# Twenty-third Annual
# University of Central Florida

# High School Programming Tournament

## *Problems*

| Problem Name | Filename |
|---|---|
| Hidden Circle | circle |
| Stock Market | stock |
| Gnome Sort | gnome |
| Family Tree | family |
| Six Pedals, Four Directions | pedals |
| Math Jeopardy | jeopardy |
| Obtain Intelligence | smart |
| Dogsledding | dogsled |
| Meme Detector | meme |
| Two Lines, Less Waiting | lines |

Call your program file: *filename*.c, *filename*.cpp, or *filename*.java
Call your input file: *filename*.in

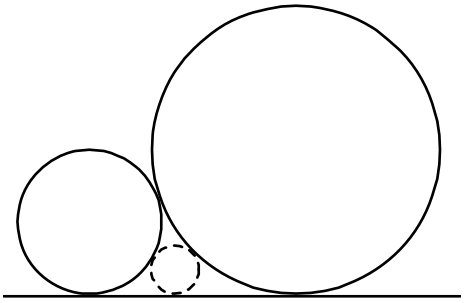
For example, if you are solving Gnome Sort:
Call your program file: gnome.c, gnome.cpp or gnome.java
Call your input file: gnome.in
Call your Java class: gnome

# Hidden Circle

*Filename:* `circle`

Ali loves difficult circle problems. Whether training in the NASA Centrifuge for the astronaut corps, organizing his CD collection or building fancy clocks, nothing makes Ali happier than a circle problem! However, Raymond has decided that he wants to challenge Ali for the circular crown given to the person who makes the most difficult circle problem. Of course, Ali does not want to relinquish his crown. He wants you to succeed in solving Raymond's challenge, which is stated below. Don't let Ali down!

Given the radii of two tangent circles sitting on the ground, Ali wants you to determine the radius of the circle that is tangent to the two given circles and the ground as illustrated in the diagram below:



**The Problem:**

Given the radii of the two large circles in the diagram above, determine the radius of the circle that would fit between them, tangent to each large circle and the ground. If you need to use $\pi$, use a value of 3.1415926535.

**The Input:**

Input will begin with a single, positive integer, $n$, on a line by itself, representing the number of circle problems to solve. Each problem will be on a single line. Each line will contain only two positive integers separated by a single space, $r$ ($r < 100$) and $s$ ($s < 100$), representing the radii of the two circles, respectively.

**The Output:**

For each circle problem in the input, output a line with the following format:

`Circle Problem #`*k*`: Radius of the small circle =` *c*

where $k$ ($1 \leq k \leq n$) represents the number of circle problem and $c$ is the desired radius rounded to two decimal places. For example, 3.454 would round to 3.45 while 3.455 would round to 3.46.

**Sample Input:**

```
3
1 1
1 2
19 88
```

**Sample Output:**

```
Circle Problem #1: Radius of the small circle = 0.25
Circle Problem #2: Radius of the small circle = 0.34
Circle Problem #3: Radius of the small circle = 8.86
```

# Stock Market

*Filename:* `stock`

With the recent recession, investors are finding that they have to improve their prediction schemes to maximize the return on their investments. Your hope is to capitalize on this need by writing some software that investors are going to snatch up.

Your goal is to read in a listing of a company's stock values and find the biggest overall gain in any time period.

For example, consider a company that had the following stock prices over a seven-day period:

12 1/2, 12 3/4, 13 1/4, 12 1/4, 12 5/8, 12 1/2, 12 7/8

Then the biggest overall gain was from day 1 to day 3 where the stock increased a total of 3/4 of a point (the next biggest overall gain was from day 4 to day 7 of 5/8 of a point). All time spans to be considered will have distinct starting and ending days.

**The Problem:**

Given a list of stock prices on several consecutive days, determine the greatest overall gain in any time period within the span. If there are multiple such time periods, determine the one that starts the earliest. If there are still multiple such time periods, determine the one that ends earliest.

**The Input:**

The first line of the input file will have a single positive integer, *n*, representing the number of time periods. For each time period, the first line will have a single integer, *m* ($1 < m < 1001$), representing the number of days of stock prices for that case. The following *m* lines will contain the price of the stock for each day in that particular input case, from day 1 to day *m*.

