUTPUT:



TASK 4

Task 4: Write a program to implement predictive parser table

Aim: To write a program to implement predictive parser table

Program:

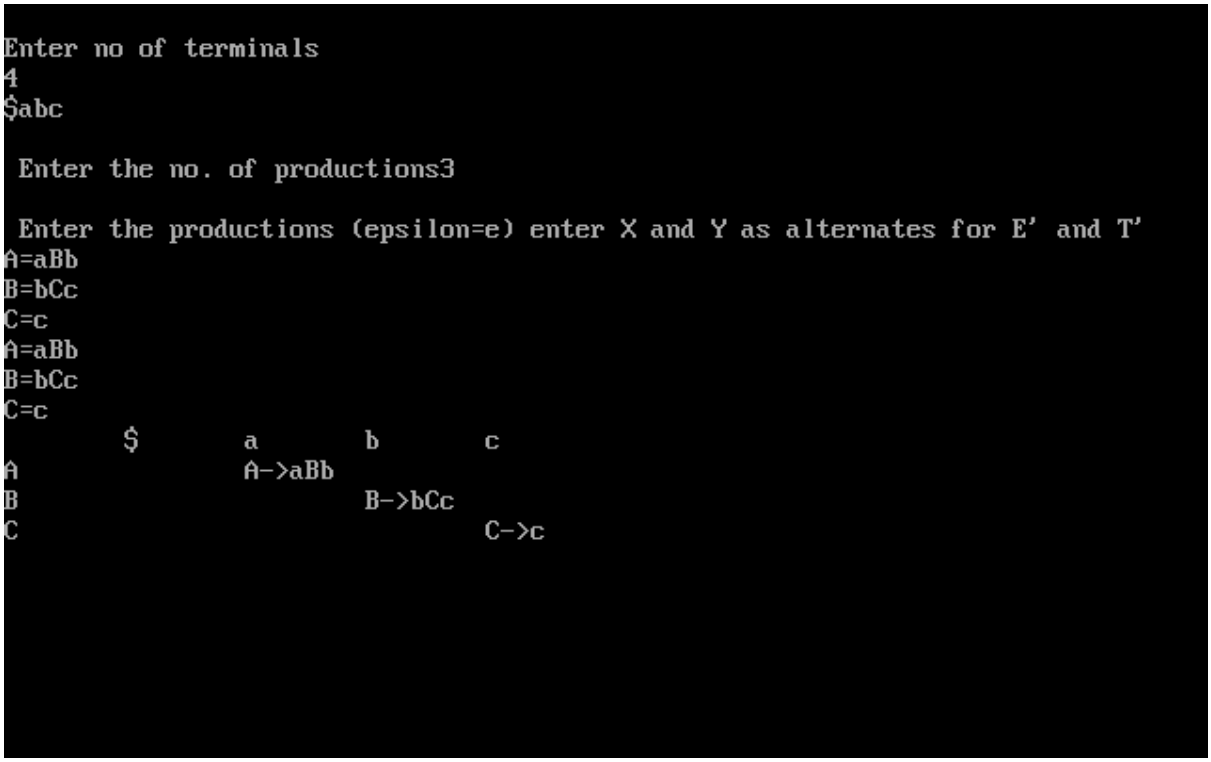
```
#include<stdio.h>
#include<string.h>
int n,z=0,m=0,p,i=0,j=0;
char a[3][10],f[10];
void follow(char c);
void first(char c);
int main()
{
        int i,j,z,n1;
        char t[10];
         int x;
        char c;
         printf("enter no of terminals");
         scanf("%d",&n1);
        scanf("%s",t);
        printf("\n enter the no. of productions");
        scanf("%d",&n);
        printf("\n enter the productions (epsilon=e) enter X and Y as alternates for E' and T'");
        for(i=0;i<n;i++)
         scanf("%s",a[i]);
         for(i=0;i<n;i++)
         printf("%s\n",a[i]);
         for(i=0;i<n1;i++)
          printf("\t%c",t[i]);
          printf("\n");
         for(i=0;i<n;i++)
        {
        if((islower(a[i][2]))&&(a[i][2]!='e'))
        {
                x=0;
                  printf("%c\t",a[i][0]);
                 while(a[i][2]!=t[x])
                {
                    printf("\t");
```

```
      x++;
    }
     printf("%c->%s\n",a[i][0],a[i]+2);
}
 if(isupper(a[i][2]))
   {
    m=0;
    first(a[i][2]);
    for(z=0;z<m;z++)
    {
      if(f[z]=='e')
        {
         for(x=0;x<m;x++)
         f[m]='\0';
          m=0;
          follow(a[i][0]);
        }
      }
     printf("%c\t",a[i][0]);
     for(z=0;z<m;z++)
     {
       x=0;
       while(f[z]!=t[x])
       {
        printf("\t");x++;
        }
        printf("%c->%s\t",a[i][0],a[i]+2);
      }
  printf("\n");
 continue;
 }
 if(a[i][2]=='e')
{
     m=0;
     follow(a[i][0]);
     printf("%c\t",a[i][0]);
     for(z=0;z<m;z++)
   {
       x=0;
       while(f[z]!=t[x])
       {
```

```
                        printf("\t");x++;
                      }
                       printf("%c->e",a[i][0]);
                    }
                printf("\n");
                continue;
            }
           }
        return 0;
}
void follow(char c)
{
        int k,l;
        if(a[0][0]==c)f[m++]='$';
        for(k=0;k<n;k++)
        {
          for(l=2;l<strlen(a[k]);l++)
           {
             if(a[k][l]==c)
            {
              if(a[k][l+1]!='\0') first(a[k][l+1]);
              if((a[k][l+1]=='\0') &&(c!=a[k][0]))
              follow(a[k][0]);
           }
         }
       }
}
void first(char c)
{
        int k;
        if(!isupper(c))
        f[m++]=c;
        for(k=0;k<n;k++)
        {
           if(a[k][0]==c)
        {
                if(a[k][2]=='e')
                follow(a[i][0]);
                else if(islower(a[k][2]))
                f[m++]=a[k][2];
                else
```

```
                first(a[k][2]);
        }
        }
}
```

