R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi / Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB RECORD

22CS502 – THEORY OF COMPUTATION (LAB INTEGRATED)

Academic Year	**:**	**2025-2026**
Regulations	**:**	**2022**
Batch	**:**	**2023-2027**
Year / Semester	**:**	**III / V**

Institute Vision and Mission

Vision

To be knowledge hub of providing quality technical education and promoting research for building up of our nation and its contribution for the betterment of humanity

Mission

To make the best use of state-of-the-art infrastructure to ensure quality technical education

To develop industrial collaborations to promote innovation and research capabilities

To inculcate values and ethics to serve humanity

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision

To be a source of knowledge in the field of Computer Science and Engineering to cater to the growing need of industry and society

Mission

To avail state-of-the art infrastructure for adopting cutting edge technologies and encouraging research activities

To promote industrial collaborations for professional competency

To nurture social responsibility and ethics to become worthy citizens

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program Educational Objectives (PEOs)

Graduates of Computer Science and Engineering Program will

1. Become a globally competent professional in all spheres and pursue higher education world over.
2. Successfully carry forward domain knowledge in computing and allied areas to solve complex real world engineering problems.
3. Continuously upgrade their technical knowledge and expertise to keep pace with the technological revolution.
4. Serve the humanity with social responsibility combined with ethics.

Program Specific Outcomes (PSOs)

Graduates of Computer Science and Engineering Program will be able to:

- Analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.
- Apply software engineering principles and practices for developing quality software for scientific and business applications.
- Implement cost-effective solutions for the betterment of both industry and society with the technological skills acquired through the Centres of Excellence.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOME

Engineering Knowledge: Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

Problem Analysis: Identify, formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

Design/ Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.

Conduct investigations of complex problems using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.

Modern Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

The Engineer and Society: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice.

Environment and Sustainability: Understand the impact of professional engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.

Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams and in multi-disciplinary settings.

Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions

Project Management and Finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Life-long Learning: Recognize the need for and have the preparation and ability to Engage in independent and life- long learning in the broadest context of technological Change.

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi / Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.

2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.

3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.

b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.

c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.

5. Execute your task in the laboratory, and record the results / output in the lab observation notebook, and get certified by the concerned faculty.

6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.

8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.

9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department

EXPERIMENTS WITH MAPPED CO

S.NO	NAME OF THE EXERCISE	Mapped CO
1	Design a Finite State Machine (FSM) that accepts all strings over input symbols {0,1} having consecutive 1's as a substring.	CO1
2	Construct Epsilon Closure from the given NFA	CO1
3	Convert the given Regular expression to given NFA	CO2
4	Design DFA Minimization form the given Transition Table.	CO2
5	Design a PDA that accepts all strings having equal number of 0's and 1's over input symbol {0,1} for the language 0^n1^n where n>=1.	CO3
6	Design a PDA to accept WCW^R where W is any binary string and W^R is reverse of that string and C is a special symbol.	CO3
7	Design a Program to create PDA achie that accept the well-formed parenthesis.	CO3
8	Design a Turing machine that's accepts the following language $a^n b^n c^n$ where n>0	CO4
9	Design a Turing machine at accepts W^R where w is any binary string and W^R is reverse of that string	CO4
10	Design the travelling salesman problem for NP Complete Problem.	CO5

Rubrics for Assessment

	Excellent	**Good**	**Average**	**Poor**
Problem Understanding & Concept Design	Clear understanding with efficient logic (3 marks)	Good logic with minor issues (2 marks)	Partial understanding (1 mark)	unclear understanding (0 mark)
Code Implementation	Error-free, well-structured, meets all requirements (3 marks)	Minor errors, mostly meets objectives (2 marks)	Major errors (1 mark)	incomplete implementation (0 mark)
Output & Viva Explanation	All test cases passed, confident explanation (4 marks)	Some test cases passed, fair explanation (3 marks)	Few test cases passed, weak explanation (2 marks)	Attempted, no test cases passed, no explanation (1 mark)

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

LIST OF EXPERIMENTS

Sl.NO	NAME OF THE EXPERIMENT
1	Design a Finite State Machine (FSM) that accepts all strings over input symbols {0,1} having consecutive 1's as a substring.
2	Construct Epsilon Closure from the given NFA
3	Convert the given Regular expression to given NFA
4	Design DFA Minimization form the given Transition Table.
5	Design a PDA that accepts all strings having equal number of 0's and 1's over input symbol {0,1} for the language 0^n1^n where n>=1.
6	Design a PDA to accept WCW^R where W is any binary string and W^R is reverse of that string and C is a special symbol.
7	Design a Program to create PDA achie that accept the well-formed parenthesis.
8	Design a Turing machine that's accepts the following language a^n b^n c^n where n>0
9	Design a Turing machine at accepts W^R where w is any binary string and W^R is reverse of that string
10	Design the travelling salesman problem for NP Complete Problem.

Ex. No: 01 **Date:**	**Design A Finite State Machine (Fsm) That Accepts All Strings Over Input Symbols {0,1} Having Consecutive 1's As A Substring.**

AIM:

To design a FSM that accepts all strings over the input symbols {0,1} having consecutive 1's as a substring.

ALGORITHM:

1. Define the States:
   - Start State (q0): The initial state where no '1' has been seen yet.
   - State (q1): A state where exactly one '1' has been seen, waiting to see if another '1' follows.
   - Accept State (q2): The final state where "11" has been detected, and thus the FSM accepts the string.

2. Define the Alphabet:
   - Input symbols are {0, 1}.

3. Define the Transition Function:
   - From q0:

     On input '0', stay in q0 (as '0' alone doesn't lead to "11"). On input '1', transition to q1 (one '1' seen so far).
   - From q1:

     On input '0', transition back to q0 (as '0' following a single '1' breaks the potential "11").

     On input '1', transition to q2 (we have detected "11").
   - From q2:

     On input '0', stay in q2 (once "11" is detected, any subsequent characters are irrelevant).

     On input '1', stay in q2 (as additional '1's still keep the FSM in the accepting state).

4. Define the Start State: The start state is q0.

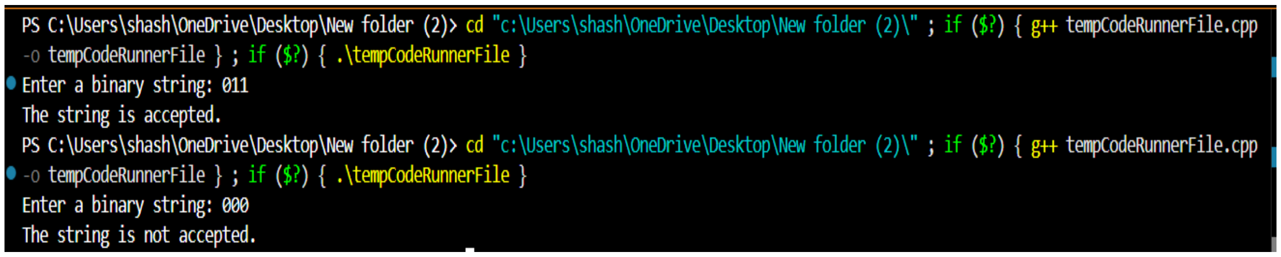
5. Define the Accept (Final) States:
   - The accepting state is q2.