The format for a stock price will be as follows:

If the price of the stock for that day is greater than or equal to one, the first part of the stock price will be a positive integer (less than 100,000) followed by a space. If not, then no characters will be present for this part of the price. All prices will have a fraction that follows. A fraction will always be composed of exactly three characters: a digit (the numerator), followed by the '/' character, followed by another digit (the denominator). The denominator will always be '1', '2', '4', or '8'. The numerator will always be non-negative and strictly less than the denominator. The numerator and denominator will not share any common factors, and the whole stock price is guaranteed to be positive (thus, if the stock value is less than one, the fractional part of it will have a positive numerator). Stock prices that are whole dollars will always have "0/1" as their fractional component (for example, 13 0/1).

**The Output:**

For each time period, output a line with the following format:

`Period` *k*`: The biggest gain was from day` *a* `to day` *b* `of` *x*`.`

where $k$ ($1 \le k \le n$) represents that number of that time period, $a$ represents the number of the day where the streak started, $b$ represents the number of the day the streak ended, and $x$ is how much the stock gained in that time span. Note that $1 \le a < b \le m$ and that $x$ must be presented in the format described in the input for stock prices. If this maximum "gain" is negative, place a single negative sign right in front of the amount lost as shown in the second period in the Sample Output below.

**Sample Input:**

```
4
8
12 5/8
12 1/2
12 3/4
13 1/4
12 3/8
12 5/8
12 1/2
13 0/1
3
14 0/1
13 5/8
12 7/8
4
9 1/2
10 1/2
11 0/1
10 3/8
2
5/8
3/4
```

**Sample Output:**

```
Period 1: The biggest gain was from day 2 to day 4 of 3/4.
Period 2: The biggest gain was from day 1 to day 2 of -3/8.
Period 3: The biggest gain was from day 1 to day 3 of 1 1/2.
Period 4: The biggest gain was from day 1 to day 2 of 1/8.
```

# Gnome Sort

*Filename:* `gnome`

The Dutch garden gnome (*tuinkabouter*) is the bane of any gardener's existence. These mischievous little creatures delight in wrecking artistic arrangements of flowerpots by arranging every line of flowerpots in non-decreasing order of size. Recently, researchers have succeeded in identifying the sorting algorithm used by the gnomes. For their next experiment, they intend to catch a gnome in the act of sorting, and so they have given you the job of writing a simulator that performs the gnome sort[1].

In order to sort a line of flowerpots, a gnome starts by standing in front of the leftmost pot in the line, and performs a series of steps. The rules are as follows:

- If he is at the leftmost pot, he takes a step to the right.
- If the pot in front of him is not smaller than the pot to his left, he takes a step to the right.
- If the pot in front of him is smaller than the pot to his left, he swaps the two and takes a step to the left.
- If there is no pot in front of him, he is done, and yells "Sorted!"

**The Problem:**

Given an arrangement of flowerpots, simulate the steps used by the gnomes in the Gnome Sort.

**The Input:**

The input contains descriptions of gardens, which each contain flowerpots to sort. The first line of each garden description contains a single positive integer, $n$ ($n \leq 500$), the number of flowerpots to be sorted. On the following line, there are $n$ integers (not necessarily distinct) indicating the sizes of the flowerpots from left to right. End of input will be indicated by a garden with 0 flowerpots. This case should not be processed.

**The Output:**

At the beginning of the output for each garden, you must print "`Garden c:`" on a line by itself where $c$ is the garden number (starting from 1). On the following lines, you must print the sequence of swaps performed by the gnome, in the form: "`The gnome swaps the pots at positions a and b.`" where $a$ and $b$ are the 1-based positions of the pots the gnome is swapping. Ensure that $a$ is always the smaller of the two positions. Each swap should be printed on a line by itself. The output for each garden should be ended by the word "`Sorted!`" on a line by itself. Leave a blank line after the output for each garden.

[1] Interesting tidbit: There is, in fact, a real sorting algorithm called gnome sort, and it works in precisely the way this problem describes. Well, except for the gnomes.

**Sample Input:**

```
5
5 4 3 2 1
3
1 2 3
6
1 7 3 2 9 8
0
```

**Sample Output:**

```
Garden 1:
The gnome swaps the pots at positions 1 and 2.
The gnome swaps the pots at positions 2 and 3.
The gnome swaps the pots at positions 1 and 2.
The gnome swaps the pots at positions 3 and 4.
The gnome swaps the pots at positions 2 and 3.
The gnome swaps the pots at positions 1 and 2.
The gnome swaps the pots at positions 4 and 5.
The gnome swaps the pots at positions 3 and 4.
The gnome swaps the pots at positions 2 and 3.
The gnome swaps the pots at positions 1 and 2.
Sorted!

Garden 2:
Sorted!

Garden 3:
The gnome swaps the pots at positions 2 and 3.
The gnome swaps the pots at positions 3 and 4.
The gnome swaps the pots at positions 2 and 3.
The gnome swaps the pots at positions 5 and 6.
Sorted!
```

# Family Tree

*Filename:* `family`

You've been studying a lot of family trees lately, and have started to get confused as to who's in whose family. The connections span many generations, and a lot of people are involved. Since you are about to go to a family reunion, and want to at least appear as if you know who's in your own family, you decide to write a program to help you sort it all out.