OUTPUT:



TASK 5

TASK 5: Write a Program To Compute FIRST() and FOLLOW() Functions.

Program:

```
// C program to calculate the First and
// Follow sets of a given grammar
#include <ctype.h>
#include <stdio.h>
#include <string.h>

// Functions to calculate Follow
void followfirst(char, int, int);
void follow(char c);

// Function to calculate First
void findfirst(char, int, int);

int count, n = 0;

// Stores the final result
// of the First Sets
char calc_first[10][100];

// Stores the final result
// of the Follow Sets
char calc_follow[10][100];
int m = 0;

// Stores the production rules
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;

int main(int argc, char** argv)
{
        int jm = 0;
        int km = 0;
        int i, choice;
        char c, ch;
        count = 8;

        // The Input grammar
        strcpy(production[0], "X=TnS");
        strcpy(production[1], "X=Rm");
```

```
strcpy(production[2], "T=q");
strcpy(production[3], "T=#");
strcpy(production[4], "S=p");
strcpy(production[5], "S=#");
strcpy(production[6], "R=om");
strcpy(production[7], "R=ST");

int kay;
char done[count];
int ptr = -1;

// Initializing the calc_first array
for (k = 0; k < count; k++) {
        for (kay = 0; kay < 100; kay++) {
                calc_first[k][kay] = '!';
        }
}
int point1 = 0, point2, xxx;

for (k = 0; k < count; k++) {
        c = production[k][0];
        point2 = 0;
        xxx = 0;

        // Checking if First of c has
        // already been calculated
        for (kay = 0; kay <= ptr; kay++)
                if (c == done[kay])
                        xxx = 1;

        if (xxx == 1)
                continue;

        // Function call
        findfirst(c, 0, 0);
        ptr += 1;

        // Adding c to the calculated list
        done[ptr] = c;
        printf("\n First(%c) = { ", c);
        calc_first[point1][point2++] = c;

        // Printing the First Sets of the grammar
        for (i = 0 + jm; i < n; i++) {
                int lark = 0, chk = 0;

                for (lark = 0; lark < point2; lark++) {

                        if (first[i] == calc_first[point1][lark]) {
                                chk = 1;
```

