LAB 3

Aim: Write a program to convert NFA to DFA
Code:

```
#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<<"************************* AAYUSH *************************"<<endl;
    cout<<"************************* NFA to DFA *************************"<<endl<<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);
}
```

LAB 4

<u>Aim</u>: Write a program to calculate first and follow a given LL(1) grammar.
<u>Code</u>:

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>
void followfirst(char, int, int);
void follow(char c);
void findfirst(char, int, int);
int count, n = 0;
char calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;
int main(int argc, char **argv)
{
   int jm = 0;
   int km = 0;
   int i, choice;
   char c, ch;
   count = 8;
   printf("************************* AAYUSH *******************************\n");
   strcpy(production[0], "X=TnS");
   strcpy(production[1], "X=Rm");
   strcpy(production[2], "T=q");
   strcpy(production[3], "T=#");
   strcpy(production[4], "S=p");
   strcpy(production[5], "S=#");
   strcpy(production[6], "R=om");
   strcpy(production[7], "R=ST");
   int kay;
   char done[count];
   int ptr = -1;
   for (k = 0; k < count; k++)
   {
        for (kay = 0; kay < 100; kay++)
        {
            calc_first[k][kay] = '!';
        }
    }
    int point1 = 0, point2, xxx;
    for (k = 0; k < count; k++)
    {
        c = production[k][0];
        point2 = 0;
        xxx = 0;
        for (kay = 0; kay <= ptr; kay++)
            if (c == done[kay])
                xxx = 1;
        if (xxx == 1)
            continue;
        findfirst(c, 0, 0);
        ptr += 1;
        done[ptr] = c;
        printf("\n First(%c) = { ", c);
        calc_first[point1][point2++] = c;
        for (i = 0 + jm; i < n; i++)
        {
            int lark = 0, chk = 0;
            for (lark = 0; lark < point2; lark++)
            {
                if (first[i] == calc_first[point1][lark])
                {
                    chk = 1;
                    break;
                }
            }
            if (chk == 0)
            {
                printf("%c, ", first[i]);
                calc_first[point1][point2++] = first[i];
            }
        }
        printf("}\n");
        jm = n;
        point1++;
    }
    printf("\n");
    printf("-----------------------------------------------"
            "\n\n");
    char donee[count];
    ptr = -1;
    for (k = 0; k < count; k++)
    {
        for (kay = 0; kay < 100; kay++)
        {
            calc_follow[k][kay] = '!';
        }
    }
    point1 = 0;
    int land = 0;
    for (e = 0; e < count; e++)
    {
        ck = production[e][0];
        point2 = 0;
        xxx = 0;
        for (kay = 0; kay <= ptr; kay++)
            if (ck == donee[kay])
                xxx = 1;
        if (xxx == 1)
            continue;
        land += 1;
        follow(ck);
        ptr += 1;
        donee[ptr] = ck;
        printf(" Follow(%c) = { ", ck);
        calc_follow[point1][point2++] = ck;
        for (i = 0 + km; i < m; i++)
        {
            int lark = 0, chk = 0;
            for (lark = 0; lark < point2; lark++)
            {
                if (f[i] == calc_follow[point1][lark])
                {
                    chk = 1;
```

```
                    break;
                }
            }
            if (chk == 0)
            {
                printf("%c, ", f[i]);
                calc_follow[point1][point2++] = f[i];
            }
        }
        printf(" }\n\n");
        km = m;
        point1++;
    }
}
void follow(char c)
{
    int i, j;
    if (production[0][0] == c)
    {
        f[m++] = '$';
    }
    for (i = 0; i < 10; i++)
    {
        for (j = 2; j < 10; j++)
        {
            if (production[i][j] == c)
            {
                if (production[i][j + 1] != '\0')
                {
                    followfirst(production[i][j + 1], i,
                                (j + 2));
                }
                if (production[i][j + 1] == '\0' && c != production[i][0])
                {
                    follow(production[i][0]);
                }
            }
        }
    }
}
```

```
void findfirst(char c, int q1, int q2)
{
    int j;
    if (!(isupper(c)))
    {
        first[n++] = c;
    }
    for (j = 0; j < count; j++)
    {
        if (production[j][0] == c)
        {
            if (production[j][2] == '#')
            {
                if (production[q1][q2] == '\0')
                    first[n++] = '#';
                else if (production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                {
                    findfirst(production[q1][q2], q1,
                              (q2 + 1));
                }
                else
                    first[n++] = '#';
            }
            else if (!isupper(production[j][2]))
            {
                first[n++] = production[j][2];
            }
            else
            {
                findfirst(production[j][2], j, 3);
            }
        }
    }
}
void followfirst(char c, int c1, int c2)
{
    int k;
    if (!(isupper(c)))
        f[m++] = c;
    else
    {
        int i = 0, j = 1;
        for (i = 0; i < count; i++)
        {
            if (calc_first[i][0] == c)
                break;
        }
        while (calc_first[i][j] != '!')
        {
            if (calc_first[i][j] != '#')
            {
                f[m++] = calc_first[i][j];
            }
            else
            {
                if (production[c1][c2] == '\0')
                {
                    follow(production[c1][0]);
                }
                else
                {
                    followfirst(production[c1][c2], c1,
                                c2 + 1);
                }
            }
            j++;
        }
    }
}
```

LAB 5

Aim: WAP to construct LL(1) parsing table for LL(1) grammar and validate the input string

Code:
```
#include <ctype.h>
#include <stdio.h>
#include <string.h>

