LAB ASSIGNMENT -2

Name: Sharan P **Reg.No:** 21BRS1582

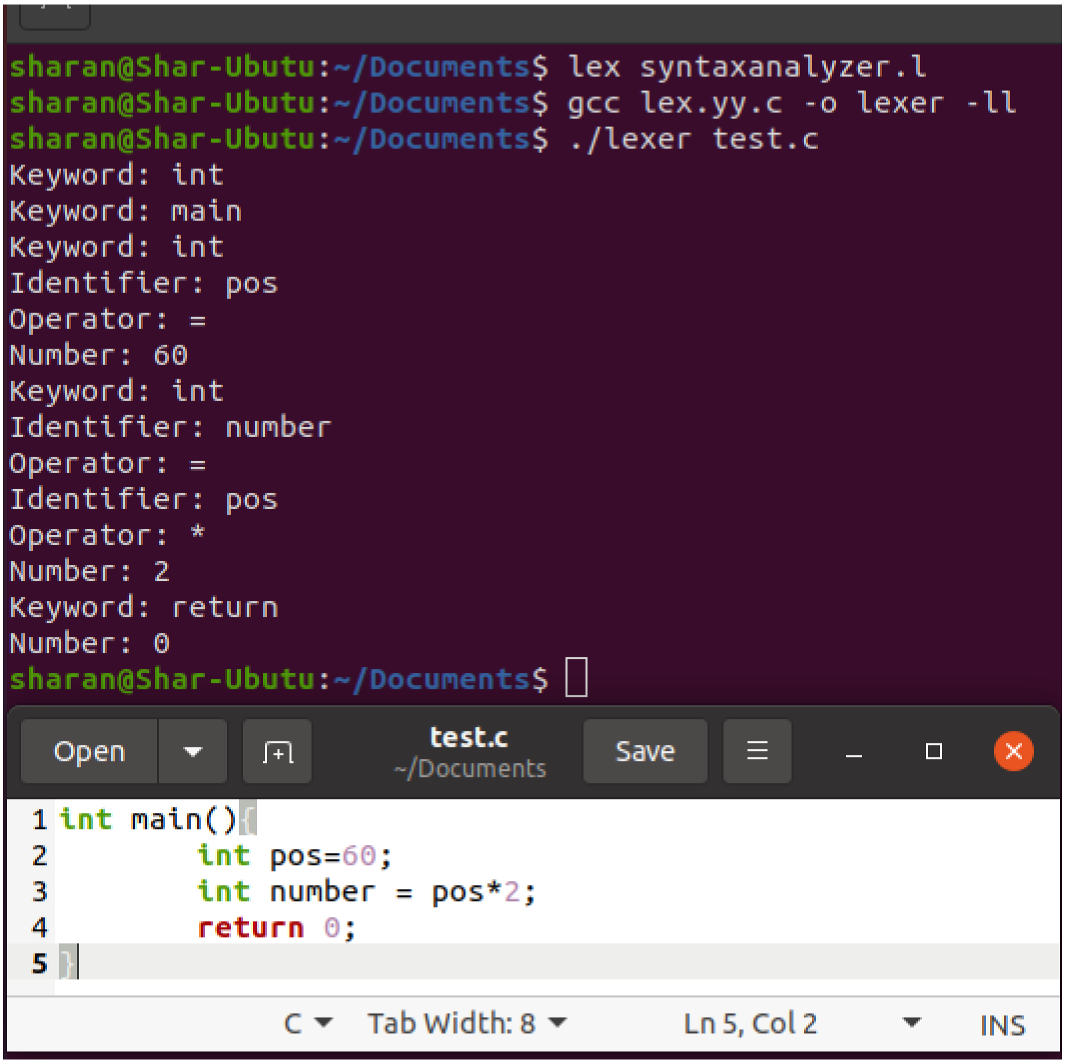
PROGRAM 1.

Aim: To create a syntax analyzer using lex tool.

Code:

```
%{
#include <stdio.h>
#include <string.h>
%}

%%

"int"|"char"|"float"|"double"|"void"|"return"|"main"    { printf("Keyword: %s\n", yytext); }
"+"|"-"|"*"|"/"|"**"|"//"|"%"|"="            {printf("Operator: %s\n",yytext);}

[a-zA-Z_][a-zA-Z0-9_]*                 { printf("Identifier: %s\n", yytext); }
";"|"("|")"|"{"|"}"                                ; //ignore coommas and brackets
[0-9]+                            { printf("Number: %s\n", yytext); }
[ \t\n]                           ; // Ignore whitespace characters
.                               { printf("Invalid character: %s\n", yytext); }


%%

int main(int argc, char *argv[]) {
   if (argc != 2) {
      printf("Usage: %s <input_file>\n", argv[0]);
      return 1;
   }

   FILE *file = fopen(argv[1], "r");
   if (!file) {
      perror("fopen");
      return 1;
   }

   yyin = file;
   yylex();
   fclose(file);
   return 0;
}
```

