CC:

1. WAP to check whether a string is accepted or not for an entered grammar.

```
#include<stdio.h>
#include<string.h>
int gen(char *str){
  int len=strlen(str);  int flag=1;  int i=0;
  while(1){
    if(((i+2)<=(len-1)) &&(str[i]=='a' && str[i+1]=='b' && str[i+2]=='b')){
      i=i+3;      continue;    }
    else if(((i+1)<=(len-1)) && (str[i]=='a' && str[i+1]=='b')){
      i=i+2;      continue;    }
    else if((i<=(len-1)) && (str[i]=='a')){
      i=i+1;      continue;    }
    else if((i==len) && str[i]=='\0')){
      break;    }
    else{  flag=0; break;    }
  }
  if(flag==0){return 0;}
  else    printf("\n accepted \n");
}
int main(){
  printf("The grammar is: S->aS, S->Sb, S->ab\n");
  printf("Enter the string to be checked: \n");
  printf("ab");  printf("%d ", gen("ab"));  return 0;}
```

Output:

```
The grammar is: S->aS, S->Sb, S->ab Enter the string to be checked:

abab

accepted

12
```

2. Write a program to count the number of tokens in an expression.

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>
int main(){
    char str[50];    int len;
    int i, a = 0, b = 0, d = 0, f = 0, var = 0, tokens = 0, constant = 0, oper = 0;
    printf("Enter string: ");
    scanf("%s", &str);
    len = strlen(str);
    for (i = 0; i < len; i++){
         if (isalpha(str[i])){ a++;  }
         if(isdigit(str[i])){
            while(isdigit(str[i])){  i++;}
            d++;    }
         if(str[i]=='%'||str[i]=='*'||str[i]=='/'||str[i]=='+'||str[i]=='-'||str[i]=='=')  f++;
   else    b++;
    }
    var=a;    constant=d;    oper=f;
    tokens=var+constant+oper;
    printf("\ntotalvar:%d ", var);
    printf("\ntotal constants: %d ", constant);
    printf("\ntotal operators: %d",oper);
    printf("\ntotal tokens: %d ", tokens);
    return 0;    getch();
}
```

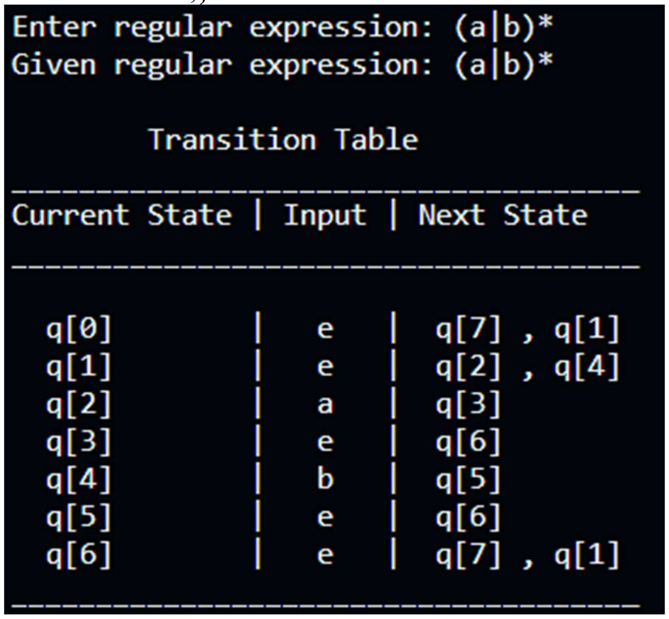
Output:

```
Enter string: x=a/b+c*d+5

totalvar:5

total constants: 1

total operators: 5

total tokens: 11
```

3. Write a program to convert regular expression to NFA.

```
#include<stdio.h> #include<string.h>
int main(){
        char reg[20]; int q[20][3],i=0,j=1,len,a,b;
        for(a=0;a<20;a++) for(b=0;b<3;b++) q[a][b]=0;
        scanf("%s",reg);  printf("Given regular expression: %s\n",reg);
        len=strlen(reg);
        while(i<len){
                if(reg[i]=='a'&&reg[i+1]!='|'&&reg[i+1]!='*') { q[j][0]=j+1; j++; }
                if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='*') {       q[j][1]=j+1; j++;        }
                if(reg[i]=='e'&&reg[i+1]!='|'&&reg[i+1]!='*') {       q[j][2]=j+1; j++;        }
                if(reg[i]=='a'&&reg[i+1]=='|'&&reg[i+2]=='b'){
                  q[j][2]=((j+1)*10)+(j+3); j++;
                  q[j][0]=j+1; j++;
                        q[j][2]=j+3; j++;
                        q[j][1]=j+1; j++;
                        q[j][2]=j+1; j++;
                        i=i+2;             }
                if(reg[i]=='b'&&reg[i+1]=='|'&&reg[i+2]=='a'){
                        q[j][2]=((j+1)*10)+(j+3); j++;
                        q[j][1]=j+1; j++;
```

