Assignment-1

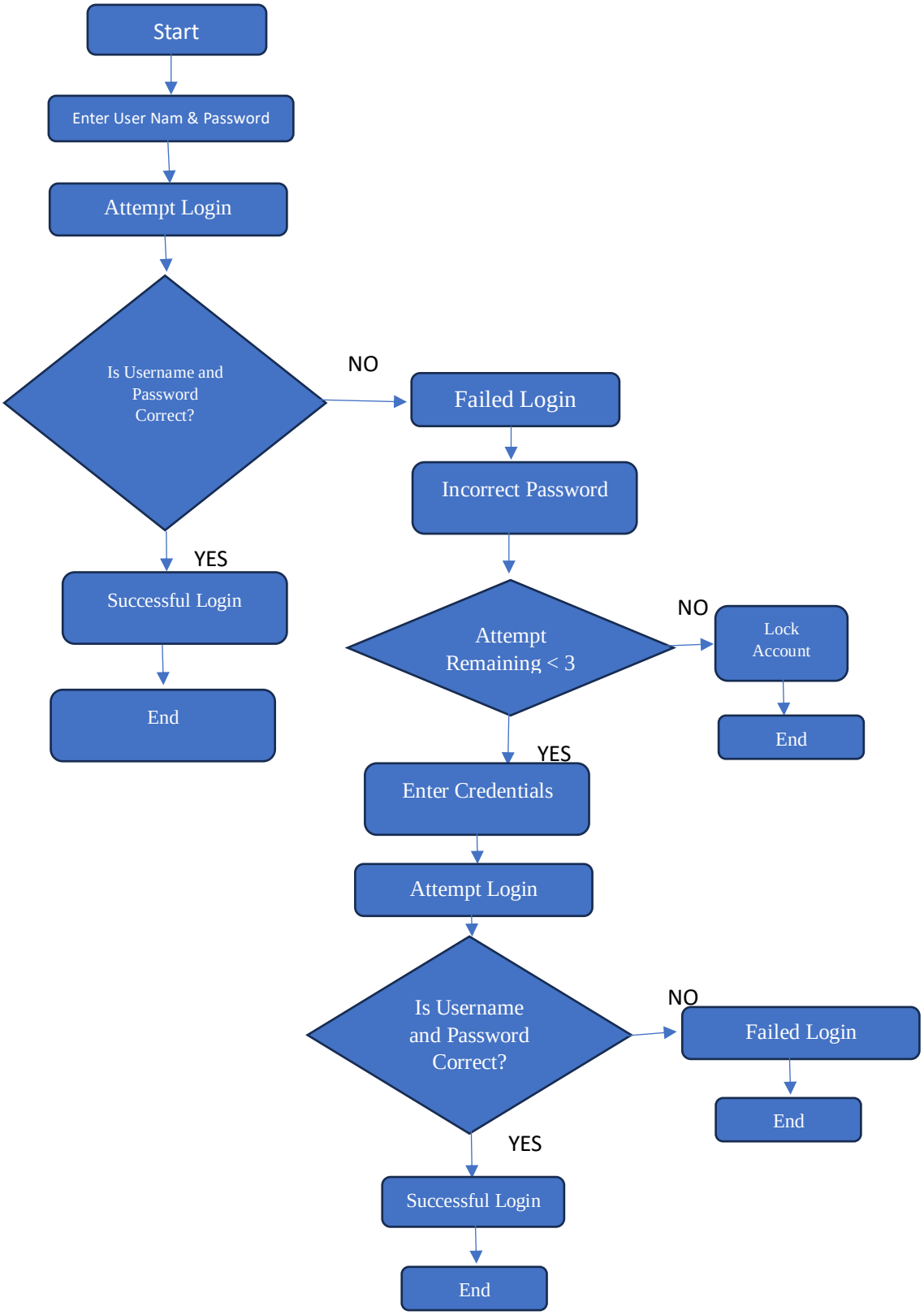
Task 1: Pseudocode Development: - Write a Detailed pseudocode for a simple program that takes a number as input. Calculates the square if it's even or the cube if it's odd, and then outputs the result. Incorporate conditional and looping constructs.

