# Programming Design, Spring 2015
# Homework 6

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

To submit your work, please upload a PDF file for Problem 1 and three CPP files for Problems 2 to 4 (optional) 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, April 20, 2014***. Please answer in either English or Chinese.

Before you start, please read Sections 7.1–7.9 and 7.11 of the textbook.[1] The TA who will prepare the solution for this homework is ***Willy Liao***.

## Problem 1

(15 points; 5 points each) In this problem, we will consider the implementation of insertion sort using pointer arithmetic. Given an unsorted array of length $n$, we will first recursively sort the last $n-1$ elements and then insert the first element to a right place. The recursive function that we implement is here:

```
void insertionSort(int array[], const int n)
{
  if(n > 1)
  {
    insertionSort(array + 1, n - 1);

    int num1 = array[0];
    int i = 1;
    for(; i < n; i++)
    {
      if(array[i] < num1)
        array[i - 1] = array[i];
      else
        break;
    }
    array[i - 1] = num1;
    // Watch here!
  }
}
```

In this function, `array` is the array to be sorted and `n` is the length of `array`.

1. Use the function to sort $(0, 6, 3, 5, 2, 7, 4, 2, 5, 6)$. Write down the values in the array every time when the comment "`// Watch here!`" is met.
2. Use your own words to explain how this function works. In particular, explain what is `array + 1` and how it is related to a subarray.
3. Use your own words to explain why the following implementation is wrong:

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

```
void insertionSort(int array[], const int n)
{
  if(n > 1)
  {
    insertionSort(array + 1, n - 1);

    int num1 = array[0];
    for(int i = 1; i < n; i++)
    {
      if(array[i] < num1)
        array[i - 1] = array[i];
      else
      {
        array[i] = num1;
        break;
      }
    }
  }
}
```

## Problem 2

(25 points) Calling by reference/pointer allows one to modify an argument's value in the caller. One obvious timing to use it, as introduced in the lecture, is to swap the values of two variables. Another timing is for the callee to *return multiple values* to the caller. In this problem, we do a simple exercise on this.

On an $n$-dimensional Cartesian coordinate system, we define the vector from $x \in \mathbb{R}^n$ to $y \in \mathbb{R}^n$ as $d = y - x$, where $d_i = y_i - x_i$ for each $i = 1, ..., n$. For example, for $\bar{x} = (4, 3, 2, 1)$ and $\bar{y} = (1, 2, 3, 4)$, the vector from $\bar{x}$ to $\bar{y}$ is $(1, 2, 3, 4) - (4, 3, 2, 1) = (-3, -1, 1, 3)$.

While this is true for general $n$, in this problem let's focus on the two-dimensional space, i.e., $n = 2$. Given two points $x$ and $y$ in $\mathbb{R}^2$, you should find the vector $y - x$ from $x$ to $y$. However, you are required to implement functions for this task. Recall that a function can only return *one* value. How would you return *two* values (the two coordinates of $y - x$) in a function? The idea is simple: Just call by reference for two variables! More precisely, you should define a function

```
void vecXY(int x1, int x2, int y1, int y2, int& d1, int& d2);
```

In this function, the first four parameters define $x$ and $y$. After some calculations, the function stores the vector in the last two parameters. Then the caller will be able to read the outcome by accessing `d1` and `d2` after the function invocation. This is the power of calling by reference.

To encourage (actually force) you to practice using pointers, you are required to implement another function to call by pointer. Your main function may call either function to complete the task. The TA will grade your program by reading your implementations.

### Input/output formats

There are 10 input files. In each file, there is a line of four integers, $x_1$, $x_2$, $y_1$, and $y_2$. There is a white space between each pair of integers. You may assume that $|x_i| \leq 1000$ and $|y_1| \leq 1000$ for all $i = 1, 2$.

Given the input file, you output two integers $d_1$ and $d_2$, where $d_i = y_1 - x_1$. These two values should be separated by a white space.

As an example, suppose that the input file contains

```
3 5 1 2
```

in a line, then the output file should contain

```
-2 -3
```

in one single line.

**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.

As we mentioned, you must implement the two functions described above.

**Grading criteria**

One special restriction for this problem is to call by reference and call by pointer as we require. If you fail to do this, the TA will give you 0 point (regardless of how many points you get on PDOGS).

Once you implement those two functions, 20 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. For each set of input data, if your program outputs correctly without violating the space limit, you get 2 points.

5 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 3

(60 points) How many friends do you have on Facebook? How many unconfirmed friend requests do you have? Let's write a program to do the counting!

Consider a social networking site. You will be given a set of $m$ users with their IDs. For user $i$, you will be given a list of IDs that user $i$ has sent friend requests to. For example, if that list is $(2, 5, 10)$ for user 1, that means user 1 has sent requests to users 2, 5, and 10. If two users both send requests to each other, they are called a pair of friends in the site.[2] If user $i$ sends a request to user $j$, but user $j$ does not send to user $i$, we say that there is a unconfirmed friend request from $i$ to $j$. Given the set of users and the sets of their friend requests, we are interested in finding the number of pairs of friends and number of unconfirmed friend requests.

One way to store all the friend requests is to build an $m \times m$ matrix $A$, where $A_{ij} = 1$ if user $i$ sends a request to user $j$ and 0 otherwise. However, this requires a lot of memory spaces when $m$ is large. This is especially wasting memory spaces when $A$ is *sparse*, i.e., containing a lot of 0s. In many cases, you are constrained on the available memory spaces your program may use, and thus an $m \times m$ matrix may not be affordable. Is there a better way?

