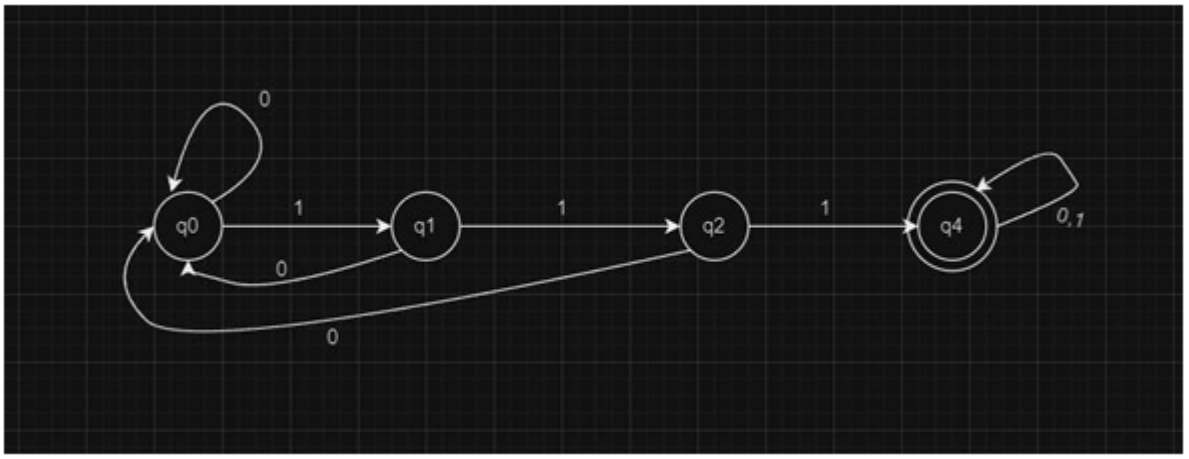
Theory Of Computation

Practical File

Submitted to:
Mr. Ravi Kumar Yadav

Submitted by:
Vinay Ruhil
BSc(H) CS (B)
16115

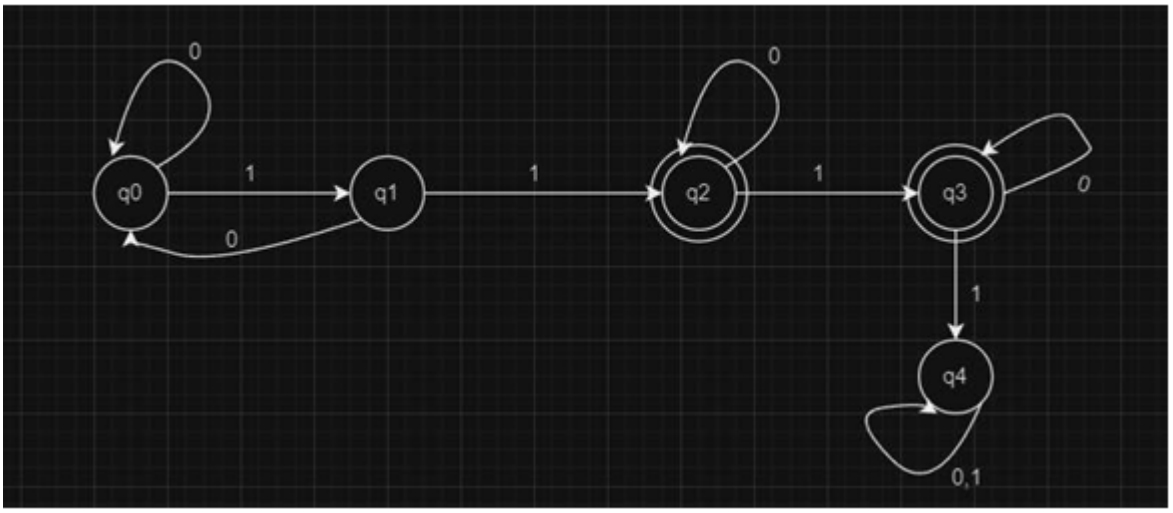
1. Design a Finite Automata (FA) that accepts all strings over S={0, 1} having three consecutive 1's as a substring. Write a program to simulate this FA.




```
#include <iostream>
 #include <string>
 using namespace std;
 // Function to simulate FA that accepts strings with three consecutive '1's
 bool simulate_FA(const string& input_string) {
    for (size_t i = 0; i < input_string.length() - 2; ++i) {
        if (input_string[i] == '1' && input_string[i+1] == '1' && input_string[i+2] == '1') {
            return true;
        }
    }
    return false;
 }
 int main() {
    // Test strings
    string test_strings[] = {"111", "011", "101", "111111", "010101", "10001"};
    cout << "Strings with three consecutive 1's:" << endl;
    for (const string& test : test_strings) {
        cout << "Input: " << test << " => " << (simulate_FA(test) ? "Accepted" : "Rejected") <<
endl;
    }
    return 0;}
```

```
Output                                                    Clear
Strings with three consecutive 1's:
Input: 111 => Accepted
Input: 011 => Rejected
Input: 101 => Rejected
Input: 111111 => Accepted
Input: 010101 => Rejected
Input: 10001 => Rejected


=== Code Execution Successful ===
```

