2nd Annual Cornell University
High School Programming Contest

2015, April 7, 9-noon

DO NOT OPEN UNTIL CONTEST BEGINS AT 9AM

Welcome to the programming contest, and good luck! In this package you'll find a set of programming problems. Even though some problems are harder than others, all problems are worth the same amount. You can upload solutions using the "pc2team" software---you receive a separate username and password for that. Don't forget to select the programming language that you are using (Java, Python, C, or C++). You should then receive feedback from our automated judges very quickly. You can also submit questions that will be answered by our live judges.

Each solution takes a single input (read from standard input), and produces a single output (written to standard output). The input for a problem will contain multiple test cases. For example, if the problem is to some up a list of integers, the input might look like:

```
3
2 5 4
4 2 1
3 3 3
```

And the output in this case might be something like

```
11
7
9
```

Spaces, new-lines, etc. are generally ignored except to delimit the input fields. Read the instructions carefully to understand the format of the input and output. We provide for each problem at least one sample input and output.

If you would like a print-out, options are unfortunately limited. Also, we're trying to save as much paper as possible. To request a print-out, submit your program to the "PrintMe" problem. It's a fake problem that we use only for printing purposes.

RULES

Team formation: To compete you should be a team of (up to) three students enrolled in the same institution. You may bring with you as much printed material as you like (books, old codes you have developed, course notes, etc.), but no electronic device will be allowed. That means that any laptop, phone, smartwatch, iphone, USB drive or any other form of media will not be allowed on the competition floor.

The problem set: The problem set will vary in difficulty. **The problems are NOT presented in order of difficulty.** You may solve your problems in any order you want, and each problem awards you with the same amount of points. More about grading later.

Submitting: After reading a problem statement, if you decide to approach it, you should code your solution in one of the accepted languages (C, C++, Java or Python). After you test your solution in your computer, and think that it is correct, you may submit your code for judging. Upon receiving your code the automatic system will:

1. Compile your code (if not in Python)
2. Execute your code with a prepared set of hard input (that are hidden from the contestants)
3. Compare the output your program generated with the expected output.

Note that this will be done by an automatic judge, no human will read your code. After the judging (usually a few seconds, but may be more depending on the judging queue) you will receive a verdict which can be:

- **NO - Compilation Error** - Your code didn't compile
- **NO - Time Limit Exceeded** - Your code took longer to run than the allowed time limit
- **NO - Wrong Answer** - Your code ran in time, but it output a wrong solution for at least one of the test cases.
- **NO - Runtime Error** - Your code ran into a runtime error, for instance, a division by 0, or a syntax error in the case of Python.
- **YES** - Your code compiled and ran correctly, and output a correct answer.
- **NO - Other (Please see staff)** - unpredictable things can happen and we have to be prepared.

We will be using the system PC2 for controlling the contest. You are encouraged to read their manual at http://www.ecs.csus.edu/pc2/doc/v9/PC2V9TeamGuide.pdf

Scoring: The first criterion for scoring is the total number of problems with a **YES** verdict. If team A solved 4 problems and team B solved 3 problems, team A will be ahead of team B in the ranking, regardless of which problems they solved, and of the time it took them.

The tie breaking is done by considering the "penalty". Penalty has two components:

1. You will receive a penalty of 20 for each "NO" submission **of a problem you eventually solve.** If you don't eventually solve the problem, you won't be penalized for wrong submissions.
2. If you receive a "YES" on a problem **X** minutes after the competition started, you will receive a penalty of **X**.

In case of ties in number of problems, teams will be ranked by smallest penalties. In the unlikely case of ties in both number or problems and penalties, the judges may decide to make a (subjective) determination based on the quality of the submitted code.

Clarifications: In case of ambiguity in a problem statement, you may ask for a clarification through the PC2 system. Clarifications go to the judge anonymously and are completely free. As a rule of thumb, in case of doubt, ask for a clarification. In the worst case, you will receive back something that is not helpful.

At your station you will have at your disposal:

- One computer (note that there will be only one computer per team of three people)
- Three copies of the problem set, printed. These copies are yours to keep, and you can write on them as you will.
- One calculator
- Sheets of paper and pencils.

Apart from that, you will be able to print anything you want (for instance, to analyze code while another team member is working on another problem on the computer). Please minimize your printing. You are ***not*** allowed to use the Internet.