**The Problem:**

Given a description of a family tree, your task is to determine whether or not two people are related. For this problem, two people are considered related if there is a path between the two in the family tree. A "path" is any series of parent-child (or child-parent) relationships that connect people in the tree (not only blood relatives are related). It is guaranteed that each child will have exactly two distinct parents, and each family tree will be valid, meaning that no child will be his/her own ancestor (luckily for you, there is no time travel allowed for this problem!).

**The Input:**

There will be multiple family trees. Each family tree will begin with an integer, $n$ ($1 \leq n \leq 100$), on a line by itself, indicating the number of connections to be listed. This will be followed by exactly $n$ lines, each of the form *parent1 parent2 child*, indicating that *parent1* and *parent2* are the parents of *child*. Each name will be separated by a single space. Each name will consist of only upper and lowercase letters, and no name will exceed 80 letters. In addition, each different person will have a unique, case-sensitive name. Each relationship will be unique (there will be no repeated relationships within a single family tree). This will be followed by a line of the form *name1 name2*, indicating the two people whose relation you are to determine. Both names will follow the same conventions as before, and will be separated by a single space. The final family will be indicated by $n = 0$, and should not be processed.

**The Output:**

For each family tree, you should output a single line beginning with "`Family #x:`" where *x* is the family tree being processed (beginning with 1). This should be followed by one space, then either "`Related.`" or "`Not Related.`" to indicate whether or not the two people are related.

(Sample Input and Sample Output are on the following page)

**Sample Input:**

```
2
Barbara Bill Ted
Nancy Ted John
John Barbara
3
Lois Frank Jack
Florence Bill Fred
Annie Fred James
James Jack
1
John Susan Billy
John Susan
2
Karen Roger Christopher
Karen Roger Michael
Christopher Michael
0
```

**Sample Output:**

```
Family #1: Related.
Family #2: Not related.
Family #3: Related.
Family #4: Related.
```

# Six Pedals, Four Directions

*Filename:* `pedals`

Rookie, a soldier in the imperial army, has just started his training in the armored legions. During his first day of training they covered driving; however, poor Rookie fell asleep during the entire lecture. Now, the unit is conducting training exercises to master their driving. Once inside the cockpit, Rookie has to ask himself one essential question: "Why are there six pedals when there are only four directions?" This is where you come in. Because you listened to the instructor, you know that pedal 1 goes forward, pedal 2 goes backward, pedal 3 goes left, pedal 4 goes right, pedal 5 rotates the tank left, and pedal 6 rotates the tank right. Using this basic knowledge, you decide to help Rookie with his driving in order to save the platoon from extra chores.

**The Problem:**

Given a series of movements, your program must tell Rookie which pedals to press.

**The Input:**

The input will start with a positive integer, $n$, on a line by itself, giving the number of courses through which the tank must drive. This will be followed by $n$ courses where each course will begin with an integer, $c$ ($1 \leq c \leq 1000$), on a line by itself, followed by $c$ lines where each line is a direction to be processed. Directions will be written in all lowercase and will be any of the following words in the table:

| Direction | Pedal |
|---|---|
| `forward` | Pedal 1 |
| `backward` | Pedal 2 |
| `left` | Pedal 3 |
| `right` | Pedal 4 |
| `rotate left` | Pedal 5 |
| `rotate right` | Pedal 6 |

The last two directions contain exactly one space between the two words.

**The Output:**

The output will start with "`Course i:`" where $i$ is an integer that will correspond to the course being processed, starting at 1. This will be followed by a series of lines containing a single number corresponding to the pedal that must be pressed for the direction given. All courses must also be separated by a blank line.