2. 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.

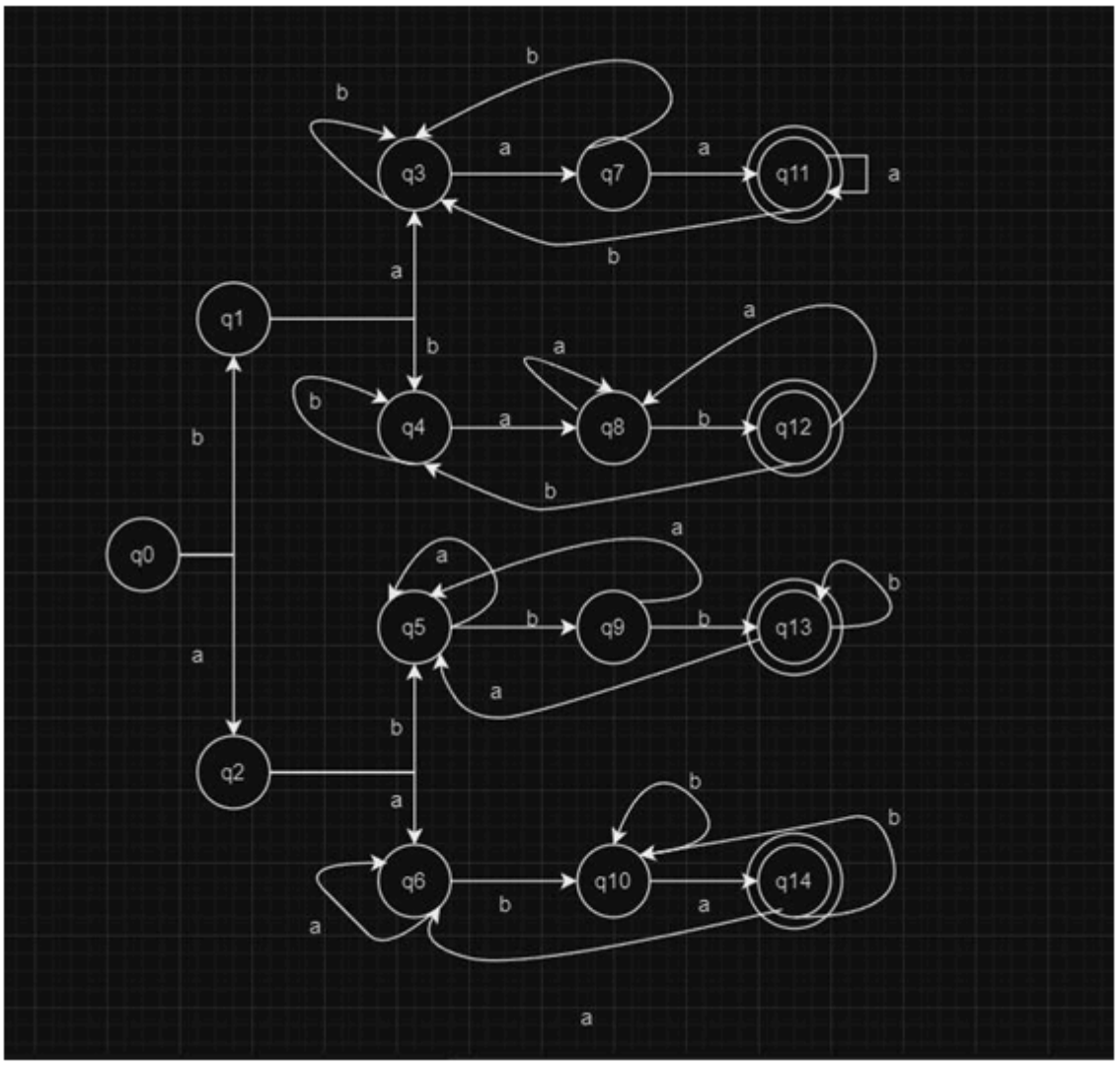


```
#include <iostream>
 #include <string>
 using namespace std;
 // Function to simulate FA for strings having exactly two or exactly three 1's
 bool simulate_FA(const string& input_string) {
 int count = 0;
 for (char c : input_string) {
       if (c == '1') {
          count++;
       }
   }
   return (count == 2 || count == 3);
 }
 int main() {
   // Test strings
   string test_strings[] = {"110", "111", "10101", "10001", "111111", "00101"};
   cout << "Strings with exactly two or exactly three 1's:" << endl;
   for (const string& test : test_strings) {
      cout << "Input: " << test << " => " << (simulate_FA(test) ? "Accepted" :
"Rejected") << endl;
   }
   return 0;
 }
```