Output Screenshot:



```
sharan@Shar-Ubutu:~/Documents$ lex syntaxanalyzer.l
sharan@Shar-Ubutu:~/Documents$ gcc lex.yy.c -o lexer -ll
sharan@Shar-Ubutu:~/Documents$ ./lexer test.c
Keyword: int
Keyword: main
Keyword: int
Identifier: pos
Operator: =
Number: 60
Keyword: int
Identifier: number
Operator: =
Identifier: pos
Operator: *
Number: 2
Keyword: return
Number: 0
sharan@Shar-Ubutu:~/Documents$ 
```

```
int main(){
        int pos=60;
        int number = pos*2;
        return 0;
}
```

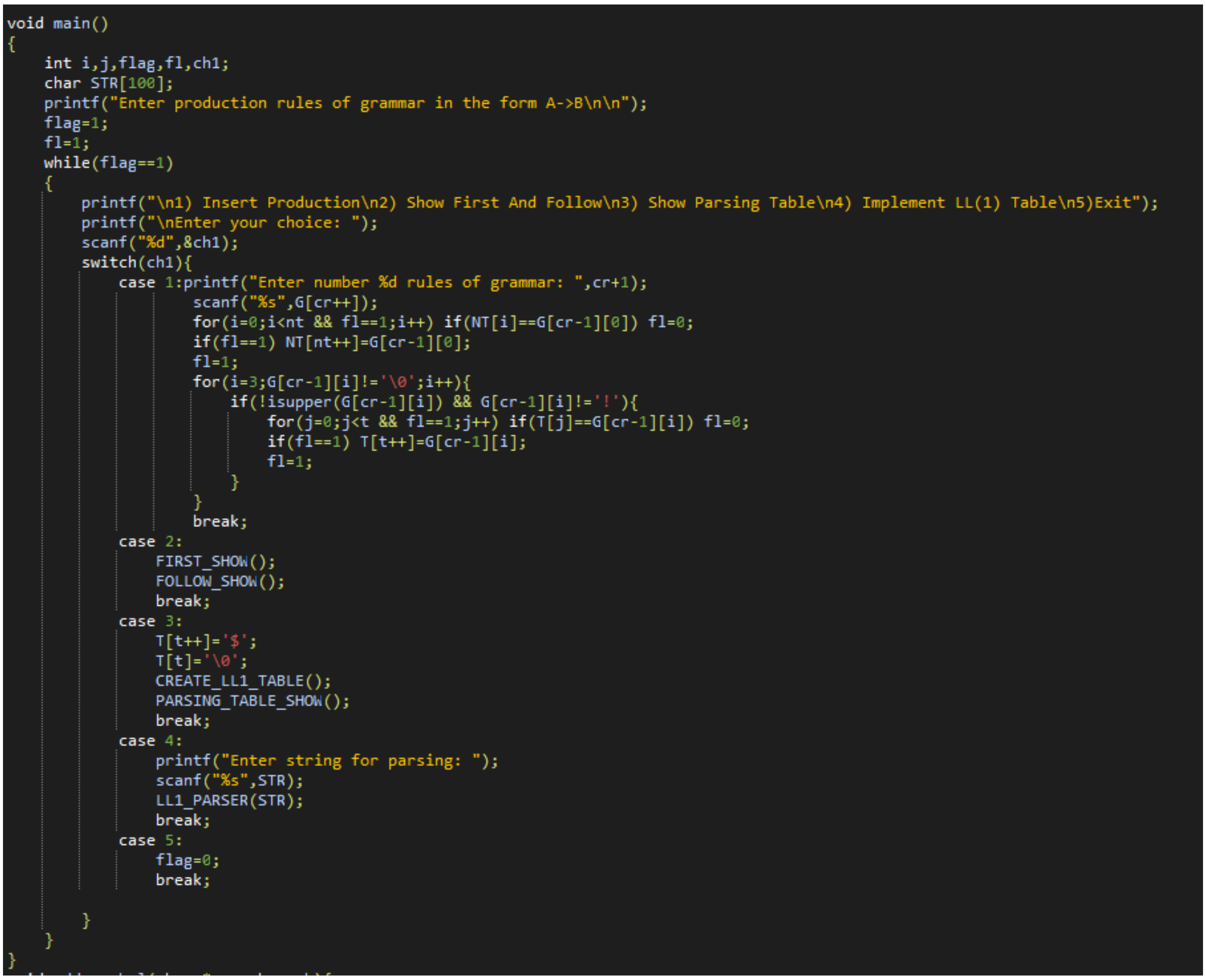
Result:

Using lex tool, syntax analyzer has been made and implemented.

PROGRAM 2.