PROGRAM:

```
#include <stdio.h>
#include <string.h>

typedef enum {
    S0,
    S1,
    S2
} State;

int isAccepted(const char *input) {
    State currentState = S0;

    for (int i = 0; i < strlen(input); ++i) {
        char c = input[i];

        switch (currentState) {
            case S0:
                if (c == '1') {
                    currentState = S1;
                } else if (c != '0') {
                    return 0;
                }
                break;

            case S1:
                if (c == '1') {
                    currentState = S2;
                } else if (c == '0') {
                    currentState = S0;
                } else {
                    return 0;
                }
                break;

            case S2:
                break;
        }
    }

    return currentState == S2;
}

int main() {
    char input[100];

    printf("Enter a binary string: ");
```

```
        scanf("%s", input);


        if (isAccepted(input)) {
            printf("The string is accepted.\n");
        } else {
            printf("The string is not accepted.\n");
        }

        return 0;
    }
```

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
-o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a binary string: 011
The string is accepted.
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
-o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a binary string: 000
The string is not accepted.
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, the design of Finite State Machine (FSM) that accepts all strings over input symbols {0,1} having consecutive 1's as a substring is successfully completed.

Ex. No: 2 **Date:**	**CONSTRUCT EPSILON CLOSURE FROM THE GIVEN NFA**

AIM:

To Construct Epsilon Closure from the given NFA

ALGORITHM:

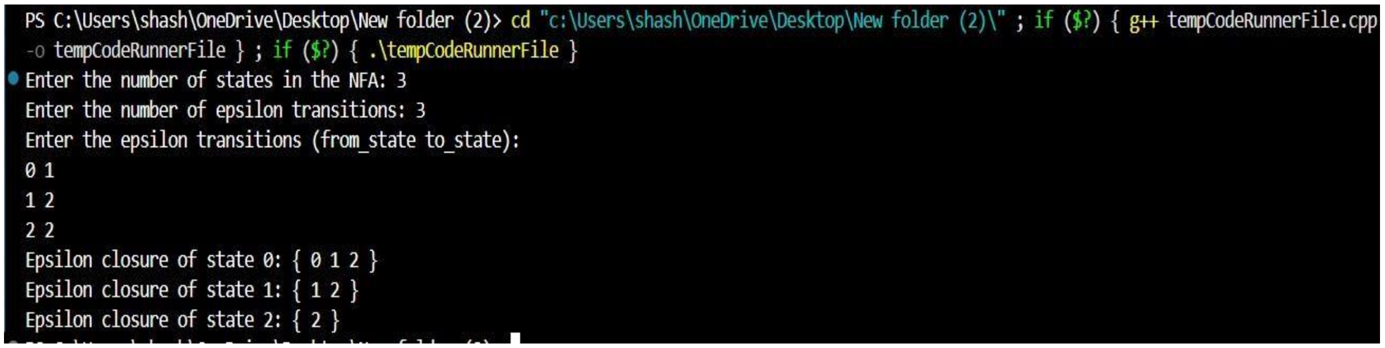
1. Data Structures:
   - NFA struct: Contains the number of states and a 2D array representing transitions. If transitions[i][j] is 1, there's an epsilon transition from state i to state j.
2. DFS Function:
   - A depth-first search (DFS) is used to find all states reachable from a given state through epsilon transitions. The visited array keeps track of which states have been reached.
3. Epsilon Closure Function:
   - This function initializes a visited array and calls the DFS function starting from the given state. It then prints all states that are reachable from the given state using epsilon transitions.
4. Main Function:
   - Reads the number of states and epsilon transitions from the user.
   - Initializes the NFA transitions matrix.
   - Reads the epsilon transitions.

- Computes and prints the epsilon closure for each state in the NFA.

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>


#define MAX_STATES 100


typedef struct {

    int numStates;

    int transitions[MAX_STATES][MAX_STATES];

} NFA;

void dfs(int state, int visited[], NFA *nfa) {

    visited[state] = 1;


    for (int i = 0; i < nfa->numStates; i++) {

        if (nfa->transitions[state][i] && !visited[i]) {

            dfs(i, visited, nfa);

        }

    }

}

void epsilonClosure(int state, NFA *nfa) {

    int visited[MAX_STATES] = {0};

    dfs(state, visited, nfa);

    printf("Epsilon closure of state %d: { ", state);

    for (int i = 0; i <  nfa->numStates; i++) {

        if (visited[i]) {
            printf("%d ", i);
        }
    }
    printf("}\n");
}

int main() {
    NFA nfa;
    int numTransitions;
    int fromState, toState;

    printf("Enter the number of states in the NFA: ");
    scanf("%d", &nfa.numStates);

    printf("Enter the number of epsilon transitions: ");
    scanf("%d", &numTransitions);

     // Initialize all transitions to 0
    for (int i = 0; i < nfa.numStates; i++) {
        for (int j = 0; j < nfa.numStates; j++) {
            nfa.transitions[i][j] = 0;
        }
    }

    printf("Enter the epsilon transitions (from_state to_state):\n");
    for (int i = 0; i < numTransitions; i++) {
        scanf("%d %d", &fromState, &toState);
        nfa.transitions[fromState][toState] = 1;
    }

    // Compute and display epsilon-closure for each state
    for (int i = 0; i < nfa.numStates; i++) {
        epsilonClosure(i,  &nfa);
    }

    return 0;
}
```

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter the number of states in the NFA: 3
Enter the number of epsilon transitions: 3
Enter the epsilon transitions (from_state to_state):
0 1
1 2
2 2
Epsilon closure of state 0: { 0 1 2 }
Epsilon closure of state 1: { 1 2 }
Epsilon closure of state 2: { 2 }
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, the Epsilon Closure for the given NFA is constructed and executed successfully.

Ex. No: 3 **Date:**	**CONVERT THE GIVEN REGULAR EXPRESSION TO NFA**

AIM:

To Convert the given regular expression to NFA.

ALGORITHM:

1. Data Structures:
    - Transition struct: Represents a transition with a from state, a to state, and a symbol.
    - NFA struct: Represents an NFA with states, transitions, start state, and accept state.
    - Regex struct: Used for converting the regex to postfix notation.
2. Functions:
    - addTransition: Adds a transition to the NFA.
    - initNFA: Initializes an NFA.
    - regexToPostfix: Converts the regular expression to postfix notation (assumes input is already in postfix for simplicity).
    - createNFAForSymbol: Creates an NFA for a single symbol.
    - createNFAForConcatenation: Creates an NFA for concatenation of two NFAs.
    - createNFAForUnion: Creates an NFA for union of two NFAs.
    - createNFAForKleeneStar: Creates an NFA for Kleene star of an NFA.
    - regexToNFA: Converts the entire regular expression to an NFA.
    - printNFA: Prints the NFA.
3. Main Function:
    - Reads a regular expression from the user.
    - Converts the regular expression to an NFA.

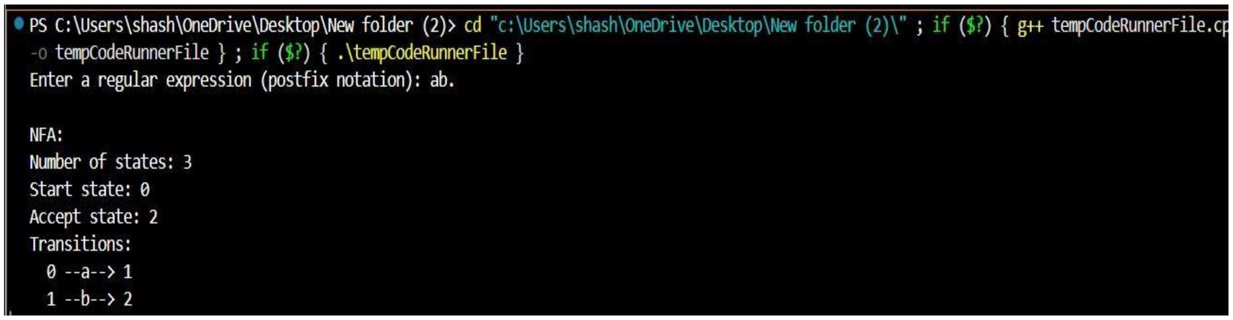
PROGRAM:

```
#include  <stdio.h>
#include  <stdlib.h>
#include <string.h>

#define MAX_STATES 100
#define MAX_TRANSITIONS 100

typedef struct {
    int fromState;
    int toState;
    char symbol;
} Transition;

typedef struct {
    int numStates;
    int startState;
    int acceptState;
    Transition transitions[MAX_TRANSITIONS];
    int numTransitions;
} NFA;

typedef struct {
    char* postfix;
    int index;
} Regex;

void addTransition(NFA* nfa, int from, int to, char symbol) {
    nfa->transitions[nfa->numTransitions].fromState = from;
    nfa->transitions[nfa->numTransitions].toState = to;
    nfa->transitions[nfa->numTransitions].symbol = symbol;
    nfa->numTransitions++;
}

void initNFA(NFA* nfa) {
    nfa->numStates = 0;
    nfa->numTransitions = 0;
    nfa->startState = 0;
    nfa->acceptState = 0;
}

void regexToPostfix(const char* regex, char* postfix) {
    // Simplified: assumes regex is already in postfix form
    strcpy(postfix, regex);
}
```

```
void createNFAForSymbol(NFA* nfa, char symbol) {
    initNFA(nfa);
    nfa->numStates = 2;
    nfa->startState = 0;
    nfa->acceptState = 1;
    addTransition(nfa, 0, 1, symbol);
}

void createNFAForConcatenation(NFA* nfa1, NFA* nfa2) {
    NFA result;
    initNFA(&result);

    result.numStates = nfa1->numStates + nfa2->numStates - 1;
    result.startState = nfa1->startState;
    result.acceptState = nfa2->acceptState + nfa1->numStates - 1;

    for (int i = 0; i < nfa1->numTransitions; i++) {
        addTransition(&result,
                      nfa1->transitions[i].fromState,
                      nfa1->transitions[i].toState,
                      nfa1->transitions[i].symbol);
    }

    for (int i = 0; i < nfa2->numTransitions; i++) {
        addTransition(&result,
                      nfa2->transitions[i].fromState + nfa1->numStates - 1,
                      nfa2->transitions[i].toState + nfa1->numStates - 1,
                      nfa2->transitions[i].symbol);
    }

    *nfa1 = result;
}

void createNFAForUnion(NFA* nfa1, NFA* nfa2) {
    NFA result;
    initNFA(&result);

    result.numStates = nfa1->numStates + nfa2->numStates + 2;
    result.startState = 0;
    result.acceptState = result.numStates - 1;


    addTransition(&result, result.startState, nfa1->startState + 1, 'e');            // ε-transitions
    addTransition(&result, result.startState, nfa2->startState + nfa1->numStates + 1, 'e');
    for (int i = 0; i < nfa1->numTransitions; i++) {
        addTransition(&result,
                     nfa1->transitions[i].fromState + 1,
                     nfa1->transitions[i].toState + 1,
                     nfa1->transitions[i].symbol);

    }

    for (int i = 0; i < nfa2->numTransitions; i++) {
        addTransition(&result,
                     nfa2->transitions[i].fromState + nfa1->numStates + 1,
                     nfa2->transitions[i].toState + nfa1->numStates + 1,
                     nfa2->transitions[i].symbol);
    }

    addTransition(&result, nfa1->acceptState + 1, result.acceptState, 'e');
    addTransition(&result, nfa2->acceptState + nfa1->numStates + 1, result.acceptState, 'e');

    *nfa1 = result;
}

void createNFAForKleeneStar(NFA* nfa) {
    NFA result;
    initNFA(&result);
    result.numStates = nfa->numStates + 2;
    result.startState = 0;
    result.acceptState = result.numStates - 1;
    addTransition(&result, result.startState, nfa->startState + 1, 'e');
    addTransition(&result, result.startState, result.acceptState, 'e');
    addTransition(&result, nfa->acceptState + 1, nfa->startState + 1, 'e');
    addTransition(&result, nfa->acceptState + 1, result.acceptState, 'e');

    for (int i = 0; i < nfa->numTransitions; i++) {
        addTransition(&result,
                     nfa->transitions[i].fromState + 1,
                     nfa->transitions[i].toState + 1,
                     nfa->transitions[i].symbol);
    }
    *nfa = result;
}
void regexToNFA(const char* regex, NFA* nfa) {
    char postfix[MAX_STATES];
    regexToPostfix(regex, postfix);

    NFA stack[MAX_STATES];
    int stackTop = -1;

    for (int i = 0; i < (int)strlen(postfix); i++) {
        char symbol = postfix[i];
        if (symbol == 'a' || symbol == 'b') {  // allowed symbols
            createNFAForSymbol(&stack[++stackTop], symbol);
        } else if (symbol == '.') {  // concatenation
            NFA nfa2 = stack[stackTop--];
            NFA nfa1 = stack[stackTop--];
            createNFAForConcatenation(&nfa1, &nfa2);
            stack[++stackTop] = nfa1;

        }  else  if  (symbol  ==  'I')  {   //  union
            NFA  nfa2   =   stack[stackTop--];
            NFA nfa1 = stack[stackTop--];
            createNFAForUnion(&nfa1,     &nfa2);
            stack[++stackTop] = nfa1;
        } else if (symbol == '*') {  // Kleene star
            NFA nfa = stack[stackTop--];
            createNFAForKleeneStar(&nfa);
            stack[++stackTop] = nfa;
        }
    }