Output Clear

```
Strings with exactly two or exactly three 1's:
Input: 110 => Accepted
Input: 111 => Accepted
Input: 10101 => Accepted
Input: 10001 => Accepted
Input: 111111 => Rejected
Input: 00101 => Accepted


=== Code Execution Successful ===
```

3. 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.



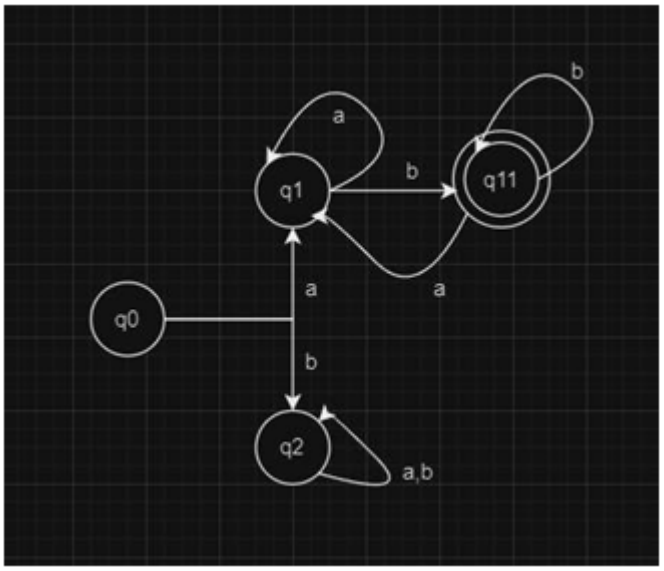

```
#include <iostream>
#include <string>
using namespace std;

// Function to simulate FA for strings where the first two characters are the same as the last two
bool simulate_FA(const string& input_string) {
    if (input_string.length() < 4) {
        return false; // Rejected if length is less than 4
    }
```

```
    return (input_string[0] == input_string[input_string.length() - 2] &&
            input_string[1] == input_string[input_string.length() - 1]);
}

int main() {
    // Test strings
    string test_strings[] = {"aabb", "abab", "baba", "abccba", "abcd", "bbaa"};

    cout << "Strings where the first two characters are the same as the last two:" << endl;

    for (const string& test : test_strings) {
        cout << "Input: " << test << " => " << (simulate_FA(test) ? "Accepted" : "Rejected") <<
endl;
    }

    return 0;
}
```

Output Clear

```
Strings where the first two characters are the same as the last two:
Input: aabb => Rejected
Input: abab => Accepted
Input: baba => Accepted
Input: abccba => Rejected
Input: abcd => Rejected
Input: bbaa => Rejected


=== Code Execution Successful ===
```

