THEORY OF COMPUTATION

PRCTICAL FILE

- **NAME – TUSHAR**
- **ROLL NUMBER – 22/28087**
- **COURSE – BSC(HONS)COMPUTER SCIENCE**
- **SEMESTER – 5TH**
- **SUBMITTED TO – DR. SHALINI AGGARWAL MADAM**

Q1. Design a Finite Automata (FA) that accepts all strings over S={0,1} having three consecutive 1's as a substring. Write a program to simulate this FA.

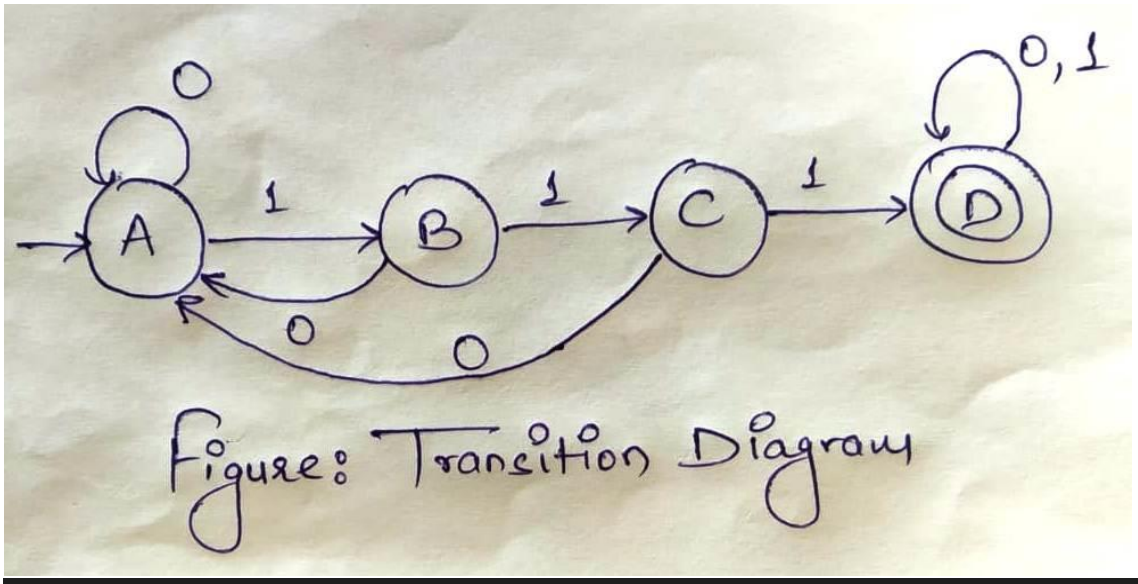


Figure: Transition Diagram

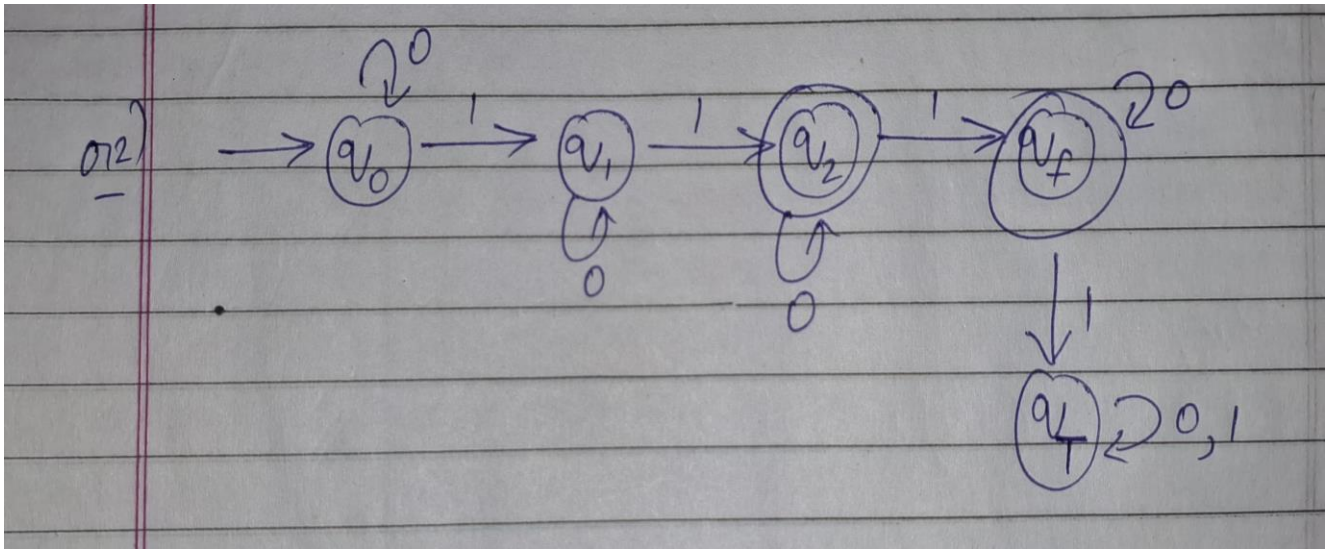
```cpp
#include <iostream>
#include <string>

using namespace std;

bool acceptsThreeConsecutiveOnes(const string& input) {
    int state = 0; // Start at state q0

    for (char ch : input) {
        switch (state) {
            case 0: // q0
                if (ch == '0') {
                    state = 0; // Stay in q0
                } else if (ch == '1') {
                    state = 1; // Move to q1
                }
                break;
            case 1: // q1
                if (ch == '0') {
                    state = 0; // Go back to q0
                } else if (ch == '1') {
                    state = 2; // Move to q2
                }
                break;
            case 2: // q2
                if (ch == '0') {
                    state = 0; // Go back to q0
                } else if (ch == '1') {
                    state = 3; // Move to q3 (accepting)
                }
                break;
            case 3: // q3 (accepting)
                // Remain in accepting state on '0' or '1'
                state = 3; // Stay in q3
                break;
        }
    }

    // Check if we are in an accepting state
    return (state == 3 || state == 4);
}
```

```
42
43  int main() {
44      string input;
45
46      cout << "Enter a binary string: ";
47      getline(cin, input);
48
49      if (acceptsThreeConsecutiveOnes(input)) {
50          cout << "Accepted: The string contains '111'." << endl;
51      } else {
52          cout << "Rejected: The string does not contain '111'." << endl;
53      }
54
55      return 0; // Indicating successful completion of the program
56  }
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desk
Enter a binary string: 011010010101
Rejected: The string does not contain '111'.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desk
Enter a binary string: 01111111111000010
Accepted: The string contains '111'.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desk
Enter a binary string: 110110110110
Rejected: The string does not contain '111'.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desk
Enter a binary string: 000011001000101001010100101110
Accepted: The string contains '111'.
```

Q2. Design a Finite Automata (FA) that accepts all strings over S={0,1} having either exactly two 1's or exactly three 1's, not more nor less. Write a program to simulate this FA.

