EXERCISE – 1

AIM: Write a C program to identify different types of Tokens in a given Program.

PROGRAM:

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
 if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||strcmp("int",str)==0||str
cmp("float",str)==0||strcmp("char",str)==0||strcmp("double",str)==0||strcmp("static",str)==0||strcmp("s
witch",str)==0||strcmp("case",str)==0)
        printf("\n%s is a keyword",str);
 else
        printf("\n%s is an identifier",str);
}
main()
{
FILE *f1,*f2,*f3;
char c,str[10],st1[10];
int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
printf("Enter the c program:\n");
f1=fopen("input","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");
f3=fopen("specialchar","w");
while((c=getc(f1))!=EOF)
{
if(isdigit(c))
{
 tokenvalue=c-'0';
 c=getc(f1);
 while(isdigit(c))
{
        tokenvalue*=10+c-'0';
        c=getc(f1);
        }
        num[i++]=tokenvalue;
        ungetc(c,f1);
}
else if(isalpha(c))
        {
        putc(c,f2);
 c=getc(f1);
        while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
        {
        putc(c,f2); c=getc(f1);
        }
        putc(' ',f2);
        ungetc(c,f1);
        }
        else if(c==' '||c=='\t')
                printf(" ");
        else if(c=='\n')
                lineno++;
        else
                putc(c,f3);
 }
fclose(f2);
fclose(f3);
fclose(f1);
printf("\nThe no's in the program are:\n");
for(j=0;j<i;j++)
printf("%d ",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("The keywords and identifiers are:\n");
while((c=getc(f2))!=EOF)
{
 if(c!=' ')
 str[k++]=c;
 else
 {
 str[k]='\0';
 keyword(str);
 k=0;
}
}
fclose(f2);
f3=fopen("specialchar","r");
 printf("\nSpecial characters are:\n");
 while((c=getc(f3))!=EOF)
        printf("%c ",c);
 printf("\n");
 fclose(f3);
 printf("Total no. of lines are: %d",lineno);
}
```

INPUT:

```
Enter the C program:
int main( )
{
int a=5;
int b=8;
int c=a+b;
printf(“\n C value is: %d”,c);
}
^z
```

OUTPUT:

```
The no’s in the program are:
5 8
The keywords and identifiers are:
int is a keyword
main is an identifier
int is keyword
a is an identifier
int is a keyword
b is an identifier
int is a keyword
c is an identifier
a is an identifier
b is an identifier
printf is an identifier
n is an identifier
C is an identifier
value is an identifier
is is an identifier
d is an identifier
c is an identifier
Special characters are:
( ){ = ; = ; = + ; ( “ \ : % ” , ) ; }
Total no. of lines are: 8
```

EXERCISE – 2

AIM: Write a Lex Program to implement a Lexical Analyzer using Lex tool.

PROGRAM:

```
%{
#include<stdio.h>
%}
DIGIT [0-9]
DIGITS {DIGIT}+
LETTER [A-Za-z]
DELIM [ \t\n]
WS {DELIM}+
NUMBER {DIGITS}(\.{DIGITS})?(E[+-]?{DIGITS})?
ID {LETTER}({LETTER}|{DIGIT})*
%%
{WS} { printf("\n WS special characters"); }
{NUMBER} { printf("\n%s Number",yytext); }
{ID} { printf("\n%s identifier",yytext); }
>|<|"<="|">="|"="|"!=" { printf("\n%s Relational Operators",yytext); }
"&&"|"||"|! { printf("\n%s Logical Operators",yytext); }
"+"|"-"|"*"|"/"|"%" { printf("\n%s Arthmetic Operator",yytext); }
%%
int yywrap()
{
 return 1;
}
int main()
{
printf("Enter any text: \n");
yylex();
return 0;
}
```

OUTPUT:

```
$ lex lexical2.l
$ cc lex.yy.c
$ ./a.out
Enter any text:
a=b+10

a identifier
= Relational Operators
b identifier
+ Arthmetic Operator
10 Number
```

EXERCISE – 3

AIM: Write a C program to Simulate Lexical Analyzer to validating a given input String.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
main()
{
int st=0,i=0,k;
char c,b[10];
FILE *fp;
clrscr();
fp=fopen("lex.c","r");
while(!feof(fp))
{
switch(st)
{
case 0:
c=getc(fp);
if(c=='(')
st=1;
else if(c=='>')
st=5;
else if(c=='=')
st=8;
else if(c=='+')
st=11;
else if(c=='-')
st=12;
else if(c=='*')
st=13;
else if(c=='%')
st=14;
else if (c==' ')
st=15;
else if (c=='\t')
st=17;
else if(isalpha(c))
{
b[i++]=c;
st=9;
}
break;
case 1:
```

```
c=getc(fp);
if(c=='-')
st=2;
else if(c=='>')
st=3;
else
st=4;
break;
case 2:
printf("\nis identifed");
st=0;
break;
case 3:
printf("\nNE is identifed");
st=0;
break;
case 4:
ungetc(c,fp);
printf("\nLT is identifed");
st=0;
break;
case 5:
c=getc(fp);
if(c=='=')
st=6;
else
st=7;
break;
case 6:
printf("\nGE is identifed ");
st=0;
break;
case 7:
ungetc(c,fp);
printf("\nGT is identifed");
st=0;
break;
case 8:
printf("\nEQ is identifed");
st=0;
break;
case 9:
c=getc(fp);
if(isalpha(c) || isdigit(c))
{
st=9;
```

```
b[i++]=c;
}
else
{
st=10;
b[i]='\0';
}
break;
case 10:
ungetc(c,fp);
k=install_id(b);
if(k==0)
printf("\n%s keyword",b);
else
printf("\n%s identifer",b);
st=0;
i=0;
break;
case 11:
printf("\n+ is identifed");
st=0;
break;
case 12:
printf("\n- is identified");
st=0;
break;
case 13:
printf("\n* is identified");
st=0;
break;
case 14:
printf("\n/ is identfied ");
st=0;
break;
case 15:
printf("\nmodule is identifed");
st=0;
break;
case 16:
printf("\nnew line operator is used ");
st=0;
break;
case 17:
c=fgetc(fp);
if(c=='/')
{
```

```
while((c=fgetc(fp))!='\n')
st=20;
}
else if (c=='*')
{
while((c=fgetc(fp))!=EOF)
if(c=='*')
{
if((c=fgetc(fp))=='/')
{
st=20;
break;
}
}
}
else
st=9;
break;
case 20:
printf("\ncomment is identified");
st=0;
break;
}
}
fclose(fp);
getch();
}
int install_id(char b[])
{
char a[20][20]={"int","char","float","switch","break","if","else","for","while","case","exit","return"};
int i=0,k=1;
for(i=0;i<10;i++)
{
if(strcmp(b,a[i])==0)
{
k=0;
return k;
}
}
return k;
}
```

INPUT: (“lex.c” file contents)

```
A+B*C
```

OUTPUT:

```
A identifier
+ identified
B identifier
* identified
C identifier
```

EXERCISE – 4

AIM: Write a C program to implement the Brute force technique of Top down Parsing.

PROGRAM:

```
#include<stdio.h>
char c[10];
int i=0;
main()
{
clrscr();
printf("\nEnter input string: ");
scanf("%s",c);
if(s()==0)
printf("The given input string is not valid");
else
printf("The given input string is valid");
getch();
}
int s()
{
if(c[i]=='c')
{
advance();
if(A())
{
if(c[i]=='d')
{
advance();
return 1;
}
}
}
return 0;
}
advance()
{
i=i+1;
}
int A()
{
int isave;
{
isave=1;
if(c[i]=='a');
{
```

```
advance();
if(c[i]=='b')
{
advance();
return 1;
}
}
i=isave;
if(c[i]=='a')
{
advance();
return 1;
}
return 0;
}
}
```

OUTPUT:

```
Enter input string: cad
The given input string is valid
```

EXERCISE – 5

AIM: Write a C program to implement a Recursive Descent Parser.

PROGRAM:

```
#include<stdio.h>
char c[10];
int isym=0,flag=0;
main()
{
clrscr();
printf("\nEnter the input string: ");
scanf("%s",c);
E();
if(flag==1)
printf("not valid");
else
printf("valid");
getch();
}
E()
{
T();
eprime();
}
eprime()
{
if(c[isym]=='+')
{
advance();
T();
eprime();
}
}
T()
{
F();
tprime();
}
F()
{
if(c[isym]=='i')
{
advance();
if(c[isym]=='i')
error();
}
else
if(c[isym]=='c')
{
advance();
E();
if(c[isym]==')')
advance();
else
error();
}
else
error();
}
tprime()
{
if(c[isym]=='*')
{
advance();
F();
tprime();
}
}
advance()
{
isym++;
}
error()
{
flag=1;
}
```

OUTPUT-1:

```
Enter the input string: i*i+i
valid
```

OUTPUT-2:

```
Enter the input string: i(i)
valid
```

OUTPUT-3:

```
Enter the input string: i*i+c
not valid
```

EXERCISE – 6

AIM: Write C program to compute the First and Follow Sets for the given Grammar.

PROGRAM 6(a):

AIM: Write a C program to compute the First set for the given Grammar.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char t[5],nt[10],p[5][5],first[5][5],temp; int i,j,not,nont,k=0,f=0;
clrscr();
printf("\nEnter the no. of Non-terminals in the grammar: "); scanf("%d",&nont);
printf("\nEnter the Non-terminals in the  grammar:\n");
for(i=0;i<nont;i++)
{       scanf("\n%c",&nt[i]);             }
printf("\nEnter the no. of Terminals in the grammar (Enter e for epsilon): "); scanf("%d",&not);
printf("\nEnter the Terminals in the grammar:\n");
for(i=0;i<not||t[i]=='$';i++)
{       scanf("\n%c",&t[i]);   }
for(i=0;i<nont;i++)
{       p[i][0]=nt[i]; first[i][0]=nt[i];
}
printf("\nEnter the productions :\n");
for(i=0;i<nont;i++)
{
scanf("%c",&temp);
printf("\nEnter the production for %c ( End the production with '$' sign ): ",p[i][0]);
for(j=0;p[i][j]!='$';)
{
j+=1;
scanf("%c",&p[i][j]);
}
}
for(i=0;i<nont;i++)
{
printf("\nThe production for %c -> ",p[i][0]);
for(j=1;p[i][j]!='$';j++)
{ printf("%c",p[i][j]); }
}
for(i=0;i<nont;i++)
{
f=0;
for(j=1;p[i][j]!='$';j++)
{
```