4. 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.




```
#include <iostream>
#include <string>
using namespace std;

// Function to simulate FA for strings of the form a(a+b)*b
bool simulate_FA(const string& input_string) {
    if (input_string.empty()) {
        return false; // Reject empty strings
    }
    // Check if the string starts with 'a' and ends with 'b'
    return (input_string[0] == 'a' && input_string[input_string.length() - 1] == 'b');
}

int main() {
    // Test strings
    string test_strings[] = {"ab", "aabb", "abab", "b", "aaab", "bb", "abc"};

    cout << "Strings of the form a(a+b)*b:" << endl;

    for (const string& test : test_strings) {
        cout << "Input: " << test << " => " << (simulate_FA(test) ? "Accepted" : "Rejected") <<
endl;
    }

    return 0;
```

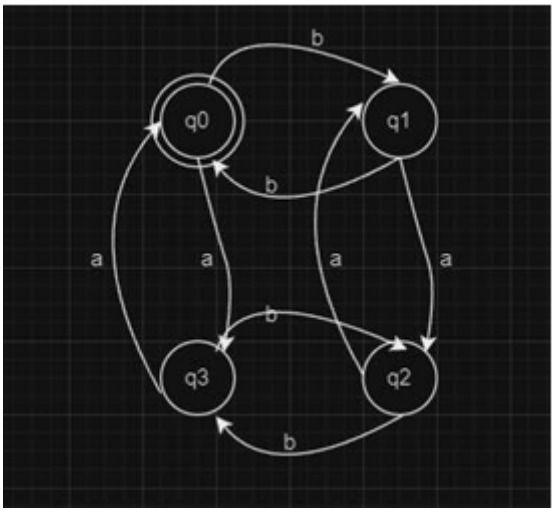
```
}
```

Output

Clear

```
Strings of the form a(a+b)*b:
Input: ab => Accepted
Input: aabb => Accepted
Input: abab => Accepted
Input: b => Rejected
Input: aaab => Accepted
Input: bb => Rejected
Input: abc => Rejected


=== Code Execution Successful ===
```

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




```
#include <iostream>
#include <string>
using namespace std;

// Function to simulate FA for strings with an even number of 'a's and an even number of 'b's
bool simulate_FA(const string& input_string) {
    int count_a = 0, count_b = 0;

    for (char c : input_string) {
        if (c == 'a') {
            count_a++;
        } else if (c == 'b') {
            count_b++;
        }
    }

    // Accepted if both counts are even
    return (count_a % 2 == 0 && count_b % 2 == 0);
}

int main() {
    // Test strings
    string test_strings[] = {"ab", "aabb", "bbaa", "aab", "abab", "aaa", "bb"};

    cout << "Strings with even number of 'a's and even number of 'b's:" << endl;

    for (const string& test : test_strings) {
```

```
        cout << "Input: " << test << " => " << (simulate_FA(test) ? "Accepted" : "Rejected") <<
endl;
    }

    return 0;
}
```

Output

Clear

```
Strings with even number of 'a's and even number of 'b's:
Input: ab => Rejected
Input: aabb => Accepted
Input: bbaa => Accepted
Input: aab => Rejected
Input: abab => Accepted
Input: aaa => Rejected
Input: bb => Accepted


=== Code Execution Successful ===
```

6. Write a program to simulate an FA that accepts
a. Union of the languages L1 and L2
b. Intersection of the languages L1 and L2
c. Language L1 L2 (concatenation).

```
#include <iostream>
#include <string>
using namespace std;

// Function to simulate FA for Language L1 (starts with 'a' and ends with 'b')
bool simulate_L1(const string& input_string) {
    if (input_string.length() < 2) {
        return false; // Rejected if string length is less than 2
    }
    return (input_string[0] == 'a' && input_string[input_string.length() - 1] == 'b');
}

// Function to simulate FA for Language L2 (contains at least one 'a')
bool simulate_L2(const string& input_string) {
    return (input_string.find('a') != string::npos); // 'a' is present in the string
}

// Function to simulate Union of L1 and L2
bool simulate_union(const string& input_string) {
    return simulate_L1(input_string) || simulate_L2(input_string); // Accepted if either L1 or L2
accepts the string
}

// Function to simulate Intersection of L1 and L2
bool simulate_intersection(const string& input_string) {
    return simulate_L1(input_string) && simulate_L2(input_string); // Accepted if both L1 and
L2 accept the string
}

// Function to simulate Concatenation of L1 and L2
bool simulate_concatenation(const string& input_string) {
    for (size_t i = 1; i < input_string.length(); i++) {
        string part1 = input_string.substr(0, i);
        string part2 = input_string.substr(i);

        // Check if part1 is accepted by L1 and part2 is accepted by L2
        if (simulate_L1(part1) && simulate_L2(part2)) {
            return true; // Accepted if concatenation of valid L1 and L2 parts
        }
    }
    return false; // Rejected if no valid split is found
```

```
}

int main() {
    // Test strings
    string test_strings[] = {"ab", "aab", "ba",  "a", "baa", "bb"};

    cout << "Union of L1 and L2:" << endl;
    for (const string& test : test_strings) {
        cout << "Input: " << test << " => " << (simulate_union(test) ? "Accepted" : "Rejected")
<< endl;
    }

    cout << "\nIntersection of L1 and L2:" << endl;
    for (const string& test : test_strings) {
        cout << "Input: " << test << " => " << (simulate_intersection(test) ? "Accepted" :
"Rejected") << endl;
    }

    cout << "\nConcatenation of L1 and L2:" << endl;
    for (const string& test : test_strings) {
        cout << "Input: " << test << " => " << (simulate_concatenation(test) ? "Accepted" :
"Rejected") << endl;
    }

    return 0;
}
```