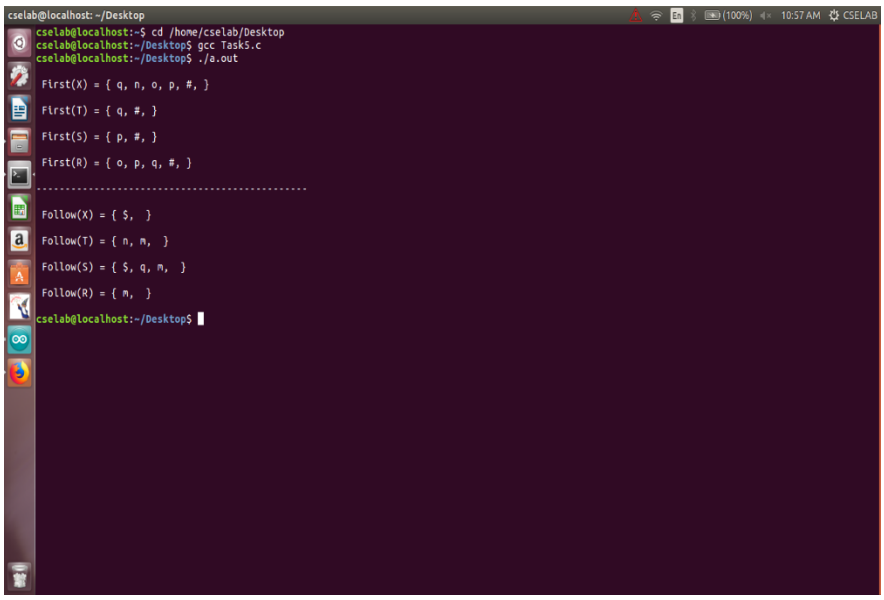
```
                                        break;
                                }
                        }
                        if (chk == 0) {
                                printf("%c, ", first[i]);
                                calc_first[point1][point2++] = first[i];
                        }
                }
                printf("}\n");
                jm = n;
                point1++;
        }
        printf("\n");
        printf("-----------------------------------------------"
                "\n\n");
        char donee[count];
        ptr = -1;

        // Initializing the calc_follow array
        for (k = 0; k < count; k++) {
                for (kay = 0; kay < 100; kay++) {
                        calc_follow[k][kay] = '!';
                }
        }
        point1 = 0;
        int land = 0;
        for (e = 0; e < count; e++) {
                ck = production[e][0];
                point2 = 0;
                xxx = 0;

                // Checking if Follow of ck
                // has already been calculated
                for (kay = 0; kay <= ptr; kay++)
                        if (ck == donee[kay])
                                xxx = 1;

                if (xxx == 1)
                        continue;
                land += 1;

                // Function call
                follow(ck);
                ptr += 1;

                // Adding ck to the calculated list
                donee[ptr] = ck;
                printf(" Follow(%c) = { ", ck);
                calc_follow[point1][point2++] = ck;
```

```
                // Printing the Follow Sets of the grammar
                for (i = 0 + km; i < m; i++) {
                        int lark = 0, chk = 0;
                        for (lark = 0; lark < point2; lark++) {
                                if (f[i] == calc_follow[point1][lark]) {
                                        chk = 1;
                                        break;
                                }
                        }
                        if (chk == 0) {
                                printf("%c, ", f[i]);
                                calc_follow[point1][point2++] = f[i];
                        }
                }
                printf(" }\n\n");
                km = m;
                point1++;
        }
}

void follow(char c)
{
        int i, j;

        // Adding "$" to the follow
        // set of the start symbol
        if (production[0][0] == c) {
                f[m++] = '$';
        }
        for (i = 0; i < 10; i++) {
                for (j = 2; j < 10; j++) {
                        if (production[i][j] == c) {
                                if (production[i][j + 1] != '\0') {
                                        // Calculate the first of the next
                                        // Non-Terminal in the production
                                        followfirst(production[i][j + 1], i,
                                                                (j + 2));
                                }

                                if (production[i][j + 1] == '\0'
                                        && c != production[i][0]) {
                                        // Calculate the follow of the
                                        // Non-Terminal in the L.H.S. of the
                                        // production
                                        follow(production[i][0]);
                                }
                        }
                }
        }
}
```

```
void findfirst(char c, int q1, int q2)
{
	int j;

	// The case where we
	// encounter a Terminal
	if (!(isupper(c))) {
		first[n++] = c;
	}
	for (j = 0; j < count; j++) {
		if (production[j][0] == c) {
			if (production[j][2] == '#') {
				if (production[q1][q2] == '\0')
					first[n++] = '#';
				else if (production[q1][q2] != '\0'
						&& (q1 != 0 || q2 != 0)) {
					// Recursion to calculate First of New
					// Non-Terminal we encounter after
					// epsilon
					findfirst(production[q1][q2], q1,
							(q2 + 1));
				}
				else
					first[n++] = '#';
			}
			else if (!isupper(production[j][2])) {
				first[n++] = production[j][2];
			}
			else {
				// Recursion to calculate First of
				// New Non-Terminal we encounter
				// at the beginning
				findfirst(production[j][2], j, 3);
			}
		}
	}
}

void followfirst(char c, int c1, int c2)
{
	int k;

	// The case where we encounter
	// a Terminal
	if (!(isupper(c)))
		f[m++] = c;
	else {
		int i = 0, j = 1;
		for (i = 0; i < count; i++) {
```

```
                if (calc_first[i][0] == c)
                        break;
        }

        // Including the First set of the
        // Non-Terminal in the Follow of
        // the original query
        while (calc_first[i][j] != '!') {
                if (calc_first[i][j] != '#') {
                        f[m++] = calc_first[i][j];
                }
                else {
                        if (production[c1][c2] == '\0') {
                                // Case where we reach the
                                // end of a production
                                follow(production[c1][0]);
                        }
                        else {
                                // Recursion to the next symbol
                                // in case we encounter a "#"
                                followfirst(production[c1][c2], c1,
                                                c2 + 1);
                        }
                }
                j++;
        }
    }
}
```

OUTPUT



TASK 6

TASK 6: Construct operator precedence parser

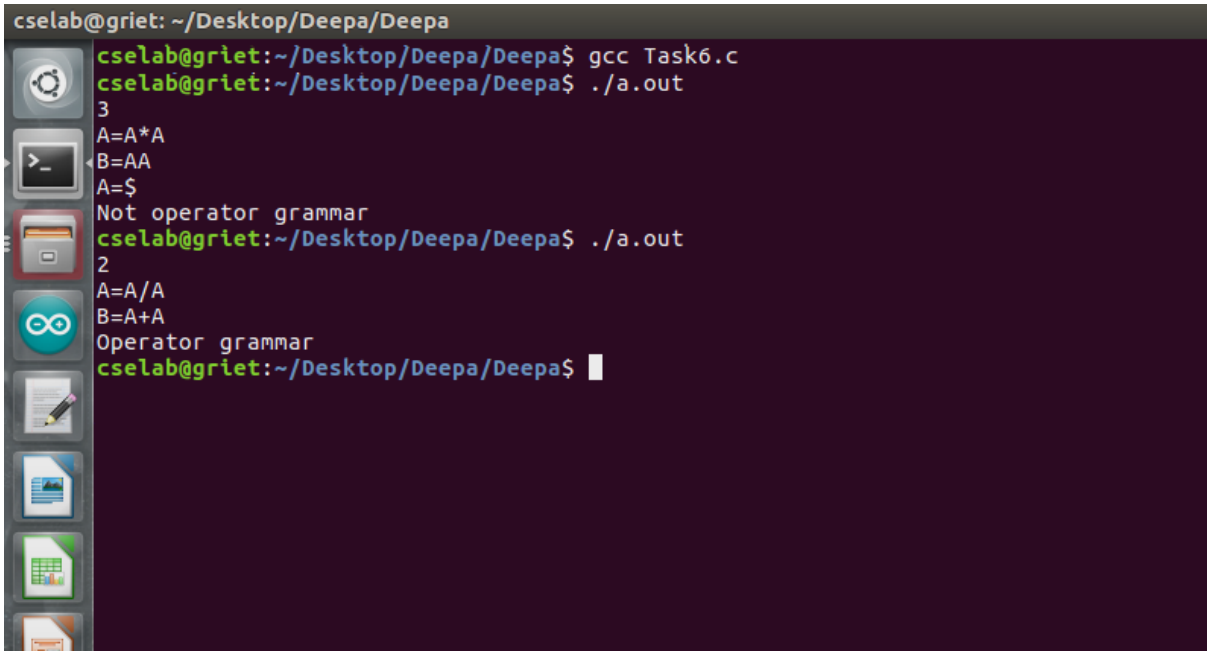
Aim: To Construct operator precedence parser