```
for(k=0;k<not;k++)
{
if(f==1)
break;
if(p[i][j]==t[k])
{
first[i][j]=t[k]; first[i][j+1]='$'; f=1;
break;
}
else if(p[i][j]==nt[k])
{
first[i][j]=first[k][j];
if(first[i][j]=='e')
continue;
first[i][j+1]='$'; f=1;
break;
}
}
}
}
}
for(i=0;i<nont;i++)
{
printf("\nThe first of %c -> ",first[i][0]);
for(j=1;first[i][j]!='$';j++)
{
printf("%c\t",first[i][j]);
}
}
getch();
}
```

OUTPUT:

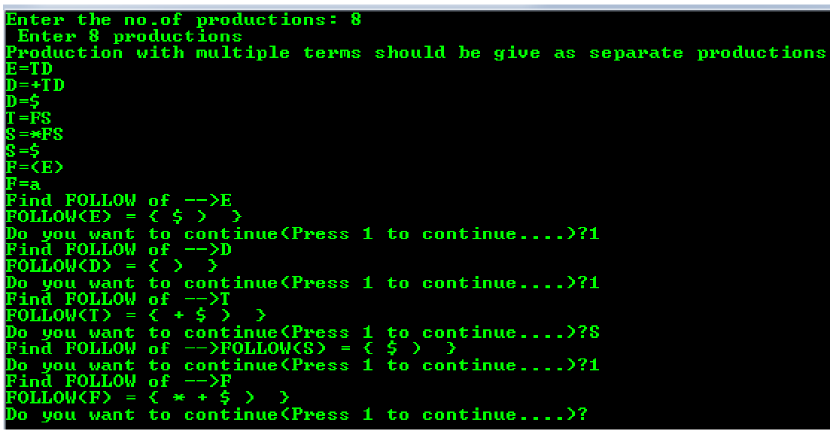
```
Enter the no. of Non-terminals in the grammar: 3
Enter the Non-terminals in the grammar: ERT
Enter the no. of Terminals in the grammar (Enter e for epsilon): 5
Enter the Terminals in the grammar: ase*+
Enter the productions :
Enter the production for E ( End the production with '$' sign ): a+s$
Enter the production for R ( End the production with '$' sign ): e$
Enter the production for T ( End the production with '$' sign ): Rs$
The production for E -> a+s
The production for R -> e
The production for T -> Rs
The first of E -> a
The first of R -> e
The first of T -> e s
```

PROGRAM 6(b):

AIM: Write a C program to find follow set for the given grammar.

```
#include<stdio.h>
#include<string.h>
int n,m=0,p,i=0,j=0;
char a[10][10],followResult[10];
void follow(char c);
void first(char c);
void addToResult(char);
int main()
{
 int i;
 int choice;
 char c,ch;
 printf("Enter the no.of productions: ");
scanf("%d", &n);
 printf(" Enter %d productions\nProduction with multiple terms should be give as separate productions \n", n);
 for(i=0;i<n;i++)
  scanf("%s%c",a[i],&ch);
    // gets(a[i]);
 do
 {
  m=0;
  printf("Find FOLLOW of -->");
  scanf(" %c",&c);
  follow(c);
  printf("FOLLOW(%c) = { ",c);
  for(i=0;i<m;i++)
   printf("%c ",followResult[i]);
  printf(" }\n");
  printf("Do you want to continue(Press 1 to continue....)?");
 scanf("%d%c",&choice,&ch);
 }
 while(choice==1);
}
void follow(char c)
{
    if(a[0][0]==c)
addToResult('$');
 for(i=0;i<n;i++)
 {
  for(j=2;j<strlen(a[i]);j++)
  {
   if(a[i][j]==c)
   {
    if(a[i][j+1]!='\0')
    first(a[i][j+1]);
    if(a[i][j+1]=='\0'&&c!=a[i][0])
     follow(a[i][0]);
   }
  }
 }
}
void first(char c)
{
      int k;
                 if(!(isupper(c)))
                     //f[m++]=c;
                     addToResult(c);
                 for(k=0;k<n;k++)
                 {
                 if(a[k][0]==c)
                 {
                 if(a[k][2]=='$') follow(a[i][0]);
                 else if(islower(a[k][2]))
                     //f[m++]=a[k][2];
                     addToResult(a[k][2]);
                 else first(a[k][2]);
                 }
                 }
}
void  addToResult(char c)
{
    int i;
    for( i=0;i<=m;i++)
        if(followResult[i]==c)
            return;
   followResult[m++]=c;
}
```

OUTPUT:



```
Enter the no.of productions: 8
 Enter 8 productions
Production with multiple terms should be give as separate productions
E=TD
D=+TD
D=$
T=FS
S=*FS
S=$
F=(E)
F=a
Find FOLLOW of -->E
FOLLOW(E) = { $ )  }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->D
FOLLOW(D) = { )  }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->T
FOLLOW(T) = { + $ )  }
Do you want to continue(Press 1 to continue....)?S
Find FOLLOW of -->FOLLOW(S) = { $ )  }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->F
FOLLOW(F) = { * + $ )  }
Do you want to continue(Press 1 to continue....)?
```

EXERCISE – 7

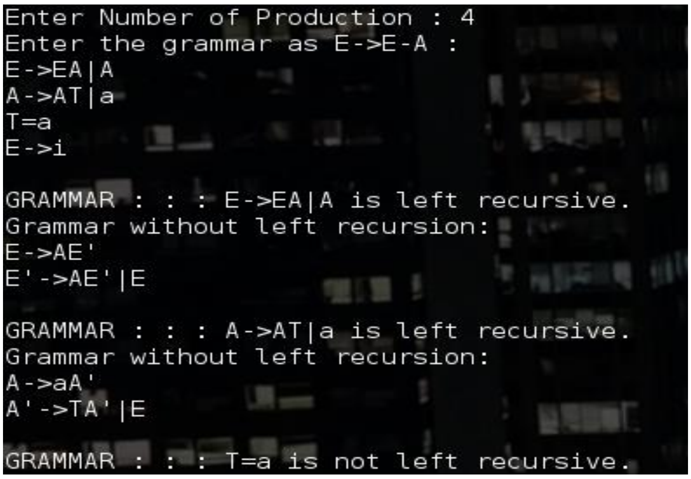
AIM: Write a C program for eliminating the left recursion and left factoring of a given grammar.

PROGRAM 7(a):

AIM: Write a C program for eliminating the left recursion of a given grammar.

```
#include<stdio.h>
#include<string.h>
#define SIZE 10
int main ()
{
  char non_terminal;
  char beta,alpha;
  int num;
  char production[10][SIZE];
  int index=3; /* starting of the string following "->" */
  printf("Enter Number of Production : ");
  scanf("%d",&num);
  printf("Enter the grammar as E->E-A :\n");
  for(int i=0;i<num;i++)
  {         scanf("%s",production[i]);    }
  for(int i=0;i<num;i++)
 {
       printf("\nGRAMMAR : : : %s",production[i]);
       non_terminal=production[i][0];
       if(non_terminal==production[i][index])
      {
          alpha=production[i][index+1];
          printf(" is left recursive.\n");
          while(production[i][index]!=0 && production[i][index]!='|')
             index++;
          if(production[i][index]!=0)
         {
              beta=production[i][index+1];
              printf("Grammar without left recursion:\n");
              printf("%c->%c%c\'",non_terminal,beta,non_terminal);
              printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
         }
         else
              printf(" can't be reduced\n");
      }
      else
         printf(" is not left recursive.\n");
      index=3;
   }
}
```

OUTPUT:



```
Enter Number of Production : 4
Enter the grammar as E->E-A :
E->EA|A
A->AT|a
T=a
E->i

GRAMMAR : : : E->EA|A is left recursive.
Grammar without left recursion:
E->AE'
E'->AE'|E

GRAMMAR : : : A->AT|a is left recursive.
Grammar without left recursion:
A->aA'
A'->TA'|E

GRAMMAR : : : T=a is not left recursive.
```

PROGRAM 7(b):

AIM: Write a C program for eliminating the left factoring of a given grammar.

```
#include<stdio.h>
#include<string.h>
int main()
{
   char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
   int i,j=0,k=0,l=0,pos;
   printf("Enter Production: A->");
   gets(gram);
   for(i=0;gram[i]!='|';i++,j++)
      part1[j]=gram[i];
   part1[j]='\0';
   for(j=++i,i=0;gram[j]!='\0';j++,i++)
      part2[i]=gram[j];
   part2[i]='\0';
   for(i=0;i<strlen(part1)||i<strlen(part2);i++){
      if(part1[i]==part2[i]){
         modifiedGram[k]=part1[i];
         k++;
         pos=i+1;
      }
   }
   for(i=pos,j=0;part1[i]!='\0';i++,j++){
      newGram[j]=part1[i];
   }
   newGram[j++]='|';
   for(i=pos;part2[i]!='\0';i++,j++){
      newGram[j]=part2[i];
   }
   modifiedGram[k]='X';
   modifiedGram[++k]='\0';
   newGram[j]='\0';
   printf("\nGrammar without Left Factoring: \n");
   printf(" A->%s",modifiedGram);
   printf("\n X->%s\n",newGram);
}
```

OUTPUT:

```
Enter Production: A->bE+acF|bE+f
Grammar without Left Factoring:
A-> bE+X
X-> acF|f
```

EXERCISE – 8

AIM: Write a C program to check the validity of input string using Predictive Parser.

PROGRAM:

```
#include<stdio.h>
int stack[20],top=-1;
void push(int item)
{
  if(top>=20)
  {
   printf("STACK OVERFLOW");
   exit(1);
  }
  stack[++top]=item;
 }
 int pop()
 {
   int ch;
   if(top<=-1)
   {
     printf("underflow");
     exit(1);
   }
   ch=stack[top--];
   return ch;
}
char convert(int item)
{
  char ch;
  switch(item)
  {
    case 0:return('E');
    case 1:return('e');
    case 2:return('T');
    case 3:return('t');
    case 4:return('F');
    case 5:return('i');
    case 6:return('+');
    case 7:return('*');
    case 8:return('(');
    case 9:return(')');
    case 10:return('$');
  }
}
void main()
```