Output

Clear

```
Union of L1 and L2:
Input: ab => Accepted
Input: aab => Accepted
Input: ba => Accepted
Input: a => Accepted
Input: baa => Accepted
Input: bb => Rejected

Intersection of L1 and L2:
Input: ab => Accepted
Input: aab => Accepted
Input: ba => Rejected
Input: a => Rejected
Input: baa => Rejected
Input: bb => Rejected

Concatenation of L1 and L2:
Input: ab => Rejected
Input: aab => Rejected
Input: ba => Rejected
Input: a => Rejected
Input: baa => Rejected
Input: bb => Rejected
```

7. 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}}.

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

bool simulatePDA(const string& input) {
    stack<char> st;         // Stack for the PDA
    string state = "q0";    // Start state

    for (char c : input) {
        if (state == "q0") {
            if (c == 'a') {
                st.push('a'); // Push 'a' onto the stack
            } else if (c == 'b') {
                if (!st.empty()) {
                    state = "q1"; // Transition to state q1
                    st.pop();     // Pop 'a' for the first 'b'
                } else {
                    return false; // Rejected: unmatched 'b'
                }
            } else {
                return false; // Rejected: Invalid character
            }
        } else if (state == "q1") {
            if (c == 'b') {
                if (!st.empty()) {
                    st.pop(); // Pop 'a' for each 'b'
                } else {
                    return false; // Rejected: unmatched 'b'
                }
            } else {
                return false; // Rejected: 'a' not allowed in state q1
            }
        }
    }

    // Accepted if in state q1 and stack is empty
    return state == "q1" && st.empty();
}

int main() {
    string testCases[] = {"ab", "aabb", "aaabbb", "aaaabbbb", "abb", "aab", "ba", ""};

    for (const string& test : testCases) {
```

```
        cout << "Input: \"" << test << "\" -> "
             << (simulatePDA(test) ? "Accepted" : "Rejected") << endl;
    }

    return 0;
}
```

Output

Clear

```
Input: "ab" -> Accepted
Input: "aabb" -> Accepted
Input: "aaabbb" -> Accepted
Input: "aaaabbbb" -> Accepted
Input: "abb" -> Rejected
Input: "aab" -> Rejected
Input: "ba" -> Rejected
Input: "" -> Rejected


=== Code Execution Successful ===
```

8. 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 }

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

bool simulateMachine(const string& input) {
    stack<char> st; // Stack to store 'w'
    size_t len = input.length();
    size_t posX = input.find('X');

    if (posX == string::npos) {
        return false; // No 'X' found, invalid string
    }

    // Process the first part (w)
    for (size_t i = 0; i < posX; ++i) {
        if (input[i] == 'a' || input[i] == 'b') {
            st.push(input[i]); // Push characters of w onto the stack
        } else {
            return false; // Invalid character
        }
    }

    // Process the second part (w^R)
    for (size_t i = posX + 1; i < len; ++i) {
        if (st.empty() || input[i] != st.top()) {
            return false; // Mismatch between w and w^R
        }
        st.pop();
    }

    // String is accepted if the stack is empty after processing
    return st.empty();
}

int main() {
    string testCases[] = {
        "aXa", "abXba", "abbXbba", "aabbXbbaa", "abXab", "aX", "X", "abXa"
    };

    for (const string& test : testCases) {
        cout << "Input: \"" << test << "\" -> "
```

```
        << (simulateMachine(test) ? "Accepted" : "Rejected") << endl;
    }

    return 0;
}
```