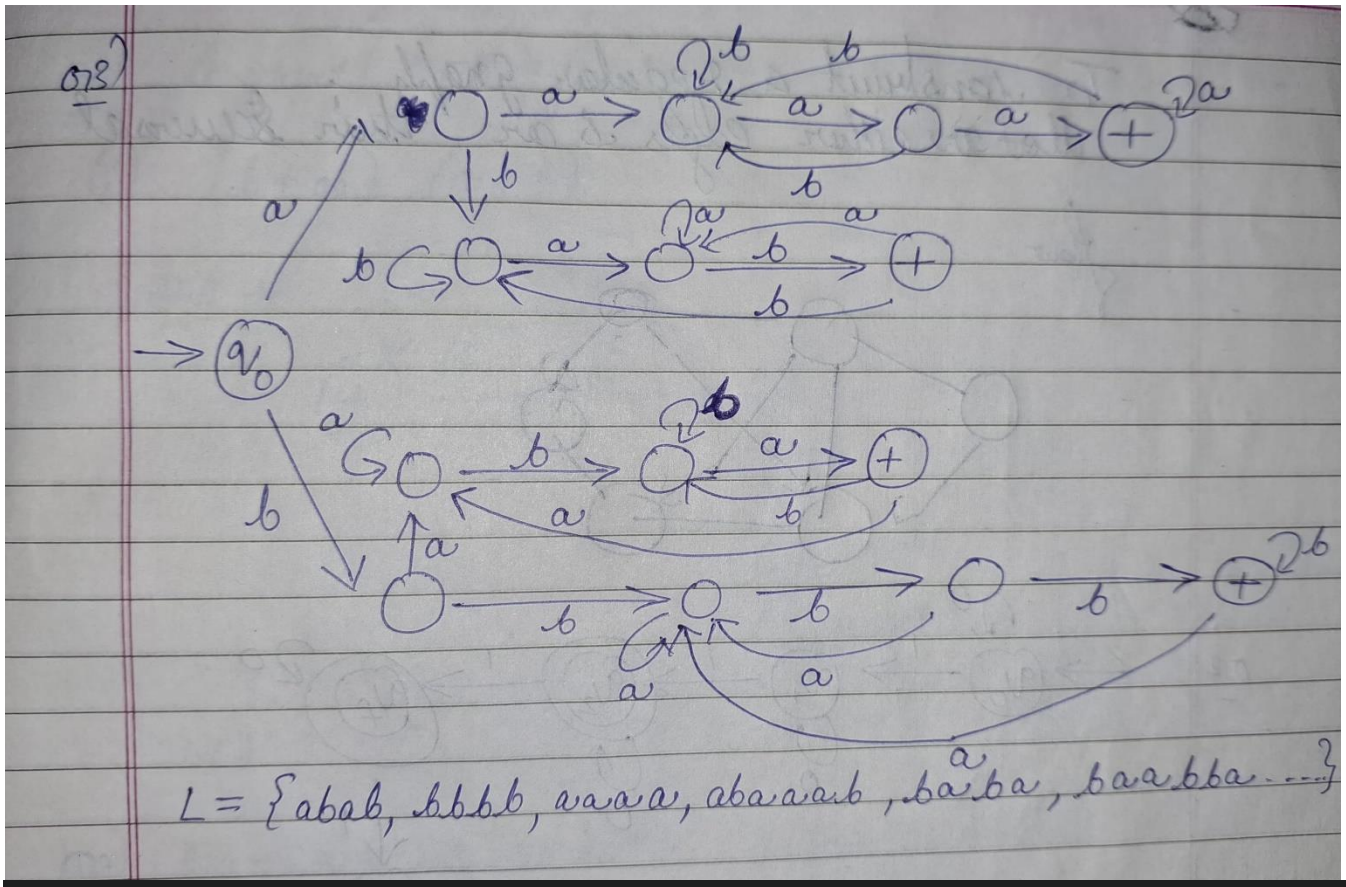


```cpp
#include <iostream>
#include <string>
using namespace std;
bool acceptsTwoOrThreeOnes(const string& input) {
    int state = 0; // Start at state q0
    int oneCount = 0; // Count of '1's

    for (char ch : input) {
        switch (state) {
            case 0: // q0
                if (ch == '0') {
                    state = 0; // Stay in q0
                } else if (ch == '1') {
                    state = 1; // Move to q1
                    oneCount++;
                }
                break;
            case 1: // q1
                if (ch == '0') {
                    state = 1; // Stay in q1
                } else if (ch == '1') {
                    state = 2; // Move to q2
                    oneCount++;
                }
                break;
            case 2: // q2
                if (ch == '0') {
                    state = 2; // Stay in q2
                } else if (ch == '1') {
                    state = 3; // Move to q3
                    oneCount++;
                }
                break;
            case 3: // q3 (accepting state)
                if (ch == '0') {
                    state = 3; // Stay in q3
                } else if (ch == '1') {
                    state = 4; // Move to q4 (dead state)
                }
                break;
            case 4: // q4 (dead state)
                // Stay in dead state on any input
                break;
        }
    }
    // The string is accepted if we end in q2 or q3
    return (state == 2 || state == 3);
}

int main() {
    string input;

    cout << "Enter a binary string: ";
    getline(cin, input);

    if (acceptsTwoOrThreeOnes(input)) {
        cout << "Accepted: The string contains exactly two or three '1's." << endl;
    } else {
        cout << "Rejected: The string does not contain exactly two or three '1's." << endl;
    }
    return 0; // Indicating successful completion of the program
}
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desktop
Enter a binary string: 01101010
Rejected: The string does not contain exactly two or three '1's.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desktop
Enter a binary string: 01100010
Accepted: The string contains exactly two or three '1's.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desktop
Enter a binary string: 100001000
Accepted: The string contains exactly two or three '1's.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desktop
Enter a binary string: 011111110001001001
Rejected: The string does not contain exactly two or three '1's.
```

Q3. Design a Finite Automata (FA) that accepts language L1, over S={a, b}, comprising of all strings (of length 4 or more) having first two characters same as the last two. Write a program to simulate this FA.

