Programming Design, Spring 2014
Homework 3

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

Submission. To submit your work, please upload the following two files to the online grading system at http://lckung.im.ntu.edu.tw/PD/.

1. A .pdf for Problems 1 to 3.
2. Your .cpp file for Problem 4.[1]

Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is ***8:00am, March 12, 2014***. Please answer in either English or Chinese.

Problem 0

(0 point) Please do the following two things:

(a) Starting from the week of March 10, we may schedule individual meetings between TAs and students so that the TAs may give suggestions to the design and style of your programs face to face. The three possible time slots are:

- 4-5pm, Wednesday.
- 9-10am, Thursday.
- 5:30-6:30pm, Friday.

If you are not available in any of the three time slots, please indicate it here and provide the reason (e.g., conflicting with another course). If you have preference, you may also indicate it. However, we do not guarantee to meet your preference.

(b) Please read Sections 6.1–6.4 of the textbook.[2] You probably will see some unfamiliar things. In this case, feel free to guess their meanings or simply skip them. Sections 6.7 and 6.8 are also helpful. If you are wondering what are "functions" in C++, you may skim through the first few pages of Chapter 5, which will formally introduced on March 10. In any case, I strongly suggest you to read the textbook thoroughly before you start to do this homework.

Problem 1

(10 points) Consider the following program:

```
int n = 10;
int num1 = 1;
int num2 = 1;
for (int i = 2; i < n; i++)
{
    num1 = num1 + num2;
    int temp = num2;
    num2 = num1;
    num1 = temp;
}
cout << num2;
```

[1] To celebrate IM Night on 3/10, bonuses are provided in Problems 1 to 3.
[2] The textbook is *C++ How to Program: Late Objects Version* by Deitel and Deitel, seventh edition.

(a) (2 points) For a general value of n (which may not be 10), what does this program output?

(b) (3 points) For the same algorithm, find another implementation that outputs the same thing but runs faster.

(c) (3 points) Implement this algorithm by using an array so that after the execution of your program, the jth element of the array contains the output of the above program with $n = j$ for all $j \in \{3, 4, ..., 10\}$ and 1 for $j \in \{1, 2\}$.[3]

(d) (2 points) Explain why the output with $n = 50$ is negative.

Problem 2

(10 points plus 10 bonus points) Consider the following pseudocode for multiplying two $n \times n$ matrices:

Algorithm for matrix multiplication

Given two $n \times n$ matrices A and B
Declare an $n \times n$ matrix C and initialize all its elements to 0
for i from 1 to n
  for j from 1 to n
  for k from 1 to n
  Update C_{ij} to $C_{ij} + A_{ik}B_{kj}$

Note that we ignore the implementation issue regarding array indices start from 0 in C++. In our pseudocode, A, B, and C are just matrices. After all, no one says that a matrix must be implemented by arrays! Also note that we avoid using statements like $C_{ij} = C_{ij} + A_{ik}B_{kj}$, which may not be meaningful outside the world of C++. We hope our pseudocode works under any implementations with any programming languages!

(a) (5 points) As a function of n, how many multiplications must be performed?

(b) (5 points) Suppose both the two input matrices A and B are diagonal matrices (i.e., $A_{ij} = B_{ij} = 0$ for all $i \neq j$), write a pseudocode for finding $C = AB$ that is faster than the above one. As a function of n, how many multiplications must be performed?

(c) (10 bonus points) Suppose both the two input matrices A and B are upper triangular matrices (i.e., $A_{ij} = B_{ij} = 0$ for all $i > j$), write a pseudocode for finding $C = AB$ that is faster than the above one. As a function of n, how many multiplications must be performed?

Problem 3

(10 bonus points) Consider the program provided on page 21 of the slides used on March 3:

```
int array[100] = {0};

for (int i = 0; i < 500; i++)
{
  cout << array[i] << " ";
  if (i % 10 == 9)
    cout << "\n";
}
```

Try to execute it (and probably modify the total number of iterations of the loop) and get the operating system force your program to terminate. Print your screen with the error message generated by the operating system. Then briefly explain what run time error results in the termination.

[3] That is, the array should contain 1, 1, 2, 3, 5, ..., and 55.

