PART B
EXPERIMENT NUMBER 3

Aim: To implement the program to remove left recursion from grammar and find first and follow the given grammar.

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded at the end of the practical)

Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 05/03/2021	**Date of Submission:** 05/03/2021
Grade:	

B.1 Software Code written by a student:
(Paste your code completed during the 2 hours of practice in the lab here)

- **LEFT_RECURSION.C**

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

int i,j,l,m,n=0,o,p,nv,z=0,x=0;
char str[10],temp,temp2[10],temp3[20],*ptr;

struct prod
{
   char lhs[10],rhs[10][10],ft[10],fol[10];
   int n;
}pro[10];

void findter()
{
   int k,t;
   for(k=0;k<n;k++)
   {
      if(temp==pro[k].lhs[0])
      {
         for(t=0;t<pro[k].n;t++)
         {
            if( pro[k].rhs[t][0]<65 || pro[k].rhs[t][0]>90 )
               pro[i].ft[strlen(pro[i].ft)]=pro[k].rhs[t][0];
            else if( pro[k].rhs[t][0]>=65 && pro[k].rhs[t][0]<=90 )
            {
               temp=pro[k].rhs[t][0];
               if(temp=='S')
                  pro[i].ft[strlen(pro[i].ft)]='#';
               findter();
            }
         }
         break;
      }
   }
}

void findfol()
{
   int k,t,p1,o1,chk;
   char *ptr1;
   for(k=0;k<n;k++)
   {
      chk=0;
      for(t=0;t<pro[k].n;t++)
      {
         ptr1=strchr(pro[k].rhs[t],temp);
         if( ptr1 )
         {
            p1=ptr1-pro[k].rhs[t];
            if(pro[k].rhs[t][p1+1]>=65 && pro[k].rhs[t][p1+1]<=90)
            {
               for(o1=0;o1<n;o1++)
                  if(pro[o1].lhs[0]==pro[k].rhs[t][p1+1])
                  {
                        strcat(pro[i].fol,pro[o1].ft);
                        chk++;
                  }
            }
            else if(pro[k].rhs[t][p1+1]=='\0')
            {
               temp=pro[k].lhs[0];
               if(pro[l].rhs[j][p]==temp)
                  continue;
               if(temp=='S')
                  strcat(pro[i].fol,"$");
               findfol();
               chk++;
            }
            else
            {
               pro[i].fol[strlen(pro[i].fol)]=pro[k].rhs[t][p1+1];
               chk++;
            }
         }
      }
      if(chk>0)
         break;
   }
}

int main()
{
   FILE *f;
   //clrscr();

   for(i=0;i<10;i++)
      pro[i].n=0;

   f=fopen("SPCC-3.txt","r"); //READ this txt file
   while(!feof(f))
   {
      fscanf(f,"%s",pro[n].lhs);
      if(n>0)
      {
         if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )
         {
            pro[n].lhs[0]='\0';
            fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);
            pro[n-1].n++;
            continue;
         }
      }
      fscanf(f,"%s",pro[n].rhs[pro[n].n]);
      pro[n].n++;
      n++;
   }
```

```
printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");
for(i=0;i<n;i++)
   for(j=0;j<pro[i].n;j++)
      printf("%s = %s\n",pro[i].lhs,pro[i].rhs[j]);

pro[0].ft[0]='#';
for(i=0;i<n;i++)
{
   for(j=0;j<pro[i].n;j++)
   {
      if( pro[i].rhs[j][0]<65 || pro[i].rhs[j][0]>90 )
      {
         pro[i].ft[strlen(pro[i].ft)]=pro[i].rhs[j][0];
      }
      else if( pro[i].rhs[j][0]>=65 && pro[i].rhs[j][0]<=90 )
      {
         temp=pro[i].rhs[j][0];
         if(temp=='S')
            pro[i].ft[strlen(pro[i].ft)]='#';
         findter();
      }
   }
}