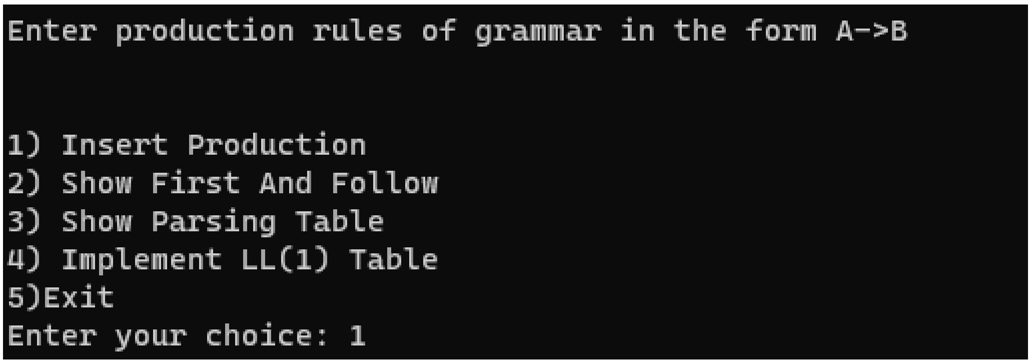
Aim: Implement a c program for LL(1) top down parser for any given LL(1) grammar. Find first(), follow(), predictive parsing table, stack implementation and parse tree. you can take any CFG grammar

Driver Code:

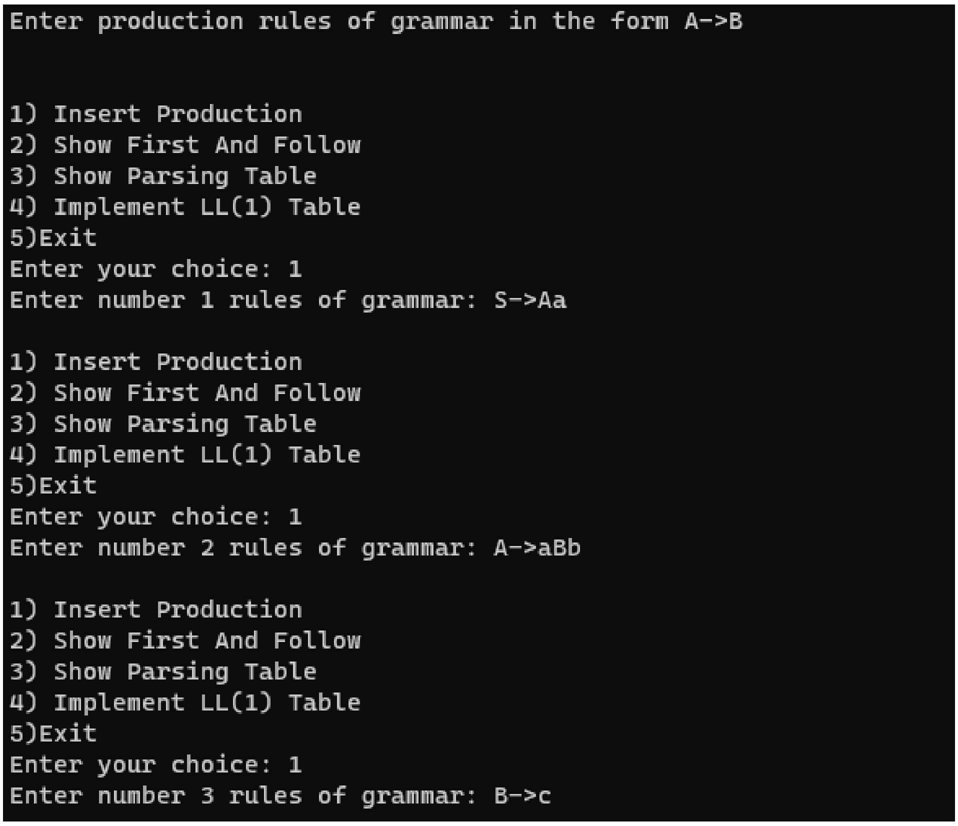


```c
void main()
{
    int i,j,flag,fl,ch1;
    char STR[100];
    printf("Enter production rules of grammar in the form A->B\n\n");
    flag=1;
    fl=1;
    while(flag==1)
    {
        printf("\n1) Insert Production\n2) Show First And Follow\n3) Show Parsing Table\n4) Implement LL(1) Table\n5)Exit");
        printf("\nEnter your choice: ");
        scanf("%d",&ch1);
        switch(ch1){
            case 1:printf("Enter number %d rules of grammar: ",cr+1);
                    scanf("%s",G[cr++]);
                    for(i=0;i<nt && fl==1;i++) if(NT[i]==G[cr-1][0]) fl=0;
                    if(fl==1) NT[nt++]=G[cr-1][0];
                    fl=1;
                    for(i=3;G[cr-1][i]!='\0';i++){
                        if(!isupper(G[cr-1][i]) && G[cr-1][i]!='!'){
                            for(j=0;j<t && fl==1;j++) if(T[j]==G[cr-1][i]) fl=0;
                            if(fl==1) T[t++]=G[cr-1][i];
                            fl=1;
                        }
                    }
                    break;
            case 2:
                FIRST_SHOW();
                FOLLOW_SHOW();
                break;
            case 3:
                T[t++]='$';
                T[t]='\0';
                CREATE_LL1_TABLE();
                PARSING_TABLE_SHOW();
                break;
            case 4:
                printf("Enter string for parsing: ");
                scanf("%s",STR);
                LL1_PARSER(STR);
                break;
            case 5:
                flag=0;
                break;

        }
    }
}
```

Output Screenshot:

