**Lab 3: Recursive Programs**

**Problem Statement**

Write a Prolog program to compute the **factorial** of a number and generate the **Fibonacci series** using recursion.

**Implementation Steps**

1. **Factorial**:

factorial(0, 1).

factorial(N, Result) :-

N > 0,

N1 is N - 1,

factorial(N1, Result1),

Result is N \* Result1.

**Implementation Steps**

1. **Base Case**:
   * The factorial of 0 is 1.

factorial(0, 1).

1. **Recursive Case**:
   * For n>0*n*>0, compute the factorial of *n*−1 and multiply it by *n*.

factorial(N, Result) :-

N > 0, % Ensure N is positive

N1 is N - 1, % Compute N-1

factorial(N1, Result1), % Recursively compute factorial of N-1

Result is N \* Result1. % Multiply N with the result of factorial(N-1)

1. **Explanation**:
   * The program first checks if *N*=0. If true, it returns 1.
   * If *N*>0, it recursively computes the factorial of *N*−1 and multiplies it by *N*.

**Sample Output**

?- factorial(5, X).

X = 120.

?- factorial(0, X).

X = 1.

?- factorial(6, X).

X = 720.

**Step-by-Step Execution of Factorial Program**

Let’s trace the execution of factorial(3, X) to understand how recursion works:

1. **Call 1**: factorial(3, X)
   * 3>0, so compute factorial(2, Result1).
2. **Call 2**: factorial(2, Result1)
   * 2>0, so compute factorial(1, Result2).
3. **Call 3**: factorial(1, Result2)
   * 1>0, so compute factorial(0, Result3).
4. **Call 4**: factorial(0, Result3)
   * Base case reached: Result3 = 1.
5. **Return to Call 3**:
   * Result2 = 1 \* 1 = 1.
6. **Return to Call 2**:
   * Result1 = 2 \* 1 = 2.
7. **Return to Call 1**:
   * X = 3 \* 2 = 6.

Final result: X = 6.

1. **Fibonacci**:

fibonacci(0, 0).

fibonacci(1, 1).

fibonacci(N, Result) :-

N > 1,

N1 is N - 1,

N2 is N - 2,

fibonacci(N1, Result1),

fibonacci(N2, Result2),

Result is Result1 + Result2.

**Sample Output**

?- factorial(5, X).

X = 120.

?- fibonacci(6, X).

X = 8.

**Implementation Steps**

1. **Base Cases**:
   * The Fibonacci of 0 is 0.
   * The Fibonacci of 1 is 1.

fibonacci(0, 0).

fibonacci(1, 1).

1. **Recursive Case**:
   * For n>1*n*>1, compute the Fibonacci of n−1*n*−1 and n−2*n*−2, and add them.

fibonacci(N, Result) :-

N > 1, % Ensure N is greater than 1

N1 is N - 1, % Compute N-1

N2 is N - 2, % Compute N-2

fibonacci(N1, Result1), % Recursively compute Fibonacci of N-1

fibonacci(N2, Result2), % Recursively compute Fibonacci of N-2

Result is Result1 + Result2. % Add the results

1. **Explanation**:
   * The program first checks if *N*=0 or *N*=1. If true, it returns 0 or 1, respectively.
   * If *N*>1, it recursively computes the Fibonacci of *N*−1 and *N*−2 and adds them.

**Sample Output**

?- fibonacci(6, X).

X = 8.

?- fibonacci(0, X).

X = 0.

?- fibonacci(1, X).

X = 1.

?- fibonacci(10, X).

X = 55.

**Step-by-Step Execution of Fibonacci Program**

Let’s trace the execution of fibonacci(4, X):

1. **Call 1**: fibonacci(4, X)
   * 4>1, so compute fibonacci(3, Result1) and fibonacci(2, Result2).
2. **Call 2**: fibonacci(3, Result1)
   * 3>1, so compute fibonacci(2, Result3) and fibonacci(1, Result4).
3. **Call 3**: fibonacci(2, Result3)
   * 2>1, so compute fibonacci(1, Result5) and fibonacci(0, Result6).
4. **Call 4**: fibonacci(1, Result5)
   * Base case: Result5 = 1.
5. **Call 5**: fibonacci(0, Result6)
   * Base case: Result6 = 0.
6. **Return to Call 3**:
   * Result3 = 1 + 0 = 1.
7. **Return to Call 2**:
   * Result1 = 1 + 1 = 2.
8. **Call 6**: fibonacci(2, Result2)
   * 2>1, so compute fibonacci(1, Result7) and fibonacci(0, Result8).
9. **Call 7**: fibonacci(1, Result7)
   * Base case: Result7 = 1.
10. **Call 8**: fibonacci(0, Result8)
    * Base case: Result8 = 0.
11. **Return to Call 6**:
    * Result2 = 1 + 0 = 1.
12. **Return to Call 1**:
    * X = 2 + 1 = 3.

Final result: X = 3.