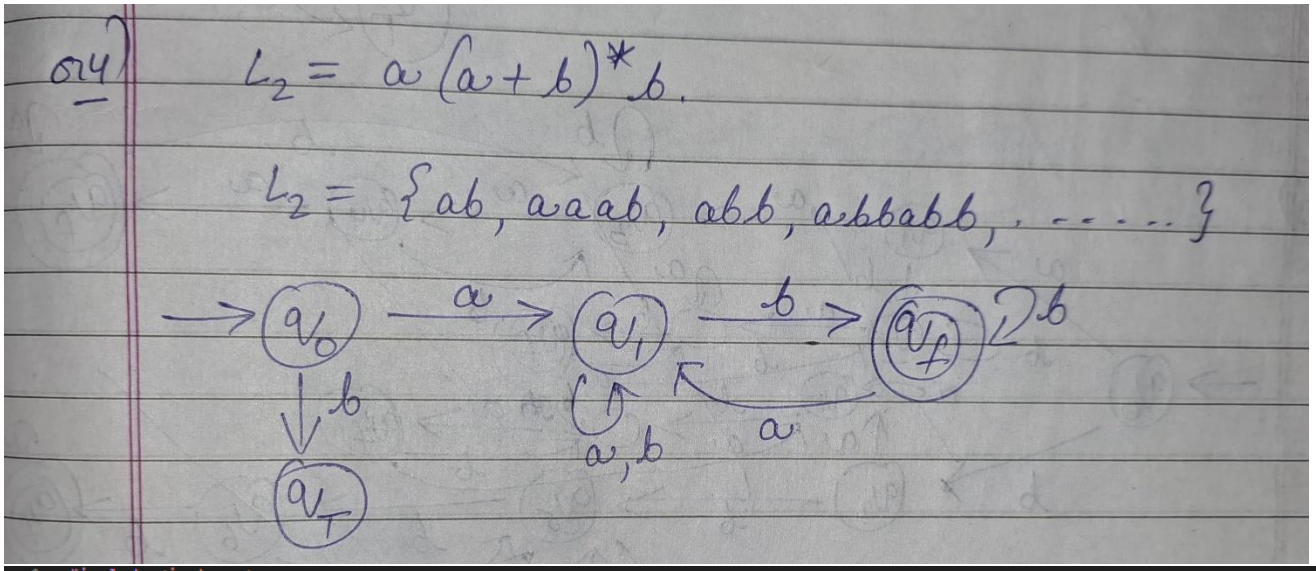



```cpp
#include <iostream>
#include <string>

bool acceptsL1(const std::string &input) {
    int length = input.length();
    // Check if length is less than 4, immediately reject
    if (length < 4) return false;

    // Check if the first two characters match the last two
    return (input[0] == input[length - 2] && input[1] == input[length - 1]);
}

int main() {
    std::string input;
    std::cout << "Enter a string over {a, b}: ";
    std::cin >> input;

    if (acceptsL1(input)) {
        std::cout << "The string is accepted by the FA.\n";
    } else {
        std::cout << "The string is rejected by the FA.\n";
    }

    return 0;
}
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Deskt
Enter a string over {a, b}: abababbababbabbbbbabababab
The string is accepted by the FA.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Deskt
Enter a string over {a, b}: baba
The string is accepted by the FA.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Deskt
Enter a string over {a, b}: bbbb
The string is accepted by the FA.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Deskt
Enter a string over {a, b}: aaaaaa
The string is accepted by the FA.
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Deskt
Enter a string over {a, b}: babbbabbabbbbbbbabbbabb
The string is rejected by the FA.
```

Q4. Design a Finite Automata (FA) that accepts language L2, over S= {a, b} where L2= a(a+b)*b. Write a program to simulate this FA.



