Lab Exp 2:

Design a DFA

1) Which accepts the substrings “abb”
2) Which accepts strings with alternating `0`s and `1`s (e.g., `"0101"`, `"101"`)

In this lab experiment, you will design and implement a **Deterministic Finite Automaton (DFA)** that accepts the sub strings `"abb"`. This exercise will help you understand the underlying principles of DFA and its implementation in a programming language like C.

Approach: C is optional , try Lex

1. **Manual Implementation in C:**
   This is an imperative approach where you manually design and implement the DFA logic using basic control structures in C. You will explicitly define the states, transitions, and acceptance conditions in code. This hands-on method helps deepen your understanding of how DFAs operate and provides foundational knowledge of automata theory, which is essential for tasks like lexical analysis in compiler construction.

2. **Using LEX:**
   Design the DFA using **LEX** (or **Flex**) . This declarative approach automates the process of defining the DFA and gives you practical experience with lexical analysis tools, which are widely used in compiler construction for pattern matching and token recognition.

Example Test Cases:

1. DFA that accepts the substring "abb"

Accepted Test Case (contains the substring "abb"):

- **Input:** `babbaa`
- **Output:** `Accepted` **(contains** `"abb"` **as a substring).**

Rejected Test Case (does not contain the substring "abb"):

- **Input:** `abab`
- **Output:** `Not Accepted` **(does not contain** `"abb"`**).**

Invalid Test Case (contains invalid characters):

- **Input:** `a@bb!`
- **Output:** `Invalid` **(contains invalid characters** `@` **and** `!`**).**

---

2. DFA that accepts strings with alternating 0s and 1s

Accepted Test Case (alternates between `0` **and** `1`**):**

- **Input:** `010101`
- **Output:** `Accepted` **(alternates between** `0` **and** `1`**).**

Rejected Test Case (contains consecutive `0`**s or** `1`**s):**

- **Input:** `1110`
- **Output:** `Not Accepted` **(contains consecutive** `1`**s).**

Invalid Test Case (contains non-binary characters):

- **Input:** `abc01`
- **Output:** `Invalid` **(contains invalid characters** `a`**,** `b`**, and** `c`**).**