void compute_follow_first(char, int, int);
void compute_follow(char c);
void compute_first(char, int, int);

int count, n = 0;
char first_set[10][100];
char follow_set[10][100];
int m = 0;
char grammar_rules[10][10];
char follow[10], first[10];
int k;
char current_char;
int e;

int main(int argc, char **argv) {
    int jm = 0;
    int km = 0;
    int i, choice;
    char c, ch;
    count = 8;
    printf("************************* AAYUSH *******************************\n");
    strcpy(grammar_rules[0], "X=TnS");
    strcpy(grammar_rules[1], "X=Rm");
    strcpy(grammar_rules[2], "T=q");
    strcpy(grammar_rules[3], "T=#");
    strcpy(grammar_rules[4], "S=p");
    strcpy(grammar_rules[5], "S=#");
    strcpy(grammar_rules[6], "R=om");
    strcpy(grammar_rules[7], "R=ST");
    int kay;
```

```
char done[count];
int ptr = -1;
for (k = 0; k < count; k++)
{
    for (kay = 0; kay < 100; kay++)
    {
        first_set[k][kay] = '!';
    }
}
int point1 = 0, point2, xxx;
for (k = 0; k < count; k++)
{
    c = grammar_rules[k][0];
    point2 = 0;
    xxx = 0;
    for (kay = 0; kay <= ptr; kay++)
        if (c == done[kay])
            xxx = 1;
    if (xxx == 1)
        continue;
    compute_first(c, 0, 0);
    ptr += 1;
    done[ptr] = c;
    printf("\n First(%c) = { ", c);
    first_set[point1][point2++] = c;
    for (i = 0 + jm; i < n; i++)
    {
        int lark = 0, chk = 0;
        for (lark = 0; lark < point2; lark++)
        {
            if (first[i] == first_set[point1][lark])
            {
                chk = 1;
                break;
            }
        }
        if (chk == 0)
        {
            printf("%c, ", first[i]);
            first_set[point1][point2++] = first[i];
        }
    }
    printf("}\n");
    jm = n;
    point1++;
}
printf("\n");
printf("-----------------------------------------------"
       "\n\n");
char donee[count];
ptr = -1;
for (k = 0; k < count; k++)
{
    for (kay = 0; kay < 100; kay++)
    {
        follow_set[k][kay] = '!';
    }
}
point1 = 0;
int land = 0;
for (e = 0; e < count; e++)
{
    current_char = grammar_rules[e][0];
    point2 = 0;
    xxx = 0;
    for (kay = 0; kay <= ptr; kay++)
        if (current_char == donee[kay])
            xxx = 1;
    if (xxx == 1)
        continue;
    land += 1;
    compute_follow(current_char);
    ptr += 1;
    donee[ptr] = current_char;
    printf(" Follow(%c) = { ", current_char);
    follow_set[point1][point2++] = current_char;
    for (i = 0 + km; i < m; i++)
    {
        int lark = 0, chk = 0;
        for (lark = 0; lark < point2; lark++)
```

```
            {
                if (follow[i] == follow_set[point1][lark])
                {
                    chk = 1;
                    break;
                }
            }
            if (chk == 0)
            {
                printf("%c, ", follow[i]);
                follow_set[point1][point2++] = follow[i];
            }
        }
        printf(" }\n\n");
        km = m;
        point1++;
    }
}

void compute_follow(char c) {
    int i, j;
    if (grammar_rules[0][0] == c)
    {
        follow[m++] = '$';
    }
    for (i = 0; i < 10; i++)
    {
        for (j = 2; j < 10; j++)
        {
            if (grammar_rules[i][j] == c)
            {
                if (grammar_rules[i][j + 1] != '\0')
                {
                    followfirst(grammar_rules[i][j + 1], i, (j + 2));
                }
                if (grammar_rules[i][j + 1] == '\0' && c != grammar_rules[i][0])
                {
                    compute_follow(grammar_rules[i][0]);
                }
            }
        }
    }
}

void findfirst(char c, int q1, int q2) {
    int j;
    if (!(isupper(c)))
    {
        first[n++] = c;
    }
    for (j = 0; j < count; j++)
    {
        if (grammar_rules[j][0] == c)
        {
            if (grammar_rules[j][2] == '#')
            {
                if (grammar_rules[q1][q2] == '\0')
                    first[n++] = '#';
                else if (grammar_rules[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                {
                    findfirst(grammar_rules[q1][q2], q1, (q2 + 1));
                }
                else
                    first[n++] = '#';
            }
            else if (!isupper(grammar_rules[j][2]))
            {
                first[n++] = grammar_rules[j][2];
            }
            else
            {
                findfirst(grammar_rules[j][2], j, 3);
            }
        }
    }
}