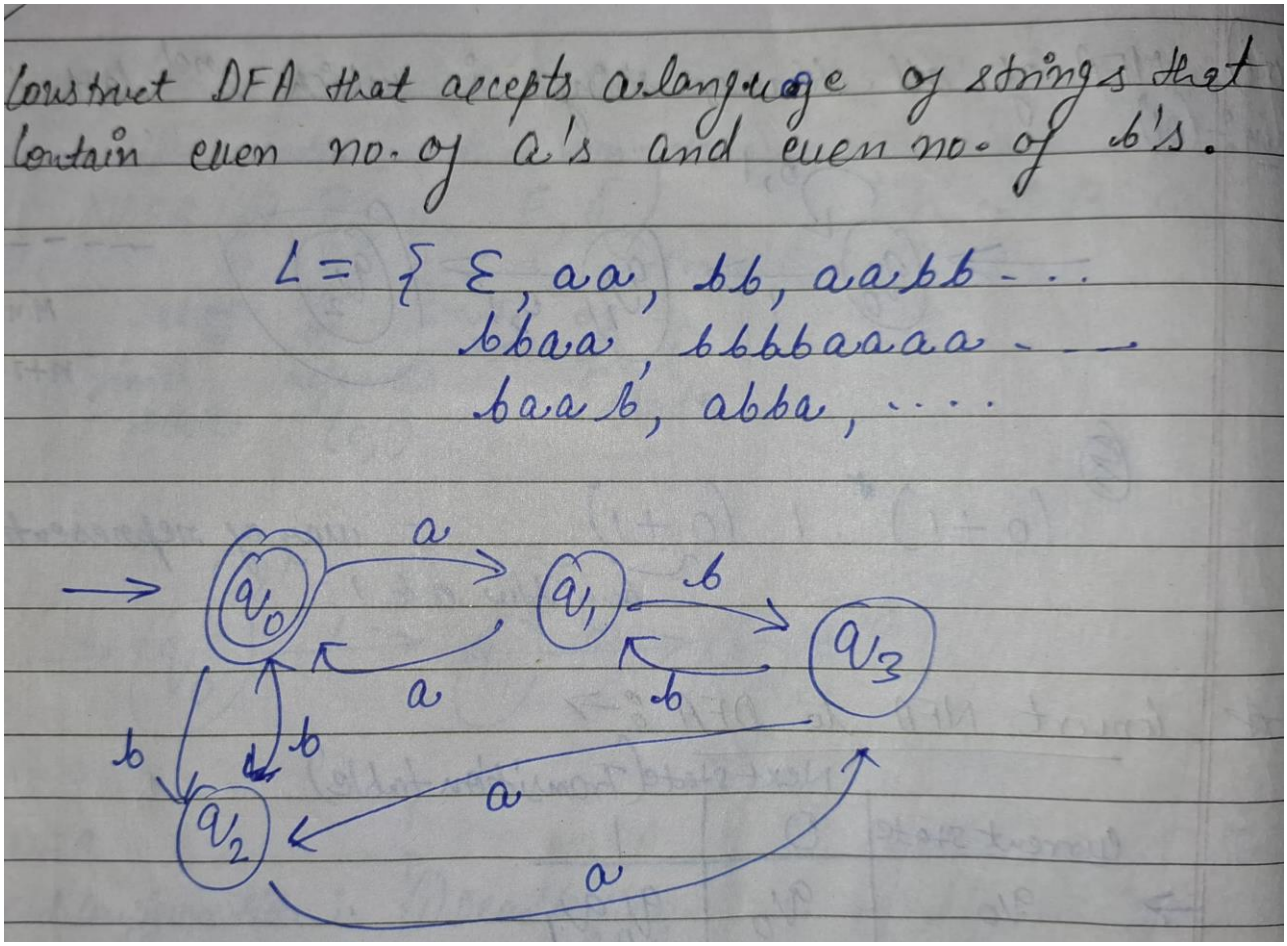

```cpp
#include <iostream>
#include <string>
using namespace std;
// Function to simulate the FA
bool simulateFA(const string &input) {
    // Define states
    enum State { q0, q1, q2 };
    State currentState = q0;

    // Process each character in the input string
    for (char ch : input) {
        switch (currentState) {
            case q0:
                if (ch == 'a') {
                    currentState = q1;  // Move to q1 on initial 'a'
                } else {
                    return false;  // Reject if it doesn't start with 'a'
                }
                break;

            case q1:
                if (ch == 'a') {
                    currentState = q1;  // Stay in q1 on 'a'
                } else if (ch == 'b') {
                    currentState = q2;  // Move to q2 on 'b'
                }
                break;

            case q2:
                if (ch == 'a') {
                    currentState = q1;  // Return to q1 on 'a'
                } else if (ch == 'b') {
                    currentState = q2;  // Stay in q2 on 'b'
                }
```

```
                break;
            }
        }
        // Accept if the final state is q2
        return (currentState == q2);
    }
    int main() {
        // Test cases
        string testStrings[] = { "ab", "aab", "abb", "aaabb", "abab", "abba", "a", "b" , "ababbab" , "abaab","aababab" };
        for (const string &test : testStrings) {
            cout << "Input: " << test << " => "
                 << (simulateFA(test) ? "Accepted" : "Rejected") << endl;
        }
        return 0;
    }
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desktop\CODING\c++\Th
Input: ab => Accepted
Input: aab => Accepted
Input: abb => Accepted
Input: aaabb => Accepted
Input: abab => Accepted
Input: abba => Rejected
Input: a => Rejected
Input: b => Rejected
Input: ababbab => Accepted
Input: abaab => Accepted
Input: aababab => Accepted
```

Q5. Design a Finite Automata (FA) that accepts language EVEN-EVEN over S={a, b}. Write a program to simulate this FA.



```cpp
#include <iostream>
#include <string>

using namespace std;

// Function to simulate the FA for EVEN-EVEN
bool simulateEvenEvenFA(const string &input) {
    // Define states
    enum State { q_ee, q_eo, q_oe, q_oo };
    State currentState = q_ee;

    // Process each character in the input string
    for (char ch : input) {
        switch (currentState) {
            case q_ee:
                if (ch == 'a') {
                    currentState = q_oe;
                } else if (ch == 'b') {
                    currentState = q_eo;
                }
                break;

            case q_eo:
                if (ch == 'a') {
                    currentState = q_oo;
                } else if (ch == 'b') {
                    currentState = q_ee;
                }
                break;

            case q_oe:
                if (ch == 'a') {
                    currentState = q_ee;
                } else if (ch == 'b') {
                    currentState = q_oo;
                }
                break;

            case q_oo:
                if (ch == 'a') {
                    currentState = q_eo;
                } else if (ch == 'b') {
                    currentState = q_oe;
                }
                break;
        }
    }

    // Accept if the final state is q_ee
    return (currentState == q_ee);
}

int main() {
    // Test cases
    string testStrings[] = { "", "aa", "bb", "aabb", "abab", "abba", "a", "b", "ab", "ba", "bbb", "aaabbbb"};
    for (const string &test : testStrings) {
        cout << "Input: " << test << " => "
             << (simulateEvenEvenFA(test) ? "Accepted" : "Rejected") << endl;
    }
    return 0;
}
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Desktop\CODING\c++\Theo
Input:  => Accepted
Input: aa => Accepted
Input: bb => Accepted
Input: aabb => Accepted
Input: abab => Accepted
Input: abba => Accepted
Input: a => Rejected
Input: b => Rejected
Input: ab => Rejected
Input: ba => Rejected
Input: bbb => Rejected
Input: aaabbbb => Rejected
```

Q6. Write a program to simulate an FA that accepts

A. Union of the languages L1 and L2.

```cpp
#include <iostream>
#include <string>

using namespace std;

// FA for L1: Accepts strings that start with 'a' and end with 'b'
bool simulateFA_L1(const string &input) {
    if (input.length() < 2) return false; // Minimum length for "a...b" is 2
    return (input[0] == 'a' && input[input.length() - 1] == 'b');
}

// FA for L2: Accepts strings that contain "aa" as a substring
bool simulateFA_L2(const string &input) {
    for (size_t i = 0; i < input.length() - 1; ++i) {
        if (input[i] == 'a' && input[i + 1] == 'a') {
            return true;
        }
    }
    return false;
}

// Union of L1 and L2: Accept if either L1 or L2 accepts the string
bool simulateFA_Union(const string &input) {
    return simulateFA_L1(input) || simulateFA_L2(input);
}

int main() {
    // Test cases
    string testStrings[] = { "ab", "aab", "aa", "bbaa", "ba", "aabb", "aaab", "bb","aaaaab" };
    for (const string &test : testStrings) {
        cout << "Input: " << test << " => "
             << (simulateFA_Union(test) ? "Accepted" : "Rejected") << endl;
    }
    return 0;
}
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\D
Input: ab => Accepted
Input: aab => Accepted
Input: aa => Accepted
Input: bbaa => Accepted
Input: ba => Rejected
Input: aabb => Accepted
Input: aaab => Accepted
Input: bb => Rejected
Input: aaaaab => Accepted
```