```
                    q[j][2]=j+3; j++;
                    q[j][0]=j+1; j++;
                    q[j][2]=j+1; j++;
                    i=i+2;          }
            if(reg[i]=='a'&&reg[i+1]=='*'){
                    q[j][2]=((j+1)*10)+(j+3); j++;
                    q[j][0]=j+1; j++;
                    q[j][2]=((j+1)*10)+(j-1); j++;          }
            if(reg[i]=='b'&&reg[i+1]=='*'){
                    q[j][2]=((j+1)*10)+(j+3); j++;
                    q[j][1]=j+1; j++;
                    q[j][2]=((j+1)*10)+(j-1); j++;          }
            if(reg[i]==')'&&reg[i+1]=='*'){
                    q[0][2]=((j+1)*10)+1;
                    q[j][2]=((j+1)*10)+1;
                    j++;            }
            i++;
        }
        printf("\n\tTransition Table \n");
        printf("_________________________________________\n");
        printf("Current State |\tInput |\tNext State");
        printf("\n_________________________________________\n");
        for(i=0;i<=j;i++){
            if(q[i][0]!=0) printf("\n  q[%d]\t      |  a   | q[%d]",i,q[i][0]);
            if(q[i][1]!=0) printf("\n  q[%d]\t      |  b   | q[%d]",i,q[i][1]);
            if(q[i][2]!=0){
                    if(q[i][2]<10) printf("\n  q[%d]\t      |  e   | q[%d]",i,q[i][2]);
                    else printf("\n  q[%d]\t      |  e   | q[%d] ,
q[%d]",i,q[i][2]/10,q[i][2]%10);
            }
        }
        return 0;}
```



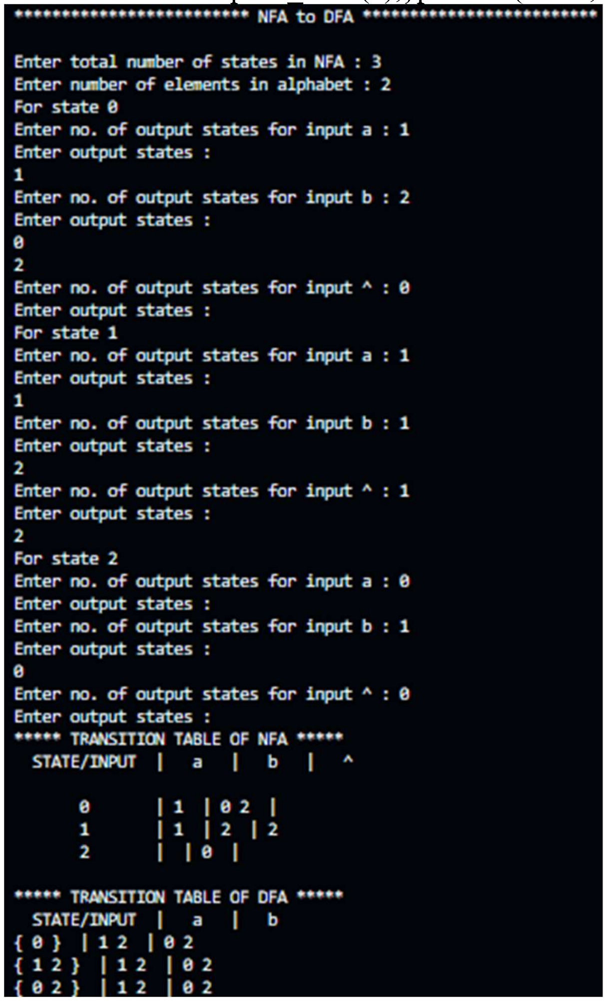
```
Enter regular expression: (a|b)*
Given regular expression: (a|b)*

        Transition Table

------------------------------------
Current State | Input | Next State
------------------------------------

  q[0]        |   e   |  q[7] , q[1]
  q[1]        |   e   |  q[2] , q[4]
  q[2]        |   a   |  q[3]
  q[3]        |   e   |  q[6]
  q[4]        |   b   |  q[5]
  q[5]        |   e   |  q[6]
  q[6]        |   e   |  q[7] , q[1]

------------------------------------
```

## 4. Write a program to convert NFA to DFA.

