Programming Design, Spring 2015
Homework 4

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

To submit your work, please upload a PDF file for Problems 1 and 2 and two CPP files for Problems 3 and 4 to PDOGS at http://pdogs.ntu.im/judge/. Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is ***8:00am, March 30, 2014***. Please answer in either English or Chinese.

Before you start, please read Sections 5.1–5.7 and 6.5–6.8 of the textbook.[1]

Problem 1

(10 points) Use your own words to explain why in C++ the function signature does not include the function return type. You may want to give an example to show the potential problems if the return type is included.

Problem 2

(15 points; 5 points each) Consider the following implementation of a function that determines whether a given integer is a prime number:

```
void prime(int n)
{
  bool isPrime = true;
  for(int i = 2; i * i <= n; i++)
  {
    if(n % i == 0)
    {
      isPrime = false;
      break;
    }
  }
  cout << isPrime << "\n";
}
```

In the main function, one may invoke `prime()` with an integer argument to see whether that argument is a prime. If it is, we see `1` on the screen; otherwise, we see `0`.

(a) This function couples calculation with the output process. Explain why we say that they are coupled.

(b) Modify the function to decouple calculation with the output process. Briefly explain how to use the new implementation.

(c) Modify the function by removing the `break` statement. Use two `return` statements instead. Briefly explain which one do you prefer.

[1]The textbook is *C++ How to Program: Late Objects Version* by Deitel and Deitel, seventh edition.

Problem 3

(45 points) Recall the knapsack problem introduced in Homework 3. For a knapsack instance with n items, a solution is a n-dimensional binary vector $(x_1, x_2, ..., x_n)$, where $x_i \in \{0, 1\}$ for all $i = 1, ..., n$. We say that a solution y is a *neighbor* of a solution x if there exist an index $j \in \{1, 2, ..., n\}$ such that

$$y_j = 1 - x_j \quad \text{and} \quad y_i = x_i \ \forall i \neq j.$$

For example, for a five-item instance, the solution $(0, 1, 1, 0, 1)$ has five neighbors

$$(1, 1, 1, 0, 1),\ (0, 0, 1, 0, 1),\ (0, 1, 0, 0, 1),\ (0, 1, 1, 1, 1),\ \text{and}\ (0, 1, 1, 0, 0).$$

In general, an n-dimensional solution has n neighbors. Of course, some of these neighbors may be infeasible. Given a solution, you may want to know whether there is any neighbor that is better than the current one. If this is the case, obviously the current one is not an optimal solution.

To compare two solutions, it is possible that the two solutions are equally good, i.e., they are both feasible and generating the same value. In this case, we say that solution x is *prior* to solution y if x selects more low-indexed items. More precisely, x is prior to y if

$$\sum_{i=1}^{n} 2^{n-i} x_i > \sum_{i=1}^{n} 2^{n-i} y_i.$$

For example, the solution $(0, 1, 1, 0, 0)$ is prior to $(0, 1, 0, 0, 1)$.

In this problem, you will be given a solution of a knapsack instance. Among all its neighbors and itself, you need to find the best one, i.e., a feasible one whose value is the highest among these solutions. If there are more than one solution that are equally good, we define the best one as the one that that is best and prior to all other equally good solutions in the considered pool.[2]

As an example, consider the following knapsack instance

$$\begin{aligned} \max \quad & 2x_1 + 4x_2 + 5x_3 + 3x_4 \\ \text{s.t.} \quad & 2x_1 + 3x_2 + 4x_3 + 3x_4 \leq 9 \\ & x_i \in \{0, 1\} \quad \forall i = 1, ..., 4. \end{aligned} \tag{1}$$

For the solution $(0, 0, 1, 1)$, there are four neighbors: $(1, 0, 1, 1)$, $(0, 1, 1, 1)$, $(0, 0, 0, 1)$, and $(0, 0, 1, 0)$. Their total weights are 9, 10, 3, and 4, respectively. This implies that the second neighbor is infeasible. Among all feasible neighbors and the solution $(0, 0, 1, 1)$ itself, the first neighbor is the best (with the highest value 10). Therefore, your program should report $(1, 0, 1, 1)$ as the output.