Program:

```
#include <stdio.h>
#include <string.h>

// function f to exit from the loop
// if given condition is not true
void f()
{
        printf("Not operator grammar");
        exit(0);
}

void main()
{
        chargrm[20][20], c;

        // Here using flag variable,
        // considering grammar is not operator grammar
        inti, n, j = 2, flag = 0;

        // taking number of productions from user
        scanf("%d", &n);
        for (i = 0; i< n; i++)
           scanf("%s", grm[i]);

        for (i = 0; i< n; i++) {
           c = grm[i][2];

    while (c != '\0') {

            if (grm[i][3] == '+' || grm[i][3] == '-'|| grm[i][3] == '*' || grm[i][3] == '/')

                flag = 1;

             else {
```

```
                        flag = 0;
                        f();
                }

                 if (c == '$')
                {
                        flag = 0;
           f();
        }

        c = grm[i][++j];
      }
   }

   if (flag == 1)
      printf("Operator grammar");
}
```

OUTPUT



TASK 7

Task 7: Implement Shift Reduce Parsing

Aim: To implement Shift Reduce Parsing

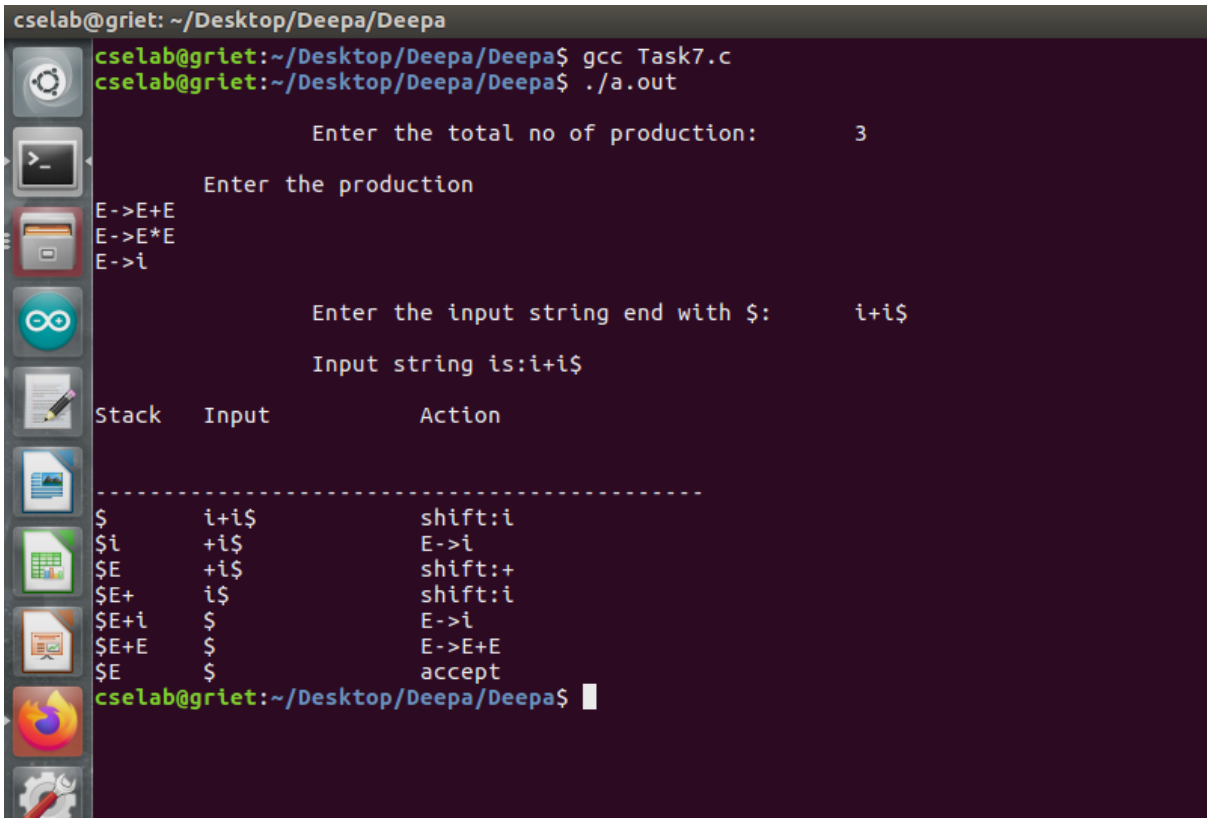
Program

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
inti,j,l,n,top=0,k,p,sl;
char in[20],apro[20],pro[20][20],sta[20],stack[20];
int s[10];
void null();
void out();
void main()
{
        clrscr();
        printf("\n\t\tEnter the total no of production:\t");
        scanf("%d",&n);
        printf("\n\tEnter the production\n");
        for(i=0;i<n;i++)
                {
                scanf("%s",pro[i]);
                s[i]=strlen(pro[i])-3;
                }
        printf("\n\t\tEnter the input string end with $:\t");
        scanf("\n%s",in);
        printf("\n\t\tInput string is:%s",in);
        printf("\n\nStack\tInput\t\tAction\n");
        printf("\n\n-----------------------------------------\n");
        stack[top]='$';
        l=0;
        out();
        while(in[l]!='$')
        null();
        if(stack[top]==pro[0][0] && stack[top+1]=='\0')
```

```
        printf("\t\taccept\n");
        else
        printf("error\n");
        getch();