    *nfa = stack[stackTop];
}

void printNFA(NFA* nfa) {
    printf("\nNFA:\n");
    printf("Number of states: %d\n", nfa->numStates);
    printf("Start state: %d\n", nfa->startState);
    printf("Accept state: %d\n", nfa->acceptState);
    printf("Transitions:\n");

    for (int i = 0; i < nfa->numTransitions; i++) {
        printf("  %d --%c--> %d\n",
               nfa->transitions[i].fromState,
               nfa->transitions[i].symbol,
               nfa->transitions[i].toState);
    }
}

int main() {
    char regex[MAX_STATES];
    NFA nfa;

    printf("Enter a regular expression (in postfix form): ");
    scanf("%s", regex);

    regexToNFA(regex, &nfa);
    printNFA(&nfa);

    return 0;
```

}

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cp
-o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a regular expression (postfix notation): ab.

NFA:
Number of states: 3
Start state: 0
Accept state: 2
Transitions:
  0 --a--> 1
  1 --b--> 2
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, the conversion of regular expression to NFA is executed successfully.

Ex. No: 4 **Date:**	**DESIGN DFA MINIMIZATION FROM THE GIVEN TRANSITION TABLE**

AIM:

To design DFA minimization form the given transition table.

ALGORITHM:

1. Remove Unreachable States:
    - Identify and eliminate states that are not reachable from the start state.
2. Create an Initial Partition:
    - Partition the set of states into two groups: final (accepting) states and non-final (non-accepting) states.
3. Refine the Partition:
    - Iteratively refine the partition until no further refinement is possible. Two states are in the same subset of the partition if and only if for every input symbol, the states transition to states within the same subset of the partition.
4. Create the Minimized DFA:
    - Each subset of the final partition represents a state in the minimized DFA.
    - Define the transitions based on the transitions of the representative states of each subset.

PROGRAM:

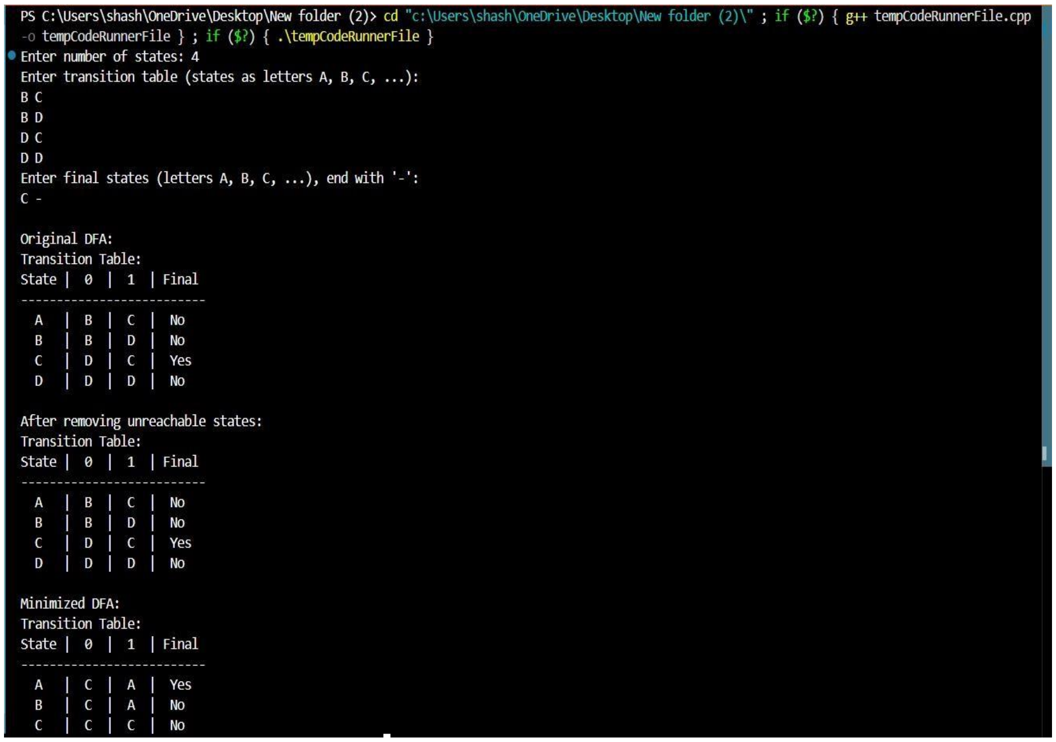
```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

#define MAX_STATES 100
#define MAX_SYMBOLS 2 // Binary DFA (symbols 0 and 1)

typedef struct {
    int transitions[MAX_STATES][MAX_SYMBOLS];
    bool is_final[MAX_STATES];
} DFA;

typedef struct {
    int *states;
    int count;
} PartitionSet;

void initialize_partition(PartitionSet *partition, int size) {
    partition->states = (int *)malloc(size * sizeof(int));
    partition->count = 0;
}

void add_to_partition(PartitionSet *partition, int state) {
    partition->states[partition->count++] = state;
}

void clear_partition(PartitionSet *partition) {
    partition->count = 0;
}

void free_partition(PartitionSet *partition) {
    free(partition->states);
}

void print_dfa(DFA *dfa, int num_states) {
    printf("Transition Table:\n");
    printf("State | 0 | 1 | Final\n");
    printf(" -----------------------------\n");
    for (int i = 0; i < num_states; i++) {
        printf("  %c  | %c | %c | %s\n",
               'A' + i,
               'A' + dfa->transitions[i][0],
               'A' + dfa->transitions[i][1],
               dfa->is_final[i] ? "Yes" : "No");
    }
}

void read_dfa(DFA *dfa, int *num_states, int *num_symbols) {
    printf("Enter number of states: ");
    scanf("%d", num_states);

    *num_symbols = MAX_SYMBOLS; // Binary DFA

    printf("Enter transition table (states as letters A, B, C, ...):\n");
    for (int i = 0; i < *num_states; i++) {
        for (int j = 0; j < *num_symbols; j++) {
            char state;
            scanf(" %c", &state);
            dfa->transitions[i][j] = state - 'A';
        }
    }

    printf("Enter final states (letters A, B, C, ...), end with '-':\n");
    for (int i = 0; i < *num_states; i++) {
        dfa->is_final[i] = false;
    }

    while (1) {
        char state;
        scanf(" %c", &state);
        if (state == '-') break;
        dfa->is_final[state - 'A'] = true;
    }
}

void remove_unreachable_states(DFA *dfa, int *num_states, int start_state) {
    bool reachable[MAX_STATES] = {false};
    reachable[start_state] = true;

    bool changed = true;
    while (changed) {
        changed = false;
        for (int i = 0; i < *num_states; i++) {
            if (reachable[i]) {
                for (int j = 0; j < MAX_SYMBOLS; j++) {
                    int next_state = dfa->transitions[i][j];
                    if (!reachable[next_state]) {
```

```
                    reachable[next_state] = true;
                    changed = true;
                }
            }
        }
    }
}

int new_num_states = 0;
int state_mapping[MAX_STATES];
for (int i = 0; i < *num_states; i++) {
    if (reachable[i]) {
        state_mapping[i] = new_num_states++;
    } else {
        state_mapping[i] = -1;
    }
}

