```
#ifndef KNAPSACK_HPP
#define KNAPSACK_HPP

#include <vector>

using vector = std::vector<int>;

//Definition contained in the corresponding cpp file
vector
knapsack(vector values, vector weights, int weight_lim);

#endif
```



```
#include <iostream>
#include <vector>
#include "knapsack.hpp"

//Definitions for vectors and matricies of integers, used to reduce
using vector = std::vector<int>;
using table = std::vector<std::vector<int>>;

//The function takes in two vectors of positive integers that hold the weights
//and values of the items to be stolen. It is assumed that the arrays are the
//same size, and elements are in the same order. For example the first value in
//the values array should also be the first weight in the weights array. This is
//not strongly enforced, but is required for the function to perform correctly.

vector
knapsack(vector values, vector weights, int weight_lim)
{
    //The two matricies store the previously computed best value combinations,
    //and when each element is excluded of included noted by a 0 or 1
    //respectively. They are padded with an extra row an column, so accessing
    //the weights requires an offset of negative one. The vector will be used
    //at the end to store the calculated answer.

    table subproblems(values.size() + 1, vector(weight_lim + 1 , 0));
    table inclusion(values.size() + 1, vector(weight_lim + 1 , 0));
    vector solution;

    //These loops iterate over every item at every possible weight from 0 up
    //to and including weight_lim.

    for(int i = 1; i <= values.size(); ++i)
    {
        for(int j = 1; j <= weight_lim; ++j)
        {
            if(weights[i-1] > j) //The item is thrown out if its weight alone is

            {                   //more than the maximum.

                subproblems[i][j] = subproblems[i-1][j];
            }
            //Determine if it is better to take the item and then mark it
            //as included if so.

            else if((subproblems[i-1][j-weights[i-1]] + values[i-1])
                                          > subproblems[i-1][j] )
            {
                subproblems[i][j] = subproblems[i-1][j-weights[i-1]] + values[i-
1];
                inclusion[i][j] = 1;
            }

            //Otherwise indicate that the previously computed row was optimal.

            else
            {
                subproblems[i][j] = subproblems[i-1][j];
            }
        }
    }

    int W = weight_lim; //A variable holding current weights to trace back
                        //through the problem.

    for(int i = inclusion.size()-1; i >= 0; --i) // This iterates backwards
    {                                            //through the inclusion array
        if(inclusion[i][W] == 1)                 //tracing the optimal solution
        {

        //If the item here is taken, add it to the solution and remove its
```

```
        //weight from the available pool.

            solution.push_back(i-1);
            W -= weights[i-1];
        }
    }

    //Write the maximum value to standard output

    std::cout << "Maximum Value: " << subproblems[values.size()][weight_lim] << st
d::endl;

    //Finally return the end result.
    return solution;
}
```



```
#include "knapsack.hpp"
#include <vector>
#include <iostream>
#include <random>

constexpr int NUM_ITEMS = 10;
constexpr int MAX_WEIGHT = 10;

int main(int argc, char* argv[])
{
    //Declare and initialize random weights and values for items
    std::vector<int> values (NUM_ITEMS), weights (NUM_ITEMS);
    std::srand(0);

    for(int i = 0; i < NUM_ITEMS; ++i)
    {
        values[i] = rand() % 20;
        weights[i] = rand() % 10;
    }
    //Display the generated data
    for(int i = 0; i < values.size(); ++i)
        std::cout << i << ": VAL: "
        << values[i] << " WT: " << weights[i] << std::endl;

    //Call the knapsack function on this data and store the result.
    auto results = knapsack(values, weights, MAX_WEIGHT);

    //Display the result
    for(int i = 0; i < results.size(); ++i)
        std::cout << i << ": " << results[i] << std::endl;

    return 0;
}
```