April 2010 Packet

Computer Science Competition
Hands-On Programming Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Point Values and Names of Problems

Number	Name	Point Value
Problem 1	A Greeting in the Dark	60
Problem 2	Objection	60
Problem 3	Submissions	60
Problem 4	A Prime Opportunity	60
Problem 5	Power Squared	60
Problem 6	Roman Homework	60
Problem 7	The Ending Sounds Like the Beginning	60
Problem 8	Possibility of Donuts	60
Problem 9	Homework Time	60
Problem 10	Balloono	60
Problem 11	Wikiracing!	60
Problem 12	Superstitious Thirteen	60
		Total 720

Problem #1
60 Points

A Greeting in the Dark

Program Name: greeting.java
Input File : none

"Hello, world!" is one of the most common programs created by programmers. A new rendition has been created, one ought to try recreating it.

Input
There is no input for this problem.

Output
Print out "Oh hai world!" in the format shown below
(The distance between the words is 6 spaces)

Example Input File
```
none
```

Example Output To Screen
```
  OO   H  H      H  H AAAA III      W   W   OO   RRRR L    DDDD !!
 O  O  H  H      H  H A  A  I       W   W  O  O  R  R L    D  D !!
O    O HHHH      HHHH AAAA  I       W W W O    O RRRR L    D  D !!
 O  O  H  H      H  H A  A  I       W W W  O  O  R R  L    D  D
  OO   H  H      H  H A  A III      WWWWW   OO   R  R LLLL DDDD !!
```

Problem #2
60 Points

Objection

Program Name: objection.java
Input File: objection.in

Once again, a programming team is arguing of the correctness of their programming solution. In order to tell contest with the judges, they will cry "Objection!" until to various points until the judges finally just cry out "HOLD IT!" at their futile attempts. The judges only can take so many objections though. However, if the Objections they can stand is less than or equal to 0, it means that the team was actually right! Thus a cry of "GUILTY!" will be cried out since the judges falsely declared the problem incorrect.

Input
The first number will tell you the number of times the program will run. Afterwards, each number will tell you how many times the team was able to cry "Objection" before a "HOLD IT!" was declared, or if the team screamed "GUILTY!" when they discovered their problem was correct. The number will always be an integer.

Output
The first integer will tell you the number of trials. From then on, the number will tell the number of "Objection!" 's that will be shouted before "HOLD IT!" is finally shouted. If the number is less than or equal to 0, a "GUILTY!" is shouted instead. A blank line will separate all trials.

Example Input File
```
3
1
5
-3
```

Example Output To Screen
```
Objection!
HOLD IT!

Objection!
Objection!
Objection!
Objection!
Objection!
HOLD IT!

GUILTY!
```

Problem #3
60 Points

Submissions

Program Name: submit.java
Input File: submit.in

You are an administrator for MLIA. Your goal is to write a program to check for the correct formatting of submitted entries. A correctly formatted entry should begin with the word "Today", and the last word should be "MLIA"

Input
First will be the number indicating the number of trials. After that, each line will contain a separate submission that needs to be verified.

Output
If the submission starts with "Today" and ends with "MLIA" (case insensitive) the program should print out `"VALID ENTRY"`
Otherwise, if the submission fails to meet that requirement, the program should print out `"INCORRECT FORMATTING, TRY ANOTHER SUBMISSION"`

Example Input File
```
5
ToDAY, I went to school.  mlia
Hehehe today mlia this shouldn't work
Today, I went to a programming contest.  Hehe.  MLIA
TODAYMLIA
T0day is a brand new day! MLIA
```

Example Output To Screen
```
VALID ENTRY
INCORRECT FORMATTING, TRY ANOTHER SUBMISSION
VALID ENTRY
VALID ENTRY
INCORRECT FORMATTING, TRY ANOTHER SUBMISSION
```

Problem #4
60 Points

A Prime Opportunity

Program Name: prime.java
Input File: prime.in

Lay Z. Bee, like many students before him has decided him math homework is taking too long to do by hand, and would rather devise a program to do it for him. His homework is to do the prime factorization of a given integer.

Input
The first number will be the number of problems Lay Z. Bee must do. After that, each number is an integer for which he must find all the prime factors for.

Output
Print out all the prime factors for each number on its own line. If a prime is factored twice out of a number, it will be printed twice. (Example: 4 is 2 2, 16 is 2 2 2 2). The factors should be in printed in ascending order.

Example Input File
```
5
7
84
121
1000
96
```

Example Output To Screen
```
7
2 2 3 7
11 11
2 2 2 5 5 5
2 2 2 2 2 3
```

Problem #5
60 Points

Power Squared

Program Name: power.java
Input File: power.in

A crazy math teacher has come up with a new math formula that he has called power squared. Given an integer x, the student must calculate x to the power of (x-1) to the power of (x-2)…all the way to the power of 1
Ex. 3 -> (3^2)^1
5 -> ((((5^4)^3)^2)^1)

Input
The first integer will tell the number of trials to come. After that, the input will be an integer that is to have power squared performed on it.