}

void null()
{
        int p=0,m;
        if(in[l]!='$')
        stack[++top]=in[l++];
        printf("\t\tshift:%c\n",stack[top]);
        out();
        while(1)
                {
                if(stack[top-p]=='$')
                break;
                else
                        {
                        for(m=top-p,k=0;m<=top;m++,k++)
                        sta[k]=stack[m];
                        sta[k]='\0';
                        sl=strlen(sta);
                        for(i=0;i<n;i++)
                                {
                                //printf("s1=%d\t%d",sl,s[0]);
                                if(sl==s[i])
                                        {
                                        for(j=3,k=0;pro[i][j]!='\0';j++)
                                                {
                                                apro[k]=pro[i][j];
                                                k++;
                                                }
                                        apro[k]='\0';
                                        if((strcmp(sta,apro))==0)
                                                {
                                                top=top-p;
                                                p=0;
                                                stack[top]=pro[i][0];
                                                stack[top+1]='\0';
                                                printf("\t\t%s\n",pro[i]);
                                                out();

                                                }
```

```
                                        }
                                    }
                                }
                        p++;
                        }
                }
void out()
        {
        inti;
        for(i=0;i<=top;i++)
        printf("%c",stack[i]);
        printf("\t");
        for(i=l;in[i]!='\0';i++)
        printf("%c",in[i]);
        }
```

OUTPUT:



TASK 8

TASK 8: Write a program to implement LALR parser

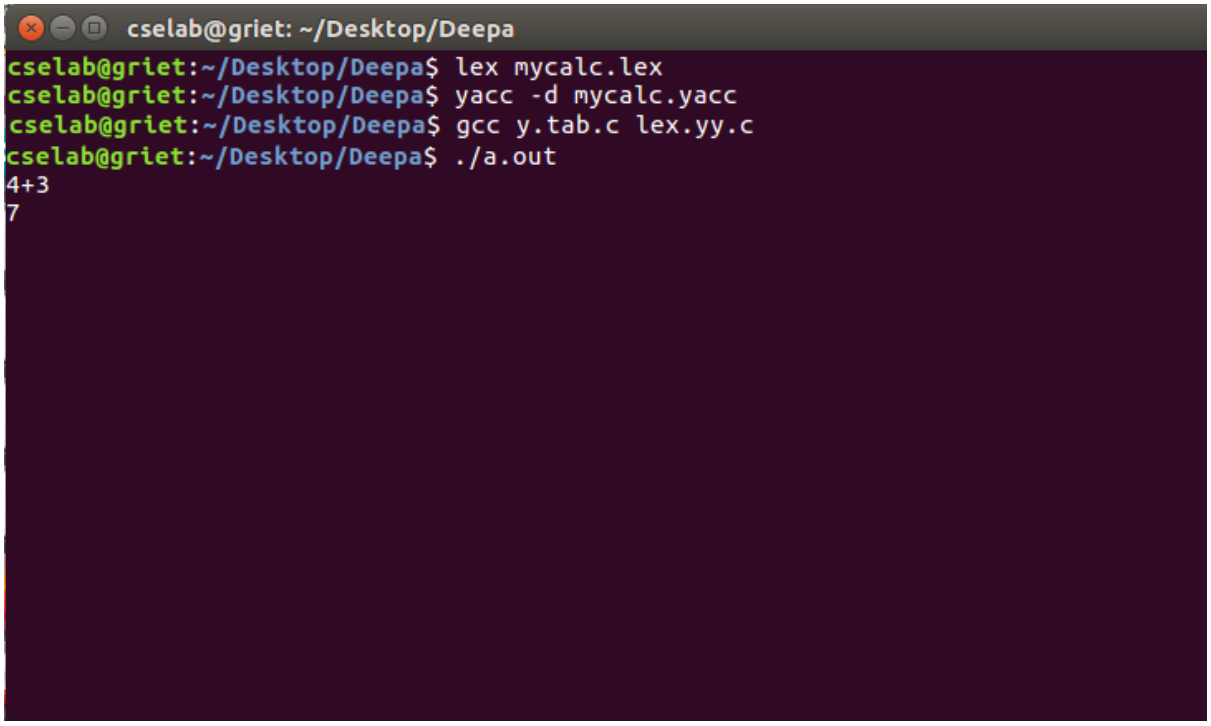
Aim : To write a program to implement LALR parser

Program:

```
%{
        #include
        #include “y.tab.h”
%}
%%
        [0-9]+ {yylval.dval=atof(yytext);
        return DIGIT;
}
\n|. returnyytext[0];
%%
%{
/*This YACC specification file generates the LALR parser for the program
c
#include
%}
%union
{
        doubledval;
}
        %token DIGIT
        %type expr
        %type term
        %type factor
        %%
        line: expr ‘\n’ {
        printf(“%g\n”,$1);
}
;
        expr: expr ‘+’ term {$$=$1 + $3 ;}
        | term
```

```
        ;
        term: term '*' factor {$$=$1 * $3 ;}
        | factor
        ;
        factor: '(' expr ')' {$$=$2 ;}
        | DIGIT
        ;
%%


int main()
{
        yyparse();
}
yyerror(char *s)
{
        printf("%s",s);
}
```

OUTPUT:



TASK 9

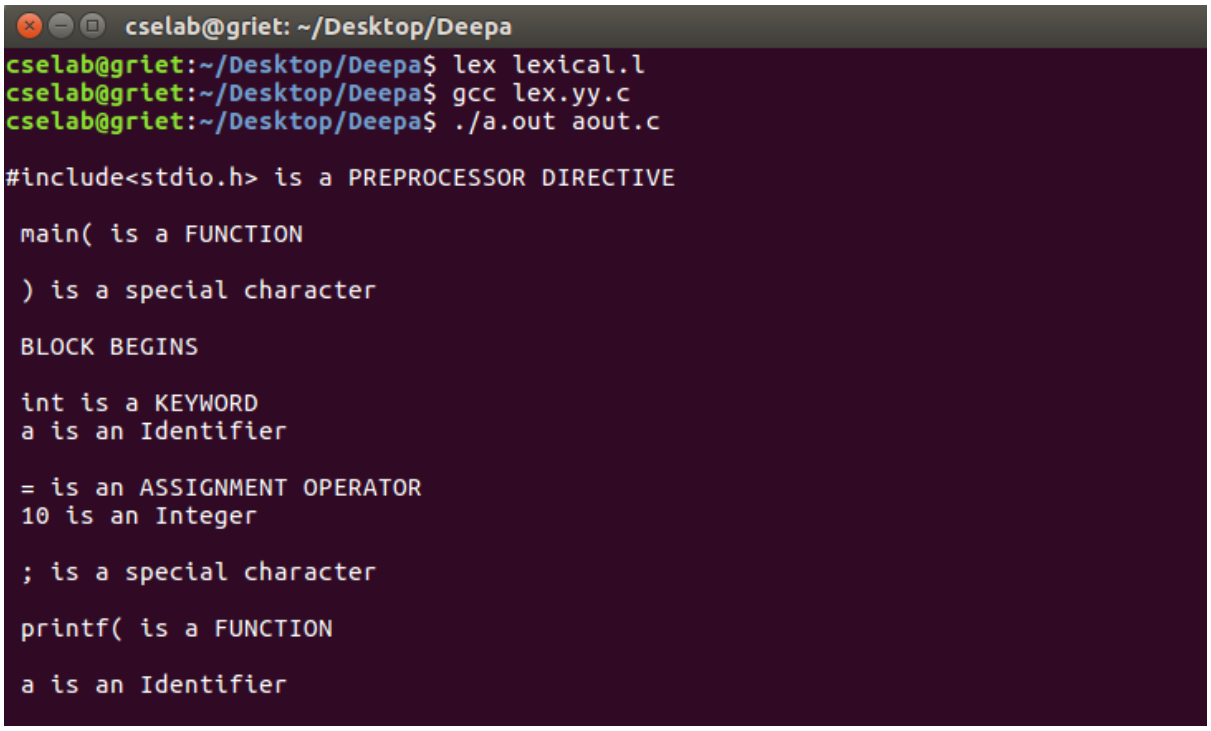
TASK 9: Implementation of Lexical Analyzer using Lex tool

Aim: To implement Lexical Analyzer using Lex tool

```
%{
        int COMMENT=0;
        intcnt=0;
%}
        identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
        int |
        float |
        char |
        double |
        while |
        for |
        do |
        if |
        break |
        continue |
        void |
        switch |
        case |
        long |
        struct |
        const |
        typedef |
        return |
        else |
goto {printf("\n\t%s is a KEYWORD",yytext);}
"/*" {COMMENT = 1;}
"*/" {COMMENT = 0; cnt++;}
{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}
\} {if(!COMMENT) printf("\n BLOCK ENDS");}
```

```
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}
\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(intargc,char **argv)
{
if (argc> 1)
{
        FILE *file;
        file = fopen(argv[1],"r");
        if(!file)
        {
                printf("could not open %s \n",argv[1]);
                exit(0);
        }
        yyin = file;
}
        yylex();
        printf("\n\n Total No.Of comments are %d",cnt);
        return 0;
}
intyywrap()
{
        return 1;