```
{
 int m[10][10],i,j,k;
 char ips[20];
 int ip[10],a,b,t;
 m[0][0]=m[0][3]=21;
 m[1][1]=621;
 m[1][4]=m[1][5]=-2;
 m[2][0]=m[2][3]=43;
 m[3][1]=m[3][4]=m[3][5]=-2;
 m[3][2]=743;
 m[4][0]=5;
 m[4][3]=809;
 clrscr();
 printf("\n enter the input string:");
 scanf("%s",ips);
 for(i=0;ips[i];i++)
 {
    switch(ips[i])
    {
    case 'E':k=0;break;
    case 'e':k=1;break;
    case 'T':k=2;break;
    case 't':k=3;break;
    case 'F':k=4;break;
    case 'i':k=5;break;
    case '+':k=6;break;
    case '*':k=7;break;
    case '(':k=8;break;
    case ')':k=9;break;
    case '$':k=10;break;
 }
 ip[i]=k;
}
ip[i]=-1;
push(10);
push(0);
i=0;
printf("\tstack\t       input \n");
while(1)
{
   printf("\t");
   for(j=0;j<=top;j++)
   printf("%c",convert(stack[j]));
   printf("\t\t");
   for(k=i;ip[k]!=-1;k++)
   printf("%c",convert(ip[k]));
```

```
    printf("\n");
    if(stack[top]==ip[i])

    {
      if(ip[i]==10)
      {
          printf("\t\t SUCCESS");
          return;
      }
      else
     {
       top--;
       i++;
     }
  }
  else if(stack[top]<=4&&stack[top]>=0)
    {
       a=stack[top];
       b=ip[i]-5;
       t=m[a][b];
       top--;
       while(t>0)
          {
            push(t%10);
            t=t/10;
       }
       }
       else
       {
           printf("ERROR");
           return;
      }
     }
  getch();
}
```

OUTPUT:

```
enter the string:i+(i*i)$

stack           input
$E              i+(i*i)$
$eT             i+(i*i)$
$etF            i+(i*i)$
$eti           i+(i*i)$
$et             +(i*i)$
$e              +(i*i)$
$eT+            +(i*i)$
$eT              (i*i)$
$etF             (i*i)$
$et)E(           (i*i)$
$et)E             i*i)$
$et)eT            i*i)$
$et)etF           i*i)$
$et)eti           i*i)$
$et)et            *i)$
$et)etF*           *i)$
$et)etF             i)$
$et)eti             i)$
$et)et              )$
$et)e               )$
$et)                )$
$et                  $
$e                   $
$                    $
```

EXERCISE – 9

AIM: Write a C program for implementation of LR parsing algorithm to accept a given input string.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
int axn[][6][2]={
          {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
          {{-1,-1},{100,6},{-1,-1},{-1,-1},{-1,-1},{102,102}},
          {{-1,-1},{101,2},{100,7},{-1,-1},{101,2},{101,2}},
          {{-1,-1},{101,4},{101,4},{-1,-1},{101,4},{101,4}},
          {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
          {{-1,-1},{101,6},{101,6},{-1,-1},{101,6},{101,6}},
          {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
          {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
          {{-1,-1},{100,6},{-1,-1},{-1,-1},{100,1},{-1,-1}},
          {{-1,-1},{101,1},{100,7},{-1,-1},{101,1},{101,1}},
          {{-1,-1},{101,3},{101,3},{-1,-1},{101,3},{101,3}},
          {{-1,-1},{101,5},{101,5},{-1,-1},{101,5},{101,5}}
};//Axn Table

int    gotot[12][3]={1,2,3,-1,-1,-1,-1,-1,-1,-1,-1,-1,8,2,3,-1,-1,-1,-1,9,3,-1,-1,10,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};  //GoTo table

int a[10];
char b[10];
int top=-1,btop=-1,i;

void push(int k)
{
  if(top<9)
    a[++top]=k;
}
void pushb(char k)
{
  if(btop<9)
    b[++btop]=k;
}
char TOS()
{
   return a[top];
}
void pop()
{
   if(top>=0)
```

```
     top--;
}
void popb()
{
   if(btop>=0)
     b[btop--]='\0';
}
void display()
{
  for(i=0;i<=top;i++)
    printf("%d%c",a[i],b[i]);
}
void display1(char p[],int m) //Displays The Present Input String
{
  int l;
  printf("\t\t");
  for(l=m;p[l]!='\0';l++)
    printf("%c",p[l]);
  printf("\n");
}
void error()
{
  printf("Syntax Error");
}
void reduce(int p)
{
   int len,k,ad;
   char src,*dest;
   switch(p)
   {
 case 1: dest="E+T"; src='E';break;
 case 2: dest="T"; src='E'; break;
 case 3: dest="T*F"; src='T'; break;
 case 4: dest="F"; src='T'; break;
 case 5: dest="(E)"; src='F'; break;
 case 6: dest="i"; src='F'; break;
 default: dest="\0"; src='\0'; break;
   }
   for(k=0;k<strlen(dest);k++)
   {
     pop();
     popb();
   }
   pushb(src);
   switch(src)
   {
```

```
 case 'E': ad=0;break;
 case 'T': ad=1;break;
 case 'F': ad=2;break;
 default: ad=-1;break;
   }
  push(gotot[TOS()][ad]);
}
int main()
{
   int j,st,ic;
   char ip[20]="\0",an;
   clrscr();
   printf("Enter any String\n");
   scanf("%s",ip);
   printf("STACK\t\tINPUT\n");
   push(0);
   display();
   printf("\t\t%s\n",ip);
   for(j=0;ip[j]!='\0';)
   {
    st=TOS();
    an=ip[j];
    if(an>='a'&&an<='z') ic=0;
    else if(an=='+') ic=1;
    else if(an=='*') ic=2;
    else if(an=='(') ic=3;
    else if(an==')') ic=4;
    else if(an=='$') ic=5;
    else
    {
     error();
     break;
    }
    if(axn[st][ic][0]==100)
    {
      pushb(an);
      push(axn[st][ic][1]);
      display();
      j++;
      display1(ip,j);
   }
  if(axn[st][ic][0]==101)
  {
   reduce(axn[st][ic][1]);
   display();
   display1(ip,j);
  }
  if(axn[st][ic][1]==102)
  {
      printf("Given String is accepted \n");
      getch();
      break;
  }
}
return 0;
}
```