printf("\n\nFIRST\n");
for(i=0;i<n;i++)
{
   printf("\n%s = ",pro[i].lhs);
   for(j=0;j<strlen(pro[i].ft);j++)
   {
      for(l=j-1;l>=0;l--)
         if(pro[i].ft[l]==pro[i].ft[j])
            break;
      if(l==-1)
         printf("%c",pro[i].ft[j]);
   }
}

for(i=0;i<n;i++)
   temp2[i]=pro[i].lhs[0];
pro[0].fol[0]='$';
for(i=0;i<n;i++)
```

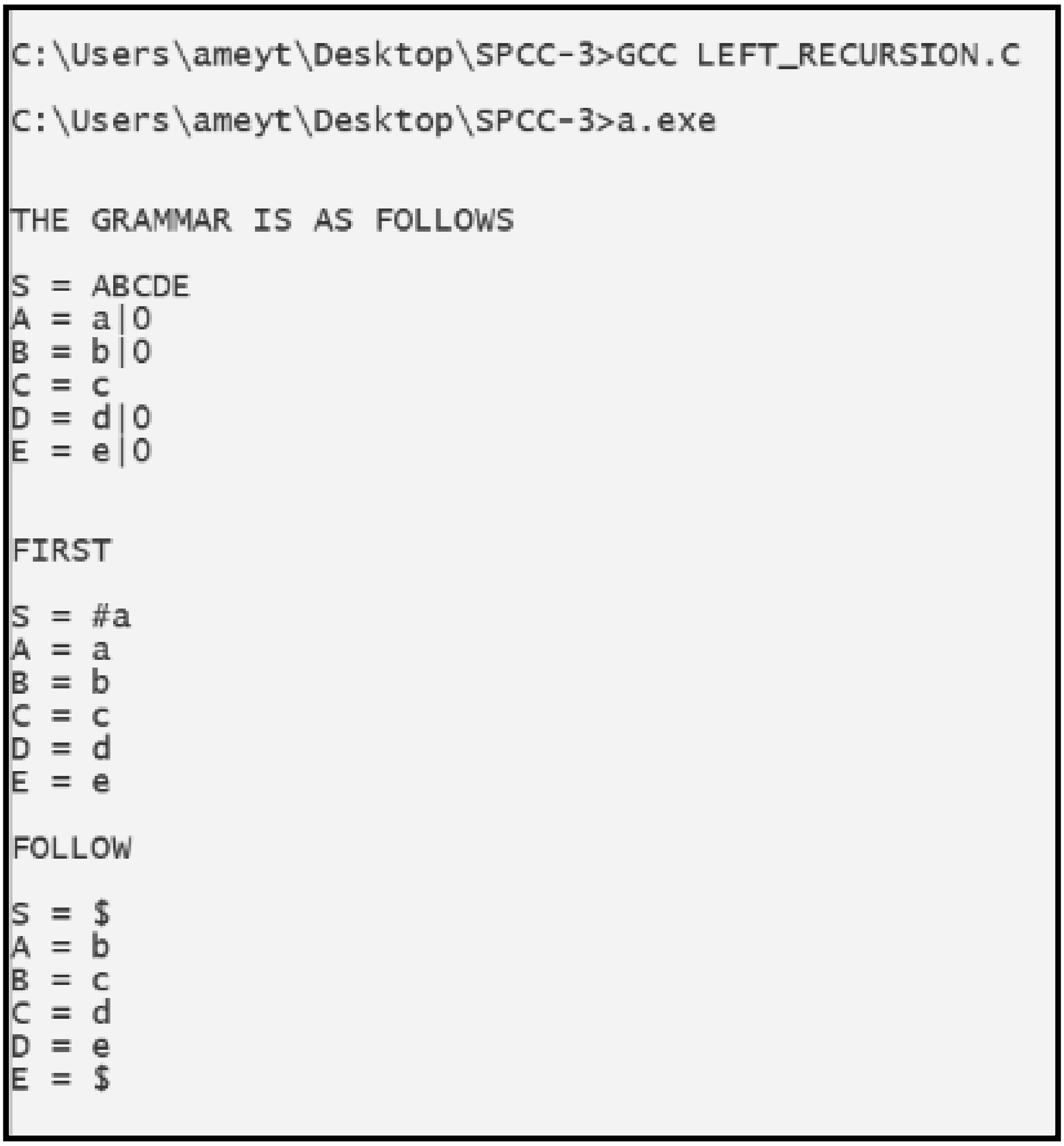
```
{
   for(l=0;l<n;l++)
   {
      for(j=0;j<pro[i].n;j++)
      {
         ptr=strchr(pro[l].rhs[j],temp2[i]);
         if( ptr )
         {
            p=ptr-pro[l].rhs[j];
            if(pro[l].rhs[j][p+1]>=65 && pro[l].rhs[j][p+1]<=90)
            {
               for(o=0;o<n;o++)
                  if(pro[o].lhs[0]==pro[l].rhs[j][p+1])
                        strcat(pro[i].fol,pro[o].ft);
            }
            else if(pro[l].rhs[j][p+1]=='\0')
            {
               temp=pro[l].lhs[0];
               if(pro[l].rhs[j][p]==temp)
                  continue;
               if(temp=='S')
                  strcat(pro[i].fol,"$");
               findfol();
            }
            else
               pro[i].fol[strlen(pro[i].fol)]=pro[l].rhs[j][p+1];
         }
      }
   }
}