Pseudocode for a simple program that takes a number as input, calculates the square if it's even, or the cube if it's odd, and then outputs the result. This pseudocode incorporates conditional and looping constructs:

```
1. Start

2. Initialize variables:
   - number
   - result

3. Input: Prompt the user to enter a number
   - Read the input and store it in the variable 'number'

4. Check if the number is positive:
   - If number < 0, display an error message and end the program
   - Else, continue to the next step

5. Check if the number is even or odd:
   - If number % 2 == 0, set 'result' to number * number (square)
   - If number % 2 != 0, set 'result' to number * number * number (cube)

6. Output: Display the result
   - Display "The result is: " + result

7. End
```

This pseudocode outlines a simple program flow that takes a number as input, determines if it's even or odd, calculates the square or cube accordingly, and then outputs the result. It incorporates conditional statements to check for even or odd numbers and a basic error handling step for negative numbers.

Task 2: Flow Chart Creation: - Design a flowchart that outlines the logic for a user login process. It should include conditional paths for successful and unsuccessful login attempts, and a loop that allows a user there attempts before locking the account.



Explanation:

- The process starts with the user entering their username and password.
- The system attempts to log in using the provided credentials.
- If the username and password are correct, the process proceeds to the "Successful Login" path, and the user gains access to the system.
- If the username and password are incorrect, the process follows the "Failed Login" path. The system checks if the user has any attempts remaining.
- If the user has attempts remaining (less than three attempts), the process loops back to the "Enter Username and Password" step to allow the user to try again.
- If the user has no attempts remaining (three failed attempts), the process proceeds to the "Lock Account" path, indicating that the account will be locked.
- The process ends after the final step.

Task 3: Function Design And Modularization: - Create a document that describes the design of two modular functions. One that returns the factorial of a number, and another that calculates the nth Fibonacci number. Include pseudocode brief explanation of how modularity in programming helps with code reuse and organization

1. Factorial Function

Description: The factorial function calculates the factorial of a given non-negative integer.

Modular Function Design:

```
Function factorial(n)
  if n = 0
    return 1
  else
    return n * factorial(n - 1)
```

Explanation:

- The factorial function is defined recursively, where the base case is when n equals 0, in which case the function returns 1.
- If n is greater than 0, the function recursively calls itself with the argument (n - 1) and multiplies the result by n.
- This design allows for a concise and efficient implementation of the factorial calculation logic.

- **2. Fibonacci Function**

- **Description:** The Fibonacci function calculates the nth Fibonacci number, where the Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding ones.

- **Modular Function Design:**

```
Function fibonacci(n)
  if n <= 1
    return n
  else
    return fibonacci(n - 1) + fibonacci(n - 2)
```

Explanation:

- Similar to the factorial function, the Fibonacci function is defined recursively.
- The base cases are when n is 0 or 1, in which case the function returns n.
- If n is greater than 1, the function recursively calls itself with the arguments (n - 1) and (n - 2) and returns the sum of the results.
- This modular design encapsulates the Fibonacci sequence generation logic in a reusable and organized manner.

Modularity in Programming:

Brief Explanation: Modularity in programming refers to the practice of breaking down a program into smaller, self-contained modules or functions that perform specific tasks. These modules can then be reused across different parts of the program or in other programs altogether. Modularity helps with code reuse and organization in several ways:

1. **Code Reuse:** Modular functions can be reused multiple times within the same program or across different programs without the need to rewrite the same logic. This reduces redundancy and promotes efficiency in development.
2. **Organization:** By dividing a program into modular functions, the code becomes more organized and easier to understand, maintain, and debug. Each function encapsulates a specific task or piece of functionality, making it easier to reason about and modify.
3. **Abstraction:** Modular functions abstract away the implementation details of a particular task, providing a higher level of abstraction. This allows developers to focus on the functionality of each module without needing to understand the internal workings of other modules.
4. **Scalability:** Modular design allows for scalability, as new functionality can be added by simply creating additional modules or extending existing ones. This promotes flexibility and adaptability in software development.

Overall, modularity enhances code reusability, maintainability, and scalability, leading to more efficient and robust software development practices.

Conclusion: In this document, we have described the design of two modular functions: one that calculates the factorial of a number and another that calculates the nth Fibonacci number. These modular functions demonstrate how modularity in programming helps with code reuse and organization by encapsulating specific tasks or functionalities into reusable and self-contained modules. By employing modular design principles, developers can create more efficient, maintainable, and scalable software solutions.