OUTPUT:

```
Enter any String
a+a*a$
STACK                   INPUT
0                       a+a*a$
0a5                     +a*a$
0F3                     +a*a$
0T2                     +a*a$
0E1                     +a*a$
0E1+6                   a*a$
0E1+6a5                 *a$
0E1+6F3                 *a$
0E1+6T9                 *a$
0E1+6T9*7                 a$
0E1+6T9*7a5                $
0E1+6T9*7F10               $
0E1+6T9                    $
0E1                        $
Given String is accepted
```

EXERCISE – 10

AIM: Write a C program for implementation of a Shift Reduce Parser using Stack Data Structure to accept a given input string of a given grammar.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
char stack[30];
int top=-1;
void push(char c)
{
top++;
stack[top]=c;
}
char pop()
{
char c;
if(top!=-1)
{
c=stack[top];
top--;
return c;
}
return 'x';
}
void printstat()
{
int i;
printf("\n$");
for(i=0;i<=top;i++)
printf("%c",stack[i]);
}
void main()
{
int i,j,k,len;
char s1[20],s2[20],ch1,ch2,ch3;
clrscr();
printf("LR PARSING\n");
printf("ENTER THE EXPRESSION\n");
scanf("%s",s1);
len=strlen(s1);
j=0;
printf("$");
for(i=0;i<len;i++)
{
```

```
if(s1[i]=='i' && s1[i+1]=='d')
{
s1[i]=' ';
s1[i+1]='E';
printstat(); printf("id");
push('E');
printstat();
}
else if(s1[i]=='+'||s1[i]=='-'||s1[i]=='*' ||s1[i]=='/' ||s1[i]=='d')
{
push(s1[i]);
printstat();
}
 }
printstat();
len=strlen(s2);
while(len)
{
ch1=pop();
if(ch1=='x')
{
printf("\n$");
break;
}
if(ch1=='+'||ch1=='/'||ch1=='*'||ch1=='-')
{
ch3=pop();
if(ch3!='E')
{
printf("errror");
exit();
}
else
{
push('E');
printstat();
}
}
ch2=ch1;
}
getch();
}
```

OUTPUT:

```
LR PARSING
ENTER THE EXPRESSION
id+id*id-id
$
$id
$E
$E+
$E+id
$E+E
$E+E*
$E+E*id
$E+E*E
$E+E*E-
$E+E*E-id
$E+E*E-E
$E+E*E-E
$E+E*E
$E
$
```

EXERCISE – 11

AIM: Simulate the calculator using LEX and YACC tool.

PROGRAM:
LEX PART:

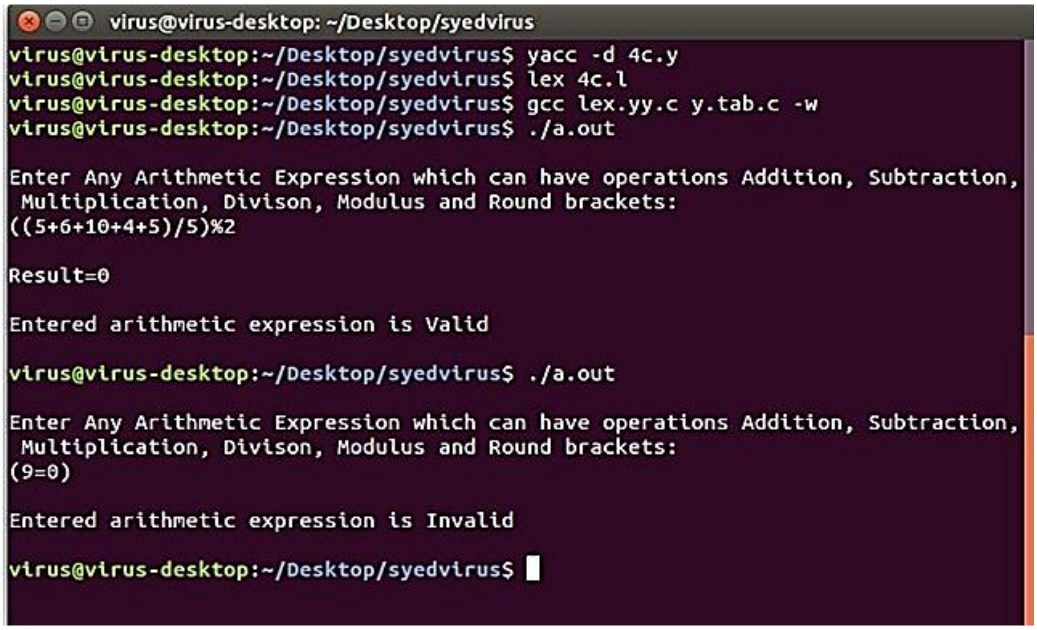
```
%{
/* Definition section */
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