Some Quick Hints

- Log in with the username and password provided to you.
- For editing, you have many choices, including
    - gedit: a simple to use graphical editor. Can be started from a terminal window.
    - Eclipse: a complete programming environment. Should be in sidebar.
    - vi: old school…

  You can ask a staff member for help with these.
- If you need to print something, submit the solution to the PrintMe problem. Please try to minimize printing.
- If you need to go to the bathroom, please ask one of the attendants to accompany you.
- Clarifications can be submitted through pc2team. This is preferred over asking attendants.
- All programs should read from standard input and write to standard output.
- There is a running time limit on each solution (typically 5 seconds, which should be plenty).
- Until 30 minutes before the end of the contest (or until we run out of helium or balloons), you will get a balloon for each solved problem, in a color that corresponds to the problem you solved. The colors of the problems are not disclosed, so unless you have a balloon of the same color you cannot tell which problems other teams have solved.
- The progress of other teams (in terms of score) can also be seen on the scoreboards until 30 minutes before the end.

Problem 1: Making the Grade?

You are getting ready to take a final exam for your Basket Weaving 101 class and you need to know the minimum score you must get on the final exam in order to obtain an average of at least 90 for the course. A weighted average consists of a number of scores along with their respective weights. For this course, there are four parts to the grade, a project that is worth 15% of the grade, a term paper that is worth 20% of the grade, a midterm exam that is worth 25% of the grade, and a final exam that worth 40% of the grade. Given scores for the project, the term paper, and the midterm exam, compute and display the minimum integer final exam score that will result in a grade of at least 90 for the course. Possible final exam scores range from 0 to 100, inclusive. If it is impossible to obtain at least 90 your program should display the appropriate message.

Your program must read a set of test cases. The input starts with an integer that contains the number of test cases. This integer is followed by that many lists of three integers representing the project score, the term paper score, and the midterm exam score, respectively. Your program must then display for each test case the lowest possible integer score for the final exam that will result in a course grade of at least 90 or, if it is not possible to obtain at least 90, your program must display the word "impossible."

Sample Input:

```
3
85 88 92
70 95 65
100 90 100
```

Sample output:

```
92
impossible
80
```

Problem 2: Eakspay igpay atinlay?

Your job is to translate English phrases into Pig Latin. There are only two rules you need to know. First, for words beginning with a consonant or consonant cluster, you must cut off the consonant or consonant cluster, attach it to the end of the word, and add the letters “a” and “y” at the end of the word. Second, for words beginning with a vowel, you must simply add the letters “y”, “a”, and “y” at the end of the word. We will consider “y” to be a vowel for this problem. You may assume the words will consist of only lower case letters.

The size of each line will be no longer than 200 characters.

Your program must read a list of phrases from standard input (which is preceded by the number of phrases), translate the phrases into Pig Latin, and display the translation.

Sample Input:
```
2
twist of fate
hotter than the fourth of july
```

Sample Output:
```
isttway ofyay atefay
otterhay anthay ethay ourthfay ofyay ulyjay
```

Problem 3: Frame by Frame

In this problem you will have to decide the final score of a game of bowling. A game of bowling has 10 frames. The tenth frame works in a slightly different way than the first 9.

For the first 9 frames, each will consist of 1 or 2 rolls. You will be presented with 10 pins. If you hit all 10 pins in the first roll you make a “strike”, and the frame is over. Otherwise you have a second roll to hit the remaining pins. If you hit all remaining pins you score a “spare”; otherwise you have an “open frame”. The number of points an open frame gives is simply the total number of pins you hit. A spare gives 10 points plus the number of pins hit on your next roll (in the next frame). A strike gives 10 points plus the number of pins hit on the next two rolls (in the next frame, or possibly the next two frames if you hit a strike in the next roll).

The last frame consists of 2 or 3 rolls. The way it works can be explained as:

- 10 pins are presented
- If you hit all 10 pins, you score a strike (10 points), and 10 pins are presented again:
    - If you hit all 10 pins again, you score a new strike, and 10 pins are presented again, for the last time.
        - If you hit 10 pins again, you score one more strike. (1)
        - If you hit fewer than 10 pins, you score that number of pins. (2)
    - If you hit fewer than 10 pins, you score that number of points and the remaining pins are presented.
        - If you hit all remaining pins, you score a spare (10 points). (3)
        - If you hit fewer than that, you score that many pins. (4)
- If you hit fewer than 10 pins on the first roll, the remaining pins are presented.
    - If you hit all remaining pins, you score a spare and 10 pins are presented again.
        - If you hit all 10 pins on the third roll, you score a strike. (5)
        - If you hit fewer than 10 pins, you score as many points as pins hit. (6)
    - If you hit fewer than the remaining pins, you score the number of pins hit in both rolls. (7)

Note that the number of points in the last frame is the total number of pins hit in all rolls.

