# Assignment 6: Solve the 0-1 Knapsack Problem (AGAIN!) using Dynamic Programming

## Learning Outcomes

* Solve the Knapsack Problem using Dynamic Programming.

## Motivation

Hopefully you realize that the brute force solution is kinda crappy. It's exhaustive and slow. It does you no favors, but it still returns the optimal results.

## Inputs

(This is an identical description in the previous assignment.)

The story of the 0-1 knapsack problem is that you are a robber and must rob a store with the greatest total value of the items in knapsack as possible without exceeding the weight limit of your knapsack.

* The first integer seen in the input will be the weight limit of your knapsack.
* The second integer seen will be the number of items in the fictional store which we are attempting to rob.
* Every input after that will be a product represented by three values.
  + A single word string representing the name of item, such as "bike" or "toycar". Notice that the name will not have any spaces in it.
  + An integer representing the weight of the item.
  + An integer representing the value of the item.

## Outputs

* Your program should report all of the items which you put into your knapsack. It should have the maximum value without exceeding the weight limit of the knapsack.
* You should report the total value.
* You should report the total weight.

## How to solve the problem.

Let's do this problem again with Dynamic Programming.

// num\_products is the number of products.  
// weight is the weight limit of the knapsack  
// Create a matrix K that is the size of (num\_products+1, weights+1)  
  
// For each cell in the matrix K, do the following:  
// i is 0 to num\_products  
// w is 0 to weight  
// if i is 0 or w is 0  
// Write 0 to K(i,w)  
// if products[i-1] can fit in the bag  
// (the weight is less than or equal to w)  
// Then find the following:  
// A: the value at K(i-1,w)  
// B: the value of product[i-1] plus K(i-1,w - products[i-1]'s weight)  
// Write the larger of the two values (A or B) to K(i,w)  
// if products[i-1] can't fit in the bag, then  
// Write K(i-1,w) to K(i-1,w)  
  
// We need to get the values back out of the K matrix.  
// Set i to num\_products  
// Set w to weight  
// while K(i,w) does not equal 0  
// while K(i-1,w) equals K(i,w) // Move Up  
// set i to i-1  
// put the element at products[i-1] into the knapsack  
// Set v to be the K(i,w) - product[i-1]'s value  
// while K(i,w) does not equal v // Move Left  
// set w to w-1  
//

## Examples.

Example 1.

5  
3  
speaker 2 100  
bike 3 120  
nicechair 4 1000

Results.

Name: nicechair Weight: 4 Value: 1000  
  
Total Weight: 4  
Total Value : 1000

Example 2.

5  
3  
speaker 2 100  
bike 3 120  
chair 4 200

Results.

Name: speaker Weight: 2 Value: 100  
Name: bike Weight: 3 Value: 120  
  
Total Weight: 5  
Total Value : 220

## Turn it in

Upload a zip file containing the files that use used or wrote yourself to the drop box on D2L:

* Make sure your name, CSCI 4270, and the Programming Assignment Number appear in comments in all of your files.
* Note: NO CREDIT will be given to programming assignments that do not compile.
* Make sure you have compiled and tested your program before handing it in.