DFA new_dfa;
for (int i = 0; i < *num_states; i++) {
    if (reachable[i]) {
        for (int j = 0; j < MAX_SYMBOLS; j++) {
            new_dfa.transitions[state_mapping[i]][j] =
                state_mapping[dfa->transitions[i][j]];
        }
        new_dfa.is_final[state_mapping[i]] = dfa->is_final[i];
    }
}

*dfa = new_dfa;
*num_states = new_num_states;
}

void minimize_dfa(DFA *dfa, int *num_states) {
    PartitionSet final_states, non_final_states;
    initialize_partition(&final_states, *num_states);
    initialize_partition(&non_final_states, *num_states);

    for (int i = 0; i < *num_states; i++) {
        if (dfa->is_final[i])
            add_to_partition(&final_states, i);
        else
            add_to_partition(&non_final_states,   i);
    }

    PartitionSet partitions[MAX_STATES];
    int num_partitions = 0;
    if (final_states.count > 0)
        partitions[num_partitions++] = final_states;
```

```
if (non_final_states.count > 0)
    partitions[num_partitions++] = non_final_states;

bool changed = true;
while (changed) {
    changed = false;
    PartitionSet new_partitions[MAX_STATES];
    int new_num_partitions = 0;

    for (int i = 0; i < num_partitions; i++) {
        PartitionSet *partition = &partitions[i];
        bool split[MAX_STATES] = {false};

        for (int j = 1; j < partition->count; j++) {
            for (int k = 0; k < MAX_SYMBOLS; k++) {
                int s1 = partition->states[0];
                int s2 = partition->states[j];
                int t1 = dfa->transitions[s1][k];
                int t2 = dfa->transitions[s2][k];

                bool in_same_partition = false;
                for (int l = 0; l < num_partitions; l++) {
                    PartitionSet *p = &partitions[l];
                    bool found_t1 = false, found_t2 = false;
                    for (int m = 0; m < p->count; m++) {
                        if (p->states[m] == t1) found_t1 = true;
                        if (p->states[m] == t2) found_t2 = true;
                    }
                    if (found_t1 && found_t2) {
                        in_same_partition = true;
                        break;
                    }
                }

                if (!in_same_partition) {
                    split[j] = true;
                    changed = true;
                    break;
                }
            }
        }

        if (changed) {
            PartitionSet new_partition1, new_partition2;
            initialize_partition(&new_partition1, partition->count);
            initialize_partition(&new_partition2, partition->count);

            add_to_partition(&new_partition1, partition->states[0]);
```

```
                for (int j = 1; j < partition->count; j++) {
                    if (split[j])
                        add_to_partition(&new_partition2, partition->states[j]);
                    else
                        add_to_partition(&new_partition1,     partition->states[j]);
                }

                new_partitions[new_num_partitions++] = new_partition1;
                new_partitions[new_num_partitions++]  = new_partition2;
            } else {
                new_partitions[new_num_partitions++]  =  *partition;
            }
        }

        memcpy(partitions, new_partitions, new_num_partitions * sizeof(PartitionSet));
        num_partitions = new_num_partitions;
    }

    int state_mapping[MAX_STATES];
    for (int i = 0; i < num_partitions; i++) {
        for (int j = 0; j < partitions[i].count; j++) {
            state_mapping[partitions[i].states[j]] = i;
        }
    }

    DFA new_dfa;
    for (int i = 0; i < num_partitions; i++) {
        int rep_state = partitions[i].states[0];
        for (int j = 0; j < MAX_SYMBOLS; j++) {
            new_dfa.transitions[i][j]  =  state_mapping[dfa->transitions[rep_state][j]];
        }
        new_dfa.is_final[i]  =  dfa->is_final[rep_state];
    }

    *dfa = new_dfa;
    *num_states  =  num_partitions;

    for (int i = 0; i < num_partitions; i++) {
        free_partition(&partitions[i]);
    }
}

int main() {
    DFA dfa;
    int num_states, num_symbols;

    read_dfa(&dfa, &num_states, &num_symbols);

    printf("\nOriginal DFA:\n");
    print_dfa(&dfa,  num_states);

    remove_unreachable_states(&dfa, &num_states, 0);
    printf("\nAfter removing unreachable states:\n");
    print_dfa(&dfa, num_states);

    minimize_dfa(&dfa, &num_states);
    printf("\nMinimized DFA:\n");
    print_dfa(&dfa, num_states);

    return 0;
}
```

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
-o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter number of states: 4
Enter transition table (states as letters A, B, C, ...):
B C
B D
D C
D D
Enter final states (letters A, B, C, ...), end with '-':
C -

Original DFA:
Transition Table:
State |  0  |  1  | Final
-------------------------
  A   |  B  |  C  |  No
  B   |  B  |  D  |  No
  C   |  D  |  C  |  Yes
  D   |  D  |  D  |  No

After removing unreachable states:
Transition Table:
State |  0  |  1  | Final
-------------------------
  A   |  B  |  C  |  No
  B   |  B  |  D  |  No
  C   |  D  |  C  |  Yes
  D   |  D  |  D  |  No

Minimized DFA:
Transition Table:
State |  0  |  1  | Final
-------------------------
  A   |  C  |  A  |  Yes
  B   |  C  |  A  |  No
  C   |  C  |  C  |  No
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus records have been retrieved using various clauses successfully.

Ex. No: 5 **Date:**	**Design a PDA that accepts all string having equal number of 0's and 1's over input symbol {0,1} for a language 0^n1^n where n>=1.**

AIM:

To design PDA that accepts all strings having equal number of 0's and 1's over the input symbol {0,1} for language 0^n1^n where n > = 1.

ALGORITHM:

Stack Operations:

- The push function adds an element to the stack.
- The pop function removes the top element from the stack.
- The peek function returns the top element without removing it.
- The is_empty function checks if the stack is

empty. PDA Acceptance Function:

- The pda_accepts function processes the input string:
  - For each 0 encountered, it pushes it onto the stack.
  - For each 1 encountered, it pops a 0 from the stack.
  - If it encounters a 1 when the stack is empty or finds any invalid symbols, it returns false.
  - Finally, it checks if the stack is empty to determine if the input string has been correctly processed.

Main Function:

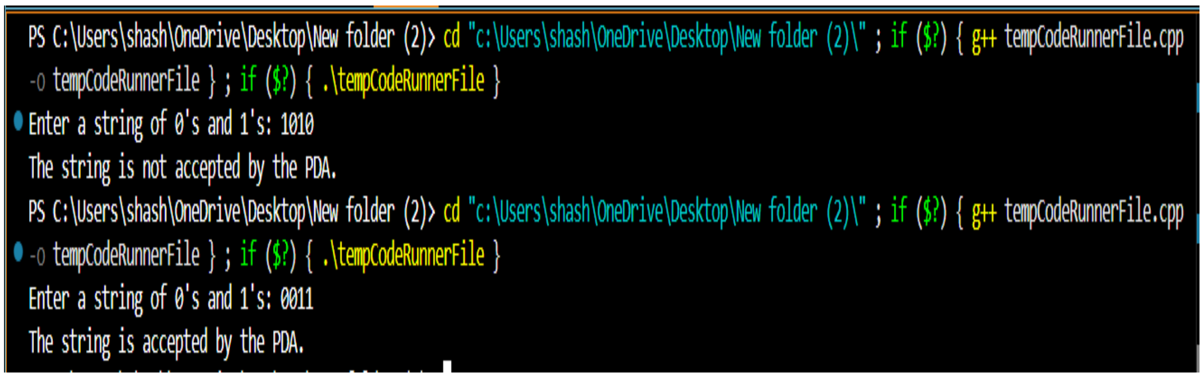
The main function takes the input string from the user, calls the pda_accepts function, and prints whether the string is accepted by the PDA.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_STACK_SIZE 100

typedef struct {
    char stack[MAX_STACK_SIZE];
    int top;
} Stack;

void push(Stack *s, char c) {
    if (s->top < MAX_STACK_SIZE - 1) {
        s->stack[++(s->top)] = c;
    } else {
        printf("Stack overflow\n");
    }
}

char pop(Stack *s) {
    if (s->top >= 0) {
        return s->stack[(s->top)--];
    } else {
        printf("Stack underflow\n");
        return '\0';
    }
}

char peek(Stack *s) {
    if (s->top >= 0) {
        return  s->stack[s->top];
    } else {
        return '\0';
    }
}

bool is_empty(Stack *s) {
    return s->top == -1;
}

bool pda_accepts(const char *input) {
    Stack stack;
    stack.top = -1;

    int length = strlen(input);

    for (int i = 0; i < length; i++) {
        if (input[i] == '0') {
            push(&stack,  '0');
        } else if (input[i] == '1') {
            if (is_empty(&stack) || pop(&stack) != '0') {
                return false;
            }
        } else {
            return false;   //INVALID  input  symbol
        }
    }

    return  is_empty(&stack);
}
int main() {
    char input[MAX_STACK_SIZE];
```

```
        printf("Enter a string of 0's and 1's: ");
        scanf("%s", input);

        if (pda_accepts(input)) {
            printf("The string is accepted by the PDA.\n");
        } else {
            printf("The string is not accepted by the PDA.\n");
        }

        return 0;
    }
```