Output
Print out the resulting power squared and then a blank line in between each trial.

Example Input File
```
3
1
3
6
```

Example Output To Screen
```
1

9

2388636399360109977557402041718133080829429159844757507642063199359529632522467783435119230976
```

Problem #6
60 Points

Roman Homework

Program Name: roman.java
Input File: roman.in

A young boy is learning addition. Unfortunately, being in the Roman Empire, he is doing addition using Roman numerals! Being a kind person, you volunteered to create a program to do it for him!

Of course, since he is a young boy, the total sum will never add up to be more than 2000 in Arabic numeral system.

Input
The first value will be the number of problems that have to be solved. Each line after that will contain two Roman numerals separated by a space that need to be added together

Output
Output the sum in Roman numeral notation

Example Input File
```
5
XL L
XXIV I
CDXLIV II
M CMIII
I X
```

Example Output To Screen
```
XC
XXV
CDXLVI
MCMIII
XI
```

Problem #7
60 Points

The Ending Sounds like the Beginning

Program Name: core.java
Input File: core.in

In String matching, sometimes the idea of a core is used. Essentially, it is when the prefix of a string is identical to the suffix of a String. (But the core cannot be the String itself).

Find a way to compute the core of a given String

Input
The first integer will tell how many Strings are to be tested. After that, on each line is a String for which the core should be calculated

Output
Output the core of the String. If a core cannot be found for that String, print out instead "NO CORE FOUND'. Each result should be printed on its own line.

Example Input File
```
5
abababa
RAWR!
abcdefghijklmnopqrstuvwxyz
haha I win! haha
catch it, did you catch it
```

Example Output To Screen
```
ababa
NO CORE FOUND
NO CORE FOUND
haha
catch it
```

Problem #8
60 Points

Possibility of Donuts

Program Name: donuts.java
Input File: donuts.in

On the way to the programming contest, a sponsor stopped on the way to pick up some donuts. Given how many donuts he or she wants to order and the number of varieties of donuts there are, calculate the number of different combinations the sponsor could order donuts (order does not matter). The sponsor can order some of the varieties, all of the varieties, only one variety. The sponsor does not have to order some donuts of each variety.

The following formula might be useful:

$$\mathbf{C}(n,k) = \mathbf{C}_n^k = \mathbf{C}_k^n = {}_nC_k = \binom{n}{k} = \frac{n!}{k!(n-k)!},$$

This is the combinations formula, which calculates the total number of ways to arrange k items out of n items. (Example: pick k people out of a possible of n for a committee)

N will always be greater than or equal to k. N and K will never equal 0.

The number of possibilities will never exceed $2.43290201 \times 10^{18}$

Input
The first line will be an integer telling how many trials will be run. On each line after that, there will be two integers, the first telling how many donuts the sponsor wants to pick out and the second will tell how many varieties of donuts there are.

Output
The number of ways the sponsor could order donuts followed by the phrase "way(s) to order donuts"

Example Input File
```
4
4 2
12 4
16 1
11 10
```

Example Output To Screen
```
5  way(s) to order donuts
455  way(s) to order donuts
1  way(s) to order donuts
167960  way(s) to order donuts
```

Problem #9
60 Points

Homework Time

Program Name: homework.java
Input File: homework.in

A college student is working over the weekend to complete his homework. He is going to work on his homework. However, he has a few "quirks" about how he does his homework. He will work on the most important assignment at that time, unless he is assigned a more important assignment. Then, he will quit his current assignment and start working on the new assignment. This student is a perfectionist though, and will only work an assignment from start to finish, so once an assignment has been stopped, when the student starts again, he must start from the beginning.

Write a program to tell how long the student will take to complete his homework and if he will have enough time.

The student has 2880 minutes to complete all of his assignments. He will work nonstop, unless he has nothing else to do.

Input
The first line of input will contain a single integer which will tell indicate the number of trials that will be run. The remainder of the input is for the trails.

The data set of each trial will be formatted as such:

1. A single integer that will indicate how many assignments the student will be assigned
2. All of the next lines will correspond to an assignment with three integers where:
    a. The first integer is equivalent to the number of minutes that have passed since the student started working. The assignments are listed in the order they come in. (A student cannot start working on the assignment until they have been assigned it)
    b. The second integer is the priority of the assignment (1-5), where 1 is a high priority assignment (worth more of the grade or in a class that is at risk)
    c. The third integer is the amount of uninterrupted time need to finish the assignment

Output
For each data set print out the following:
The number of minutes it will take to finish followed by the phrase " minutes." If the amount of time required is more than 2880, then print out `Not enough time to complete all the homework!`. If there is enough time, print out `There is enough time to finish homework.` (All of this is on the same line)

Other:
If multiple assignments are awaiting completion, the one with the higher priority is finished first. If multiple assignments of the same highest priority, then the one with the shortest completion time is completed first. An assignment is only interrupted when a higher priority assignment is assigned. When this happens, the work on the lower priority assignment will be halted and the student will start working on the higher priority assignment. An assignment must be worked on uninterrupted for the number of minutes specified to be successful completed.