**Sample Input:**

```
2
4
left
forward
right
backward
5
rotate right
left
left
rotate left
forward
```

**Sample Output:**

```
Course 1:
3
1
4
2

Course 2:
6
3
3
5
1
```

# Math Jeopardy

*Filename:* `jeopardy`

Your math teacher, Alexi Trebekk, is a huge Jeopardy fan and has the annoying tendency to give oral quizzes in the same way as his favorite television show, where the show's host gives the answer to a question, and the contestant responds with the question. After a few quizzes, you think you've figured out his pattern. First, his answers always correspond to the same pair of problems, a simple addition and simple subtraction. Second, he only accepts questions that involve non-negative integers, and the know-it-all students that answer in negative fractions always fail the quizzes. Finally, he always wants the numbers in non-increasing order.

You've decided to write a program on your SJ-500 calculator that will just give you the question that corresponds with Trebekk's answer. For example, last week, his answer was just "45 and 5." The correct question was "What are the sum and difference of 25 and 20?"

However, once Trebekk accidentally gave an answer that had no question given the constraints of his own problem: "8 and 3." Another student in the class, Shawn Connerly (the class clown) jumped up and shouted "NOT SO FAST, TREBEKK!"

This moment has been immortalized in school history, and whenever Trebekk makes this mistake, it is traditional to make the same response.

**The Problem:**

Given Trebekk's two answers, representing the addition and subtraction of two numbers, determine the two numbers that fit in the question.

**The Input:**

There will be multiple quiz answers, each quiz answer given on a single line. Each line contains two integers, *i* and *j*, separated by a single space. The number *i* is the answer to the addition of two numbers, and *j* is the answer to the subtraction of two numbers. Both *i* and *j* will be between 0 and 100000, inclusive. The end of input will be indicated by values of -1 for both *i* and *j*, which should not be processed.

**The Output:**

Your program should either output:

`What are the sum and difference of` *`a`* `and` *`b`*`?`

where *a* and *b* are the two non-negative integers that are the question to the problem. If the problem answer does not fit the criteria, output:

```
NOT SO FAST, TREBEKK!
```

**Sample Input:**

```
45 5
8 3
12 0
11 11
44 5
-1 -1
```

**Sample Output:**

```
What are the sum and difference of 25 and 20?
NOT SO FAST, TREBEKK!
What are the sum and difference of 6 and 6?
What are the sum and difference of 11 and 0?
NOT SO FAST, TREBEKK!
```

# Obtain Intelligence

*Filename:* `smart`

Jimmie Flowers, known as Agent 13, is preparing for his next big mission. He must go undercover in the Russian Circus as a professional knife thrower. In order to not blow his cover, he is practicing throwing knives at a target so that his accuracy improves (his first throws were off by an embarrassing amount!).

**The Problem:**

Given the position of a target and the position where Agent 13 actually throws his knife, determine if Agent 13 hits his target. Each position is represented by a single integer.

**The Input:**

Input will begin with a single, positive integer, *n,* on a line by itself, representing the number of throws made by Agent 13. For each throw, there will be two positive integers, t and *p* (*t*, *p* ≤ 1000), on a single line each separated by a single space where *t* represents the position of the target and *p* represents the position of Agent 13's throw.

**The Output:**

For each throw, if the target is hit, output "`The Old Knife-Hits-the-Target Trick`" or output "`Missed it by THAT much`" if it is not. Each output should be on a separate line.

**Sample Input:**

```
2
5 5
7 1
```

**Sample Output:**

```
The Old Knife-Hits-the-Target Trick
Missed it by THAT much
```

# Dogsledding

*Filename:* `dogsled`

Pat is planning a cool vacation up to the Arctic Circle. To get a true cultural experience, he wants to try out dogsledding. Unfortunately, he has never gone dogsledding before and isn't comfortable with the commands given to a dog team. He wants to simulate his journey to see whether he makes it from the Camp to the Old Silver Mine.

Commands to dog teams are as follows:

| Command | Meaning |
|---|---|
| Hike! | Move forward 50 feet |
| Haw! | Turn left by 10 degrees |
| Gee! | Turn right by 10 degrees |
| Whoa! | Stop |

**The Problem:**

Given a sequence of commands to a dog team, determine whether the team ends up at the Old Silver Mine. The team is represented on an (x, y)-grid by a point and initially starts at (0, 0) facing East (i.e. along the positive x-axis) for each simulation. The Old Silver Mine is always represented by a square of 30 feet wide, centered at (500, 0). The team is considered to make it to the mine if it ends its movement either inside the mine or within $10^{-5}$ feet of its border (passing through the mine, but not ending movement within it, is not sufficient). If you need to use $\pi$, use a value of 3.1415926535.