```
#include<iostream> #include<bits/stdc++.h>
using namespace std;
void print(vector<vector<vector<int> > > table){
        cout<<"  STATE/INPUT  |";  char a='a';
        for(int i=0;i<table[0].size()-1;i++){
                cout<<"   "<<a++<<"   |";       }
        cout<<"   ^   "<<endl<<endl;
        for(int i=0;i<table.size();i++){
                cout<<"       "<<i<<"      ";
                for(int j=0;j<table[i].size();j++){
                        cout<<" | ";
                        for(int k=0;k<table[i][j].size();k++){
                                cout<<table[i][j][k]<<" ";}
                cout<<endl;    }
}
void printdfa(vector<vector<int> > states, vector<vector<vector<int> > > dfa){
        cout<<"  STATE/INPUT  ";   char a='a';
        for(int i=0;i<dfa[0].size();i++){
                cout<<"|   "<<a++<<"   ";}
        cout<<endl;
        for(int i=0;i<states.size();i++){
                cout<<"{ ";
                for(int h=0;h<states[i].size();h++)
                        cout<<states[i][h]<<" ";
                if(states[i].empty()){
                        cout<<"^ ";}
                cout<<"} ";
                for(int j=0;j<dfa[i].size();j++){
                        cout<<" | ";
                        for(int k=0;k<dfa[i][j].size();k++){
                                cout<<dfa[i][j][k]<<" ";}
                        if(dfa[i][j].empty()){cout<<"^ ";}
                }cout<<endl;   }
}
vector<int> closure(int s,vector<vector<vector<int> > > v){
        vector<int> t;  queue<int> q;  t.push_back(s);
        int a=v[s][v[s].size()-1].size();
        for(int i=0;i<a;i++){
                t.push_back(v[s][v[s].size()-1][i]); q.push(t[i]);}
        while(!q.empty()){
                int f=q.front(); q.pop();
                if(!v[f][v[f].size()-1].empty()){
                        int u=v[f][v[f].size()-1].size();
                        for(int i=0;i<u;i++){
                                int y=v[f][v[f].size()-1][i];
                                if(find(t.begin(),t.end(),y)==t.end()){
                                        t.push_back(y); q.push(y);}
                        }       }       }       return t;
```

```
}
int main(){
        int n,alpha;
        cout<<"************ NFA to DFA ************"<<endl;
        cout<<"Enter total number of states in NFA : ";        cin>>n;
        cout<<"Enter number of elements in alphabet : ";     cin>>alpha;
        vector<vector<vector<int> > > table;
        for(int i=0;i<n;i++){
                cout<<"For state "<<i<<endl;
                vector< vector< int > > v; char a='a';  int y,yn;
                for(int j=0;j<alpha;j++){
                        vector<int> t;
                        cout<<"Enter no. of output states for input "<<a++<<" : ";
                        cin>>yn;
                        cout<<"Enter output states :"<<endl;
                        for(int k=0;k<yn;k++){
                                cin>>y; t.push_back(y);}
                        v.push_back(t);
                }
                vector<int> t;
                cout<<"Enter no. of output states for input ^ : "; cin>>yn;
                cout<<"Enter output states :"<<endl;
                for(int k=0;k<yn;k++){
                        cin>>y;           t.push_back(y);}
                v.push_back(t);  table.push_back(v);
        }
        cout<<"***** TRANSITION TABLE OF NFA *****"<<endl;
        print(table);
        cout<<endl<<"***** TRANSITION TABLE OF DFA *****"<<endl;
        vector<vector<vector<int> > > dfa;   vector<vector<int> > states;
        states.push_back(closure(0,table));
        queue<vector<int> > q;          q.push(states[0]);
        while(!q.empty()){
                vector<int> f=q.front(); q.pop();
                vector<vector<int> > v;
                for(int i=0;i<alpha;i++){
                        vector<int> t;             set<int> s;
                        for(int j=0;j<f.size();j++){
                                for(int k=0;k<table[f[j]][i].size();k++){
                                        vector<int> cl= closure(table[f[j]][i][k],table);
                                        for(int h=0;h<cl.size();h++){
                                                if(s.find(cl[h])==s.end())
                                                s.insert(cl[h]);  }
                                }
                        }
                        for(set<int >::iterator u=s.begin(); u!=s.end();u++)
                                t.push_back(*u); v.push_back(t);
                        if(find(states.begin(),states.end(),t)==states.end()){
                                states.push_back(t);    q.push(t);}
                }
```

```
                    dfa.push_back(v);}printdfa(states,dfa); }
```



```
************************* NFA to DFA *************************

Enter total number of states in NFA : 3
Enter number of elements in alphabet : 2
For state 0
Enter no. of output states for input a : 1
Enter output states :
1
Enter no. of output states for input b : 2
Enter output states :
0
2
Enter no. of output states for input ^ : 0
Enter output states :
For state 1
Enter no. of output states for input a : 1
Enter output states :
1
Enter no. of output states for input b : 1
Enter output states :
2
Enter no. of output states for input ^ : 1
Enter output states :
2
For state 2
Enter no. of output states for input a : 0
Enter output states :
Enter no. of output states for input b : 1
Enter output states :
0
Enter no. of output states for input ^ : 0
Enter output states :
***** TRANSITION TABLE OF NFA *****
  STATE/INPUT  |   a   |   b   |   ^