}
```

OUTPUT


cselab@griet: ~/Desktop/Deepa
cselab@griet:~/Desktop/Deepa$ lex lexical.l
cselab@griet:~/Desktop/Deepa$ gcc lex.yy.c
cselab@griet:~/Desktop/Deepa$ ./a.out aout.c
#include<stdio.h> is a PREPROCESSOR DIRECTIVE
main(is a FUNCTION
) is a special character
BLOCK BEGINS
int is a KEYWORD
a is an Identifier
= is an ASSIGNMENT OPERATOR
10 is an Integer
; is a special character
printf(is a FUNCTION
a is an Identifier

TASK 10

TASK 10: Design a simple arithmetic calculator using LEX

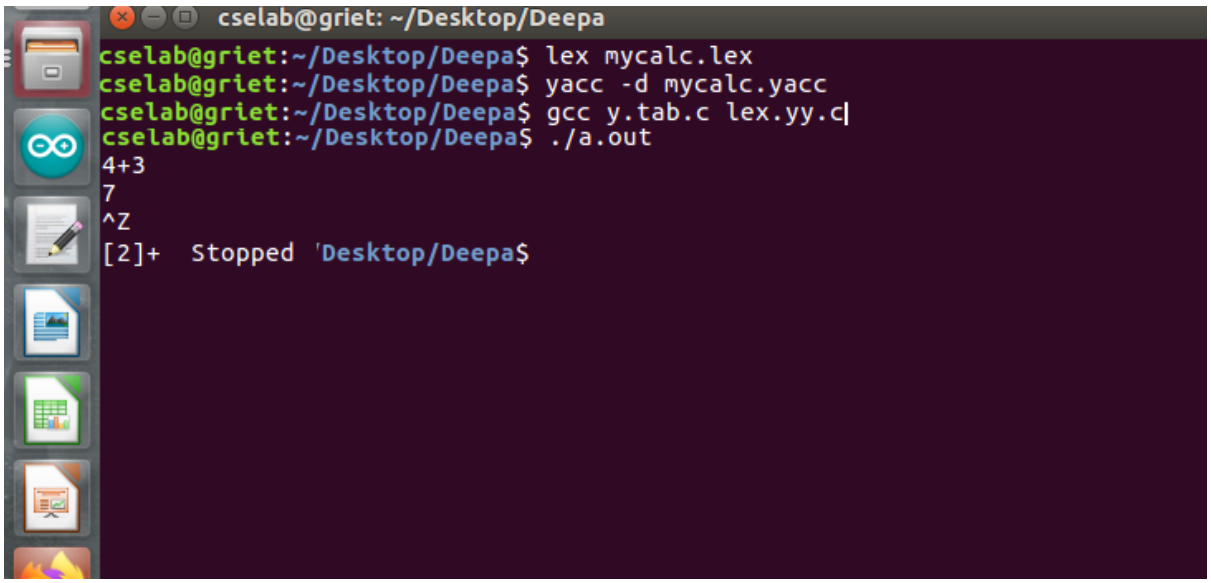
Aim: To design a simple arithmetic calculator using LEX

Program:

```
%{
        int op=0,i;
        floata,b;
%}

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

%%
        {dig} {digi();} /*** digi() is a user defined function ***/
        {add} {op=1;}
        {sub} {op=2;}
        {mul} {op=3;}
        {div} {op=4;}
        {pow} {op=5;}
        {ln} {printf("\n the result :%f\n\n",a);}

%%
digi()
{
        if(op==0)
        a=atof(yytext); /*** atof() is used to convert the ASCII input to float***/
        else
        {
        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;
        }
}
main(intargv,char *argc[])
{
yylex();
}
yywrap()
{
return 1;
}
```

OUTPUT:



TASK 11

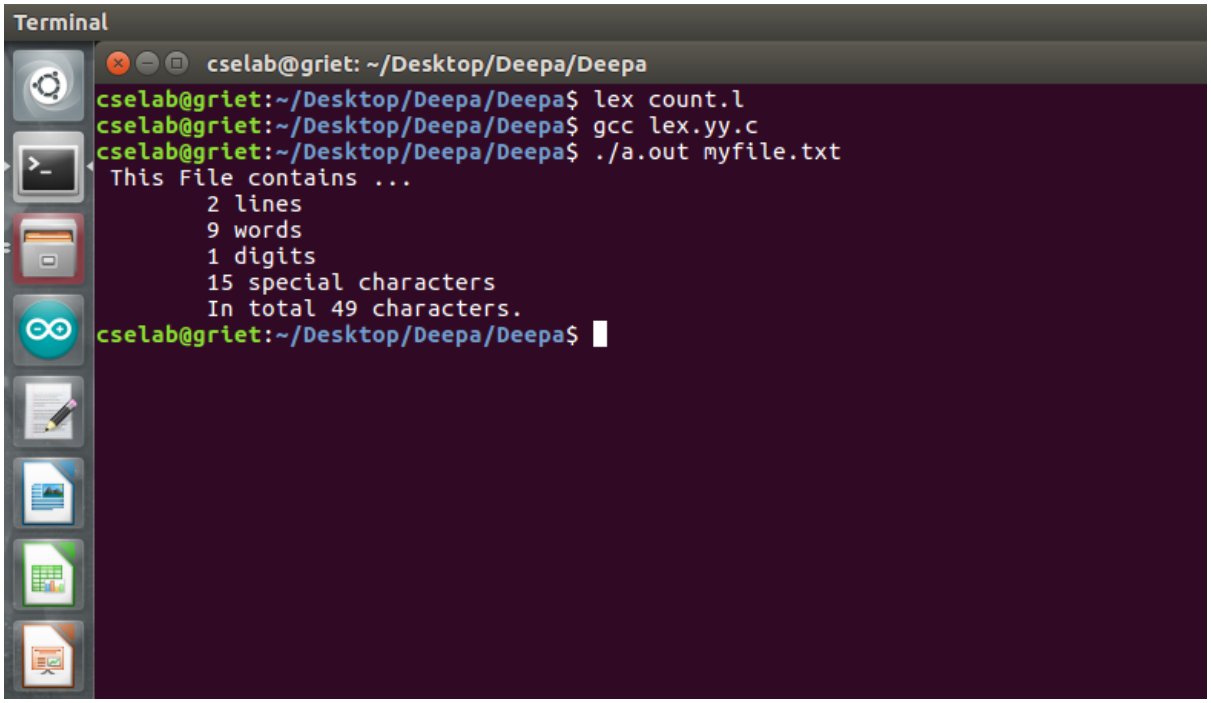
TASK 11: Lex program to count the number of characters, words, lines, and special characters in a file

Aim: To write a lex program to count the number of characters, words, lines, and special characters in a file

Program:

```
%{
        #include<stdio.h>
        int lines=0, words=0,s_letters=0,c_letters=0, num=0, spl_char=0,total=0;
%}
%%

        \n { lines++; words++;}
        [\t ' '] words++;
        [A-Z] c_letters++;
        [a-z] s_letters++;
        [0-9] num++;
        . spl_char++;
%%

main(void)
{
        yyin= fopen("myfile.txt","r");
        yylex();
        total=s_letters+c_letters+num+spl_char;
        printf(" This File contains ...");
        printf("\n\t%d lines", lines);
        printf("\n\t%dwords",words);
        printf("\n\t%d digits", num);
        printf("\n\t%d special characters",spl_char);
        printf("\n\tIn total %d characters.\n",total);
}

intyywrap()
{
        return(1);
}
```

OUTPUT:


Terminal
cselab@griet: ~/Desktop/Deepa/Deepa
cselab@griet:~/Desktop/Deepa/Deepa$ lex count.l
cselab@griet:~/Desktop/Deepa/Deepa$ gcc lex.yy.c
cselab@griet:~/Desktop/Deepa/Deepa$ ./a.out myfile.txt
This File contains ...
2 lines
9 words
1 digits
15 special characters
In total 49 characters.
cselab@griet:~/Desktop/Deepa/Deepa$

TASK 12

TASK 12: Implement code optimization techniques

Aim: To implement code optimization techniques

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct regis
{
 char var;
}reg[10];
int noreg=0;
char st[10][10];

int nost;
char* opcode[10]={"ADD","SUB","MUL","DIV"};
char oper[10]={'+','-','*','/'};
char* toopcode(char opert)
{
            int i;
           for(i=0;i<4;i++)
           {
            if(oper[i]==opert)
           {
            return(opcode[i]);
           }
           }
}
```

```
int isinregister(char var)
{
        int i;
        for(i=1;i<=noreg;i++)
        {
        if(var==reg[i].var)
          {
           return(i);
        }
        }
        return(0);
}

void main()
{
        int i,regno2=0,regno1=1,k,j;
        int nost1,n,flag=0;

        clrscr();
        printf("TO GENERATE OPTIMIZED TARGET MACHINE CODE FOR AN
        INTERMEDIATE CODE");
        printf("\nEnter the no. of statements:");
        scanf("%d",&nost);
        nost1=nost;
        printf("Enter the statements:");
        for(i=0;i<nost;i++)
        {
        scanf("%s",st[i]);

        }

        for(k=0;k<nost-1;k++)
        {
          for(j=2;j<5;j++)
            {
              if(st[k][j]!=st[k+1][j])
              flag=1;
```

```
                }

        }
printf("If the machine architecture is having the following format \n OPERATIONS
SOURCE  TARGET\n");
printf("ADD var/reg var/reg :--> MOV b,r1 ,variable b contents are moved to register r1\n");
printf("\n\n\tTstatements    \t\t\t targetcode");
for(i=0;i<nost;i++)
{
 printf("\n\t%s",st[i]);

if((!regno1==isinregister(st[i][2])))
{
        printf("\n\t\t\t\t MOV %c,r%d",st[i][2],++noreg);
        reg[noreg].var=st[i][2];
        regno1=noreg;
}
if((!regno2==isinregister(st[i][4])))
{
        printf("\n\t\t\t\t %s r%d,r%d\n",toopcode(st[i][3]),regno2,regno1);
        reg[regno1].var=st[i][0];
}
else
{
        printf("\n\t\t\t\t %s %c,r%d\n",toopcode(st[i][3]),st[i][4],regno1);
        reg[regno1].var=st[i][0];
}
//if(i==nost-1)
{
printf("\t\t\t\t MOV r%d,%c\n",regno1,st[i][0]);
}

if(flag==0)
 {
 //printf("\t\t\t\t MOV r%d,%c\n",regno1,st[i+1][0]);
 printf("\t%s\t\t\t MOV r%d,%c\n",st[i+1],regno1,st[i+1][0]);
 break;
```

```
  }

}
printf("\n CODE GENERATION COMPLETED\n\n");
}
```

OUTPUT: