Corrupted clerks

Submission deadline:	**2011-12-04 23:59:59**	442282.584 sec
Evaluation:	**0.0000**	
Max. assessment:	**5.0000** (Without bonus points)	
Submissions:	0 / 10 Free retries + 20 Penalized retries (-2 % penalty each retry)	
Advices:	0 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)	

Your task is to develop a function which helps corrupted clerks maximize their illegal income (bribes).

The clerks of Corruptistan who manage the state contracts do have ideal times. They are assigned a certain budget at the beginning of a year and they are responsible to spend the money. The key point is that no one actually cares how the money were spent. The clerks know it, thus they assign the contracts with only one objection in their minds - to maximize their bribery income. Every contractor in Corruptistan knows that practice and pays the clerk a bribe if the contract is assigned. To avoid anarchy, the Central Committee of Corrupted Clerks published mandatory tables describing the amount of bribe paid based on the contract price. Both clerks and contractors know the table, thus the system is operating smoothly.

Budget is an integer number representing the total budget in millions. A clerk may prepare as many contract as he/she wishes, provided that the total sum of the contracts is equal to the budget. Moreover, each individual contract amount must be a multiple of a whole million.

Clerks are paid bribes for the assigned contracts. The bribe paid depends on the table (above). The table is represented in an array, where the `i-th` element is the bribe paid toe a contract of `i` millions. For example, the contractor of a 4 million contract will pay bribe `array[4]` to the clerk. The table size is limited, amounts exceeding the table size are not allowed to be valid contracts.

The problem is to divide the budget to be bribed as much as possible. The computation is done by a function:

```
int sumBribes ( int budget, int * table, int tableLen )
```

budget
: is the budget, in millions,

table
: is the bribe table, as discussed above. The values are stored in thousands, i.e. the bribe for a 4 million contract will be `table[4]` thousand crowns,

tableLen
: is the size of the table

return value
: is the optimal (maximal) sum of bribes.

Submit a source file containing the implementation of the `sumBribes` function. The source file must contain the function itself and all your supplementary functions needed (called from) the function. On the other hand, the source file shall not contain `#include` preprocessor directives and `main` function (if the `#include` definitions and `main` function are inside a conditional compile block, they may stay in the submitted file). Use the code below as a basis for your development. If the preprocessor definitions and conditional compilation remain unmodified, the file may be submitted to the Progtest.

```
#ifndef __PROGTEST__
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#endif /* __PROGTEST__ */

/* your supplementary functions (if any) */
int sumBribes ( int budget, int * table, int tableLen )
 {
   /* implementation */
 }


#ifndef __PROGTEST__
int main ( int argc, char * argv [] )
 {
   /* your tests */
 }
#endif /* __PROGTEST__ */
```

Your implementation will be included in a testing environment. There will be both time and memory limits when testing your implementation. The time limit may be a problem in this assignment. The time limits are set such that a correct implementation of a naive algorithm passes all mandatory tests. Yo pass the bonus tests, the algorithm must be improved.

Sample function usage:

```
int  r;
int  test1 [] = { 0, 100, 0, 0, 350, 0, 750 };
int  test2 [] = { 0, 1, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 53 };
int  test3 [] = { 0, 0, 0, 0, 0, 0, 0, 1 };
r = sumBribes ( 1, test1, 7 ); /* r = 100 */
r = sumBribes ( 2, test1, 7 ); /* r = 200 */
r = sumBribes ( 3, test1, 7 ); /* r = 300 */
r = sumBribes ( 4, test1, 7 ); /* r = 400 */
r = sumBribes ( 5, test1, 7 ); /* r = 500 */
r = sumBribes ( 6, test1, 7 ); /* r = 750 */
r = sumBribes ( 7, test1, 7 ); /* r = 850 */
r = sumBribes ( 8, test1, 7 ); /* r = 950 */
r = sumBribes ( 9, test1, 7 ); /* r = 1050 */
r = sumBribes ( 10, test1, 7 ); /* r = 1150 */
r = sumBribes ( 11, test1, 7 ); /* r = 1250 */
r = sumBribes ( 12, test1, 7 ); /* r = 1500 */
r = sumBribes ( 13, test1, 7 ); /* r = 1600 */
r = sumBribes ( 14, test1, 7 ); /* r = 1700 */
r = sumBribes ( 15, test1, 7 ); /* r = 1800 */
r = sumBribes ( 16, test1, 7 ); /* r = 1900 */
r = sumBribes ( 17, test1, 7 ); /* r = 2000 */
r = sumBribes ( 1, test2, 16 ); /* r = 1 */
r = sumBribes ( 2, test2, 16 ); /* r = 7 */
r = sumBribes ( 3, test2, 16 ); /* r = 8 */
r = sumBribes ( 4, test2, 16 ); /* r = 14 */
r = sumBribes ( 5, test2, 16 ); /* r = 15 */
r = sumBribes ( 6, test2, 16 ); /* r = 21 */
r = sumBribes ( 7, test2, 16 ); /* r = 22 */
r = sumBribes ( 8, test2, 16 ); /* r = 28 */
r = sumBribes ( 9, test2, 16 ); /* r = 29 */
r = sumBribes ( 10, test2, 16 ); /* r = 35 */
r = sumBribes ( 11, test2, 16 ); /* r = 36 */
r = sumBribes ( 12, test2, 16 ); /* r = 42 */
r = sumBribes ( 13, test2, 16 ); /* r = 43 */
r = sumBribes ( 14, test2, 16 ); /* r = 49 */
r = sumBribes ( 15, test2, 16 ); /* r = 53 */
r = sumBribes ( 16, test2, 16 ); /* r = 56 */
r = sumBribes ( 17, test2, 16 ); /* r = 60 */
r = sumBribes ( 1, test3, 8 ); /* r = 0 */
r = sumBribes ( 2, test3, 8 ); /* r = 0 */
r = sumBribes ( 3, test3, 8 ); /* r = 0 */
r = sumBribes ( 4, test3, 8 ); /* r = 0 */
r = sumBribes ( 5, test3, 8 ); /* r = 0 */
r = sumBribes ( 6, test3, 8 ); /* r = 0 */
r = sumBribes ( 7, test3, 8 ); /* r = 1 */
r = sumBribes ( 8, test3, 8 ); /* r = 1 */
r = sumBribes ( 9, test3, 8 ); /* r = 1 */
r = sumBribes ( 10, test3, 8 ); /* r = 1 */
r = sumBribes ( 11, test3, 8 ); /* r = 1 */
r = sumBribes ( 12, test3, 8 ); /* r = 1 */
r = sumBribes ( 13, test3, 8 ); /* r = 1 */
r = sumBribes ( 14, test3, 8 ); /* r = 2 */
r = sumBribes ( 15, test3, 8 ); /* r = 2 */
r = sumBribes ( 16, test3, 8 ); /* r = 2 */
r = sumBribes ( 17, test3, 8 ); /* r = 2 */
```

Help:

- Naive algorithm tries to divide the budget in all possible ways, sums the bribes and returns the maximum. This algorithm is acceptable for input sizes (budgets) up to approx. 20.
- Use recursion for the division.
- Non-recursive solution is possible, for instance dynamic programming could be used.
- It is not a good idea to solve the (bribe:amount ratio), sort the inputs and divide the budget in a descending order.

Submit: [redacted] **Submit**

Reference