      0        | 1  | 0 2  |
      1        | 1  | 2  | 2
      2        |  | 0  |

***** TRANSITION TABLE OF DFA *****
  STATE/INPUT  |   a   |   b
{ 0 }  | 1 2  | 0 2
{ 1 2 }  | 1 2  | 0 2
{ 0 2 }  | 1 2  | 0 2
```

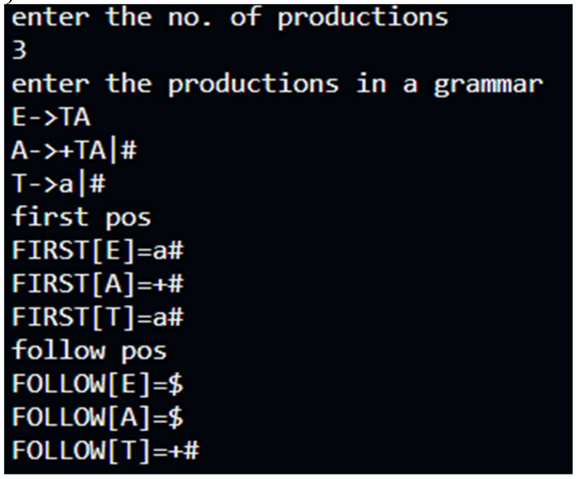
## 5.	Write a program to calculate first and follow

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main(){
    char fin[10][20], st[10][20], ft[20][20], fol[20][20];
    int a = 0, e, i, t, b, c, n, k, l = 0, j, s, m, p;
    printf("enter the no. of productions\n");
    scanf("%d", &n);
    printf("enter the productions in a grammar\n");
    for (i = 0; i < n; i++)
        scanf("%s", st[i]);
    for (i = 0; i < n; i++)
        fol[i][0] = '\0';
    for (s = 0; s < n; s++)     {
        for (i = 0; i < n; i++)         {
            j = 3;          l = 0;          a = 0;
        l1:
```

```
        if (!((st[i][j] > 64) && (st[i][j] < 91))){
            for (m = 0; m < l; m++){
                if (ft[i][m] == st[i][j])
                    goto s1; }
            ft[i][l] = st[i][j];
            l = l + 1;
        s1:
            j = j + 1;
        }
        else{
            if (s > 0){
                while (st[i][j] != st[a][0]){
                    a++;}
                b = 0;
                while (ft[a][b] != '\0'){
                    for (m = 0; m < l; m++){
                        if (ft[i][m] == ft[a][b])
                            goto s2;
                    }
                    ft[i][l] = ft[a][b];
                    l = l + 1;
                s2:
                    b = b + 1;
                }
            }
        }
        while (st[i][j] != '\0'){
            if (st[i][j] == '|'){
                j = j + 1;
                goto l1;      }
            j = j + 1;           }
        ft[i][l] = '\0';
    }
}
printf("first pos\n");
for (i = 0; i < n; i++)
    printf("FIRST[%c]=%s\n", st[i][0], ft[i]);
fol[0][0] = '$';
for (i = 0; i < n; i++){
    k = 0;    j = 3;
    if (i == 0)      l = 1;
    else           l = 0;
k1:
    while ((st[i][0] != st[k][j]) && (k < n)){
        if (st[k][j] == '\0'){
            k++;      j = 2;  }
        j++;       }
    j = j + 1;
    if (st[i][0] == st[k][j - 1]){
        if ((st[k][j] != '|') && (st[k][j] != '\0')){
```

```
        a = 0;
        if (!((st[k][j] > 64) && (st[k][j] < 91))){
            for (m = 0; m < l; m++){
                if (fol[i][m] == st[k][j])
                    goto q3;
            }
            fol[i][l] = st[k][j];
            l++;
        q3:
            p++;
        }
        else{
            while (st[k][j] != st[a][0]){
                a++;
            }
            p = 0;
            while (ft[a][p] != '\0'){
                if (ft[a][p] != 'e'){
                    for (m = 0; m < l; m++){
                        if (fol[i][m] == ft[a][p])
                            goto q2;
                    }
                    fol[i][l] = ft[a][p];
                    l = l + 1;
                }
                else
                    e = 1;
            q2:
                p++;
            }
            if (e == 1){
                e = 0;       goto a1;
            }
        }
    }
    else{
    a1:
        c = 0;    a = 0;
        while (st[k][0] != st[a][0]){
            a++;   }
        while ((fol[a][c] != '\0') && (st[a][0] != st[i][0])){
            for (m = 0; m < l; m++){
                if (fol[i][m] == fol[a][c])
                    goto q1;                    }
            fol[i][l] = fol[a][c];
            l++;
        q1:
            c++;
        }
    }
```