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a string of 0's and 1's: 1010
The string is not accepted by the PDA.
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a string of 0's and 1's: 0011
The string is accepted by the PDA.
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, the PDA accepts the given language is executed successfully.

Ex. No: 6 **Date:**	**Design a PDA to accept WCW^R where w is any binary string and W^R is reverse of that string and C is a special symbol.**

AIM:

To design a PDA to accept WCW^R where w is any binary string and W^R is reverse of that string and C is a special symbol.

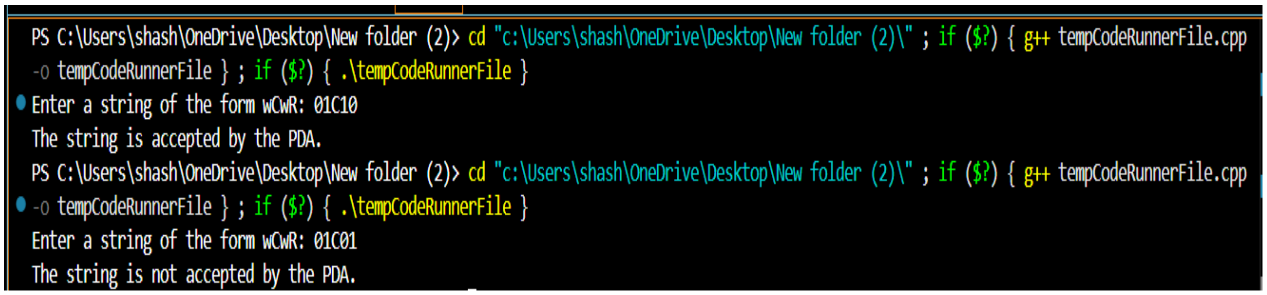
ALGORITHM:

1. Stack Operations:
   - push adds an element to the stack.
   - pop removes the top element from the stack.
   - is empty checks if the stack is empty.
2. PDA Acceptance Function:
   - The PDA accepts function processes the input string in two phases:
     - Push Phase: It pushes characters 0 and 1 onto the stack until it encounters C.
     - Pop and Compare Phase: After C, it pops the stack and compares each character with the input. If they don't match or if any invalid symbol is encountered, it returns false.
   - Finally, it checks if the stack is empty to determine if the input string has been correctly processed.
3. Main Function:
   - The main function takes the input string from the user, calls the pda_accepts function, and prints whether the string is accepted by the PDA

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_STACK_SIZE 100

typedef struct {
    char stack[MAX_STACK_SIZE];
    int top;
} Stack;

void push(Stack *s, char c) {
    if (s->top < MAX_STACK_SIZE - 1) {
        s->stack[++(s->top)] = c;
    } else {
        printf("Stack overflow\n");
    }
}

char pop(Stack *s) {
    if (s->top >= 0) {
        return  s->stack[(s->top)--];
    } else {
        printf("Stack underflow\n");
        return '\0';
    }
}

bool is_empty(Stack *s) {
    return s->top == -1;
}

bool pda_accepts(const char *input) {
    Stack stack;
    stack.top = -1;

    int length = strlen(input);
    int i = 0;

    // Push phase: Push 'w' onto the stack until 'C' is found
    while (i < length && input[i] != 'C') {
        if (input[i] == '0' || input[i] == '1') {
            push(&stack, input[i]);
        } else {
            return false; // Invalid input symbol
        }
        i++;
    }

    if (i == length || input[i] != 'C') {
        return false;
    }

    // Move past 'C'
    i++;

    // Pop and compare phase: Compare each character of wR with the stack
    while (i < length) {
        if (input[i] == '0' || input[i] == '1') {
            if (is_empty(&stack) || pop(&stack) != input[i]) {
                return false;
            }
        } else {
            return false; // Invalid input symbol
        }
        i++;
    }

    // Accept if stack is empty after full match
    return is_empty(&stack);
}

int main() {
    char input[MAX_STACK_SIZE];

    printf("Enter a string of the form wCwR: ");
    scanf("%s", input);

    if (pda_accepts(input)) {
        printf("The string is accepted by the PDA.\n");
    } else {
        printf("The string is not accepted by the PDA.\n");
    }

