ABHINAND K PRASAD

S7 CSA

ROLL NO: 4

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a node in a linked list

struct node {

    int st;                // State number

    struct node \*link;     // Pointer to the next node

};

// Define a structure for a new state in the DFA

struct node1 {

    int nst[20];          // Array to hold state numbers (up to 20 states)

};

// Function prototypes

void insert(int, char, int);

int findalpha(char);

void findfinalstate(void);

int insertdfastate(struct node1);

int compare(struct node1, struct node1);

void printnewstate(struct node1);

// Global variables

static int set[20], nostate, noalpha, s, notransition, nofinal, start, finalstate[20], c, r, buffer[20];

int complete = -1;              // Indicator for complete processing of DFA

char alphabet[20];              // Array to hold the alphabet symbols

static int eclosure[20][20] = {0}; // Epsilon closure (not utilized in current code)

struct node1 hash[20];          // Array to store the DFA states

struct node \*transition[20][20] = {NULL}; // Transition table

// Main function

void main() {

    int i, j, k, m, t, n, l;

    struct node \*temp;

    struct node1 newstate = {0}, tmpstate = {0}; // Current and temporary states

    // Input for number of alphabets

    printf("Enter the number of alphabets?\n");

    printf("NOTE:- [ use letter e as epsilon]\n");

    printf("NOTE:- [e must be last character ,if it is present]\n");

    printf("\nEnter No of alphabets and alphabets?\n");

    scanf("%d", &noalpha);

    getchar(); // Consume the newline character

    for (i = 0; i < noalpha; i++) {

        alphabet[i] = getchar(); // Read each alphabet

        getchar(); // Consume the newline character

    }

    // Input for number of states and the start state

    printf("Enter the number of states?\n");

    scanf("%d", &nostate);

    printf("Enter the start state?\n");

    scanf("%d", &start);

    // Input for final states

    printf("Enter the number of final states?\n");

    scanf("%d", &nofinal);

    printf("Enter the final states?\n");

    for (i = 0; i < nofinal; i++)

        scanf("%d", &finalstate[i]);

    // Input for transitions

    printf("Enter no of transition?\n");

    scanf("%d", &notransition);

    printf("NOTE:- [Transition is in the form–> qno alphabet qno]\n");

    printf("NOTE:- [States number must be greater than zero]\n");

    printf("\nEnter transition?\n");

    for (i = 0; i < notransition; i++) {

        scanf("%d%c%d", &r, &c, &s); // Read transitions in the form: qno alphabet qno

        insert(r, c, s); // Insert the transition into the transition table

    }

    // Initialize the hash array to zero

    for (i = 0; i < 20; i++) {

        for (j = 0; j < 20; j++)

            hash[i].nst[j] = 0;

    }

    complete = -1; // Reset complete indicator

    i = -1; // State index

    printf("\nEquivalent DFA.....\n");

    printf("Transitions of DFA\n");

    newstate.nst[start] = start; // Initialize the start state

    insertdfastate(newstate); // Insert the start state into the DFA

    // Main loop to construct the DFA

    while (i != complete) {

        i++; // Increment to process the next state

        newstate = hash[i]; // Get the current state

        for (k = 0; k < noalpha; k++) { // For each alphabet

            c = 0; // Reset counter

            for (j = 1; j <= nostate; j++)

                set[j] = 0; // Reset the set for the new state

            // Compute the next state based on current state and transition

            for (j = 1; j <= nostate; j++) {

                l = newstate.nst[j]; // Get the state number

                if (l != 0) {

                    temp = transition[l][k]; // Get the transition linked list

                    while (temp != NULL) {

                        if (set[temp->st] == 0) {

                            c++; // Increment counter if state is new

                            set[temp->st] = temp->st; // Mark the state as visited

                        }

                        temp = temp->link; // Move to the next transition

                    }

                }

            }

            printf("\n");

            if (c != 0) { // If new state is found

                for (m = 1; m <= nostate; m++)

                    tmpstate.nst[m] = set[m]; // Prepare the new state

                insertdfastate(tmpstate); // Insert the new state into the DFA

                printnewstate(newstate); // Print current state

                printf("%c\t", alphabet[k]); // Print current alphabet

                printnewstate(tmpstate); // Print next state

                printf("\n");

            } else { // No new state found

                printnewstate(newstate); // Print current state

                printf("%c\t", alphabet[k]);

                printf("NULL\n"); // No transition

            }

        }

    }

    // Print final DFA states and details

    printf("\nStates of DFA:\n");

    for (i = 0; i <= complete; i++)

        printnewstate(hash[i]); // Print all states

    printf("\n Alphabets:\n");

    for (i = 0; i < noalpha; i++)

        printf("%c\t", alphabet[i]); // Print all alphabets

    printf("\n Start State:\n");

    printf("q%d", start); // Print start state

    printf("\nFinal states:\n");

    findfinalstate();

    printf("\n");// Print final states

}

// Function to insert a new state into the DFA hash

int insertdfastate(struct node1 newstate) {

    int i;

    for (i = 0; i <= complete; i++) {

        if (compare(hash[i], newstate)) // Check for duplicates

            return 0; // Already exists

    }

    complete++; // Increment to add new state

    hash[complete] = newstate; // Add new state

    return 1; // Successfully added

}

// Function to compare two states

int compare(struct node1 a, struct node1 b) {

    int i;

    for (i = 1; i <= nostate; i++) {

        if (a.nst[i] != b.nst[i]) // If any state differs

            return 0; // Not the same

    }

    return 1; // States are identical

}

// Function to insert a transition into the linked list

void insert(int r, char c, int s) {

    int j;

    struct node \*temp;

    j = findalpha(c); // Find the index of the alphabet

    if (j == 999) { // Error if not found

        printf("error\n");

        exit(0);

    }

    // Create a new transition node

    temp = (struct node \*) malloc(sizeof(struct node));

    temp->st = s; // Set the state number

    temp->link = transition[r][j]; // Link to the previous transitions

    transition[r][j] = temp; // Update the transition table

}

// Function to find the index of an alphabet

int findalpha(char c) {

    int i;

    for (i = 0; i < noalpha; i++)

        if (alphabet[i] == c)

            return i; // Return index if found

    return (999); // Not found

}

// Function to find and print final states in the DFA

void findfinalstate() {

    int i, j, k;

    for (i = 0; i <= complete; i++) { // Iterate through DFA states

        for (j = 1; j <= nostate; j++) { // Check each state

            for (k = 0; k < nofinal; k++) { // Check against final states

                if (hash[i].nst[j] == finalstate[k]) {

                    printnewstate(hash[i]); // Print if it is a final state

                    printf("\t");

                    j = nostate; // Break out of loop if found

                    break;

                }

            }

        }

    }

}

// Function to print the states in the DFA

void printnewstate(struct node1 state) {

    int j;

    printf("{");

    for (j = 1; j <= nostate; j++) {

        if (state.nst[j] != 0)

            printf("q%d,", state.nst[j]); // Print each state in format qX

    }

    printf("}\t"); // Close the set representation

}

**OUTPUT**