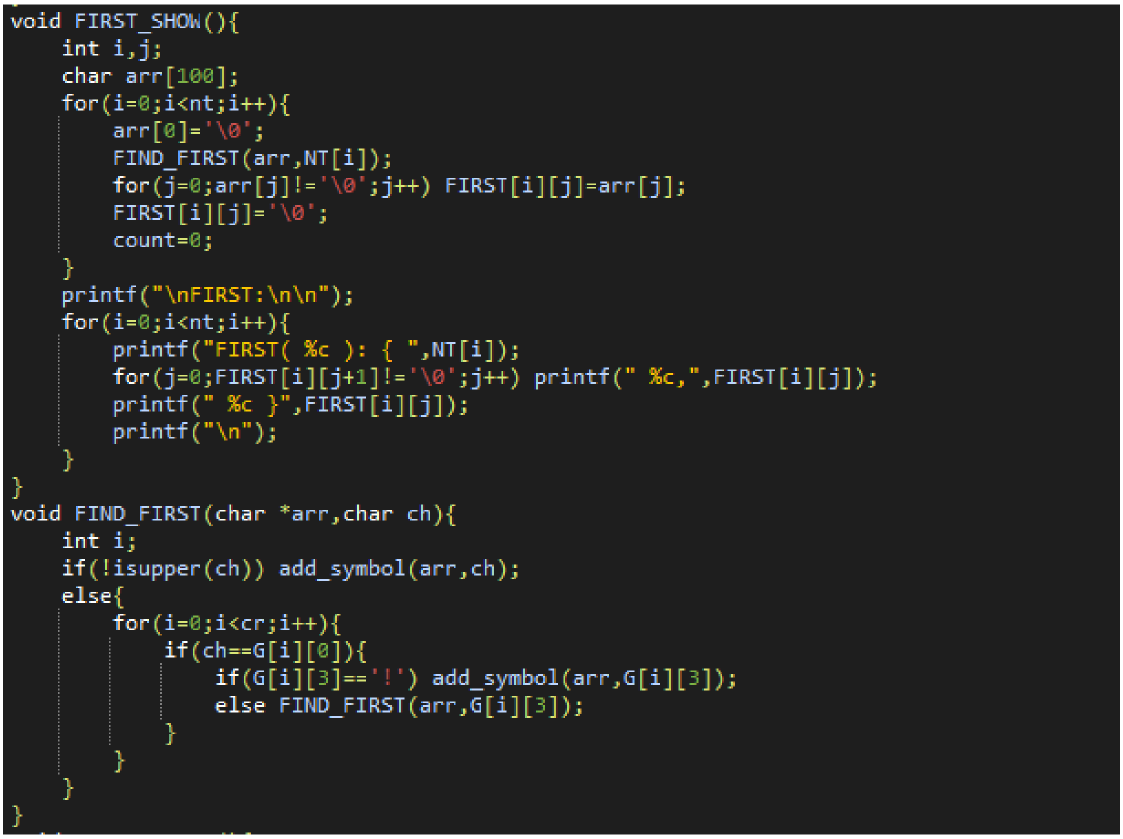


```
Enter production rules of grammar in the form A->B


1) Insert Production
2) Show First And Follow
3) Show Parsing Table
4) Implement LL(1) Table
5)Exit
Enter your choice: 1
```