**The Input:**

Input will begin with a single, positive integer, *n,* on a line by itself, representing the number of simulations to check. Each simulation is a series of commands, each given on a separate line with no whitespace. The command "`Whoa!`" tells the dog sled team to stop and represents the end of that simulation.

**The Output:**

For each simulation, output "`Simulation #`*`i`*`:`" where *i* is the number of the simulation (beginning with 1). This is followed by a single space. Then, determine if the dog team will end up at the Old Silver Mine (meaning inside the mine or on its border). If the team makes it, output "`Made it!`". Otherwise, output "`We're lost, eh?`" if it does not. The output for each simulation should be on a separate line.

**Sample Input:**

```
2
Hike!
Hike!
Hike!
Hike!
Hike!
Hike!
Hike!
Hike!
Hike!
Hike!
Whoa!
Hike!
Haw!
Hike!
Gee!
Hike!
Gee!
Hike!
Hike!
Whoa!
```

**Sample Output:**

```
Simulation #1: Made it!
Simulation #2: We’re lost, eh?
```

# Meme Detector

*Filename:* `meme`

Dagg is a popular Internet social news site where users can submit stories they find around the Internet for everyone to see. Other users can then comment on the stories where the site's creator, Kalvin Rose, hopes intellectual discourse can occur. This rarely happens in practice, primarily due to what's called an Internet meme. Such a meme is a word or phrase that quickly spreads across the Internet and, according to Kalvin, prevents intellectual discourse. Unscrupulous users frequently call upon these memes while commenting on stories to trivialize a debate and, as a result, no intelligent conversation occurs. Because Kalvin wants the Internet to become a true place of intellectual discourse, a place where residents can come together and share thoughts of morality, local and world events, and all sorts of other deeply meaningful things, he has been working hard on a solution to remove these ubiquitous memes from the comments.

So far, Kalvin has produced a dictionary of common memes found throughout the Internet and a text parser which pulls suspicious phrases from the comments of a story. Kalvin is not naïve; he knows that if he does a direct comparison between each meme and a user's comment the users will catch on and begin making small changes to their post to get past the filter. He has determined a series of common changes which are likely to occur, and are detailed by the following Rules:

1. The user will change the capitalization of any number of letters in the meme. For example: "rickroll" can be changed into "RickRoll".
2. The user will change a single occurrence of a single letter in the meme to a symbol, one of @, #, $, %, *, &, or !. For example "Ninja Cat" can change into "Ninja C@t" but not into "N!nja C@t" because more than one letter would change into a symbol. Note that any letter could be changed this way and there is no apparent pattern; any letter is equally likely to change into any of the 7 symbols.
3. The user will repeat a single instance of a single non-space character in the meme any number of times. The new characters added will always be adjacent to the source character. This character can also be the character affected by Rule 2. For example, "Leeroy Jenkins" can be changed into "Leeeeeeeroy Jenkins" or "Leeroy Jenkinsssss" but not "Leeeeroy Jennnkinnnnsss" since no more than one character can be duplicated. If "Leeroy Jenkins" was changed to "Leeroy Jenk!ns" due to Rule 2, it could then be changed by this rule to "Leeroy Jenk!!!!ns". The repeated character can have mixed case due to Rule 1, so "LeeeeEEeroy Jenkins" is a valid change.
4. The user will change the ordering of the words. For example, "THIS IS SPARTA" could be changed into "SPARTA IS THIS". A word is defined as any series of consecutive non-space characters separated by a single space. Any word can be switched to any position with this rule. There will never be more than one space between words.

Any, none or all of these Rules can be applied to a meme in any order, for example: "IT IS OVER NINE THOUSAND" can be changed to "nine OVER it is THOUsand" or "It is over NINE THOUSAAAAAA@AAAAAND". However, each rule can be applied only once to a given meme. Given a list of memes and a series of comment phrases, Kalvin has charged you with determining which (if any) are, in fact, an internet meme.

**The Problem:**

Given a list of phrases from comment threads and a list of known memes, determine whether any of the phrases are memes that have been changed using the rules above. Because Kalvin wants to check the comment threads from multiple stories, there will be multiple input cases each with a different dictionary of memes (the Internet is a volatile place, and memes phase in and out all of the time).

**The Input:**

The first line of the input begins with an integer, *n*, specifying the number of comment threads. For each comment thread, there will be a positive integer, *a* ($a < 20$), specifying the number of memes in the dictionary for the current comment thread. The following *a* lines will contain known memes, each on their own line. You are guaranteed that each meme will contain at least 1 and no more than 50 characters, including spaces. Each meme will only contain the characters a-z, A-Z, and space. There will be no leading or trailing spaces, and no more than a single space between consecutive series of letters. It will not be possible to transform one meme into another with the given rules.