    return 0;
}
```

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a string of the form wCwR: 01C10
The string is accepted by the PDA.
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a string of the form wCwR: 01C01
The string is not accepted by the PDA.
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, the PDA for the given string is executed successfully.

Ex. No: 7 Date:	**DESIGN A PROGRAM TO CREATE PDA MACHINE THAT ACCEPT WELL FORMED PARENTHESIS.**

AIM:

To design and create PDA machine that accept the well-formed parenthesis.

ALGORITHM:

1. Stack Operations:

- push adds an element to the stack.
- pop removes the top element from the stack.
- is_empty checks if the stack is empty.

2. PDA Acceptance Function:

- The pda_accepts function processes the input string:
    - For each (encountered, it pushes it onto the stack.
    - For each) encountered, it pops the stack and checks if the top element is a matching (. If the stack is empty or the top element doesn't match, it returns false.
    - If any invalid symbols are encountered, it returns false.
- Finally, it checks if the stack is empty to determine if the input string has been correctly processed (i.e., all opening parentheses have been matched by closing parentheses).

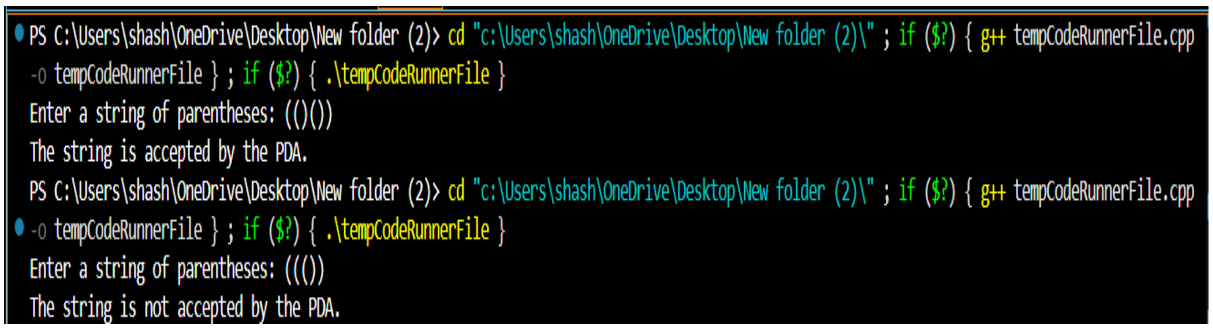
3. Main Function:

- The main function takes the input string from the user, calls the pda_accepts function, and prints whether the string is accepted by the PDA.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX_STACK_SIZE 100

typedef struct {
    char stack[MAX_STACK_SIZE];
    int top;
} Stack;

void push(Stack *s, char c) {
    if (s->top < MAX_STACK_SIZE - 1) {
        s->stack[++(s->top)] = c;
    } else {
        printf("Stack overflow\n");
    }
}

char pop(Stack *s) {
    if (s->top >= 0) {
        return  s->stack[(s->top)--];
    } else {
        printf("Stack underflow\n");
        return '\0';
    }
}

// Check if the stack is empty
bool is_empty(Stack *s) {
    return s->top == -1;
}

bool pda_accepts(const char *input) {
    Stack stack;
    stack.top = -1;
    int length = strlen(input);

    for (int i = 0; i < length; i++) {
        if (input[i] == '(') {
            push(&stack,  '(');
        } else if (input[i] == ')') {
            if (is_empty(&stack) || pop(&stack) != '(') {
                return false;
            }
        } else {
            return false; // Invalid input symbol
        }
    }

    return is_empty(&stack);
}

int main() {
    char input[MAX_STACK_SIZE];

    printf("Enter a string of parentheses: ");
    scanf("%s", input);

    if (pda_accepts(input)) {
        printf("The string is accepted by the PDA.\n");
    } else {
        printf("The string is not accepted by the PDA.\n");
    }

    return 0;
}
```

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a string of parentheses: (()())
The string is accepted by the PDA.
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a string of parentheses: ((())
The string is not accepted by the PDA.
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, the program for PDA machine that accepts the parenthesis is executed successfully.

Ex. No: 8 **Date:**	**Design a Turing Machine that's accepts the following language $a^nb^nc^n$ where n>0.**

AIM :

To design TM that accepts the language $a^nb^nc^n$ where n>0.

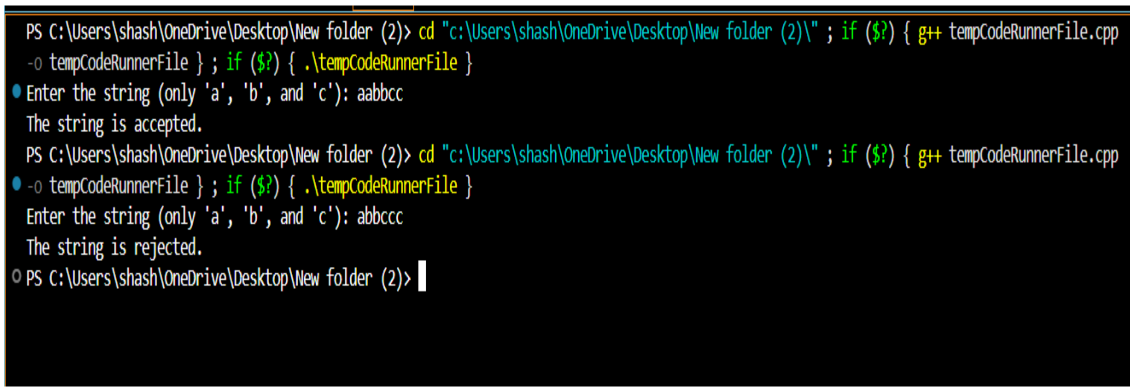
ALGORITHM:

1. A string consisting of a's followed by b's followed by c's.
2. Scan for the first 'a', replace it with 'X', and move to the first 'b'.
3. For each 'a' replaced with 'X', find a corresponding 'b' and 'c', replace them with 'Y' and 'Z' respectively.
4. If all characters are matched and replaced correctly, the machine should reach an accepting state.
5. Define transitions between states based on the current character being read.

PROGRAM:

```
#include <stdio.h>
#include <string.h>

#define MAX_LENGTH 100

// Function to check if the string is of the form a^n b^n c^n
int isAccepted(char *input) {
    int length = strlen(input);
    int a_count = 0, b_count = 0, c_count = 0;
    int i;

    // Count 'a's
    for (i = 0; i < length && input[i] == 'a'; i++) {
        a_count++;
    }

    // Count 'b's
    for (; i < length && input[i] == 'b'; i++) {
        b_count++;
    }

    // Count 'c's
    for (; i < length && input[i] == 'c'; i++) {
        c_count++;
    }

    // Check validity: all counts must be equal and no extra characters
    if (i != length || a_count == 0 || a_count != b_count || b_count != c_count) {
        return 0;
    }

    return 1;
}

int main() {
    char input[MAX_LENGTH];

    printf("Enter the string (only 'a', 'b', and 'c'): ");
    scanf("%s", input);

    if (isAccepted(input)) {
        printf("The string is accepted.\n");
    } else {
        printf("The string is rejected.\n");
    }

    return 0;
}
```

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
-o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter the string (only 'a', 'b', and 'c'): aabbcc
The string is accepted.
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
-o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter the string (only 'a', 'b', and 'c'): abbccc
The string is rejected.
PS C:\Users\shash\OneDrive\Desktop\New folder (2)>
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, the program for TM that accepts the language $a^nb^nc^n$ is executed successfully.

Ex. No: 9 **Date:**	**Design a Turing Machine to accept W^R where w is any binary string and W^R is reverse of that string**

AIM :

To design TM to accept W^R where w is any binary string and W^R is reverse of that string.