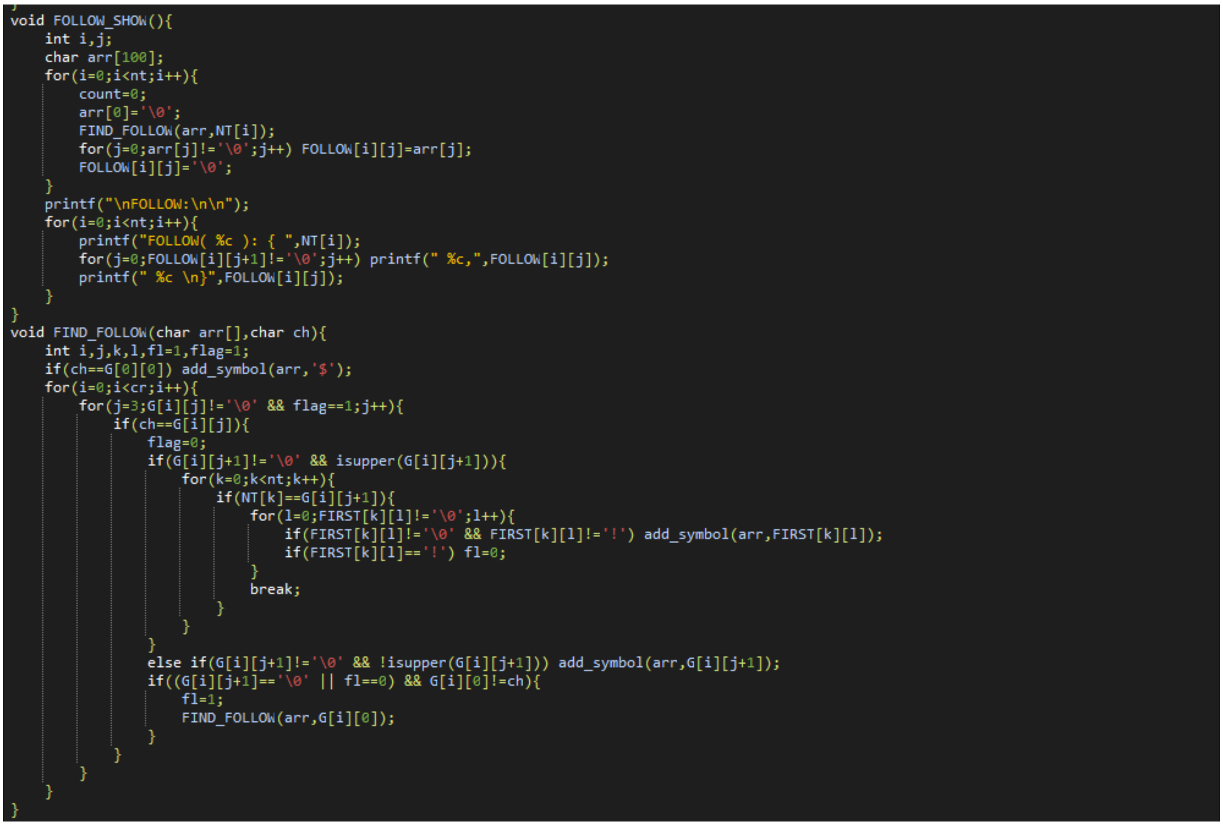


```
Enter production rules of grammar in the form A->B


1) Insert Production
2) Show First And Follow
3) Show Parsing Table
4) Implement LL(1) Table
5)Exit
Enter your choice: 1
Enter number 1 rules of grammar: S->Aa

1) Insert Production
2) Show First And Follow
3) Show Parsing Table
4) Implement LL(1) Table
5)Exit
Enter your choice: 1
Enter number 2 rules of grammar: A->aBb

1) Insert Production
2) Show First And Follow
3) Show Parsing Table
4) Implement LL(1) Table
5)Exit
Enter your choice: 1
Enter number 3 rules of grammar: B->c
```

FIRST() ANF FOLLOW() CODE:

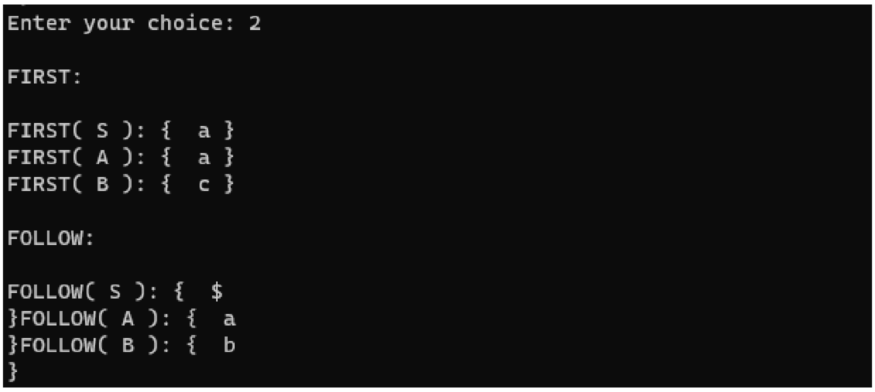


```
void FIRST_SHOW(){
    int i,j;
    char arr[100];
    for(i=0;i<nt;i++){
        arr[0]='\0';
        FIND_FIRST(arr,NT[i]);
        for(j=0;arr[j]!='\0';j++) FIRST[i][j]=arr[j];
        FIRST[i][j]='\0';
        count=0;
    }
    printf("\nFIRST:\n\n");
    for(i=0;i<nt;i++){
        printf("FIRST( %c ): { ",NT[i]);
        for(j=0;FIRST[i][j+1]!='\0';j++) printf(" %c,",FIRST[i][j]);
        printf(" %c }",FIRST[i][j]);
        printf("\n");
    }
}
void FIND_FIRST(char *arr,char ch){
    int i;
    if(!isupper(ch)) add_symbol(arr,ch);
    else{
        for(i=0;i<cr;i++){
            if(ch==G[i][0]){
                if(G[i][3]=='!') add_symbol(arr,G[i][3]);
                else FIND_FIRST(arr,G[i][3]);
            }
        }
    }
}
```



