CS19001/CS19002 PROGRAMMING AND DATA STRUCTURES LABORATORY
Assignment No: 4
Last Date of Submission: 02-March-2015

Let A be an array of n positive integers. A positive integer s is said to be a *subset sum* from A if there exist indices

$$0 \leq i_1 < i_2 < \ldots < i_k \leq n - 1$$

such that

$$s = A[i_1] + A[i_2] + \ldots + A[i_k].$$

Notice that the array A need not be sorted, and may contain duplicate entries. Moreover, a given sum s may be realized in multiple ways.

Part I

Write a function to populate the array A. Pass the array A as the only parameter. The function first reads the number n of elements that the array will store ($1 \leq n \leq 20$). The elements $A[0], A[1], \ldots, A[n-1]$ are then supplied one by one by the user. The function should return the array size n.

Part II

The user enters a positive integer s. You determine whether s is a subset sum from A. For this, write another function which takes three arguments: A, n and s. If s is a subset sum from A, the function prints a way in which the sum s is realized. If s is not a subset sum from A, a message is printed to that effect. This function does not return anything.

This function should implement the algorithm sketched here. Vary a counter c in the range $[0, 2^{n-1}]$. The n-bit binary representations of all these values of c are precisely all the bit strings of length n. Each bit string is naturally identified with a subset of the array indices. For example, let $n = 5$. The counter value $c = 25 = 2^4 + 2^3 + 2^0 = (11001)_2$ is identified with the array indices $\{0, 3, 4\}$, and the subset sum corresponding to this is $A[0] + A[3] + A[4]$. If, for any of the 2^n values of c, the subset sum equals s, then the search is successful.

In order to find the n-bit binary representation of c, you may repeatedly divide n by 2. Another possibility is to use bit-wise operators. For example, the i-th bit of c is one if and only if c & (1U << i) is non-zero. Here << is left shift by i bits, and & is the bit-wise AND operator.

Part III

In this part, you write a *recursive* function in order to find out all subset sums coming from A. Let S be the sum of all the n elements of A. Since A consists of positive integers only, the subset sums from A must be in the range $[0, S - 1]$ (here, the zero subset sum corresponds to the null set of indices). B is an array of size $S + 1$. The array entry $B[s]$ is intended to store a count c in the range $[0, 2^{n-1}]$ such that s is the subset sum corresponding to the count c (see Part II for the explanation). Each entry in B is initialized to -1. The recursive function takes the following arguments:

A	The input array
n	The size of A
i	An index in A
B	The array B as introduced above
s	A subset sum from the subarray $(A[0], A[1], \ldots, A[i-1])$
c	A count corresponding to the subset sum s

The function works as follows (in pseudocode). It modifies $B[\ ]$ as the only effect, and needs to return nothing.

If $B[s]$ equals -1, record in $B[s]$ the count c.
If no further recursion is possible, return.
Make the first recursive call in which $A[i]$ is included in the subset sum.
Make the second recursive call in which $A[i]$ is excluded from the subset sum.

After the recursive function returns to *main*() (or a wrapper function), look at the array B. For each index s in the range $[0, S]$ with $B[s] \neq -1$, a subset realizing the sum s is stored as $c = B[s]$. Print the subset corresponding to c.

Submit a single C source file.

Sample output

```
Enter array size (between 1 and 20): 5
A[0] =  6
A[1] =  4
A[2] =  7
A[3] =  4
A[4] =  9
Enter sum: 23
  23 = A[0] + A[1] + A[3] + A[4] = 6 + 4 + 4 + 9
21 different sums are possible:
   0
   4 = A[1] = 4
   6 = A[0] = 6
   7 = A[2] = 7
   8 = A[1] + A[3] = 4 + 4
   9 = A[4] = 9
  10 = A[0] + A[1] = 6 + 4
  11 = A[1] + A[2] = 4 + 7
  13 = A[0] + A[2] = 6 + 7
  14 = A[0] + A[1] + A[3] = 6 + 4 + 4
  15 = A[0] + A[4] = 6 + 9
  16 = A[2] + A[4] = 7 + 9
  17 = A[0] + A[1] + A[2] = 6 + 4 + 7
  19 = A[0] + A[1] + A[4] = 6 + 4 + 9
  20 = A[1] + A[2] + A[4] = 4 + 7 + 9
  21 = A[0] + A[1] + A[2] + A[3] = 6 + 4 + 7 + 4
  22 = A[0] + A[2] + A[4] = 6 + 7 + 9
  23 = A[0] + A[1] + A[3] + A[4] = 6 + 4 + 4 + 9
  24 = A[1] + A[2] + A[3] + A[4] = 4 + 7 + 4 + 9
  26 = A[0] + A[1] + A[2] + A[4] = 6 + 4 + 7 + 9
  30 = A[0] + A[1] + A[2] + A[3] + A[4] = 6 + 4 + 7 + 4 + 9
```