printf("\n\nFOLLOW\n");
for(i=0;i<n;i++)
{
   printf("\n%s = ",pro[i].lhs);
   for(j=0;j<strlen(pro[i].fol);j++)
   {
      for(l=j-1;l>=0;l--)
         if(pro[i].fol[l]==pro[i].fol[j])
            break;
      if(l==-1)
         printf("%c",pro[i].fol[j]);
   }
```

```
    }
    printf("\n");
    //getch();
}
```

- **SPCC-3.TXT**

```
S ABCDE
A a|0
B b|0
C c
D d|0
E e|0
```

B.2 Input and Output:



```
C:\Users\ameyt\Desktop\SPCC-3>GCC LEFT_RECURSION.C

C:\Users\ameyt\Desktop\SPCC-3>a.exe


THE GRAMMAR IS AS FOLLOWS

S = ABCDE
A = a|0
B = b|0
C = c
D = d|0
E = e|0


FIRST

S = #a
A = a
B = b
C = c
D = d
E = e

FOLLOW

S = $
A = b
B = c
C = d
D = e
E = $
```

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ subpart assigned)

Thus we observed the method to remove left recursion along with various types of recursion also various other methods.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

Thus we have studied various recursions to be removed from the given grammar.

B.5 Question of Curiosity

(To be answered by a student based on the practical performed and learning/ observations)

1. What is the need for a Predictive parser?

Ans:

- ➔ The goal of predictive parsing is to construct a top-down parser that never backtracks. To do so, we must transform grammar in two ways:
    1. Eliminate left recursion, and
    2. Perform left factoring.
- ➔ These rules eliminate most common causes for backtracking although they do not guarantee a completely backtrack-free parsing

2. Difference between top-down and bottom-up parser?

Ans:

Sr. No.	Key	Top-Down Parsing	Bottom-Up Parsing
1	**Strategy**	The top-down approach starts evaluating the parse tree from the top and moves downwards for parsing other nodes.	The bottom-up approach starts evaluating the parse tree from the lowest level of the tree and moves upwards for parsing the node.
2	**Attempt**	Top-down parsing attempts to find the leftmost derivation for a given string.	Bottom-up parsing attempts to reduce the input string to the first symbol of the grammar.
3	**Derivation Type**	Top down parsing uses leftmost derivation.	Bottom-up parsing uses the rightmost derivation.

4	**Objective**	Top down parsing searches for a production rule to be used to construct a string.	Bottom-up parsing searches for a production rule to be used to reduce a string to get a starting symbol of the grammar.

3. Why is there a necessity of removing a left recursion?

Ans:

➔ Left recursion often poses problems for parsers, either because it leads them into infinite recursion (as in the case of most top-down parsers) or because they expect rules in a normal form that forbids it (as in the case of many bottom-up parsers, including the CYK algorithm). Therefore, grammar is often preprocessed to eliminate the left recursion.