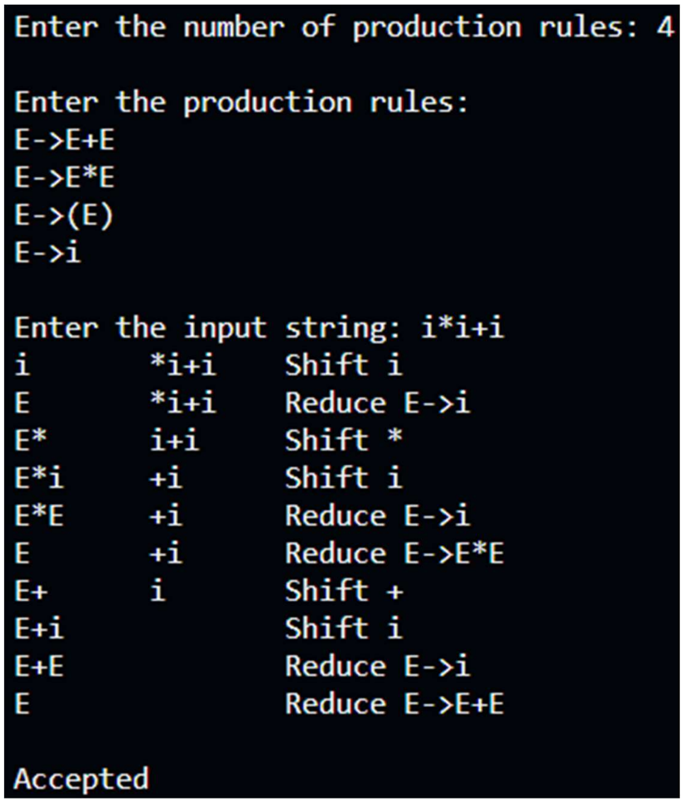
```
            goto k1;
        }
        fol[i][l] = '\0';
    }
    printf("follow pos\n");
    for (i = 0; i < n; i++)
        printf("FOLLOW[%c]=%s\n", st[i][0], fol[i]);
    printf("\n");
}
```



```
enter the no. of productions
3
enter the productions in a grammar
E->TA
A->+TA|#
T->a|#
first pos
FIRST[E]=a#
FIRST[A]=+#
FIRST[T]=a#
follow pos
FOLLOW[E]=$
FOLLOW[A]=$
FOLLOW[T]=+#
```

## 6. Write a program to design SR parser for E ->E + E | E * E | (E) | Id.

```
#include <stdio.h>
#include <string.h>
struct ProductionRule{
    char left[10];
    char right[10];};
int main(){
    char input[20], stack[50], temp[50], ch[2], *token1, *token2, *substring;
    int i, j, stack_length, substring_length, stack_top, rule_count = 0;
    struct ProductionRule rules[10];
    stack[0] = '\0';
    printf("\nEnter the number of production rules: ");
    scanf("%d", &rule_count);
    printf("\nEnter the production rules: \n");
    for (i = 0; i < rule_count; i++)    {
        scanf("%s", temp);
        token1 = strtok(temp, "->");
        token2 = strtok(NULL, "->");
        strcpy(rules[i].left, token1);
        strcpy(rules[i].right, token2);    }
    printf("\nEnter the input string: ");    scanf("%s", input);
    i = 0;
    while (1)    {
        if (i < strlen(input))        {
            ch[0] = input[i];
        ch[1] = '\0';
        i++;
        strcat(stack, ch);
        printf("%s\t", stack);
        for (int k = i; k < strlen(input); k++)            {
            printf("%c", input[k]);            }
        printf("\tShift %s\n", ch);
    }
    for (j = 0; j < rule_count; j++)        {
        substring = strstr(stack, rules[j].right);
        if (substring != NULL) {
            stack_length = strlen(stack);
            substring_length = strlen(substring);
            stack_top = stack_length - substring_length;
            stack[stack_top] = '\0';
            strcat(stack, rules[j].left);
            printf("%s\t", stack);
            for (int k = i; k < strlen(input); k++){
                printf("%c", input[k]);     }
            printf("\tReduce %s->%s\n", rules[j].left, rules[j].right);
            j = -1;
        }
    }
    if (strcmp(stack, rules[0].left) == 0 && i == strlen(input)) {
        printf("\nAccepted");   break; }
    if (i == strlen(input)) {
        printf("\nNot Accepted"); break; }
}
return 0;}
```

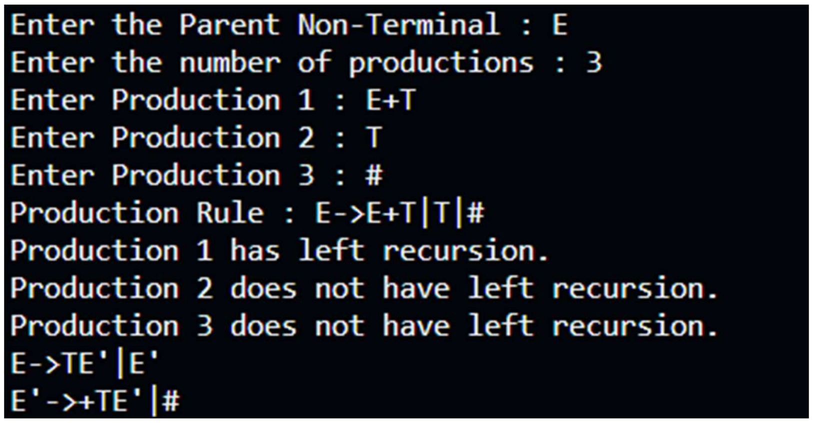


```
Enter the number of production rules: 4

Enter the production rules:
E->E+E
E->E*E
E->(E)
E->i

Enter the input string: i*i+i
i       *i+i    Shift i
E       *i+i    Reduce E->i
E*      i+i     Shift *
E*i     +i      Shift i
E*E     +i      Reduce E->i
E       +i      Reduce E->E*E
E+      i       Shift +
E+i             Shift i
E+E             Reduce E->i
E               Reduce E->E+E

Accepted
```