Output

Clear

```
Input: "aXa" -> Accepted
Input: "abXba" -> Accepted
Input: "abbXbba" -> Accepted
Input: "aabbXbbaa" -> Accepted
Input: "abXab" -> Rejected
Input: "aX" -> Rejected
Input: "X" -> Accepted
Input: "abXa" -> Rejected


=== Code Execution Successful ===
```

9. Simulate a Turing Machine that accepts the language anbncn where n >0.

```
#include <iostream>
 #include <string>
 using namespace std;
 // Function to simulate the Turing Machine
 bool simulateTuringMachine(string tape) {
    size_t head = 0;
    while (true) {
        // Step 1: Mark the first unmarked 'a'
        while (head < tape.size() && tape[head] != 'a') {
            if (tape[head] != 'X' && tape[head] != 'b' && tape[head] != 'Y' && tape[head] != 'c' &&
tape[head] != 'Z') {
                return false; // Invalid character found
            }
            head++;
        }
        if (head >= tape.size()) break;
        tape[head] = 'X';
        head++;
        // Step 2: Find and mark the first unmarked 'b'
        while (head < tape.size() && tape[head] != 'b') {
            if (tape[head] != 'Y' && tape[head] != 'c' && tape[head] != 'Z') {
                return false; // Invalid character or order
            }
            head++;
        }
        if (head >= tape.size()) return false;
        tape[head] = 'Y';
        head++;
        // Step 3: Find and mark the first unmarked 'c'
        while (head < tape.size() && tape[head] != 'c') {
            if (tape[head] != 'Z') {
                return false; // Invalid character or order
            }
            head++;
        }
        if (head >= tape.size()) return false;
        tape[head] = 'Z';
        head = 0;
    }
    for (char ch : tape) {
        if (ch != 'X' && ch != 'Y' && ch != 'Z') {
            return false;
        }
    }
```

```
    return true;
}
 int main() {
    string testCases[] = {"abc", "aabbcc", "aaabbbccc", "abccba", "aabbc", "abcabc", ""};
    for (const string& test : testCases) {
        cout << "Input: \"" << test << "\" -> "
             << (simulateTuringMachine(test) ? "Accepted" : "Rejected") << endl;
    }
    return 0;
 }
```

Output

Clear

```
Input: "abc" -> Accepted
Input: "aabbcc" -> Rejected
Input: "aaabbbccc" -> Rejected
Input: "abccba" -> Rejected
Input: "aabbc" -> Rejected
Input: "abcabc" -> Accepted
Input: "" -> Accepted


=== Code Execution Successful ===
```

10. Simulate a Turing Machine which will increment the given binary number by 1.

```
#include <iostream>
 #include <string>
 using namespace std;
 string incrementBinary(string tape) {
    int head = tape.length() - 1; // Start at the rightmost bit (LSB)
    while (head >= 0) {
        if (tape[head] == '0') {
            // If the current bit is 0, change it to 1 and halt
            tape[head] = '1';
            return tape;
        } else if (tape[head] == '1') {
            // If the current bit is 1, change it to 0 and move left
            tape[head] = '0';
        } else {
            // Invalid character in the tape
            throw invalid_argument("Invalid binary number");
        }
        head--; // Move the head left
    }
    return "1" + tape;
 }
 int main() {
    string testCases[] = {"0", "1", "10", "11", "101", "111", "1000", ""};
    for (const string& test : testCases) {
        try {
cout << "Input: \"" << test << "\" -> Output: \"" << incrementBinary(test) << "\"" << endl;
 } catch (const invalid_argument& e) {
 cout << "Input: \"" << test << "\" -> Error: " << e.what() << endl;
 }
 }
 return 0;
 }
```

Output

Clear

```
Input: "0" -> Output: "1"
Input: "1" -> Output: "10"
Input: "10" -> Output: "11"
Input: "11" -> Output: "100"
Input: "101" -> Output: "110"
Input: "111" -> Output: "1000"
Input: "1000" -> Output: "1001"
Input: "" -> Output: "1"


=== Code Execution Successful ===
```