/* Rule Section */
%%
[0-9]+ {
          yylval=atoi(yytext);
          return NUMBER;
       }
[\t] ;
[\n] return 0;
. return yytext[0];
%%

int yywrap()
{
 return 1;
}
```

YACC PART:

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

%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;}
;

%%
void main()
{
   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication,
  Divison, 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:



virus@virus-desktop: ~/Desktop/syedvirus

```
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 4c.y
virus@virus-desktop:~/Desktop/syedvirus$ lex 4c.l
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Divison, Modulus and Round brackets:
((5+6+10+4+5)/5)%2

Result=0

Entered arithmetic expression is Valid

virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Divison, Modulus and Round brackets:
(9=0)

Entered arithmetic expression is Invalid

virus@virus-desktop:~/Desktop/syedvirus$ 
```

EXERCISE – 12

AIM: Generate YACC specification for a few syntactic categories.
(A) Program that recognize a valid arithmetic expression that uses operator +,-,* and /
(B) Program to recognize a valid variable which starts with a letter followed by any number of letters or digits

PROGRAM 12(A):

AIM: program that recognize a valid arithmetic expression that uses operator +,-,* and /

Program name: arith_id.l

```
%{
/* This LEX program returns the tokens for the expression */
#include “y.tab.h”
%}

%%
“=” {printf(“\n Operator is EQUAL”);}
“+” {printf(“\n Operator is PLUS”);}
“-“ {printf(“\n Operator is MINUS”);}
“/” {printf(“\n Operator is DIVISION”);}
“*” {printf(“\n Operator is MULTIPLICATION”);}

[a-z A-Z]*[0-9]* {
printf(“\n Identifier is %s”,yytext);
return ID;
}
return yytext[0];
\n return 0;
%%

int yywrap()
{
return 1;
}
```

Program Name: arith_id.y

```
%{
#include
/* This YYAC program is for recognizing the Expression */
%}
%%
statement: A’=’E
| E {
printf(“\n Valid arithmetic expression”);
$$ = $1;
};

E: E’+’ID
| E’-’ID
| E’*’ID
| E’/’ID
| ID
;
%%
extern FILE *yyin;
main()
{
do
{
yyparse();
}while(!feof(yyin));
}

yyerror(char*s)
{
}
```

OUTPUT:

```
[root@localhost]# lex arith_id.1
[root@localhost]# yacc –d arith_id.y
[root@localhost]# gcc lex.yy.c y.tab.c
[root@localhost]# ./a.out
x=a+b;

Identifier is x
Operator is EQUAL
Identifier is a
Operator is PLUS
Identifier is b
```

PROGRAM 12(B):

AIM: Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.

Program name: variable_test.l

```
%{
/* This LEX program returns the tokens for the Expression */
#include "y.tab.h"
%}
%%
"int " {return INT;}
"float" {return FLOAT;}
"double" {return DOUBLE;}
[a-zA-Z]*[0-9]*{
printf("\nIdentifier is %s",yytext);
return ID;
}
return yytext[0];
\n return 0;
int yywrap()
{
return 1;
}
```

Program name: variable_test.y

```
%{
#include
/* This YACC program is for recognizing the Expression*/
%}
%token ID INT FLOAT DOUBLE
%%
D;T L
;
L:L,ID
|ID
;
T:INT
|FLOAT
|DOUBLE
;
%%
extern FILE *yyin;
main()
{
do
{
```

```
yyparse();
}while(!feof(yyin));
}
yyerror(char*s)
{
}
```

OUTPUT:

```
[root@localhost]# lex variable_test.I
[root@localhost]# yacc –d variable_test.y
[root@localhost]# gcc lex.yy.c y.tab.c
[root@localhost]# ./a.out
int a,b;