7. Write a program to remove the Left Recursion from a given grammar.

```
#include <iostream>
#include <string>
using namespace std;
int main(){
  string ip, op1, op2, temp;
  int sizes[10] = {};    char c;
  int n, j, l;
  cout << "Enter the Parent Non-Terminal : ";
  cin >> c;
  ip.push_back(c);
  op1 += ip + "\'->";
  ip += "->";
  op2 += ip;
  cout << "Enter the number of productions : ";
  cin >> n;
  for (int i = 0; i < n; i++){
    cout << "Enter Production " << i + 1 << " : ";
    cin >> temp;
    sizes[i] = temp.size();
    ip += temp;
    if (i != n - 1)
      ip += "|";
  }
  cout << "Production Rule : " << ip << endl;
```

```
    for (int i = 0, k = 3; i < n; i++){
        if (ip[0] == ip[k]){
            cout << "Production " << i + 1 << " has left recursion." << endl;
            if (ip[k] != '#'){
                for (l = k + 1; l < k + sizes[i]; l++)
                    op1.push_back(ip[l]);
                k = l + 1;
                op1.push_back(ip[0]);
                op1 += "\'|";
            }
        }
        else{
            cout << "Production " << i + 1 << " does not have left recursion." << endl;
            if (ip[k] != '#'){
                for (j = k; j < k + sizes[i]; j++)
                    op2.push_back(ip[j]);
                k = j + 1;
                op2.push_back(ip[0]);
                op2 += "\'|";
            }
            else{
                op2.push_back(ip[0]);
                op2 += "\'";
            }
        }
    }
    op1 += "#";
    cout << op2 << endl;
    cout << op1 << endl;
    return 0;}
```



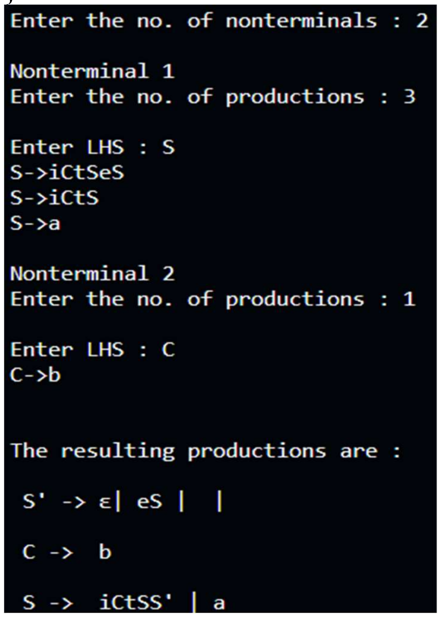
```
Enter the Parent Non-Terminal : E
Enter the number of productions : 3
Enter Production 1 : E+T
Enter Production 2 : T
Enter Production 3 : #
Production Rule : E->E+T|T|#
Production 1 has left recursion.
Production 2 does not have left recursion.
Production 3 does not have left recursion.
E->TE'|E'
E'->+TE'|#
```

8. Write a program to remove ambiguity from a given grammar.

```
#include <stdio.h>
#include <string.h>
int main(){
    char a[20];
    int i, t = 0, x;
    printf("Enter number of ambiguous productions: ");    scanf("%d", &x);
    while (x != 0) {
        printf("Enter ambiguous production: E->");        scanf("%s", a);
        char b[5] = {'E', 'P', 'Q', 'R', 'S'};
        for (i = 0; a[i] != '\0';){
            if (a[i] == '|'){
                i++; }
            else if (a[i] == 'i' && a[i + 1] == 'd') {
                printf("%c->id", b[t]);   i = i + 2; }
            else {
                if (a[i + 1] == '^'){
                    printf("%c->%c^%c|%c\n", b[t], b[t + 1], b[t], b[t + 1]);
                    t++;     i = i + 3;  }
                else{
                    printf("%c->%c%c%c|%c\n", b[t], b[t], a[i + 1], b[t + 1], b[t + 1]);
                    t++;   i = i + 3;   }
            }
        }
        x--;    }}
```

