Climbing Stairs in C++

```
#include <iostream>
#include <vector>
#include <climits> // For INT_MAX

using namespace std;

void printMinSteps(vector<int>& arr) {
    int n = arr.size();
    vector<int> dp(n + 1, INT_MAX); // Use INT_MAX for initialization

    dp[n] = 0; // Base case: 0 steps needed from the end

    for (int i = n - 1; i >= 0; i--) {
        if (arr[i] > 0) {
            int minSteps = INT_MAX;
            for (int j = 1; j <= arr[i] && (i + j) < dp.size(); j++) {
                if (dp[i + j] != INT_MAX) {
                    minSteps = min(minSteps, dp[i + j]);
                }
            }
            if (minSteps != INT_MAX) {
                dp[i] = minSteps + 1;
            }
        }
    }

    // Printing the dp array
    for (int i = 0; i < dp.size(); i++) {
        cout << " " << dp[i];
    }
    cout << endl;
}

int main() {
    vector<int> arr = {1, 5, 2, 3, 1};
    printMinSteps(arr);

    return 0;
}
```

Given:

vector<int> arr = {1, 5, 2, 3, 1};

The length of arr is **5**, so dp is initialized as:

dp = [INT_MAX, INT_MAX, INT_MAX, INT_MAX, INT_MAX, 0] // (size = 6, last element is 0)

Dry Run with Iteration Table

The loop iterates from **i = n - 1 to 0**, checking possible jumps and updating dp[i].

Iteration (i)	arr[i]	Possible Jumps	Min Steps from Reachable Positions	Updated dp[i]
4 (last)	1	(4→5)	dp[5] = 0 → min(∞, 0)	dp[4] = 1
3	3	(3→4, 3→5)	dp[4] = 1, dp[5] = 0 → min(∞, 1, 0)	dp[3] = 1
2	2	(2→3, 2→4)	dp[3] = 1, dp[4] = 1 → min(∞, 1, 1)	dp[2] = 2
1	5	(1→2, 1→3, 1→4, 1→5)	dp[2] = 2, dp[3] = 1, dp[4] = 1, dp[5] = 0 → min(∞, 2, 1, 1, 0)	dp[1] = 1
0 (first)	1	(0→1)	dp[1] = 1 → min(∞, 1)	dp[0] = 2

Final dp Array

After all iterations, the **dp array** will be:

dp = [2, 1, 2, 1, 1, 0]

Output:

2 1 2 1 1 0

Output:-

- Printed dp: 2 1 2 1 1 0
- The minimum steps to reach the end starting from index 0 is dp[0] = 2.