Identifier is a
Identifier is b
```

EXERCISE – 13

AIM: Write a C program for generating the three address code of a given expression/statement.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
void pm();
void plus();
void div();
int i,ch,j,l,addr=100;
char ex[10], exp[10] ,exp1[10],exp2[10],id1[5],op[5],id2[5];
void main()
{
clrscr();
while(1)
{
printf("\n1.assignment\n2.arithmetic\n3.relational\n4.Exit\nEnter the choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nEnter the expression with assignment operator: ");
scanf("%s",exp);
l=strlen(exp);
exp2[0]='\0';
i=0;
while(exp[i]!='=')
{
i++;
}
strncat(exp2,exp,i);
strrev(exp);
exp1[0]='\0';
strncat(exp1,exp,l-(i+1));
strrev(exp1);
printf("Three address code:\ntemp=%s\n%s=temp\n",exp1,exp2);
break;

case 2:
printf("\nEnter the expression with arithmetic operator: ");
scanf("%s",ex);
strcpy(exp,ex);
l=strlen(exp);
exp1[0]='\0';
for(i=0;i<l;i++)
{
if(exp[i]=='+'||exp[i]=='-')
{
if(exp[i+2]=='/'||exp[i+2]=='*')
{
pm();
break;
}
else
{
plus();
break;
}
}
else if(exp[i]=='/'||exp[i]=='*')
{
div();
break;
}
}
break;

case 3:
printf("Enter the expression with relational operator: ");
scanf("%s%s%s",&id1,&op,&id2);
if(((strcmp(op,"<")==0)||(strcmp(op,">")==0)||(strcmp(op,"<=")==0)||(strcmp(op,">=")==0)||(strcmp(o
p,"==")==0)||(strcmp(op,"!=")==0))==0)
printf("Expression is error");
else
{
printf("\n%d\tif %s%s%s goto %d",addr,id1,op,id2,addr+3);
addr++;
printf("\n%d\t T:=0",addr);
addr++;
printf("\n%d\t goto %d",addr,addr+2);
addr++;
printf("\n%d\t T:=1",addr);
}
break;
case 4:
exit(0);
}
    }
}
void pm()
{
```

```
strrev(exp);
j=l-i-1;
strncat(exp1,exp,j);
strrev(exp1);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[j+1],exp[j]);
}
void div()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
void plus()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
```

OUTPUT:

```
1. assignment
2. arithmetic
3. relational
4. Exit
Enter the choice: 1
Enter the expression with assignment operator: a=b
Three address code:
temp=b
a=temp

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice: 2
Enter the expression with arithmetic operator: a+b-c
Three address code:
temp=a+b
temp1=temp-c

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice: 2
Enter the expression with arithmetic operator: a*b-c
Three address code:
temp=a*b
```

```
temp1=temp-c

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice: 3
Enter the expression with relational operator: a<=b

100 if a<=b goto 103
101 T:=0
102 goto 104
103 T:=1

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:4
```

EXERCISE – 14

AIM: Write a C program for implementation of a Code Generation Algorithm of a given expression/statement.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
typedef struct
{
char var[10];
int alive;
}
regist;
regist preg[10];
void substring(char exp[],int st,int end)
{
 int i,j=0;
 char dup[10]="";
 for(i=st;i<end;i++)
  dup[j++]=exp[i];
 dup[j]='\0';
 strcpy(exp,dup);
}
int getregister(char var[])
{ int i;
for(i=0;i<10;i++)
{
if(preg[i].alive==0)
{
strcpy(preg[i].var,var);
break;
}}
return(i);
}
void getvar(char exp[],char v[])
{ int i,j=0;
char var[10]="";
for(i=0;exp[i]!='\0';i++)
if(isalpha(exp[i]))
var[j++]=exp[i];
else
break;
strcpy(v,var);
}
void main()
```

```
{
 char basic[10][10],var[10][10],fstr[10],op;
 int i,j,k,reg,vc,flag=0;
 clrscr();
 printf("\nEnter the Three Address Code:\n");
 for(i=0;;i++)
 {
  gets(basic[i]);
  if(strcmp(basic[i],"exit")==0)
    break;
 }
 printf("\nThe Equivalent Assembly Code is:\n");
 for(j=0;strcmp(basic[j],"exit")!=0;j++)
 {
  getvar(basic[j],var[vc++]);
  strcpy(fstr,var[vc-1]);
  substring(basic[j],strlen(var[vc-1])+1,strlen(basic[j]));
  getvar(basic[j],var[vc++]);
  reg=getregister(var[vc-1]);
  if(preg[reg].alive==0)
  {
   printf("\nMov R%d,%s",reg,var[vc-1]);
   preg[reg].alive=1;
  }
  op=basic[j][strlen(var[vc-1])];
  substring(basic[j],strlen(var[vc-1])+1,strlen(basic[j]));
  getvar(basic[j],var[vc++]);
  switch(op)
  {
   case '+': printf("\nAdd"); break;
   case '-': printf("\nSub"); break;
   case '*': printf("\nMul"); break;
   case '/': printf("\nDiv"); break;
  }
  flag=1;
  for(k=0;k<=reg;k++)
  {
   if(strcmp(preg[k].var,var[vc-1])==0)
   {
     printf("R%d, R%d",k,reg);
     preg[k].alive=0;
     flag=0;
     break;
   }
  }
  if(flag)
  {
   printf(" %s,R%d",var[vc-1],reg);
   printf("\nMov %s,R%d",fstr,reg);
  }
  strcpy(preg[reg].var,var[vc-3]);
 }
 getch();
}
```

OUTPUT:

```
Enter the Three Address Code:
a=b+c
c=a*c
exit

The Equivalent Assembly Code is:
Mov  R0,b
Add   c,R0
Mov  a,R0
Mov  R1,a
Mul   c,R1
Mov  c,R1
```