Programming Design, Spring 2014
Lab Exam 3

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

Submission and grading. In this exam, there are three problems. You need to write a C++ program for each problem. 100% of your grades will be based on the correctness of your outputs. The online grading system will input in total 50 lines of testing data and then check your outputs. These 50 lines count for 100 points, i.e., 2 points for each set. Before the due time of the exam, you may upload your programs multiple times. Only the last one you upload will be graded. Unlike what happens with your homework, you will not see your scores during the exam.

To submit your work, please upload the three .cpp files, one for each problem, to the online grading system at http://lckung.im.ntu.edu.tw/PD/.

Discussing the exam with anyone during the exam period, directly or indirectly, is definitely cheating.

Problem 1

(30 points) In a baseball game, one hitter may have several at bats (AB) and several hits (H). One's batting average (AVG) is the number of hits divided by the number of at bats. In this problem, you will be given data of at bats and hits for several hitters. Your goal is to record these data for evaluating hitters' performance.

Input/output formats

The input consists of many "record" lines and 15 "batting average" lines. A record line starts with a letter `R`, followed by a hitter's name (a sequence of English letters with no space), a number of at bats (an integer between 1 and 6),[1] and a number of hits (an integer between 0 and the number of at bats). This is the player's batting record in a game. The four pieces of data are separated by three white spaces. Your program should read these record lines and update these hitters' accumulated records accordingly. A batting average line starts with a letter `A`, followed by a hitter's name. When you see this, output the hitter's name, accumulated at bats, accumulated hits, and then a single letter `Y` if his batting average is no less than 0.3 or `N` otherwise. These four pieces of information should be separated by two spaces. You may assume that the name in a batting average line appears in at least one record line above it.

For example, suppose we have

```
R George 4 2
R Tom 3 1
R George 2 2
R Tom 3 0
A Tom
A George
R Tom 4 2
A Tom
```

as input, the output should be

```
Tom 6 1 N
George 6 4 Y
Tom 10 3 Y
```

[1] In a real baseball game, one may get no at bat. To make the problem easier, let's assume one gets at least one at bat.

with a new line character appended at the end of each line. Note that for the second time we see `A Tom`, Tom's batting average is $\frac{3}{10}$, which is exactly (and thus no less than) 0.3. Therefore, we output `Y` rather than `N`.

Note. To avoid the precision issue of division, you are suggested to multiply 0.3 by the number of at bats and compare the result with the number of hits.

Problem 2

(20 points) For this problem, each line of input contains one single positive integer $n \leq 40$. Your program should output $f(n) - n^2$, where $f(n)$ is the nth Fibonacci number (the first five Fibonacci numbers are 1, 1, 2, 3, and 5).

Note. There is no trick in this problem; it is as easy as you think!

Problem 3

(50 points) In this problem, we will consider some 5×5 mazes with 25 squares, whose labels are shown in Figure 1. For all the mazes, we look for a route from square 1 to square 25. At a square, one may move to an adjacent square (two adjacent squares share one edge; e.g., squares 13 and 14 are adjacent, but squares 13 and 19 are not). If all squares can be passed through, the problem is very simple. However, if some squares are blocked, finding a route becomes more challenging. As an example, consider the maze in Figure 2, in which blocked squares are crossed over. Our goal is to find a route that goes from square 1 to square 25 and do not pass through any blocked squares. Of course, there may be multiple routes (e.g., routes $(1, 2, 3, 4, 5, 10, 15, 14, 13, 18, 23, 24, 25)$ and $(1, 6, 11, 12, 13, 18, 23, 24, 25)$ are two routes). In this case, finding one route is enough.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Figure 1: Maze 1

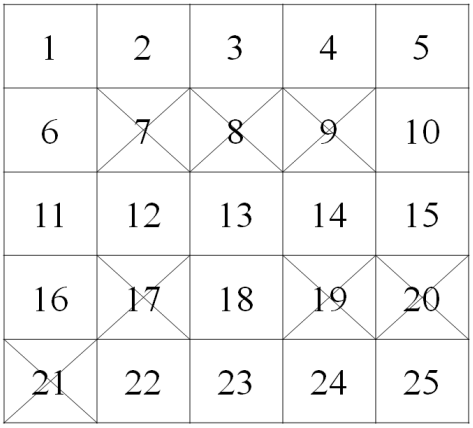

Figure 2: Maze 2

Input/output formats

The input consists of 25 lines. In each line, there are $k+1$ integer k, b_1, b_2, ..., and b_k. All these integers are between 1 and 25. Two consecutive integers are separated with a white space. With such a line, your program should create a maze with squares b_1, b_2, ..., and b_k blocked. For example, a line

```
7 7 8 9 17 19 20 21
```

creates a maze in Figure 2. You may assume that $b_1 < b_2 < \cdots < b_k$ and all given mazes have at least one route. For each maze, your program should output a route from square 1 to square 25 without passing

through any blocked squares. To output a route, list all the squares you pass through in order with white spaces separating two consecutive output numbers. For the maze in Figure 2, you may output either

```
1 2 3 4 5 10 15 14 13 18 23 24 25
```

or

```
1 6 11 12 13 18 23 24 25
```

as a route. While the second one is shorter than the first one, both of them are treated as correct and earn you two points.

As another example, if an input line

```
5 4 8 12 19 24
```

is given, your program may output

```
1 6 11 16 17 18 13 14 15 20 25
```

or any other route as a solution.

To make it easier to earn points, the 25 lines are organized as follows:

- For each of the first five lines, there is just one block.
- For each of the sixth to fifteenth lines, there are one to four blocks.
- For each of the last ten lines, there is no restriction on the number of blocks.