Climb Stairs

The "Climb Stairs" problem is a classic algorithmic challenge frequently encountered in computer science and programming interviews. The problem statement goes as follows: Given a staircase with n steps, you can climb either 1 or 2 steps at a time. You need to determine the number of distinct ways to reach the top of the staircase. This problem can be solved using various approaches such as dynamic programming, recursion with memoization, or even simple iterative methods. It's a great exercise for understanding recursion, dynamic programming concepts, and problem-solving skills. Let's delve deeper into some of the common strategies used to tackle this problem.

Code in C++

```
#include <iostream>
#include <vector>

using namespace std;

int climbStairs(int n) {
    if (n == 1) {
        return 1;
    }
    if (n == 2) {
        return 2;
    }

    vector<int> dp(n + 1, 0);
    dp[1] = 1;
    dp[2] = 2;

    for (int i = 3; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }

    return dp[n];
}
```

```
int main() {
    int n = 5;
    cout << "Number of distinct ways to climb " << n << " stairs: " <<
climbStairs(n) << endl;
    return 0;
}
```

Explanation

Function Declaration:
The code begins by declaring a function climbStairs that takes an integer n as input and returns an integer. This function calculates the number of distinct ways to climb n stairs.

Base Cases Handling:
Inside the climbStairs function, it checks for two base cases: if n is 1 or 2. If n is 1, there is only one way to climb the stairs, so it returns 1. If n is 2, there are two ways to climb the stairs, so it returns 2.

Dynamic Programming Approach:
The function uses dynamic programming to efficiently solve the problem for larger values of n.
It creates a vector dp of size n+1 to store the number of distinct ways to climb i stairs, where dp[i] will hold the solution for i stairs.
Initialize dp[1] to 1 and dp[2] to 2, as discussed in the base cases handling.

Filling the DP Table:
Then, it iterates from i = 3 up to n.
For each i, it calculates the number of distinct ways to climb i stairs by adding the number of ways to climb i-1 stairs and i-2 stairs. This is the essence of dynamic programming, where solutions to smaller subproblems are used to solve larger problems.

Returning the Result:
Finally, the function returns dp[n], which holds the number of distinct ways to climb n stairs.

Main Function:
The main() function demonstrates the usage of the climbStairs function.
It sets n = 5 as an example and prints the result of climbStairs(n).

All The Best! Happy Coding!
TCET SHASTRA CODING CLUB