**Assignment3:  
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 pseudo code and  
a brief explanation of how modularity in programming helps with code reuse  
and organization.**

Factorial of a Number:

Algorithm:

1. Start

2. Declare a variable n to store the input number

3. Declare a variable result to store the factorial value

4. Read the value of n from the user

5. If n is equal 0 or n is equal to 1 then set result is 1

6. If 'n' is greater than 1 go to step 7 otherwise, go to step 10

7. Repeat steps 8-9 until n becomes 0

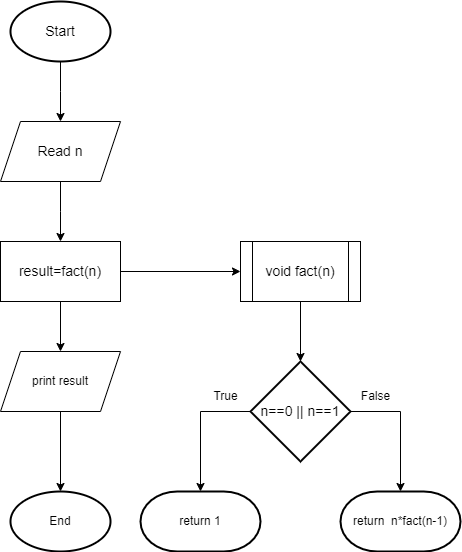
8. Multiply result by n

9. Decrement n by 1

10. Output result

11. End

Flowchart:



Psedocode:

Function Main  
  
    ...This program asks the user for a number and calculates its factorial  
  
    ... Give the result of the factorial of the given number  
  
    ... And display the result  
  
1. Begin  
  
2. Read n  
  
3. result = factorial(n)  
  
4. Output "Factorial of " & n & " is " & result  
  
5. End  
  
Function factorial(n)  
  
    If n is equal to 0:  
  
        Return 1  
  
    Else:  
  
        Return n \* factorial(n - 1)

Calculates the nth Fibonacci number:

Algorithm:

1. Start
2. Read input n
3. Call fibonacci(n) and assign the result to 'result'
4. Output "Fibonacci series up to " & n & " terms:"
5. Set i = 0
6. If i < n, go to step 7; otherwise, go to step 8
7. Output fibonacci(i)
8. Increment i by 1
9. If i < n, repeat from step 6; otherwise, go to step 10
10. End

Function fibonacci(n)

1. If n is equal to 0:

Return 0

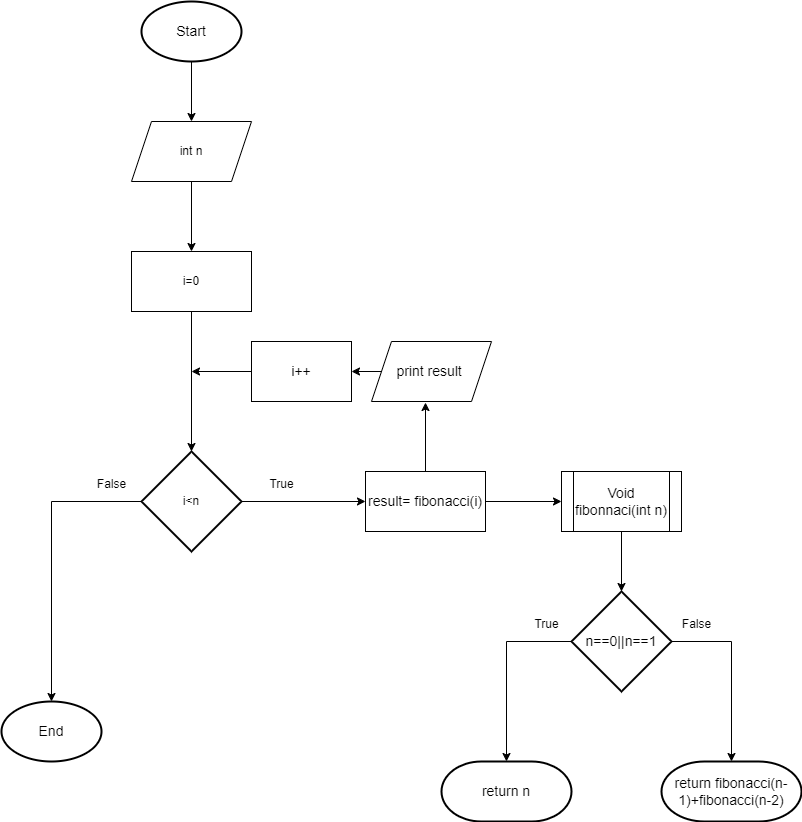
Else if n is equal to 1:

Return 1

Else:

Return fibonacci(n - 1) + fibonacci(n - 2)

Flowchart:



Psedocode:

Function Main

1. Begin

2. Read n

3. Output "Fibonacci series up to " & n & "th value:"

4. For i from 0 to n-1

Output fibonacci(i) & " "

5. End for

6. End

Function fibonacci(n)

If n is equal to 0:

Return 0

Else if n is equal to 1:

Return 1

Else:

Return fibonacci(n - 1) + fibonacci(n - 2)

**Modularity in Programming:**  
  
Modularity in programming refers to the practice of breaking down a program into smaller, independent modules or functions. Each module performs a specific task, and these modules can be reused in different parts of the program or even in different programs altogether.  
  
**Benefits of Modularity:**  
  
**Code Reusability**: Modular functions can be reused in different parts of the program or in different programs, saving time and effort in coding.  
**Ease of Maintenance**: Modular code is easier to understand and maintain, as each module focuses on a specific task, making it easier to debug and update.  
**Improved Collaboration**: Modular code allows multiple developers to work on different modules simultaneously, enhancing collaboration and productivity.  
**Scalability**: Modularity allows for easy scalability of the codebase, as new features or functionalities can be added by creating new modules without affecting existing ones.  
By designing modular functions, programmers can achieve better code organization, readability, and maintainability, leading to more efficient and reliable software development.