9. Write a program to remove left factoring from a given grammar.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
void main(){
    char ch, lhs[20][20], rhs[20][20][20], temp[20], temp1[20];
    int n, n1, count[20], x, y, i, j, k, c[20];
    printf("\nEnter the no. of nonterminals : ");
    scanf("%d", &n);
    n1 = n;
    for (i = 0; i < n; i++)    {
        printf("\nNonterminal %d \nEnter the no. of productions : ", i + 1);
        scanf("%d", &c[i]);
        printf("\nEnter LHS : ");
        scanf("%s", lhs[i]);
        for (j = 0; j < c[i]; j++){
            printf("%s->", lhs[i]);   scanf("%s", rhs[i][j]);    }
    }
    for (i = 0; i < n; i++){
        count[i] = 1;
        while (memcmp(rhs[i][0], rhs[i][1], count[i]) == 0)  count[i]++;
    }
    for (i = 0; i < n; i++){
        count[i]--;
        if (count[i] > 0){
            strcpy(lhs[n1], lhs[i]);        strcat(lhs[i], "'");
            for (k = 0; k < count[i]; k++)  temp1[k] = rhs[i][0][k];
            temp1[k++] = '\0';
            for (j = 0; j < c[i]; j++) {
                for (k = count[i], x = 0; k < strlen(rhs[i][j]); x++, k++)
                    temp[x] = rhs[i][j][k];
                temp[x++] = '\0';
                if (strlen(rhs[i][j]) == 1)
                    strcpy(rhs[n1][1], rhs[i][j]);
                strcpy(rhs[i][j], temp);
            }
            c[n1] = 2;
            strcpy(rhs[n1][0], temp1);
            strcat(rhs[n1][0], lhs[n1]);
            strcat(rhs[n1][0], "");
            n1++;
        }
    }
    printf("\n\nThe resulting productions are : \n");
    for (i = 0; i < n1; i++){
        if (i == 0)            printf("\n %s -> %c|", lhs[i], (char)238);
        else            printf("\n %s -> ", lhs[i]);
        for (j = 0; j < c[i]; j++){
            printf(" %s ", rhs[i][j]);    if ((j + 1) != c[i])    printf("|");  }
        printf("\b\b\b\n");
    }
}
```



```
Enter the no. of nonterminals : 2

Nonterminal 1
Enter the no. of productions : 3

Enter LHS : S
S->iCtSeS
S->iCtS
S->a

Nonterminal 2
Enter the no. of productions : 1

Enter LHS : C
C->b


The resulting productions are :

