```
/*AIM:
         To design predictive parser for given parsing table and input string.

DESCRIPTION:
          It has an input tape to hold given input string which is terminated with
dollar,a stack which is first initialised with $ and on top of it,starting root
node is pushed.The parsing table entries of following grammer are used.
                                                        E->E+T
                                                        E->T
                                                        T->T*F
                                                        T->F
                                                        F->(E)/id
After removing left recursion,
                                                        E->TE'
                                                        E'->+TE'/e
                                                       T->FT'
                        T'->*FT'/e
                        F->(E)/id
*/

#include <stdio.h>
#include <stdlib.h>
#include<ctype.h>
#include<string.h>
char stack[50];
void push(char);
void pop();
char topf();
int top=-1;
int main()
{
    int i,j,k,l,s,m,n,ip=0;
    char w[20],X,ch,prod[10];
    char term[30];
    char nterm[50];
    char pt[10][20][30];
    top++;
    stack[0]='$';
    top++;
    stack[1]='E';
    printf("enter the no. of non terminals:");
    scanf("%d",&n);
    printf("enter the non terminals:");
    for(i=0;i<n;i++)
      scanf(" %c",&nterm[i]);
    printf("enter the no. of terminals:");
    scanf("%d",&m);
    printf("enter the terminals:");
    for(i=0;i<m;i++)
       scanf(" %c",&term[i]);
    term[i]='$';
    m++;
    printf("enter the parsing table values:\n");
    for(i=0;i<n;i++)
   {
      for(j=0;j<m;j++)
     {
      scanf("%s",pt[i][j]);
      }
```

```
    }
printf("the parsing table contents are\n");
    for(i=0;i<n;i++){
     for(j=0;j<m;j++)
       printf("%s\t",pt[i][j]);
       printf("\n");
     }
    printf("enter the input string:");
    scanf("%s",w);


//-------------------------------------------------
    X=stack[top];
    ch=w[ip];
    while(((X=stack[top])!='$'))    x=E  ch=id
    {
      ch=w[ip];
      printf("X=%c,ch=%c\n",X,ch);
      if((!isupper(X))&&(!isupper(ch)))
       { if(X==ch)
       {
        ip++;
        pop();
       }
       else
       {
        printf("invalid string\n");
        break;
       }
       }
       else if(isupper(X))
       {
       for(i=0;i<n;i++) x=T ch=id
       {
         if(X==nterm[i])
          k=i;
       }
       for(i=0;i<m;i++)
       {
         if(ch==term[i])
          l=i;
       }
       strcpy(prod,pt[k][l]);
       if(!strcmp(prod,"#"))
          {
            printf("invalid\n");
            break;
          }
       else if(!strcmp(prod,"@"))
           pop();
       else
       {

           pop();
           for(i=(strlen(prod)-1);i>=0;i--)  1-1=0 i>=0 i--
            push(prod[i]);
       }
   }
 }
```

```
 if((X=='$')&&(w[ip]=='$'))
 printf("string valid");
 else
    printf("invalid string");
 return 1;
}
void pop()
{
 top--;
}
void push(char x)
{
 top++;
 stack[top]=x;
}
```