B. Intersection of the languages L1 and L2

```cpp
#include <iostream>
#include <string>

using namespace std;

// FA for L1: Accepts strings that start with 'a' and end with 'b'
bool simulateFA_L1(const string &input) {
    if (input.length() < 2) return false; // Minimum length for "a...b" is 2
    return (input[0] == 'a' && input[input.length() - 1] == 'b');
}

// FA for L2: Accepts strings that contain "aa" as a substring
bool simulateFA_L2(const string &input) {
    for (size_t i = 0; i < input.length() - 1; ++i) {
        if (input[i] == 'a' && input[i + 1] == 'a') {
            return true;
        }
    }
    return false;
}

// Intersection of L1 and L2: Accept if both L1 and L2 accept the string
bool simulateFA_Intersection(const string &input) {
    return simulateFA_L1(input) && simulateFA_L2(input);
}

int main() {
    // Test cases
    string testStrings[] = { "ab", "aab", "aa", "aabb", "aaab", "aba", "ba", "bbaa" , "abababbbb" , "abaababbbbbbbab" };
    for (const string &test : testStrings) {
        cout << "Input: " << test << " => "
             << (simulateFA_Intersection(test) ? "Accepted" : "Rejected") << endl;
    }
    return 0;
}
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tu
Input: ab => Rejected
Input: aab => Accepted
Input: aa => Rejected
Input: aabb => Accepted
Input: aaab => Accepted
Input: aba => Rejected
Input: ba => Rejected
Input: bbaa => Rejected
Input: abababbbb => Rejected
Input: abaababbbbbbbab => Accepted
```

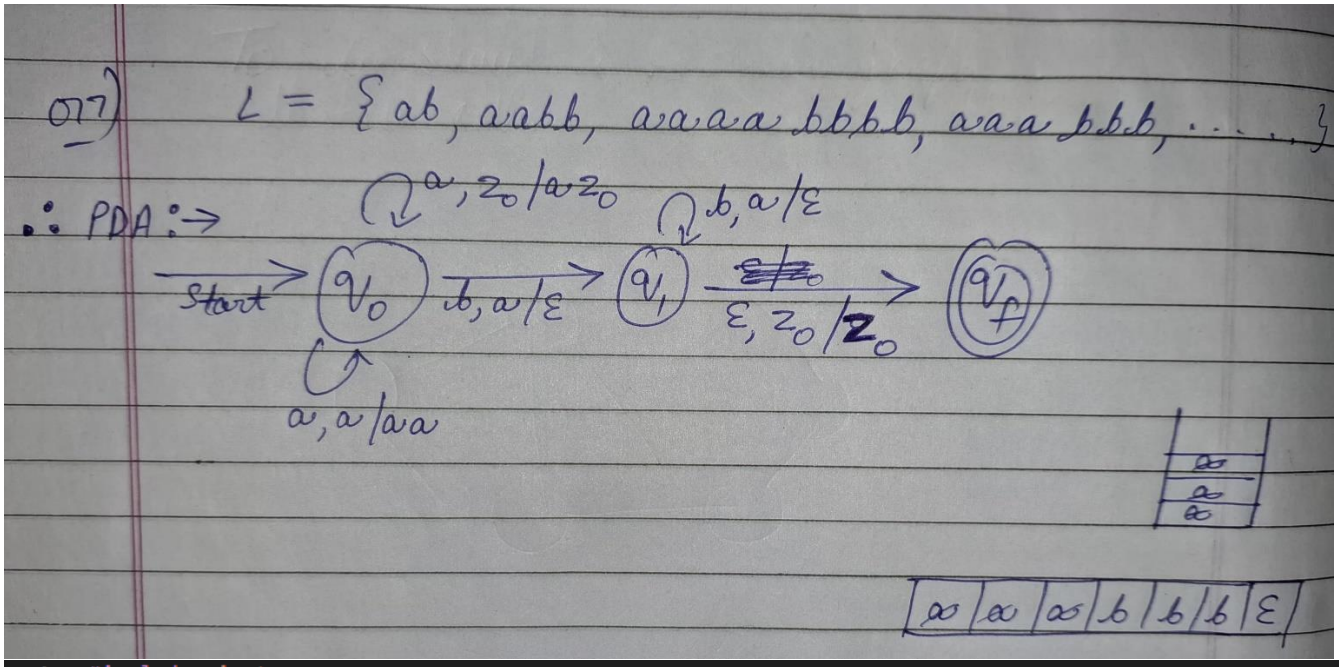
C. Language L1 L2 (concatenation)

```cpp
#include <iostream>
#include <string>

using namespace std;

// FA for L1: Accepts strings that start with 'a' and end with 'b'
bool simulateFA_L1(const string &input) {
    if (input.length() < 2) return false; // Minimum length for "a...b" is 2
    return (input[0] == 'a' && input[input.length() - 1] == 'b');
}

// FA for L2: Accepts strings that contain "aa" as a substring
bool simulateFA_L2(const string &input) {
    for (size_t i = 0; i < input.length() - 1; ++i) {
        if (input[i] == 'a' && input[i + 1] == 'a') {
            return true;
        }
    }
    return false;
}

// Concatenation of L1 and L2: Accept if input can be split such that
// the first part is accepted by L1 and the second part is accepted by L2
bool simulateFA_Concatenation(const string &input) {
    for (size_t i = 1; i < input.length(); ++i) {
        string part1 = input.substr(0, i);    // First part for L1
        string part2 = input.substr(i);       // Second part for L2
        if (simulateFA_L1(part1) && simulateFA_L2(part2)) {
            return true;
        }
    }
    return false;
}

int main() {
    // Test cases
    string testStrings[] = { "ab", "aab", "aabaa", "aabba", "aabaaa", "aaab", "abb", "aaabb", "abbabbbabaab" , "aabbbab"};
    for (const string &test : testStrings) {
        cout << "Input: " << test << " => "
             << (simulateFA_Concatenation(test) ? "Accepted" : "Rejected") << endl;
    }
    return 0;
}
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\
Input: ab => Rejected
Input: aab => Rejected
Input: aabaa => Accepted
Input: aabba => Rejected
Input: aabaaa => Accepted
Input: aaab => Rejected
Input: abb => Rejected
Input: aaabb => Rejected
Input: abbabbbabaab => Accepted
Input: aabbbab => Rejected
```

Q7. Design a PDA and write a program for simulating the machine which accepts the language {a^n b^n where n>0, S= {a, b}}.