There is one way to save memory spaces: For each user, record only those IDs that a friend request has been sent to. Consider the following example. Suppose we have 1000 users, and each user has sent 3 friend requests in average. If we create an $1000times1000$ integer matrix, we need $1000 \times 1000 \times 4$ bytes, which is approximately 4 MB. Alternatively, we may use a Boolean matrix rather than an integer one. The memory space we need drops to 1 MB. A much better way in this case, however, is for each user only store the targets of her/his friend requests. Then we only need $1000 \times 3 \times 4$ bytes, or roughly 12 KB. Not bad, right?

To implement this, you may create a two-dimensional dynamic array. The number of rows is 1000; the numbers of elements in each row are different from row to row. You will determine the length only

[2] In many social networking sites, one confirms another one's request rather than sending an independent friend request. To make this problem easier, we simply consider that confirmation as a friend request.

after you know the number of requests sent by a given user. The online grading environment has been set with quite limited amounts of available memory spaces. Please complete the task while not wasting unnecessary spaces!

**Input/output formats**

There are 20 input files. In each file, there are $m + 1$ lines of integers. The first line contains a single integer $m > 0$, the number of users. The next $m$ lines contains user IDs and their friend requests. In particular, in line $i$ there are positive integers $i$, $n_i$, $f_{i,1}$, $f_{i,2}$, ..., and $f_{i,n_i}$. Here $i$ is the user ID, $n_i$ is the number of user friends she/he sends, and $f_{ij}$ is the user ID of the target of the $j$th request sent by user $i$. There is a white space between each pair of integers. You may assume that $m \leq 1000$, $0 \leq n_i < m$ for all $i = 1, ..., m$, $f_{ij} \neq i$ for all $i = 1, ..., m$, and $1 \leq f_{ij} \leq m$ for all $i = 1, ..., m$, $1 \leq j \leq n_i$. Moreover, you have $f_{i,1} < f_{i,2} < \cdots < f_{i,n_i}$ for all $i = 1, ..., m$.

Given the input file, you output two integers $a$ and $b$, where $a$ is the number of pairs of friends and $b$ is the number of unconfirmed friend requests. These two values should be separated by a white space.

As an example, suppose that the input file contains

```
4
1 2 2 3
2 1 1
3 0
4 3 1 2 3
```

in five lines, then the output file should contain

```
1 4
```

in one single line.

**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**

One special restriction for this problem is the amount of memory spaces that you may use. The limit is set according to the following principle: If you create an $m \times m$ array, you exceed the limit; if you create a two-dimensional dynamic array as described above, you are passed.

40 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. For each set of input data, if your program outputs correctly without violating the space limit, you get 2 points.

20 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 (bonus)

(20 points) Continue from Problem 3. Given the birthdays of users, you now need to do the some counting by excluding those users who were born before a given date.

**Input/output formats**

There are 10 input files. In each file, there are $m + 2$ lines of integers. The first line contains a single integer $m > 0$, the number of users. The next $m$ lines contains user IDs and their friend requests. In particular, in line $i$ there are positive integers $i$, $b_{i,1}$, $b_{i,2}$, $b_{i,3}$, $n_i$, $f_{i,1}$, $f_{i,2}$, ..., and $f_{i,n_i}$. Here $i$ is the user ID, $b_{i,1}$ is the birth year (YYYY), $b_{i,2}$ is the birth month (M or MM), and $b_{i,3}$ is the birth date (D or DD), $n_i$ is the number of user friends she/he sends, and $f_{ij}$ is the user ID of the target of the $j$th request sent by user $i$. There is a white space between each pair of integers. The last line contains three positive integers $y_1$, $y_2$, and $y_3$. $y_1$ is a year (YYYY), $y_2$ is a month (M or MM), and $y_3$ is date (D or DD). You may assume that $m \leq 1000$, $0 \leq n_i < m$ for all $i = 1, ..., m$, $f_{ij} \neq i$ for all $i = 1, ..., m$, and $1 \leq f_{ij} \leq m$ for all $i = 1, ..., m$, $1 \leq j \leq n_i$. You may also assume that $b_{i,1}$, $b_{i,2}$, and $b_{i,3}$ form a reasonable birthday for a person whose age is between 1 and 100 for all $i = 1, ..., m$. Finally, you may assume that $y_1$, $y_2$, and $y_3$ form a reasonable date. Moreover, you have $f_{i,1} < f_{i,2} < \cdots < f_{i,n_i}$ for all $i = 1, ..., m$.

Given the input file, you output two integers $a$ and $b$, where $a$ is the number of pairs of friends and $b$ is the number of unconfirmed friend requests *only* for among those who were born before (including) the date represented by $y_1$, $y_2$, and $y_3$. These two values should be separated by a white space.

As an example, suppose that the input file contains

```
4
1 2001 2 4 2 2 3
2 2000 1 1 1 1
3 1999 8 31 0
4 1998 10 8 3 1 2 3
2000 1 1
```

in six lines, then the output file should contain

```
0 2
```

in one single line. Note that as user 1 was born after January 1, 2000, all the friend requests related to her/him (including those sent to her/him) are not counted. Also note that user 2 is counted because those who were born on January 1, 2000 should be counted.

**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 may use any techniques you prefer. You should write relevant comments for your codes.

**Grading criteria**

20 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. For each set of input data, if your program outputs correctly without violating the space limit, you get 2 points.