Your job is to compute the total score for a game of bowling given the scores for each of ten frames. A strike is represented by an “X”, a spare is represented by the number of pins hit in the first roll followed by a slash (“/”) and an open frame is represented by 2 numbers: the number of pins hit in each roll. Similarly, the tenth frame is represented either as: (1) “X X X”, or (2) “X X a”, or (3) “X a /” or (4) “X a b” or (5) “a / X” or (6) “a / b” or (7) “a b”, where a and b are numbers.

Your program must read the notated scores for each of a set of games (preceded by the number of games) and compute the total scores. Note that the frames are not delimited in any way---it's your job to figure out if a frame has one, two, or three rolls.

Sample Input:	**Sample Output:**

`3`
`2 6 5 0 4 4 0 9 3 4 8 1 1 4 3 2 5 4 6 1`
`3 / 4 4 X 5 3 9 0 1 / X 8 / 7 2 X 4 3`
`8 1 X X 3 / 4 5 1 / X X 3 / 4 / 8`
	`72`
	`140`
	`170`

For the third input the game went like this:

Frame	**Notation**	**Points**	**Total**
1	`8 1`	8 + 1 = 9	9
2	`X`	10 + 10 + 3 = 23	32
3	`X`	10 + 3 + 7 = 20	52
4	`3 /`	10 + 4 = 14	66
5	`4 5`	4 + 5 = 9	75
6	`1 /`	10 + 10 = 20	95
7	`X`	10 + 10 + 3 = 23	118
8	`X`	10 + 3 + 7 = 20	138
9	`3 /`	10 + 4 = 14	152
10	`4 / 8`	4 + 6 + 8 = 18	170

Problem 4: Crushing Confections

A popular app has players line up a set of three candies of the same color in a row in order to remove them from the screen and continue playing. The candies come in six colors: blue, yellow, green, orange, purple, and red. They are arranged in a two-dimensional grid with horizontal rows and vertical columns. Once a player has moved candies to complete and remove a set of candies, the game must automatically find and remove any remaining sets of three candies of the same color in a row.

Your program must find and display the location of a valid set of three candies of the same color in a row (or column) or display “no set found.” If there is more than one valid set, your program may display the location of any one of the valid sets. The rows are numbered from 1 starting at the top. The columns are numbered from 1 starting at the left.

The input will start with a number representing the number of test cases. Each test case consists of one line containing two integers representing the number of rows and columns of the grid followed by a grid representing the colors of the candies. Your output should be three ordered pairs of integers representing row numbers and column numbers, respectively, of a valid set if there is a valid set found. If there is no valid set found, your output should consist of the message “no set found.”

The dimensions of the board will be at most 100 by 100.

Sample Input:

```
3

4 7
R R Y B O G P
P G R Y O G B
B G R Y O B Y
P R G B Y B R

5 5
P G O B R
G O B R P
O B R P G
B R P G O
R P G O B

6 4
R O B Y
Y O B R
O B R Y
G B G B
R Y Y Y
B G O R
```

Sample Output:

```
1 5 2 5 3 5
no set found
5 2 5 3 5 4
```

Problem 5: Waiting for Change

Each student in a certain high school Calculus class must pay $15 for the latest review book. The problem is that not all of the students have exact change and the teacher, of course, has no money at all. Every student has the money to pay but some do not have exact change. To keep it simple, each student has exactly one $5 bill and one $10 bill or has exactly one $20 bill. Eager to begin their review, the students line up to purchase their new review books. This line will be called line A. The teacher forms a second line as needed for students who will have to wait for the teacher to make change. The second line is line B and will start out empty.

Review books will be purchased as follows. If the teacher is able to give the correct change, s/he will sell a review book to the first student in line B, if there is one, and give back the correct change. Otherwise, if the first student in line A has exact change, the teacher will sell that student a review book. However, if the first student in line A does not have exact change, then that student will go to the back of line B. Sadly, it is possible that not every student will be able to get a review book because there might not be enough $5 bills in the room.

Your program must print out the names of the students in line B when line B is at its longest.

