Answer

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

using namespace std;
struct Item{
  int weight;
  int value;
};
// Return the maximum value that can be put in a knapsack
of capacity W int knapsack(int W, vector<Item> and items,
vector <string> and chosen_items) {
  int N = item.size();
  vector<vector<int>>dp(N+1,vector<int>(W+1));
// Build table in bottom up manner.
  for(int i=0; i<=N; i++){
  for(int W=0; W<=W; W++){
  if (i==0 || W==0)
  dp[i][W]=0;
  else if (items[i-1] weight<=W)
  dp[i][W]=max(items[i-1].value + dp[i-1][W-items]
[i-1][W-items[i-1].weight],dp[i-1][W]);
```

```
        else.
          dp[i][W] = dp[i-1][W];
        }
    }
    // stores the result of knapsack
    int res = dp[N][W];
    int w = W;
    for (int i = N; i > 0 && res > 0; i--) {
    if (res == dp[i-1][W])
    continue;
    else {
      // This item is included.
      chosen_items.push_back(to_string(i)); // item index,
      // since this weight is included, its value is deduc
      res = res - items[i-1].value;
      w = w - items[i-1].weight;
    }
    }
    reverse(chosen_item.begin(), chosen_items.
    return dp[N][W];
    }
```

```
int main() {
 // Weight and value of items. The index will represent
 the item name vector<Item> items={{2,10},{1,7},
 {4,15},{2,8},{1,6}};
  int W=6;

 vector<string> chosen_items;
int max_value = knapsack(W, items, chosen_items);
cout<<" Maximum value in knapsack="<<max_
                                    value<<
                                    endl;
cout<<" Items to be chosen (by index): ";
for(const auto & item_index : chosen_items){
  cout<< item_index<<" ";
}
 cout<< "item_index<<"
 cout<< endl;
   return 0 ;
}
```

Output is : 31