Problem 4

(80 points) In this homework, we want to start our journey for building a computer game "IM Crush", a simpler version of the computer game "Candy Crush Saga".[4] Our game is played in a 5×5 board, where its 25 squares are labeled as

$$\begin{bmatrix} 21 & 22 & 23 & 24 & 25 \\ 16 & 17 & 18 & 19 & 20 \\ 11 & 12 & 13 & 14 & 15 \\ 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}. \tag{1}$$

In other words, the left-bottom square is labeled as the 1st square, the square to its right is labeled as the 2nd one, etc. When the game starts, candies with four different colors will randoms be placed on the board. These four types of candies will be called candies 1, 2, 3, and 4, respectively. As an example, one possible initial state of a game is

$$\begin{bmatrix} 1 & 2 & 3 & 1 & 4 \\ 4 & 2 & 4 & 2 & 2 \\ 3 & 4 & 2 & 1 & 2 \\ 2 & 4 & 3 & 1 & 3 \\ 3 & 2 & 2 & 1 & 2 \end{bmatrix}. \tag{2}$$

As in Candy Crush, in IM Crush several candies will be removed together when they form a row or a column of three, four, or five same-color candies. In our example, the three candies in squares 4, 9, and 14 form a column of three same-color candies and should be removed from the board. Suppose the three new candies that will fall down are 2, 4, and 3, the next state of this game will be

$$\begin{bmatrix} 1 & 2 & 3 & 3 & 4 \\ 4 & 2 & 4 & 4 & 2 \\ 3 & 4 & 2 & 2 & 2 \\ 2 & 4 & 3 & 1 & 3 \\ 3 & 2 & 2 & 2 & 2 \end{bmatrix}. \tag{3}$$

For this state, we see that the four candies in squares 2, 3, 4, and 5 and the three candies in squares 13, 14, and 15 should be removed at the same time. In IM Crush, points are earned when candies are removed. When n candies are removed at the same time, $n - 2$ points are earned. For example, when we remove the three candies in (2), we earn 1 point; when we remove the seven candies in (3), we earn 5 points.

Sometimes a row and a column of three or more same-color candies intersect. In this case, all those candies are removed together. For example, if a state is

$$\begin{bmatrix} 1 & 2 & 3 & 3 & 4 \\ 4 & 2 & 4 & 4 & 2 \\ 3 & 4 & 2 & 2 & 2 \\ 2 & 4 & 2 & 1 & 3 \\ 3 & 2 & 2 & 2 & 2 \end{bmatrix}, \tag{4}$$

then candies in squares 2, 3, 4, 5, 8, 13, 14, and 15 are removed at the same time. Six points are then earned.

In the true IM Crush game, we will allow a player to exchange candies neighboring to each other. We will leave this to the future and focus on detecting candies to remove, counting points, and replenishing candies. Given a sequence of integers between 1 and 4, your program needs to determine the initial state of the game (according to the first 25 integers), which candies to remove, the state after removing candies (according to the n following integers if n candies are removed), and so on until no more candies can be removed. During the process, you need to calculate the total points earned.

[4] http://www.candycrushsaga.com/.

Input/output formats

The input will consist of 35 lines of integers between 1 and 4. In each line, there are $m+1$ integers, where two consecutive integers are separated by a white space. It is given that $m \geq 25$, but you do not know the value of m. Let $(x_1, x_2, ..., x_m, 0)$ be the integers given in a line, where $x_i \in \{1, 2, 3, 4\}$ is the ith integer. Below we describe the steps your program needs to follow:

- **Step 1.** The board is initialized by taking the first 25 integers and let them fall down to the board, starting from the left-most column. In other words, the initial state is

$$\begin{bmatrix} x_5 & x_{10} & x_{15} & x_{20} & x_{25} \\ x_4 & x_9 & x_{14} & x_{19} & x_{24} \\ x_3 & x_8 & x_{13} & x_{18} & x_{23} \\ x_2 & x_7 & x_{12} & x_{17} & x_{22} \\ x_1 & x_6 & x_{11} & x_{16} & x_{21} \end{bmatrix}. \tag{5}$$

  The total points earned is initialized to $p = 0$ and the index of the last used candy is set to be $k = 25$.

- **Step 2.** Find those candies that should be removed. If no candy can be removed, this game ends. Otherwise, some candies are removed and those corresponding squares become empty. Let n be the number of candies removed, the total points earned are then updated to $p + n - 2$. Remaining candies should fall down to fill up empty squares until no empty space has any candy above it.

- **Step 3.** New candies should be taken from the sequence $\{x_i\}_{i=1,2,...}$ to fill up empty squares. Given the index of the last used candy k, if n candies were removed in Step 2, integers $x_{k+1}, x_{k+2}, ..., x_{k+n}$ should be taken in order from the input and be placed on the board. If the m integers are all used (i.e., 0 is read) and the board cannot be filled, the game ends immediately. Otherwise, these integers fall down in order to fill empty squares starting from the left-most column, then the second left-most column, ..., and lastly the right-most column. Your program then go back to Step 2.