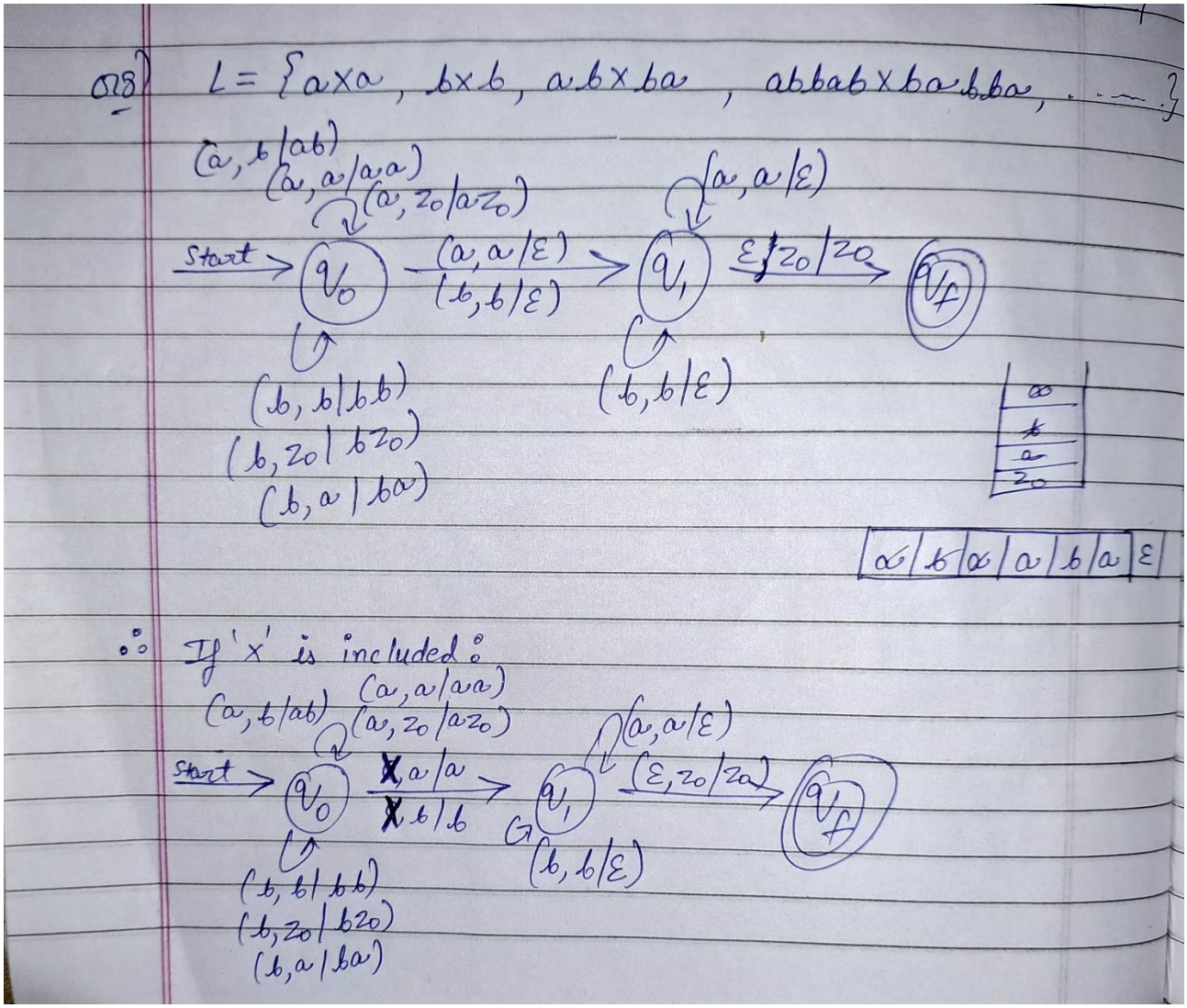



```cpp
#include <iostream>
#include <stack>
#include <string>

using namespace std;

// Function to simulate the PDA
bool simulatePDA(const string &input) {
    stack<char> pdaStack;

    // Step 1: Push 'A' onto the stack for each 'a' in the input
    size_t i = 0;
    while (i < input.length() && input[i] == 'a') {
        pdaStack.push('A');  // Push a marker for each 'a'
        i++;
    }

    // Step 2: Pop 'A' from the stack for each 'b' in the input
    while (i < input.length() && input[i] == 'b') {
        if (pdaStack.empty()) {
            return false; // More 'b's than 'a's, so reject
        }
        pdaStack.pop(); // Pop a marker for each 'b'
        i++;
    }

    // Step 3: Check if the stack is empty and we have processed the entire input
    return (pdaStack.empty() && i == input.length());
}

int main() {
    // Test cases
    string testStrings[] = { "ab", "aabb", "aaabbb", "aaaabbbb", "aaabb", "aabbb", "abbb", "a" };
    for (const string &test : testStrings) {
        cout << "Input: " << test << " => "
             << (simulatePDA(test) ? "Accepted" : "Rejected") << endl;
    }
    return 0;
}
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha
Input: ab => Accepted
Input: aabb => Accepted
Input: aaabbb => Accepted
Input: aaaabbbb => Accepted
Input: aaabb => Rejected
Input: aabbb => Rejected
Input: abbb => Rejected
Input: a => Rejected
```

Q8. Design a PDA and write a program for simulating the machine which accepts the language {wXwr| w is any string over S = {a, b} and wr is reverse of that string and X is a special symbol }.

