0/1 KnapSack in C++

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

using namespace std;

class ZeroOneKnapsack {
public:
    int knapsack(int n, vector<int>& vals,
vector<int>& wts, int cap) {
        vector<vector<int>> dp(n + 1, vector<int>(cap +
1, 0));

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= cap; j++) {
                if (j >= wts[i - 1]) {
                    int remainingCap = j - wts[i - 1];

                    if (dp[i - 1][remainingCap] + vals[i - 1] >
dp[i - 1][j]) {
                        dp[i][j] = dp[i - 1][remainingCap] +
vals[i - 1];
                    } else {
                        dp[i][j] = dp[i - 1][j];
                    }
                } else {
                    dp[i][j] = dp[i - 1][j];
                }
            }
        }

        return dp[n][cap];
    }
};

int main() {
    ZeroOneKnapsack solution;

    // Input parameters
    int n = 5;
    vector<int> vals = {15, 14, 10, 45, 30};
    vector<int> wts = {2, 5, 1, 3, 4};
    int cap = 7;

    // Compute maximum value using knapsack
function
    int maxVal = solution.knapsack(n, vals, wts, cap);

    // Output the maximum value
    cout << "Maximum value that can be obtained: " <<
maxVal << endl;

    return 0;
}
```

Dry Run

Input:

- Number of items: n = 5
- Values: vals = {15, 14, 10, 45, 30}
- Weights: wts = {2, 5, 1, 3, 4}
- Capacity: cap = 7

Steps:

1. **Initialize the DP Table**:
    - dp is a 2D table of size (n+1) × (cap+1) (i.e., 6 × 8).
    - Initially, all entries are 0.

DP Table Construction

Base Case:

dp[0][j] = 0 for all j
dp[i][0] = 0 for all i

DP Transitions:

- **Row 1 (i = 1, item with value = 15, weight = 2)**:
    - For j = 1: dp[1][1] = 0 (weight exceeds capacity).
    - For j = 2: dp[1][2] = 15 (item included).
    - For j = 3 to 7: dp[1][j] = 15 (item included).
- **Row 2 (i = 2, item with value = 14, weight = 5)**:
    - For j = 1 to 4: dp[2][j] = dp[1][j].
    - For j = 5: dp[2][5] = max(dp[1][5], vals[1] + dp[1][5 - wts[1]]) = max(15, 14) = 15.
    - For j = 6: dp[2][6] = max(15, 14) = 15.
    - For j = 7: dp[2][7] = max(15, 15 + 14) = 29.
- **Row 3 (i = 3, item with value = 10, weight = 1)**:
    - Updates based on the new item's inclusion.
- **Row 4 (i = 4, item with value = 45, weight = 3)**:
    - Updates based on the new item's inclusion.
- **Row 5 (i = 5, item with value = 30, weight = 4)**:
    - Updates based on the new item's inclusion.

	Final DP Table:
	
	dp = {
	{ 0, 0, 0, 0, 0, 0, 0, 0 },
	{ 0, 0,15,15,15,15,15,15 },
	{ 0, 0,15,15,15,15,29,29 },
	{ 0,10,15,25,25,25,29,40 },
	{ 0,10,15,45,55,55,70,70 },
	{ 0,10,15,45,55,55,70,75 },
	}

Output: Maximum value that can be obtained: 75 The maximum value that can be obtained is stored in dp[5][7] = 75.	