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

Program:-

```
#include<stdio.h>
char c[10];
inti=0;
main()
{
clrscr();
printf("\n enter 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()
{
intisave;
{
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
```

5. Write a C program to implement a Recursive Descent Parser.
PROGRAM:

```
#include<stdio.h>
char c[10];
intisym=0,flag=0;
main()
{
clrscr();
printf("\n enter the input string");
scanf("%s",c);
E();
if(flag==1)
printf("notvalid");
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:

```
enter the input stringi*i+i
valid

enter the input stringi(i)
valid

enter the input stringi*i+c
notvalid
```

6(a). PROGRAM FOR COMPUTATION OF FIRST

```
#include<stdio.h>
#include<conio.h>
#include<string.h>        void
main()
{
          char          t[5],nt[10],p[5][5],first[5][5],temp;
          inti,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();
}
```

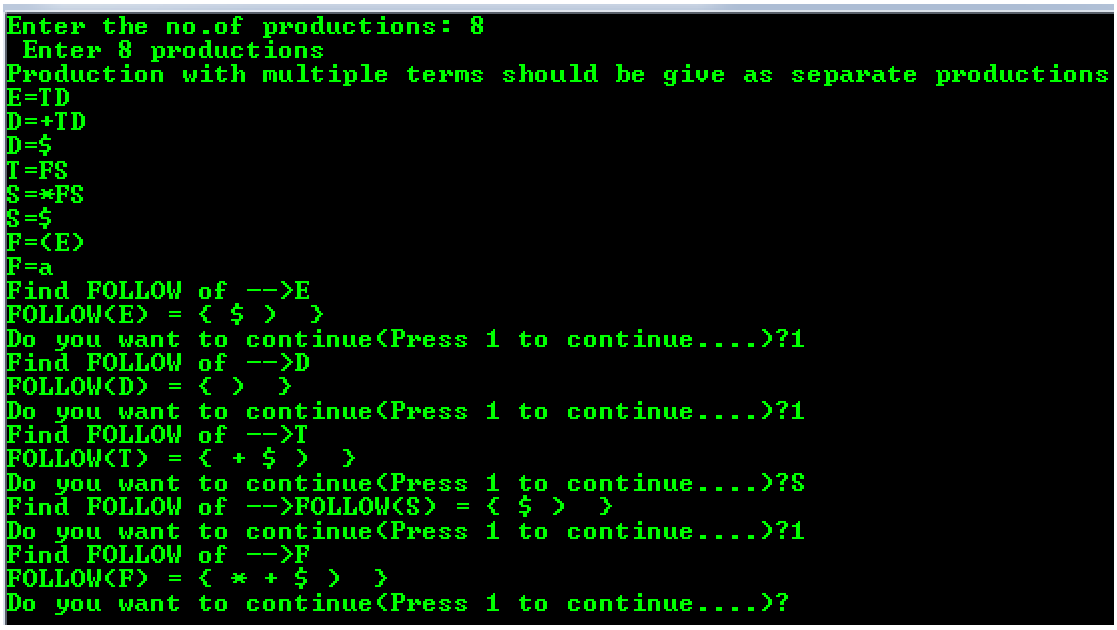
OUTPUT

```
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
```

6(b)Write a C program to find follow of a given grammar

```
#include<stdio.h>
#include<string.h>
intn,m=0,p,i=0,j=0;
char a[10][10],followResult[10];
void follow(char c);
void first(char c);
voidaddToResult(char);
int main()
{
inti;
int choice;
charc,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)
{
inti;
for(i=0;i<=m;i++)
if(followResult[i]==c)
return;
followResult[m++]=c;
}
```



```
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....)?
```

7a) Write a C program for eliminating the left recursion of a given grammar

What is left recursion?

Left Recursion:

Consider,

E->E+T

E=a

T=b

In it's parse tree E will grow left indefinitely, so to remove it

E=Ea | b

we take as

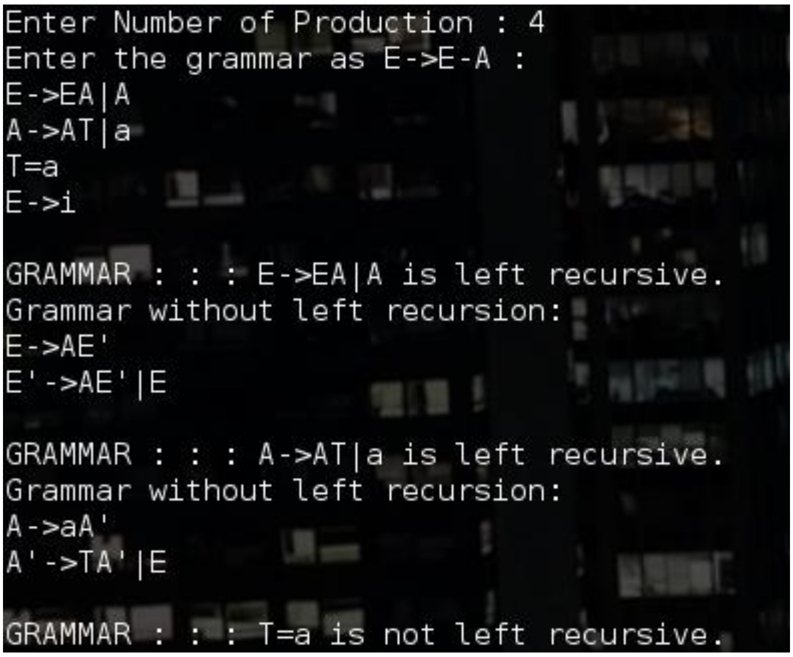
E=bE'

E'= aE'|E

Program :

```
#include<stdio.h>
#include<string.h>
#define SIZE 10
int main ()
{
charnon_terminal;
charbeta,alpha;
intnum;
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(inti=0;i<num;i++)
{
scanf("%s",production[i]);
  }
for(inti=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.
```

7 b) Write a C program for eliminating the left factoring of a given grammar

In LL(1) Parser in Compiler Design, Even if a context-free grammar is unambiguous and non-left-recursion it still cannot be a LL(1) Parser. That is because of Left Factoring.

What is Left Factoring?
Consider a part of regular grammar,

E->aE+bcD
E->aE+cBD

Here, grammar is non-left recursive, and unambiguous but there is left factoring.

How to resolve?
E=aB | aC | aD| ....

then,
E=aX
X=B | C | D |...........

So, the above grammar will be as :
E=aE+X
X=bcD | cBD

Program:

```
#include<stdio.h>
#include<string.h>
int main()
{
   char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
inti,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
```

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

```
/*program to implement PREDICTIVE PARSER */
#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()
 {
intch;
if(top<=-1)
   {
printf("underflow");
exit(1);
   }
ch=stack[top--];
returnch;
}
char convert(int item)
{
charch;
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;
charips[20];
intip[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
       $E               i+(i*i)$
       $eTi+(i*i)$
       $etFi+(i*i)$
$eti            i+(i*i)$
       $et              +(i*i)$
       $e               +(i*i)$
       $eT+              +(i*i)$
       $eT                (i*i)$
       $etF               (i*i)$
       $et)E(             (i*i)$
       $et)E               i*i)$
       $et)eT              i*i)$
       $et)etF             i*i)$
       $et)eti             i*i)$
       $et)et              *i)$
       $et)etF*             *i)$
       $et)etF                i)$
       $et)eti                i)$
$et)et                )$
       $et)e                  )$
       $et)                   )$
       $et                      $
       $e                       $
       $                        $
```