 S' -> ε| eS |  |

 C ->  b

 S ->  iCtSS' | a
```

AI:

1. Write a Python script to implement Depth First Search (DFS).

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

2. Write a Python script to implement 8 puzzle problem using BFS.

Given a 3x3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configurations using the empty space. We can slide 4 adjacent (left, right, above and below) tiles into empty space.
Breadth First Search (BFS): A breadth first search can be performed on the state space tree.
The searches begin by visiting the root node of the search tree, given by the initial state.
1. First, we remove a node from the frontier set.
2. Second, we check the state against the goal state to determine if a solution has been found.
3. Finally, if the result of the check is negative, we then expand the node. To expand a given node, we generate successor nodes adjacent to the current node, and add them to the frontier set. Note that if these successor nodes are already in the frontier, or have already been visited, then they should not be added to the frontier again.

3. Write a Python script to implement water jug problem using BFS.

In the water jug problem, we are given a m litre jug and a n litre jug where $0<m<n$. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. We are to use the jugs to measure d litres of water where $d<n$ and determine the minimum number of operations to be performed to obtain d litres of water in one of the jugs.

States: Amount of water in each respective jug where the states are represented by [S(J1), S(J2)] and S(J1) is the amount in the first jug and S(J2) is the amount in the second jug.

Capacity of Jug 1 (J1): 3 litres

Capacity of Jug 2 (J2): 4 litres

Initial state: [0, 0]

Goal state: The user gives the input of the amount of water required in the jug J2 i.e. [2, y] or [2, 0].

Operators:

Fill the first jug.

Fill the second jug.

Empty the first jug.

Empty the second jug.

Pour the first jug into the second jug.

Pour the second jug into the second jug.

Branching factor: 6 (as we have 6 operators)

4. Write a Python script to implement 8 puzzle game using A* algorithm.

An instance of the n-puzzle game consists of a board holding n2-1 distinct movable tiles, plus an empty space. The tiles are numbers from the set 1,..,(n2-1). For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. The blank space is going to be represented with the number 0. Given an initial state of the board, the combinatorial search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order 0, 1,..,(n2-1).

The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions {'Up', 'Down', 'Left', 'Right'}, one move at a time.

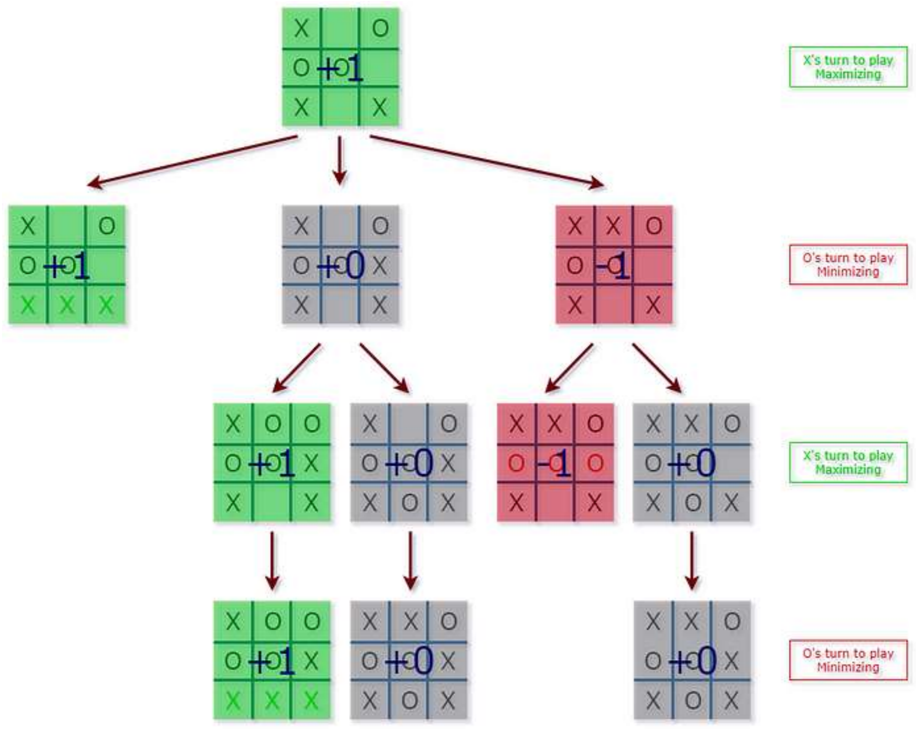
Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

A* algorithm is a searching algorithm that searches for the shortest path between the initial and the final state. It is used in various applications such as maps. The A* algorithm is used to calculate the shortest distance between the source (initial state) and the destination (final state). It has 3 parameters:

1. g: the cost of moving from the initial cell to the current cell.
2. h (heuristic value): the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We must make sure that there is never an over estimation of the cost.
3. f: sum of g and h. So, f = g + h

5. Write a Python script to implement Tic-Tac-Toe game using Minimax algorithm.

Minimax is an artificial intelligence applied in two player games, such as tic-tac-toe, checkers, chess and go. These games are known as zero-sum games, because in a mathematical representation: one player wins (+1) and another player loses (-1) or both of anyone not to win (0). Minimax is a type of adversarial search algorithm for generating and exploring game trees. It is mostly used to solve zero-sum games where one side’s gain is equivalent to other side’s loss, so adding all gains and subtracting all losses end up being zero. Adversarial search differs from conventional searching algorithms by adding opponents into the mix. Minimax algorithm keeps playing the turns of both player and the opponent optimally to figure out the best possible move.


X's turn to play Maximizing
O's turn to play Minimizing
X's turn to play Maximizing
O's turn to play Minimizing
+1
+1
+0
-1
+1
+0
-1
+0
+1
+0
+0

6. Write a Python script to implement graph coloring problem.

Graph coloring refers to the problem of coloring vertices of a graph in such a way that no two adjacent vertices have the same color. This is also called the vertex coloring problem. If coloring is done using at most m colors, it is called m-coloring.

7. Write a Python script to implement brute force approach to knapsack problem.

Given N items where each item has some weight and profit associated with it and also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible.

8. Write a Python script to implement DFS algorithm for water jug problem.

The Water Jug Problem is approached using the Depth-First Search (DFS) algorithm. The algorithm starts with an empty stack and pushes the initial state (both jugs empty) onto the stack. While the stack is not empty, it pops a state from the stack, checks if the state represents the desired quantity, generates all possible next states from the current state, and pushes them onto the stack. This process continues until the stack becomes empty or the desired quantity is found.

9. Write a Python script to implement pre-processing of text.

Text Processing pertains to the analysis of text data using a programming language such as Python. Text Processing is an essential task in NLP as it helps to clean and transform raw data into a suitable format used for analysis or modeling. Whenever we have textual data, we need to apply several processing and pre-processing steps to the data to transform words into numerical features that work with ML algorithms.

Stopwords are words that do not contribute to the meaning of a sentence. Hence, they can safely be removed without causing any change in the meaning of the sentence. The NLTK library has a set of stopwords and we can use these to remove stopwords from our text and return a list of word tokens.

Stemming is the process of getting the root form of a word. Stem or root is the part to which inflectional affixes (-ed, -ize, -de, -s, etc.) are added. The stem of a word is created by removing the prefix or suffix of a word. So, stemming a word may not result in actual words.

Lemmatization also converts a word to its root form. The only difference is that lemmatization ensures that the root word belongs to the language. We will get valid words if we use lemmatization. In NLTK, we use the WordNetLemmatizer to get the lemmas of words. We also need to provide a context for the lemmatization. So, we add the part-of-speech as a parameter.

In tokenization, text data needs to be broken down into smaller units, such as words or phrases, for analysis. Tokenization divides text into meaningful units, facilitating subsequent processing steps like feature extraction.

Bag of Words model is used to preprocess the text by converting it into a bag of words, which keeps a count of the total occurrences of most frequently used words.