/\* Knapsack Dynamic Approach \*/

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int knapsack(int W, const vector<int>& wt, const vector<int>& val, int n) {

// Create a 2D table to store the maximum value that can be obtained for different weights and items

vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

// Build the dp table in a bottom-up manner

for (int i = 1; i <= n; i++) {

for (int w = 1; w <= W; w++) {

// If the current item's weight is less than or equal to the current capacity, include it

if (wt[i - 1] <= w) {

dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);

} else {

// If the current item's weight is more than the current capacity, exclude it

dp[i][w] = dp[i - 1][w];

}

}

}

// Return the maximum value that can be obtained

return dp[n][W];

}

int main() {

int n;

cout << "Enter the number of items: ";

cin >> n;

vector<int> val(n);

vector<int> wt(n);

cout << "Enter the values and weights of the items:" << endl;

for (int i = 0; i < n; i++) {

cout << "Item " << i + 1 << ": ";

cin >> val[i] >> wt[i];

}

int W;

cout << "Enter the capacity of the knapsack: ";

cin >> W;

int maxValue = knapsack(W, wt, val, n);

cout << "Maximum value that can be obtained: " << maxValue << endl;

return 0;

}