The input consists of several test cases. The input starts with the number of test cases. Each test case starts with the number of students followed by a list of students names and the amount that they can use to pay for the book (either 15 or 20). Fortunately, there are never two students by the same name.

For each test case, your output should be a list of the names of the students waiting in line B when it is at its longest, from the front of the line to the back. If there is a tie, it should print the state of the line the first time it reached its maximum length. If no students ever get in line B, your output should print “Line B stayed empty.”

There will be at most 100 students in line A.

Sample Input:

```
3

7
Alfred 15
Beth 15
Calista 20
Desdemona 20
Ezekiel 15
Fred 20
Georgina 15

11
Hazel 20
Izzy 15
James 15
Karl 20
Lucinda 20
Malia 20
Nicholas 15
Oscar 20
Petra 20
Quantasia 20
Redd 15

8
Sasha 20
Tabitha 20
Ursula 20
Victor 15
Wanda 15
Xavier 20
Yolanda 20
Zane 15
```

Sample Output:

```
Line B stayed empty.
Malia Oscar Petra Quantasia
Sasha Tabitha Ursula
```

Problem 6: Cornell Party

One-hundred and fifty years ago, Ezra Cornell and A. D. White hosted a party in the newly opened Cornell University. They were at the door greeting people as they came, and they also made notes of who was coming in. This party was very peculiar in that a guest could pass through the door several times, and by doing so, would be counted by Cornell and White multiple times. They were very smart men, so knowing that this could happen, they mentally assigned a number to each invited guest, so every time a guest entered the door both Cornell and White would make note of the ID of that guest. They had very good memory, so they wouldn't write the ID down at the moment a guest comes in. As a consequence, their lists not necessarily are in the order of arrival.

At the end they compared their notes and realized they were completely different. They immediately understood what was going on. Cornell and White assigned different IDs for the same person. While Cornell would say that John Smith is person 5, White would say that he is person 8, for example. They are not convinced that this was their only mistake, though. Can you help them?

Your task is to tell Cornell and White if their only mistake was to give different IDs to the same person, or if one of them got something wrong. For instance, if Cornell's list is: "1 3 1" and White's list is "5 5 6", it can be the case that Cornell's person 1 is White's person 5, and Cornell's person 3 is White's person 6. In this case their only mistake was using different IDs. On the other hand if the lists are "1 1 1" and "1 2 3" there is no way that they didn't make any other mistake.

The input consists of several test cases. The input starts with the number of test cases. Each test case starts with the number *N* of names in the lists. The following line will contain all the N numbers in Cornell's list, and the next line will contain all the *N* numbers in White's list. $1 \leq N \leq 100{,}000$, and each number inside the lists is between 1 to 1,000,000 (inclusive).

For each test case, your output should be "what a lovely party" in case their only mistake was to give different IDs or "you've messed up, Cornell", otherwise.

Sample Input:	**Sample Output:**

3
 	
3
1 2 1
5 5 6
 	
3
1 1 1
1 2 3
 	
5
11 12 13 14 15
11 12 13 14 15
	what a lovely party
	you’ve messed up, Cornell
	what a lovely party

Problem 7: Cornell Party - Retry

Ezra Cornell and A. D. White learned their lesson from their last party, so this time they decided to use names instead of identifiers. They also realized that they shouldn’t trust their memories so much, so they’ve decided to write down the guest names as they arrive. After a few hours they got tired of staying at the door and decided to mingle. They did, however, continued to take note of every single person they encountered. Since they were separated, they could have encountered a completely different set of people.

At the end of their party they decided to figure out how many people were there. They both put their list on the table, and soon realized that they can only determine the **minimum** number of people that went. Basically, if a name is in either list, this person went to the party. Fortunately, no two people with the same name attended the party. Can you help Cornell and White again?

The input consists of several test cases. The input starts with the number of test cases. Each test case starts with two numbers *N* and *M* in a line, the size of Cornell’s list and White’s list respectively. The following line will contain the *N* names in Cornell’s list, and the next line will contain the *M* names in White’s list. $1 \leq N \leq 100{,}000$, $1 \leq M \leq 100{,}000$. The names will consist only of lower case letters and will consist of at most 10 characters.

For each test case you should output the minimum number of people who attended the party.

Sample Input:

```
3

3 3
agnes clelia agnes
cynara cynara elvira

3 4
zilla zilla zilla
zilla mason primrose mason

4 5
ramona ludmilla ramona anthea
ludmilla anthea briette anthea ramona
```

Sample Output:

```
4
3
4
```