Solution using Dynamic Programming (continued)

- memo[i, W] is defined and filled as follows:

for brevity called m instead of memo in this definition.

- m[0, W] = 0
- if ($W_i > W$)
  m[i, W] = m[i-1, W]
- if ($W_i \leq W$)
  m[i, W] = max(m[i-1, W], m[i-1, W-W_i] + V_i)

m

i \ W	0	1	2	...	W
0	0	0	0	0	0
1					
...					
N					

m[i-1, W]: We don't take item i ⇒ We neither gain its value V_i, nor do we gain its weight W_i

m[i-1, W-W_i] + V_i: We take the current item i ⇒ we gain its value V_i ⇒ we lose W_i "space" because we gain its weight

Pseudocode (DP)

```
KNAPSACK (values, weights, N, W) {
for (w from 0 to W)
    memo[0][w] = 0

for (i from 1 to N)
    for (w from 0 to W)
        if (weights[i] > w)
            memo[i][w] = memo[i-1][w]
        else // weights[i] ≤ w
            memo[i][w]
            = max(memo[i-1][w],
                  memo[i-1][w - weights[i]] + values[i])
return memo[N][W] }
```

Complexity (DP)

This solution runs in
$O(N \cdot W)$ time, and
$O(N \cdot W)$ space.