Example Input File

```
3
4
0 5 120
60 3 480
700 4 60
1000 1 1200
3
2800 5 10
2809 2 60
2810 1 1
6
0 5 1000
0 5 100
110 3 240
400 2 360
500 1 1000
2860 4 20
```

Example Output To Screen

```
2200 minutes. There is enough time to finish homework.
2881 minutes. Not enough time to complete all the homework!
2880 minutes. There is enough time to finish homework.
```

Problem #10
60 Points

Balloono

Program Name: balloono.java
Input File: balloono.in

You've recently been online playing a lot of Balloono. (Balloono is a lot like the game Bomberman – After you get hit by a balloono you have to float 5 seconds in a balloon and in that time an opponent can pop you). Assuming an opponent can move 1 space per second – figure out if you're going to get popped. The opponent must be at your space at the 5th second.

'O' = Your character floating in a balloon
'X' = An opponent (An opponent can share a space with another opponent)
'#' = A wall – an opponent cannot walk through this
'B' = A balloon – an opponent cannot walk through this
'.' = An open space – anyone can walk on this

Input
The first line will contain an integer tell the number of trials to be run. For each following data set for a trial: the first part will contain an integer that will tell the size of the board. (The board is a square) The following lines (the same number as the first integer will contain the board)

Output
Print out "SAFE!" if your character will not be reached in time or "POPPED!" if an opponent will reach your character.

Example Input File

```
6
15
###############
###############
##O#############
##.......##X###
##.........###
#B....#...###.#
#....X........#
#...###...#####
#.#.#.#.#.#.#.#
#.#.#.#.#.#.#.#
#.#.#.#.#.#.#.#
#.#.#.#.#.#.#.#
#.#.#.#.#.#.#.#
#.#.#.#.#.#.#.#
###############
4
####
#O.#
#.X#
####
```

```
9
#########
#O#.#.#.#
#.....X.#
#.#.#.#.#
#.......#
#X#.#.#.#
#.....B.#
#.#.#.#.#
#########
3
###
#O#
###
9
#########
#..O....#
#..#....#
#..#....#
#..#....#
#.X.X...#
#.......#
#.......#
#########
5
#####
#O###
#..X#
#.X.#
#####
```

Example Output To Screen

```
SAFE!
POPPED!
POPPED!
SAFE!
POPPED!
POPPED!
```

Problem #11
60 Points

Wikiracing!

Program Name: wiki.java
Input File: wiki.in

This is a race! Your friend and you are in an intense wikiracing contest. Your goal is to get from one article to another article as fast as you can by only clicking links. Being the conniving sort of person, you want to know the way that will require the fewest clicks. Occasionally though, you will be looking for or starting at an article that doesn't exist.

Input
The first line is an integer. It will tell how many articles will be loaded into the wiki. For each article, links are bidirectional (If A links to B, then B will also link back to A) On the line for an article, the first String will tell the name of the article, and all the following Strings will be all the articles it links too.

After loading the wiki, there will be another line with an integer. This single integer will tell how many races are to occur. On the line for a race, the first String is the starting article and the second String is the ending point.

Output
Print out
The fastest route was (# of fastest route) clicks.
If the starting or ending article doesn't exist print
One or more articles does not exist

(If the starting and ending article exist, there will be a way to get from one to another)

Example Input File
```
6
orange apple fruit color health
apple red health fruit doctor
doctor disease health day
disease staph swine_flu health
day moon sun school sleep work
sleep zzz dreams pillow naps
8
orange zzz
staph color
doctor swine_flu
day sun
ham swine_flu
orange apple
moon rabbit
apple apple
```

Example Output To Screen
```
The fastest route was 5 clicks.
The fastest route was 4 clicks.
The fastest route was 2 clicks.
The fastest route was 1 clicks.
One or more articles does not exist.
The fastest route was 1 clicks.
```

```
One or more articles does not exist.
The fastest route was 0 clicks.
```

Problem #12
60 Points

Superstitious Thirteen

Program Name: thirteen.java
Input File: thirteen.in

Along with the 2012 apocalypse theories coming about, a new superstition has erupted. The dread of the worst day ever – the 13th Friday the 13th. Given a month (starting on the first) find out when the 13th Friday the 13th will occur. (If a Friday the 13th happens the given month – it is counted as the first instance)

Input
The first line will contain an integer telling how many trials are to follow. On the line for a trial, the first part will tell the month and the second part will be the year.

The earliest date given will be January 1900 – which was a Saturday.

Output
Output the month and year when the 13th Friday the 13th will occur.

Example Input File
```
6
January 2000
August 1990
October 3456
May 1996
February 2004
May 2008
```

Example Output To Screen
```
August 2007
April 1998
April 3463
July 2003
June 2011
March 2015
```