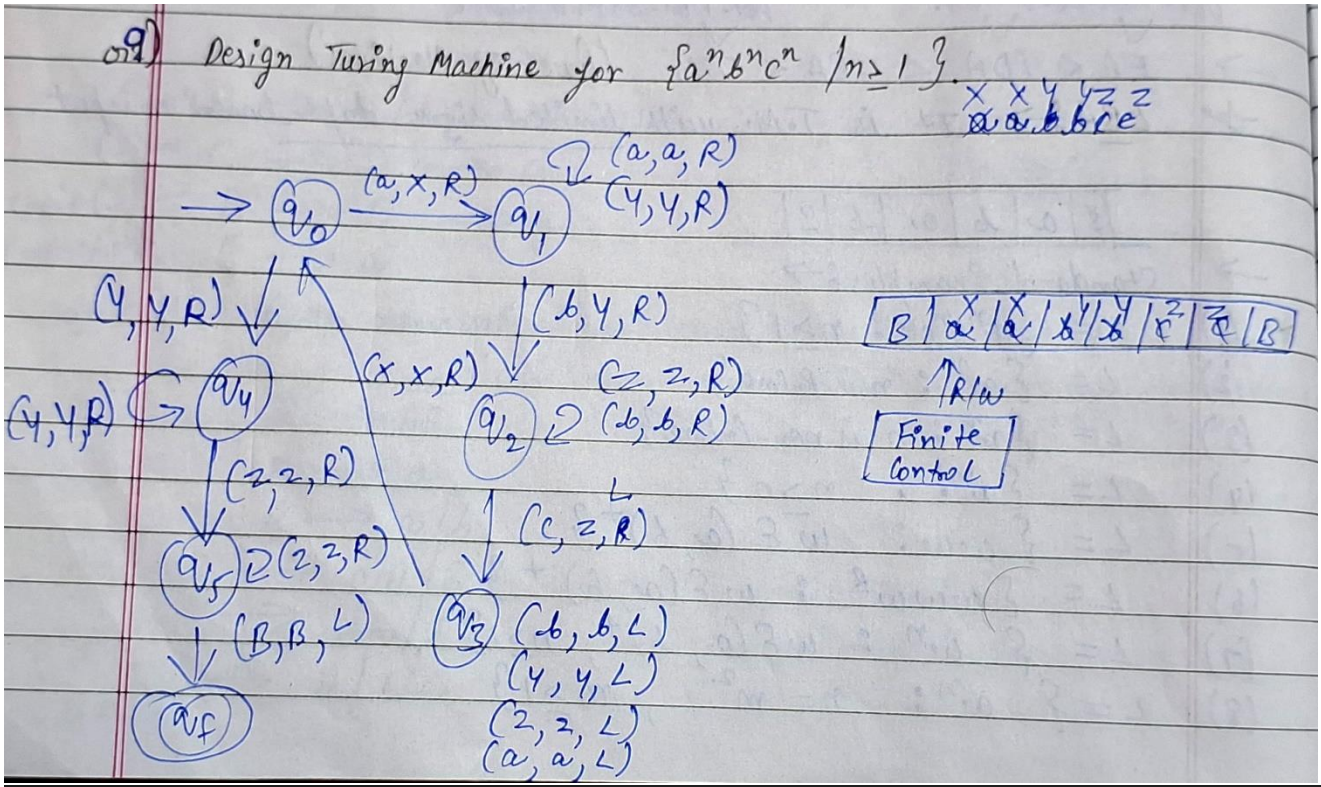


```cpp
#include <iostream>
#include <stack>
#include <string>

using namespace std;

// Function to simulate the PDA
bool simulatePDA(const string &input) {
    stack<char> pdaStack;
    size_t i = 0;

    // Step 1: Push characters of `w` onto the stack until we reach 'X'
    while (i < input.length() && input[i] != 'X') {
        pdaStack.push(input[i]);
        i++;
    }

    // Step 2: Check for the presence of 'X' in the middle
    if (i == input.length() || input[i] != 'X') {
        return false; // Reject if there's no 'X' separating `w` and `w^r`
    }

    // Skip the 'X' character
    i++;

    // Step 3: Pop from the stack and match with `w^r`
    while (i < input.length()) {
        if (pdaStack.empty() || pdaStack.top() != input[i]) {
            return false; // Mismatch or stack empty before finishing `w^r`
        }
        pdaStack.pop();
        i++;
    }

    // Step 4: Accept if the stack is empty after processing all input
    return pdaStack.empty();
}

int main() {
    // Test cases
    string testStrings[] = { "aXa", "abbaXabba", "abXba", "abbXaab", "bXb", "aabXbaa", "aaXaa" , "aaabbaXabbaaa"};
    for (const string &test : testStrings) {
        cout << "Input: " << test << " => "
             << (simulatePDA(test) ? "Accepted" : "Rejected") << endl;
    }
    return 0;
}
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tu
Input: aXa => Accepted
Input: abbaXabba => Accepted
Input: abXba => Accepted
Input: abbXaab => Rejected
Input: bXb => Accepted
Input: aabXbaa => Accepted
Input: aaXaa => Accepted
Input: aaabbaXabbaaa => Accepted
```

Q9. Design and simulate a Turing Machine that accepts the language a^n b^n c^n where n >0.