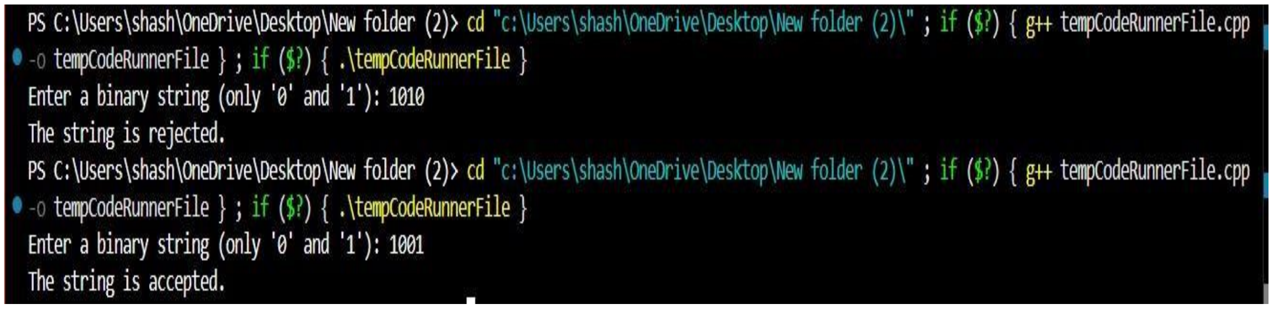
ALGORITHM:

1. A string that consists of a binary string followed by its reverse (e.g., 110011 where 110 is w and 011 is w^R).
2. Split the input into two halves.
3. Compare the first half with the reversed second half.

PROGRAM:

```
#include <stdio.h>
#include <string.h>

#define MAX_LENGTH 100

// Function to check if the binary string is of the form ww^R
int isPalindromeBinary(char *input) {
    int length = strlen(input);

    // Check if the length is even; if odd, it can't be in the form ww^R
    if (length % 2 != 0) {
        return 0; // Reject
    }

    int mid = length / 2;

    // Compare the first half with the reverse of the second half
    for (int i = 0; i < mid; i++) {
        if (input[i] != input[length - 1 - i]) {
            return 0; // Reject
        }
    }

    return 1; // Accept
}

int main() {
    char input[MAX_LENGTH];

    printf("Enter a binary string (only '0' and '1'): ");
    scanf("%s", input);

    if (isPalindromeBinary(input)) {
        printf("The string is accepted.\n");
    } else {
        printf("The string is rejected.\n");
    }

    return 0;
}
```

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a binary string (only '0' and '1'): 1010
The string is rejected.
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter a binary string (only '0' and '1'): 1001
The string is accepted.
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, the program for TM to accept W^R where w is any binary string and W^R is reverse of that string is executed successfully.

Ex. No: 10 Date:	**Design the travelling salesman problem for NP Complete Problem**

AIM:

Design the travelling salesman problem for NP complete problems.

ALGORITHM:

▪

1. Exact Algorithms:
   - Brute Force: Generate all possible permutations of cities and calculate the total distance for each permutation.
   - Dynamic Programming: Use dynamic programming with bitmasking (Held- Karp algorithm).
2. Approximation Algorithms:
   - Nearest Neighbor: Construct a tour by repeatedly visiting the nearest unvisited city.
   - Minimum Spanning Tree (MST): Use a minimum spanning tree to create an approximate solution.
3. Heuristics and Metaheuristics:
   - Genetic Algorithms: Use evolutionary techniques to evolve solutions.
   - Simulated Annealing: Use probabilistic techniques to explore the solution space.

For demonstration purposes, let's implement the Brute Force approach and the Dynamic Programming approach using C.

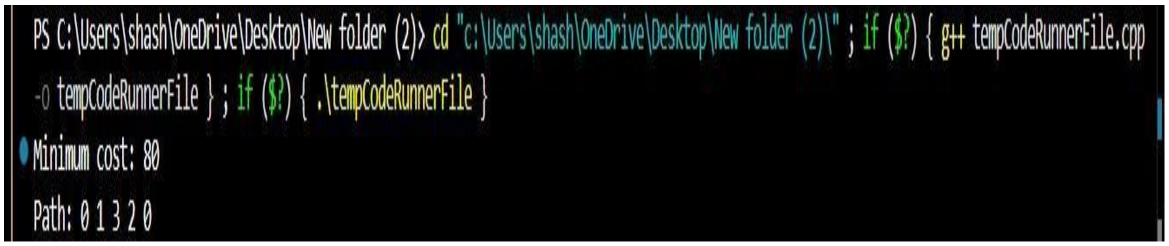
Brute Force Approach

The Brute Force approach is straightforward but computationally expensive, as it requires calculating all possible permutations of the cities.

PROGRAM:

```
#include <stdio.h>
#include <limits.h>

#define V 4 // Number of cities

// Function prototype (so compiler knows about it)
int next_permutation(int *array, int n);

int tsp(int graph[][V], int s) {
    int vertex[V - 1];
    int perm[V - 1];
    int i, j, min_path = INT_MAX, current_pathweight;

    // Store all vertices except the start vertex
    for (i = 0, j = 0; i < V; i++) {
        if (i != s) {
            vertex[j++] = i;
        }
    }

    // Permute all possible paths
    do {
        current_pathweight = 0;
        int k = s;

        for (i = 0; i <  V - 1; i++) {
            current_pathweight += graph[k][vertex[i]];
            k = vertex[i];
        }

        current_pathweight += graph[k][s]; // return to start

        if (current_pathweight < min_path) {
            min_path = current_pathweight;
            for (i = 0; i < V - 1; i++) {
                perm[i] = vertex[i];
            }
        }
    } while (next_permutation(vertex,  V - 1));

    printf("Minimum cost: %d\n", min_path);
    printf("Path: %d ", s);
    for (i = 0; i < V - 1; i++) {
        printf("%d ", perm[i]);
    }
    printf("%d\n",  s);
    return min_path;
}

int next_permutation(int *array, int n) {
    int i = n - 1;

    while (i > 0 && array[i - 1] >= array[i]) {
        i--;
    }

    if (i <= 0) {
        return 0;
    }

    int j = n - 1;
    while (array[j] <= array[i - 1]) {
        j--;
    }

    int temp = array[i - 1];
    array[i - 1] = array[j];
    array[j] = temp;

    j = n - 1;
    while (i < j) {
        temp = array[i];
        array[i] = array[j];
        array[j] = temp;
        i++;
        j--;
    }

    return 1;
}

int main() {
    int graph[V][V] = {
        {0, 10, 15, 20},
        {10, 0, 35, 25},
        {15, 35, 0, 30},
        {20, 25, 30, 0}
    };

    int s = 0; // Starting city
    tsp(graph, s);

    return 0;
}
```

OUTPUT:



```
PS C:\Users\shash\OneDrive\Desktop\New folder (2)> cd "c:\Users\shash\OneDrive\Desktop\New folder (2)\" ; if ($?) { g++ tempCodeRunnerFile.cpp
 -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Minimum cost: 80
Path: 0 1 3 2 0
```

Problem understanding and Design (3 marks)	
Code Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, the TSP for NP Complete problem minimum cost and path is executed uccessfully