The next line will contain a positive integer, *b* ($b < 20$), representing the number of phrases to be checked in the comment thread. The next *b* lines will contain a phrase, each phrase on its own line. Each phrase will contain at least 1 and no more than 100 characters. Each phrase will contain only the characters a-z, A-Z, @, #, $, %, *, &, !, and space. There will be no leading or trailing spaces in the list, and no more than one space between consecutive series of characters. It is guaranteed that no phrase will map to no more than one meme.

**The Output:**

For each comment thread, print out the following: "`Comment thread #x:`" where *x* is the current input case number (starting at 1). Then, for each meme detected in the comment thread, print out the original meme found preceded by three spaces. If the same meme is detected more than once in the thread, only print it one time. In addition, print out the memes in the order they appear in the meme dictionary. If no meme is found, print "`No memes detected! Let intellectual discourse continue!`" preceded by 3 spaces. The output for each comment thread should be separated by a single blank line.

(Sample Input and Sample Output are on following page)

**Sample Input:**

```
2
3
Never gonna give you up
All your base are belong to us
At least I got chicken
3
chicken I G*t least &t
Neverrrr gonna give YOU u@
base ARE belong us to all your
2
lolcats
I for one welcome our new ant overlords
3
llll@lllcats
welcome OUR new overlords for one
This is not a meme at all
```

**Sample Output:**

```
Comment thread #1:
   Never gonna give you up
   All your base are belong to us

Comment thread #2:
   No memes detected! Let intellectual discourse continue!
```

# Two Lines, Less Waiting

*Filename:* `lines`

Roller coaster and thrill-ride enthusiast, Michael Henry Roe (MHR), loves traveling across the country, going from theme park to theme park to ride the best roller coasters the parks have to offer. His latest destination is his old favorite, Walt Flags Studios, to try out their recently opened thrill-ride, Universe Cruiser Frigate, a twisting, turning indoor coaster on a runaway space frigate, ending with the coaster rocketing through a black hole.

Of course, MHR hates the unwanted wait in very long lines. In order to help alleviate the stress for single riders and help speed up ride movement, Walt Flags Studios followed the example of other theme parks and created a single-rider line, which is separate from the regular line. But MHR is unsure if the new single-rider line necessarily means a shorter wait.

The ride attendants have a set procedure for handling passengers. They will first try to fit the entire party that's at the front of the regular line. If that party can be seated, they move on to the next party in the regular line until the coaster is full or the next party from the regular line cannot all be seated. If not, all remaining seats are filled with people from the single-rider line. Then, off the ride goes, and a new coaster comes in. Luckily for MHR, he's made a lot of friends with the operators and surveillance crews for the rides, being famous around the park. They recommended you to help MHR with his conundrum.

**The Problem:**

Given the composition of the parties in the regular line and the number of people in the single rider line, help MHR figure out which line will get him into the ride more quickly. Since he would like to ride the coaster multiple times, he needs you to process multiple waiting scenarios.

**The Input:**

The input will have multiple waiting scenarios. Each scenario begins with 3 non-negative integers on a single line, $p$ ($p \leq 100$), $s$ ($s \leq 100$), and $c$ ($1 \leq c \leq 25$), where $p$ is the number of parties in the multi-rider line, $s$ is the number of single riders, and $c$ is the capacity of the ride. The next line will contain $p$ positive integers, where each do not exceed $c$, showing the size and the order of the parties in the regular line. The end of input will be indicated by values of zero for $p$, $s$ and $c$, which should not be processed.

**The Output:**

For each of the waiting scenarios, output on a single line the following message:

`Scenario #`*`x`*`: MHR rides coaster #`*`t`*`, using `*`m`*`.`

where $x$ is the current waiting scenario (starting with 1), $t$ is the number of the coaster that MHR rides (starting with 1), and $m$ is either "`the regular line`", "`the single rider line`", or "`either line`" depending on which, if either, is faster.

**Sample Input:**

```
5 10 6
5 3 4 2 4
6 3 10
6 5 3 4 2 8
1 1 3
1
0 0 0
```

**Sample Output:**

```
Scenario #1: MHR rides coaster #4, using the regular line.
Scenario #2: MHR rides coaster #1, using the single rider line.
Scenario #3: MHR rides coaster #1, using either line.
```