Name: Pradyumn R Pai
Roll No: 50
Class: CS7A

PROGRAM CODE

nfa_ds.c:

```c
#include <stdlib.h>

#include <stdbool.h>

#include <string.h>


struct TransitionNode {

    int target_state;

    char input;

    struct TransitionNode* next;

};


struct State {

    int id;

    struct TransitionNode* transitionListHead;

    bool finalState;

};


struct NFA {

    int stateNum;

    char * inputAlphabet;

    struct State* stateList;

};


struct NFA* init_NFA(int n, char* inputAlphabet){

    struct NFA* out = malloc(sizeof(struct NFA));

    if (!out){

        return NULL; //failed allocation

    }
```

```c
    out->stateNum = n;

    out->inputAlphabet = inputAlphabet;

    out->stateList = malloc(sizeof(struct State)*n);

    if (!out->stateList){

        free(out);

        return NULL;

    }


    for (int i=0;i<n;++i){

        out->stateList[i].id = i;

        out->stateList[i].transitionListHead = NULL;

        out->stateList[i].finalState = false;

    }


    return out;

}


void addTransitionNFA(struct NFA* n, int s, int t, char c){

    struct TransitionNode** head = &(n->stateList[s].transitionListHead);

    while (*head){

        if ((*head)->input==c && (*head)->target_state==t){

            return ; //avoid duplicates

        }

        head = &((*head)->next);

    }

    *head = malloc(sizeof(struct TransitionNode));

    if (!*head){

        return; // allocation failed

    }

    (*head)->target_state = t;
```

```c
    (*head)->input = c;

    (*head)->next = NULL;

}


void freeStateNFA(struct State s){

    struct TransitionNode* head = s.transitionListHead;

    while (head){

        struct TransitionNode* next = head->next;

        free(head);

        head = next;

    }

}


void freeNFA(struct NFA* n){

    if (!n) return;

    for (int i=0;i<(n->stateNum);++i){

        freeStateNFA(n->stateList[i]);

    }


    free(n->inputAlphabet);

    free(n->stateList);

    free(n);

}


void printNFA(struct NFA* nfa){

    printf("The transition table is as follows:\n");

    int n = nfa->stateNum;

    int m = strlen(nfa->inputAlphabet);

    printf("\t");

    for (int i=0;i<m;++i){

        printf("%c\t",nfa->inputAlphabet[i]);
```

```c
    }
    printf("epsilon\n");
    for (int i=0;i<n;++i){
        if (i==0){
            printf("->");
        }
        if (nfa->stateList[i].finalState){
            printf("*");
        }
        printf("q%d\t",i);
        struct State s = nfa->stateList[i];
        for (int j=0;j<=m;++j){
            char c = nfa->inputAlphabet[j];
            if (j==m){
                c = 'e';
            }
            for (struct TransitionNode *current=s.transitionListHead;current;current=current->next){
                if (current->input==c){
                    printf("q%d",current->target_state);
                }
            }
            printf("\t");
        }
        printf("\n");
    }
}




struct NFA* readNFA() {
    // read input
```

```c
int n, m, t, f;
scanf("%d%d%d%d", &n, &f, &m, &t);
if (f<0 || f>n){
   printf("Invalid number of final states\n");
   return NULL;
}


int finalStates[f];
for (int i=0;i<f;++i){
   scanf("%d",finalStates+i);
   if (finalStates[i]<0 || finalStates[i]>=n){
      printf("Invalid final state %d\n",finalStates[i]);
      return NULL;
   }
}



char* inputChars = malloc(sizeof(char)*(m+1));
if (!inputChars) {
   printf("Failed to allocate memory for input characters\n");
   return NULL;
}


scanf("%s\n", inputChars);
if (strlen(inputChars) != m) {
   free(inputChars);
   printf("Input characters length mismatch\n");
   return NULL;
}


struct NFA *nfa = init_NFA(n,inputChars);
```

```c
    if (!nfa) {
        free(inputChars);
        printf("Failed to initialize NFA\n");
        return NULL;
    }


    for (int i=0;i<f;++i){
        nfa->stateList[finalStates[i]].finalState = true;
    }


    for (int i = 0; i < t; ++i) {
        int a, b;
        char c;
        scanf("q%d q%d %c\n", &a, &b, &c);
        if (a < 0 || a >= n || b < 0 || b >= n) {
            printf("Invalid transition from %d to %d\n", a, b);
            freeNFA(nfa);
            return NULL;
        }
        bool validChar = false;
        for (int j = 0; j < m; ++j) {
            if (inputChars[j] == c) {
                validChar = true;
                break;
            }
        }
        if (!validChar && c != 'e') { // 'e' for epsilon transition
            printf("Invalid input character '%c' for transition from %d to %d\n", c, a, b);
            freeNFA(nfa);
            return NULL;
        }
```

```c
        addTransitionNFA(nfa, a, b, c);
    }
    return nfa;
}
```

enfa_functions.c:

```c
#include "nfa_ds.c"
void dfs_closure(struct NFA* nfa,int state, bool visited[]){
    if (visited[state]) return;
    visited[state] = true;
    for (struct TransitionNode* current = (nfa->stateList[state]).transitionListHead;current;current = current->next){
        if (current->input=='e'){
            dfs_closure(nfa,current->target_state,visited);
        }
    }
}


bool* find_epsilon_closure(struct NFA* nfa, int state){
    int n = nfa->stateNum;
    int m = strlen(nfa->inputAlphabet);
    bool* closure = malloc(sizeof(bool)*n);
    for (int i=0;i<n;++i){
        closure[i] = false;
    }
    dfs_closure(nfa,state,closure);
    return closure;
}
```

epsilon_closure.c:

```c
#include <stdio.h>
#include "enfa_functions.c"
```

```c
void print_epsilon_closure(struct NFA* nfa,int state){
    bool* closure = find_epsilon_closure(nfa,state);


    printf("The epsilon closure of state %d is: {",state);
    bool flag = false;
    for (int i=0;i<nfa->stateNum;++i){
        if (!closure[i]) continue;
        if (flag){
            printf(",");
        }
        flag = true;
        printf("q%d",i);
    }
    printf("}\n");


    free(closure);
}

int main(){
    struct NFA* nfa = readNFA();
    if (!nfa) return 1;
    printNFA(nfa);


    // epsilon closure
    for (int i=0;i<nfa->stateNum;++i){
        print_epsilon_closure(nfa,i);
    }


    freeNFA(nfa);
    return 0;
```

```
}
```

OUTPUT:

input.txt:

5 1 2 7

2

01

q0 q1 1

q1 q0 1

q0 q2 e

q2 q3 0

q3 q2 0

q2 q4 1

q4 q2 0

output:

The transition table is as follows:

	0	1	epsilon
->q0		q1	q2
q1		q0	
*q2	q3	q4	
q3	q2		
q4	q2		

The epsilon closure of state 0 is: {q0,q2}

The epsilon closure of state 1 is: {q1}

The epsilon closure of state 2 is: {q2}

The epsilon closure of state 3 is: {q3}

The epsilon closure of state 4 is: {q4}