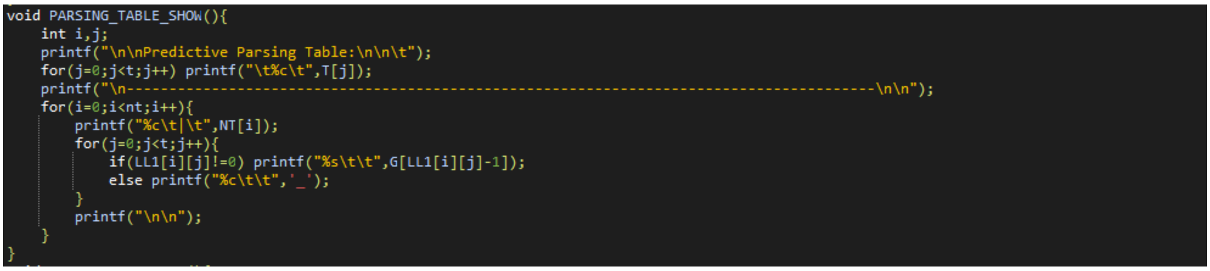
```c
void FOLLOW_SHOW(){
    int i,j;
    char arr[100];
    for(i=0;i<nt;i++){
        count=0;
        arr[0]='\0';
        FIND_FOLLOW(arr,NT[i]);
        for(j=0;arr[j]!='\0';j++) FOLLOW[i][j]=arr[j];
        FOLLOW[i][j]='\0';
    }
    printf("\nFOLLOW:\n\n");
    for(i=0;i<nt;i++){
        printf("FOLLOW( %c ): { ",NT[i]);
        for(j=0;FOLLOW[i][j+1]!='\0';j++) printf(" %c,",FOLLOW[i][j]);
        printf(" %c \n}",FOLLOW[i][j]);
    }
}
void FIND_FOLLOW(char arr[],char ch){
    int i,j,k,l,fl=1,flag=1;
    if(ch==G[0][0]) add_symbol(arr,'$');
    for(i=0;i<cr;i++){
        for(j=3;G[i][j]!='\0' && flag==1;j++){
            if(ch==G[i][j]){
                flag=0;
                if(G[i][j+1]!='\0' && isupper(G[i][j+1])){
                    for(k=0;k<nt;k++){
                        if(NT[k]==G[i][j+1]){
                            for(l=0;FIRST[k][l]!='\0';l++){
                                if(FIRST[k][l]!='\0' && FIRST[k][l]!='!') add_symbol(arr,FIRST[k][l]);
                                if(FIRST[k][l]=='!') fl=0;
                            }
                            break;
                        }
                    }
                }
                else if(G[i][j+1]!='\0' && !isupper(G[i][j+1])) add_symbol(arr,G[i][j+1]);
                if((G[i][j+1]=='\0' || fl==0) && G[i][0]!=ch){
                    fl=1;
                    FIND_FOLLOW(arr,G[i][0]);
                }
            }
        }
    }
}
```

OUTPUT:

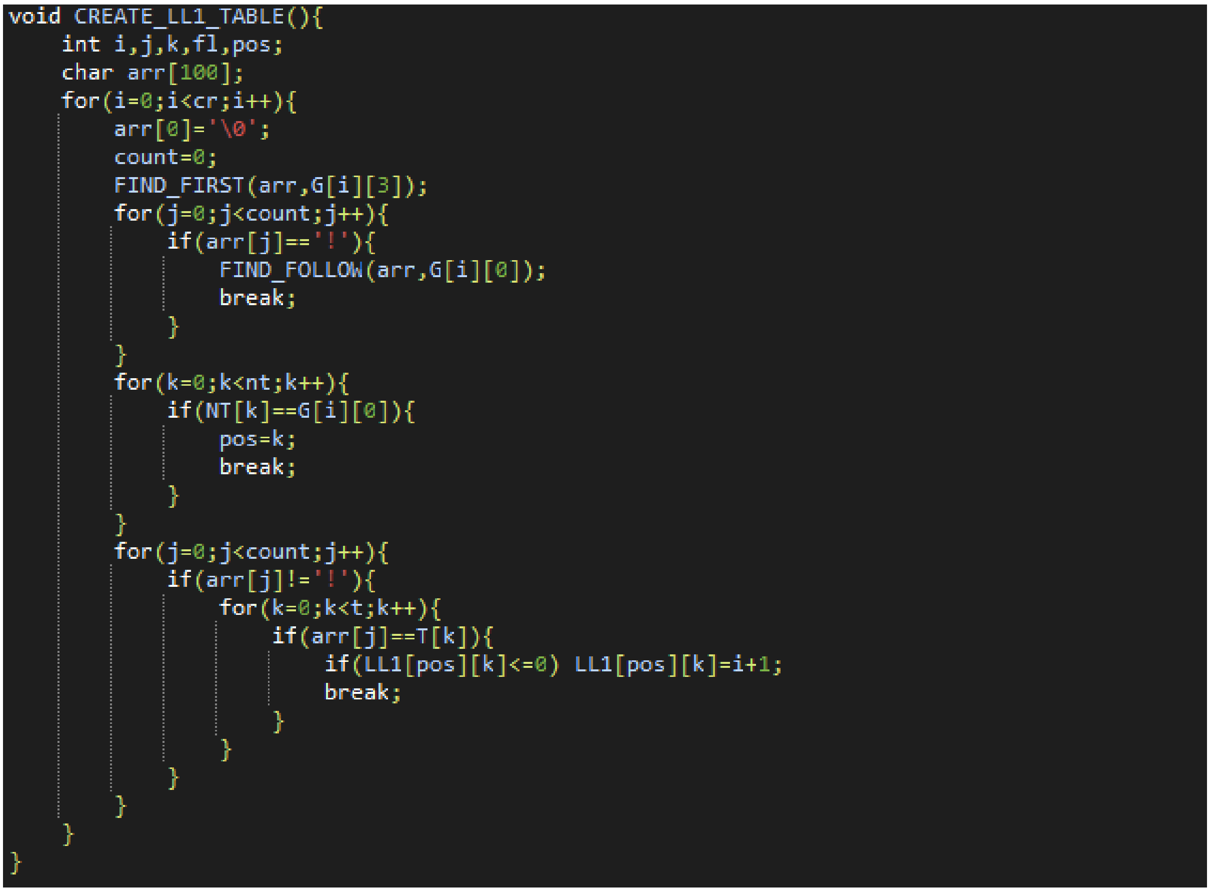


```
Enter your choice: 2

FIRST:

FIRST( S ): {  a }
FIRST( A ): {  a }
FIRST( B ): {  c }

FOLLOW:

FOLLOW( S ): {  $
}FOLLOW( A ): {  a
}FOLLOW( B ): {  b
}
```