Input/output formats

There are 10 input files. In each file, there are 4 lines of values. The first three lines define a knapsack instance in the same way as in Homework 3. The fourth line contains a sequence of n binary values $x_i \in \{0, 1\}$, $i = 1, ..., n$ followed by a newline character. Each two values are separated by a white space. This line defines a knapsack solution. You may assume that the given solution is feasible.

Given the input file, you output n binary values in a single line in your output file to represent the best solution among the given solution and its neighbors. The ith value you output is the ith value of that best solution. Each two values should be separated by a white space.

What should be in your source file

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. Moreover, you MUST implement a function

[2] If you want to calculate 2^i, be aware of *overflow*!

```
int knapsackValue(const int value[], const int weight[], int B, int n,
                  const bool sol[])
```

which calculates the total value generated by a given solution. The first four parameters are the values of items, weights of items, the knapsack size, and the number of items; the last parameter is the given solution. If the given solution is feasible, return its value; otherwise, return -1. Your main function should repeatedly invoke this function to complete the above task. For this problem, you are NOT allowed to use techniques not covered in lectures. You should write relevant comments for your codes.

Grading criteria

- 30 points for this program will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each correct output gives you 3 points.
- 5 points for this program depends on whether you declare the required function. If you fail to do that, you lose these 5 points.
- 10 points for this program will be based on how you write your program, including the logic and format. Please try to write a robust, efficient, and easy-to-read program.

Problem 4

(30 points) Consider the knapsack problem again. The knapsack problem is a well-known *NP-hard* problem, which means that most researchers in the world tend to believe that there is no efficient way to find an optimal subset of items. In this case, people may design some *heuristic algorithms* to generate a hopefully near-optimal feasible solution. Many heuristic algorithms are *greedy*, i.e., they apply some myopic search principles to improve a given solution locally but may fail to find a global solution.

In this problem, you will implement a heuristic algorithm based on the concept of neighbors introduced in Problem 3. The algorithm runs in the following way:

```
initialize two knapsack solutions $x = 0$ and $y = 0$ (both selecting no item)
do
    let $x$ be $y$
    let $y$ be the best solution among $x$'s neighbors and $x$
while $y \neq x$
output $y$
```

As an example, suppose that you are working on the knapsack instance in (1). According to the algorithm, you should start from $(0, 0, 0, 0)$, then moves to $(0, 0, 1, 0)$, then $(0, 1, 1, 0)$, and finally stops at $(1, 1, 1, 0)$. Your program should output $(1, 1, 1, 0)$ as its reported solution.[3]

In short, you start from the solution selecting no item. You keep moving to a better candidate which is the best among all your neighbor. You stop when none of your neighbor gives you any improvement. Note that the task inside the `while` loop has been done in Problem 3! This means that you may finish this problem by reusing your codes for Problem 3. Would you make those codes a function and invoke it repeatedly?

Input/output formats

There are 10 input files. In each file, there are 3 lines of values. They define a knapsack instance in the same way as in Homework 3.

[3]For this instance, we are lucky that this algorithm finds an optimal solution. Will we always be lucky to find an optimal solution? In general, no. It is well known that there is a pseudopolynomial-time dynamic programming algorithm that can solve the knapsack problem. For this problem, however, you are required to implement the given algorithm.

Given the input file, you output n binary values in a single line in your output file to represent the solution you find with the given algorithm. The ith value you output is the ith value of your reported solution. Each two values should be separated by a white space.

What should be in your source file

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you are NOT allowed to use techniques not covered in lectures. You should write relevant comments for your codes.

Grading criteria

30 points for this program will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each correct output gives you 3 points.