void followfirst(char c, int c1, int c2) {
    int k;
    if (!(isupper(c)))
```

```
            follow[m++] = c;
        else
        {
            int i = 0, j = 1;
            for (i = 0; i < count; i++)
            {
                if (calc_first[i][0] == c)
                    break;
            }
            while (calc_first[i][j] != '!')
            {
                if (calc_first[i][j] != '#')
                {
                    follow[m++] = calc_first[i][j];
                }
                else
                {
                    if (grammar_rules[c1][c2] == '\0')
                    {
                        compute_follow(grammar_rules[c1][0]);
                    }
                    else
                    {
                        followfirst(grammar_rules[c1][c2], c1, c2 + 1);
                    }
                }
                j++;
            }
        }
    }
```

LAB 6

Aim: WAP to construct operator precedence parsing table for the given grammar and check validity of the string.

Code:

```
#include <stdio.h>
#include<stdlib.h>
#include <string.h>
char *input;
int i = 0;
char lasthandle[6], stack[50], handles[][5] = {")E(", "E*E", "E+E", "i", "E^E"};

int top = 0, l;
char prec[9][9] = {'>','>','<','<','<','<','<','>','>',/* - */ '>','>','<','<','<','<','<','>','>',/* * */
'>','>','>','>','<','<','<','>','>',/* / */ '>','>','>','>','<','<','<','>','>',/* ^ */
'>','>','>','>','<','<','<','>','>',/* i */ '>','>','>','>','>','e','e','>','>',/* ( */
'<','<','<','<','<','<','<','>','e',/* ) */ '>','>','>','>','>','e','e','>','>',/* $ */
'<','<','<','<','<','<','<','<','>',};
int getindex(char c)
{
   switch (c)
   {
   case '+':
      return 0;
   case '-':
      return 1;
   case '*':
      return 2;
   case '/':
      return 3;
   case '^':
      return 4;
   case 'i':
      return 5;
   case '(':
      return 6;
   case ')':
      return 7;
```

```
    case '$':
        return 8;
    }
}
int shift()
{
    stack[++top] = *(input + i++);
    stack[top + 1] = '\0';
}
int reduce()
{
    int i, len, found, t;
    for (i = 0; i < 5; i++) {
        len = strlen(handles[i]);
        if (stack[top] == handles[i][0] && top + 1 >= len) {
            found = 1;
            for (t = 0; t < len; t++){
                if (stack[top - t] != handles[i][t]){
                    found = 0;
                    break;
                }
            }
            if (found == 1) {
                stack[top - t + 1] = 'E';
                top = top - t + 1;
                strcpy(lasthandle, handles[i]);
                stack[top + 1] = '\0';
                return 1; }
        }
    }
    return 0;
}
void dispstack()
{
    int j;
    for (j = 0; j <= top; j++)
        printf("%c", stack[j]);
}
void dispinput()
{
    int j;
    for (j = i; j < l; j++)
    printf("%c", *(input + j));
}
int main()
{
    printf("************************ AAYUSH ********************************");
    int j;
    input = (char *)malloc(50 * sizeof(char));
    printf("\nEnter the string\n");
    scanf("%s", input);
    input = strcat(input, "$");
    l = strlen(input);
    strcpy(stack, "$");
    printf("\nSTACK\tINPUT\tACTION");
    while (i <= l){
        shift();
        printf("\n");
        dispstack();
        printf("\t");
        dispinput();
        printf("\tShift");
        if (prec[getindex(stack[top])][getindex(input[i])] == '>'){
            while (reduce()) {
                printf("\n");
                dispstack();
                printf("\t");
                dispinput();
                printf("\tReduced: E->%s", lasthandle);}}}
    if (strcmp(stack, "$E$") == 0)
        printf("\nAccepted;");
    else
        printf("\nNot Accepted;");
}
```

LAB 7

Aim: WAP to construct recursive descent parser
Code:

```
print("Recursive Desent Parsing For following grammar\n")
print("E->TE'\nE'->+TE'/@\nT->FT'\nT'->*FT'/@\nF->(E)/i\n")
print("Enter the string want to be checked\n")
global s
s=list(input())
global i
i=0
def match(a):
   global s
   global i
   if(i>=len(s)):
      return False
   elif(s[i]==a):
      i+=1
      return True
   else:
      return False
def F():
   if(match("(")):
      if(E()):
         if(match(")")):
            return True
         else:
            return False
      else:
         return False
   elif(match("i")):
      return True
   else:
      return False
def Tx():
   if(match("*")):
      if(F()):
         if(Tx()):
            return True
            else:
                return False
        else:
            return False
    else:
        return True
def T():
    if(F()):
        if(Tx()):
            return True
        else:
            return False
    else:
        return False
def Ex():
    if(match("+")):
        if(T()):
            if(Ex()):
                return True
            else:
                return False
        else:
            return False
    else:
        return True
def E():
    if(T()):
        if(Ex()):
            return True
        else:
            return False
    else:
        return False
if(E()):
    if(i==len(s)):
        print("String is accepted")
    else:
         print("String is not accepted")
    else:
    print("string is not accepted")
```