PARSING TABLE CODE:

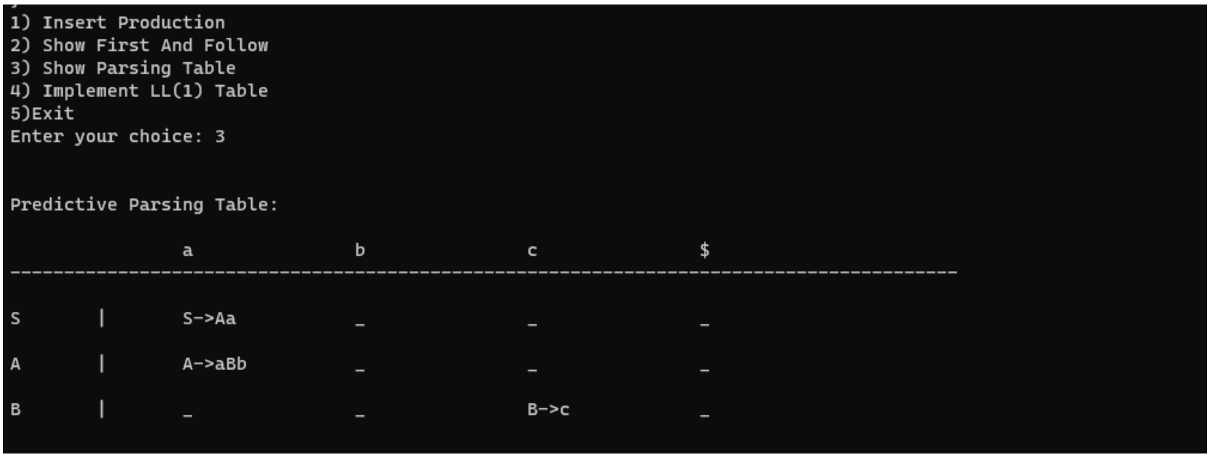


```c
void PARSING_TABLE_SHOW(){
    int i,j;
    printf("\n\nPredictive Parsing Table:\n\n\t");
    for(j=0;j<t;j++) printf("\t%c\t",T[j]);
    printf("\n----------------------------------------------------------------------------------------\n\n");
    for(i=0;i<nt;i++){
        printf("%c\t|\t",NT[i]);
        for(j=0;j<t;j++){
            if(LL1[i][j]!=0) printf("%s\t\t",G[LL1[i][j]-1]);
            else printf("%c\t\t",'_');
        }
        printf("\n\n");
    }
}
```



```
void CREATE_LL1_TABLE(){
    int i,j,k,fl,pos;
    char arr[100];
    for(i=0;i<cr;i++){
        arr[0]='\0';
        count=0;
        FIND_FIRST(arr,G[i][3]);
        for(j=0;j<count;j++){
            if(arr[j]=='!'){
                FIND_FOLLOW(arr,G[i][0]);
                break;
            }
        }
        for(k=0;k<nt;k++){
            if(NT[k]==G[i][0]){
                pos=k;
                break;
            }
        }
        for(j=0;j<count;j++){
            if(arr[j]!='!'){
                for(k=0;k<t;k++){
                    if(arr[j]==T[k]){
                        if(LL1[pos][k]<=0) LL1[pos][k]=i+1;
                        break;
                    }
                }
            }
        }
    }
}
```

OUTPUT:



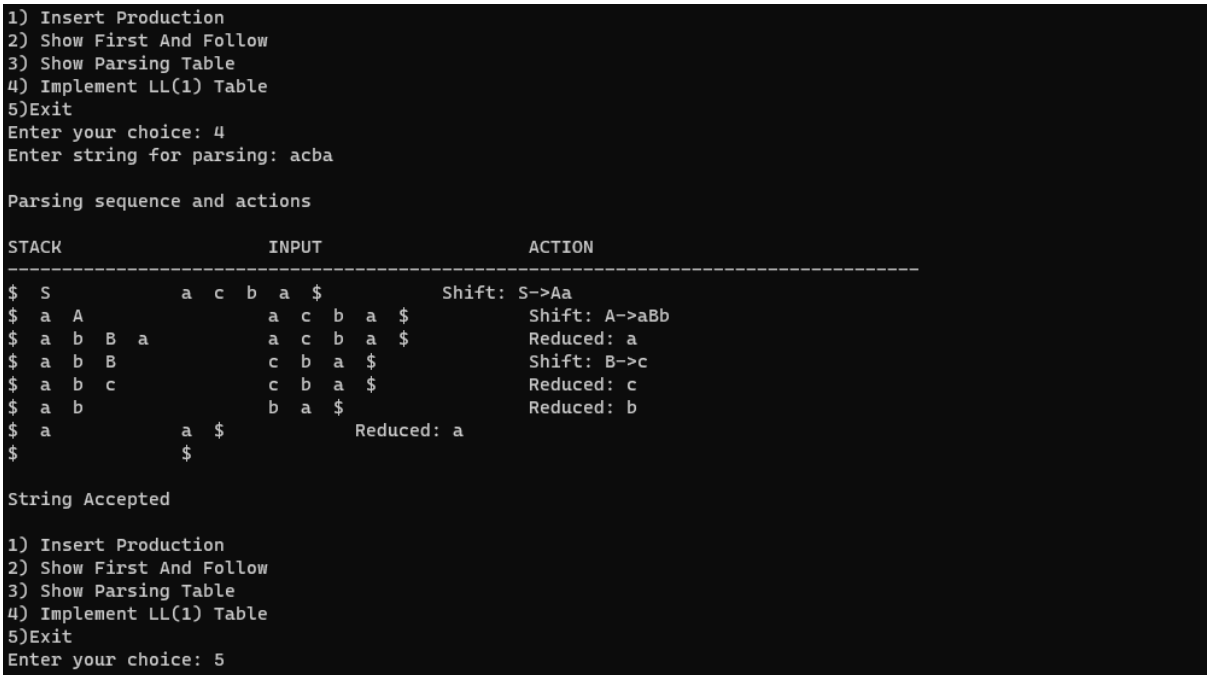
```
1) Insert Production
2) Show First And Follow
3) Show Parsing Table
4) Implement LL(1) Table
5)Exit
Enter your choice: 3


Predictive Parsing Table:

                a               b               c               $
----------------------------------------------------------------------------------------

S       |       S->Aa           _               _               _

A       |       A->aBb          _               _               _

B       |       _               _               B->c            _
```

STACK IMPLEMENTATION AND PARSING:

```
void LL1_PARSER(char *STR)
{
   int i=0,j,pos,pos1,n,k;
   STR[strlen(STR)]='$';
   STACK[top++]='$';
   STACK[top]=G[0][0];
   printf("\nParsing sequence and actions\n\n");
   printf("STACK\t\t\tINPUT\t\t\tACTION");
   printf("\n-------------------------------------------------------------------------------------\n");
   i=0;
   while(STACK[top]!='$'){
      for(j=0;STACK[j]!='\0';j++) printf("%c  ",STACK[j]);
      printf("\t\t");
      for(j=i;STR[j]!='\0';j++) printf("%c  ",STR[j]);
      if(STR[i]==STACK[top]){
         printf("\t\tReduced: %c",STACK[top]);
         STACK[top]='\0';
         top=top-1;
         i=i+1;
      }
      else{
         for(j=0;j<nt;j++){
            if(STACK[top]==NT[j]){
               pos=j;
               break;
            }
         }
         for(j=0;j<t;j++){
            if(STR[i]==T[j]){
               pos1=j;
               break;
            }
         }
         n=LL1[pos][pos1];
         if(G[n-1][3]=='!'){
            STACK[top]='\0';
            top--;
         }
         else{
            for(j=3;G[n-1][j]!='\0';j++) k=j;
            STACK[top]='\0';
            for(j=k;j>2;j--) STACK[top++]=G[n-1][j];
            top--;
         }
         printf("\t\tShift: %s",G[n-1]);
      }
      printf("\n");
   }
   for(j=0;STACK[j]!='\0';j++) printf("%c  ",STACK[j]);
   printf("\t\t");
   for(j=i;STR[j]!='\0';j++) printf("%c  ",STR[j]);
   printf("\n");
   if(STACK[top]=='$' && STR[i]=='$') printf("\nString Accepted\n");
}
```

Output:

For Input String : acba



```
1) Insert Production
2) Show First And Follow
3) Show Parsing Table
4) Implement LL(1) Table
5)Exit
Enter your choice: 4
Enter string for parsing: acba

Parsing sequence and actions

STACK                   INPUT                   ACTION
------------------------------------------------------------------------------------
$  S            a  c  b  a  $           Shift: S->Aa
$  a  A                 a  c  b  a  $           Shift: A->aBb
$  a  b  B  a           a  c  b  a  $           Reduced: a
$  a  b  B              c  b  a  $              Shift: B->c
$  a  b  c              c  b  a  $              Reduced: c
$  a  b                 b  a  $                 Reduced: b
$  a            a  $            Reduced: a
$               $

String Accepted

1) Insert Production
2) Show First And Follow
3) Show Parsing Table
4) Implement LL(1) Table
5)Exit
Enter your choice: 5
```

Result:

LL(1) parser has been implemented which prints out the first(), follow() , predicative parsing table and shows the stack implementation for a given string.

GRAMMAR USED:

S->Aa

A->aBb

B->c

END