Let's use an example to illustrate the rule. A line of input which starts with

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

results in the initial state in (2). In Step 2, we find that candies in squares 4, 9, and 14 should be removed. after removing these three candies, the board becomes

$$\begin{bmatrix} 1 & 2 & 3 & & 4 \\ 4 & 2 & 4 & & 2 \\ 3 & 4 & 2 & & 2 \\ 2 & 4 & 3 & 1 & 3 \\ 3 & 2 & 2 & 2 & 2 \end{bmatrix}, \tag{6}$$

where squares 14, 19, and 24 are empty. We update the total points earned to 1. As there are three empty squares, in Step 3 we read the following three integers 2, 4, and 3 and let them fall down to these empty squares and obtain

$$\begin{bmatrix} 1 & 2 & 3 & 3 & 4 \\ 4 & 2 & 4 & 4 & 2 \\ 3 & 4 & 2 & 2 & 2 \\ 2 & 4 & 3 & 1 & 3 \\ 3 & 2 & 2 & 2 & 2 \end{bmatrix}. \tag{7}$$

We then go back to Step 2 and find that seven candies should be removed. We remove them and obtain

$$\begin{bmatrix} 1 & & & & \\ 4 & 2 & & & \\ 3 & 2 & 3 & 3 & 4 \\ 2 & 4 & 4 & 4 & 2 \\ 3 & 4 & 3 & 1 & 3 \end{bmatrix} \tag{8}$$

while updating the total points earned to $1+5=6$. As there are seven empty squares, we do step 3 and read the following seven integers 1, 2, 3, 4, 1, 2, and 3. They fill up the empty squares and make the board become

$$\begin{bmatrix} 1 & 1 & 3 & 1 & 3 \\ 4 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 4 \\ 2 & 4 & 4 & 4 & 2 \\ 3 & 4 & 3 & 1 & 3 \end{bmatrix}. \tag{9}$$

We continue the process and observe that candies in squares 7, 8, and 9 should be removed. We do so, update the total points earned to 7, and place the next three integers 2, 3, and 4 into the board. We obtain

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 3 \\ 4 & 1 & 3 & 1 & 2 \\ 3 & 2 & 2 & 4 & 4 \\ 2 & 2 & 3 & 3 & 2 \\ 3 & 4 & 3 & 1 & 3 \end{bmatrix} \tag{10}$$

and find that no candy should be removed. The game then terminates with 7 points and 38 integers used in this game. Please note that it is possible that some integers in a line are not used when that game ends. In this case, you need to skip the remaining integers (including the 0 at the end) and move to the next line directly.[5]

Besides finding no more candies to remove, another possibility for terminating a game is when 0 (the last integer in each line) is read. For example, suppose a line of input consists of

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

and the initial state is

$$\begin{bmatrix} 1 & 2 & 2 & 3 & 3 \\ 4 & 2 & 1 & 1 & 2 \\ 3 & 1 & 4 & 4 & 1 \\ 2 & 1 & 3 & 3 & 4 \\ 1 & 1 & 2 & 2 & 3 \end{bmatrix}. \tag{11}$$

We find that the candies in squares 2, 7, and 12 can be removed and the state after removing them is

$$\begin{bmatrix} 1 & & 2 & 3 & 3 \\ 4 & & 1 & 1 & 2 \\ 3 & & 4 & 4 & 1 \\ 2 & 2 & 3 & 3 & 4 \\ 1 & 2 & 2 & 2 & 3 \end{bmatrix}. \tag{12}$$

Note that the three candies in squares 2, 3, and 4 can be removed BUT, according to our rule, we need to first go to Step 3 to fill up empty squares. While we need three more integers, we find that we have only one integer (4) left. As the board cannot be filled, the game ends immediately and no more candy should be removed.

After processing a line of input, your program should output two integers, separated with a white space, to indicate the total points earned and number of integers used in the game. For our first example, the output should be

```
7 38
```

with a new line character appended at the end. For our second example, the output should be

```
1 26
```

[5] One C++ function `cin.getline()`, defined in `<iostream>`, provides an alternative for skipping unused integers. For an introduction to `cin.getline()`, see, e.g., http://www.cplusplus.com/reference/istream/istream/getline/.

with a new line character appended at the end. Note that when a game ends because 0 is read, the second output should be m instead of $m + 1$.

More input/output examples are provided in the associated .txt files. In the input numbers are contained in "PDSp14_hw03_input.txt", your program should output exactly those things contained in "PDSp14_hw03_output.txt".

What should be in your source file

For this problem, your .cpp source file should contain C++ codes that will both read testing data and complete the above task. You are welcome to use any technique you know. Finally, you should write relevant comments for your codes.

Grading criteria

- 70% of your grades for this program will be based on the correctness of your output. PDGOS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each fully correct line of output gives you 2 points.
- 30% of your grades 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.