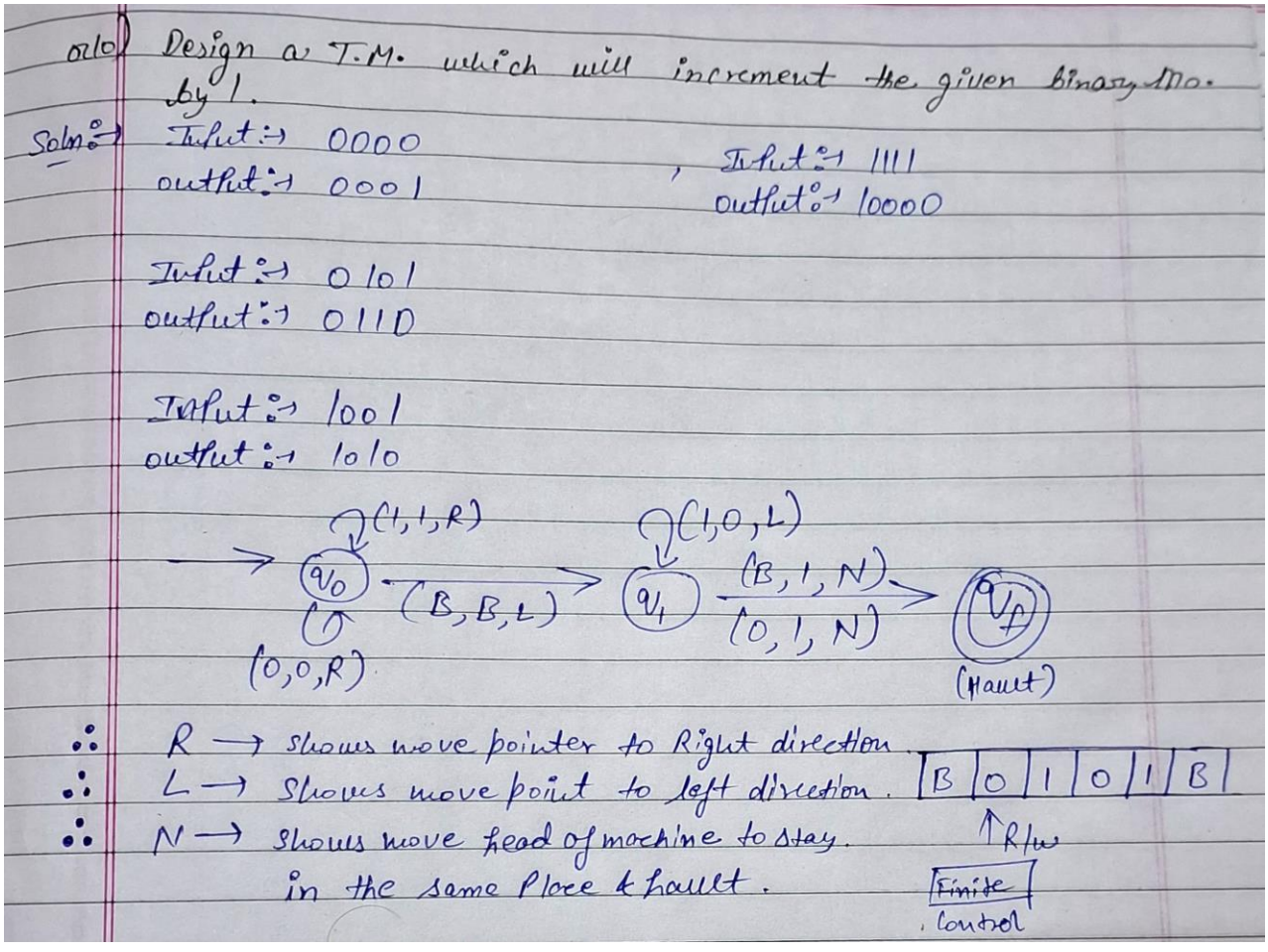

```cpp
#include <iostream>
#include <string>
bool simulateTM(std::string tape) {
    int head = 0;  // Start at the beginning of the tape

    while (true) {
        // Find the first unmarked 'a' and mark it as 'X'
        head = tape.find('a');
        if (head != std::string::npos) {
            tape[head] = 'X';
        } else {
            // If there are no more 'a's, we should also have no 'b's or 'c's left
            if (tape.find('b') == std::string::npos && tape.find('c') == std::string::npos) {
                return true;  // Accepted
            } else {
                return false; // Rejected due to mismatched count
            }
        }

        // Find the first unmarked 'b' after marking 'a' and mark it as 'Y'
        head = tape.find('b', head);
        if (head != std::string::npos) {
            tape[head] = 'Y';
        } else {
            return false; // Rejected if no matching 'b' is found for 'a'
        }

        // Find the first unmarked 'c' after marking 'b' and mark it as 'Z'
        head = tape.find('c', head);
        if (head != std::string::npos) {
            tape[head] = 'Z';
        } else {
            return false; // Rejected if no matching 'c' is found for 'a' and 'b'
        }
    }
}

```

```cpp
int main() {
    std::string input;
    std::cout << "Enter a string: ";
    std::cin >> input;

    if (simulateTM(input)) {
        std::cout << "Accepted" << std::endl;
    } else {
        std::cout << "Rejected" << std::endl;
    }

    return 0;
}
```

TERMINAL OUTPUT:

```
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha
Enter a string: abbbbbbccccc
Rejected
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha
Enter a string: abc
Accepted
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha
Enter a string: aaabbbccc
Accepted
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha
Enter a string: aabbbaaabbbac
Rejected
```

Q10. Design and simulate a Turing Machine which will increment the given binary number by 1.



```cpp
#include <iostream>
#include <string>
#include <algorithm>
class TuringMachine {
public:
    TuringMachine(const std::string& input) : tape(input), head(0) {}

    void increment() {
        // Move to the end of the binary number
        head = tape.size() - 1;

        // State: Starting from the rightmost bit
        while (head >= 0) {
            if (tape[head] == '0') {
                // Change '0' to '1' and we're done
                tape[head] = '1';
                return;
            } else {
                // Change '1' to '0' and carry over
                tape[head] = '0';
                head--;
            }
        }
        // If we carried out from the leftmost bit, we need to add '1' at the start
        tape = '1' + tape;
    }

    std::string getResult() const {
        return tape;
    }

private:
    std::string tape;  // The tape with the binary number
    int head;          // The position of the head
};

int main() {
    std::string binaryNumber;
    std::cout << "Enter a binary number: ";
    std::cin >> binaryNumber;

    // Ensure the input is a valid binary number
    if (binaryNumber.find_first_not_of("01") != std::string::npos) {
        std::cerr << "Invalid binary number!" << std::endl;
        return 1;
    }

    TuringMachine tm(binaryNumber);
    tm.increment();
    std::cout << "Incremented binary number: " << tm.getResult() << std::endl;
    return 0;
}
```

TERMINAL OUTPUT:

```
Enter a binary number: 1101
Incremented binary number: 1110
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Deskt
}
Enter a binary number: 0
Incremented binary number: 1
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Deskt
}
Enter a binary number: 111
Incremented binary number: 1000
PS C:\Users\tusha\OneDrive\Desktop\CODING\c++\Theory_of_computation> cd "c:\Users\tusha\OneDrive\Deskt
}
Enter a binary number: 0101
Incremented binary number: 0110
```