

# # 1 : Are Your Ducks in a Row?

## 1.1 The Story So Far

Janet is a duck-herd. At one time, she had grouped the ducks into flocks of 100 ducks each, and then numbered each duck with its ordinal 1 through 100. However, in the time since, the duck-wolves have had their way with the flocks, and many flocks have far fewer than their starting 100. Despite this, Janet still insists that the ducks stay lined up perfectly in order at all times.

## 1.2 Your Task

Your job is to write a computer program that detects whether or not a flock of ducks is `in order`. The input will consist of a line containing the number  $N$  that is the number of flocks you must consider. Following that, there will be  $N$  lines (one for each flock), consisting of  $M$ , the number of ducks in that flock, followed by their numbers. For each flock, your program must output `in order` or `not in order` if the numbers are strictly increasing or not, respectively.

## 1.3 Constraints and Assumptions

You may assume the following:

- $N \in [1, 100]$ . There will be no more than 100 and no fewer than 1 flocks.

2

- $M \in [1, 100]$ . Each flock will have no more than 100 and no fewer than 1 ducks.
- Each duck will be numbered with an integer not larger than 100 or smaller than 1. The numbers are unique within each flock.

## 1.4 Sample Input

```
1 5
2 1 47
3 10 1 2 3 4 5 6 7 8 9 10
4 10 1 2 3 4 5 6 8 7 9 10
5 2 99 98
6 2 1 100
```

## 1.5 Correct Output

```
1 in order
2 in order
3 not in order
4 not in order
5 in order
```

# # 2 : What Time is it?

## 2.1 The Story So Far

Peter is a salesman for his company's Java(tm)-powered "Hello World" programs. The company is about to launch a new global marketing campaign, and wants Peter to spearhead it. Unfortunately, Peter has a terrible head for numbers, and he rarely calculates new timezones correctly. To make matters worse, the (very expensive) clock on his corporate jet only gives the current time in "military time", which Peter never can get right.

## 2.2 Time Formats

Military time is defined as an integer from 000 to 2359. The last two digits are the minutes, and the remaining digits are the hours. If the hours are from 0 to 11, it is in the A.M.. Otherwise, if the hours are from 12 to 23, it is in the P.M.

For example, the time 230 is equivalent to 2:30am. The time 1430 is equivalent to 2:30pm.

Note that 000 in military time (12:00 midnight) is considered to be in the A.M. Likewise, 1200 (12:00 noon) is in the P.M.

## 2.3 Your Task

Your task is to write a program to help Peter compute the local time of each country he visits. Your program will be given a number  $N$ , which is the number of stops his corporate jet will make. There will be  $N$  lines following (each line is a stop). Each line has  $M$ , the stop's timezone's hour-offset from

4

GMT, expressed as a (possibly negative) integer, followed by  $T$ , the current GMT time in military format.

For each stop, your program must output the local time expressed in standard US time format.

## 2.4 Constraints and Assumptions

You may assume the following:

- $1 \leq N \leq 100$ . There will be no more than 100 and no fewer than 1 stop.
- $-12 \leq M \leq 12$ . Each zone's offset will be an integer from -12 to +12.

## 2.5 Sample Input

```
1 4
2   3 2115
3  -3 2115
4 -12 230
5   2 001
```

## 2.6 Correct Output

```
1 12:15am
2 6:15pm
3 2:30pm
4 2:01am
```

# # 3 : Mission: Failure

## 3.1 The Story So Far

Microsoft Matlab(tm) is one of the most expensive mathematics software packages on the market. But with all that money, you get a lot of features. For example, Matlab has a web mode, where it runs as a web server. Anybody on the Internet can surf in and type in Matlab functions, then the server computes and displays the results.

Unfortunately, James, a developer, realized that the soon-to-be-released version of their software, Matlab v. $\infty$ , is missing several important functions for statistics. Since all the developers are all currently busy fixing bugs, Microsoft hired an independent contractor to write the function that computes averages and medians. James emphasized to the contractor that (because Matlab can run as a web server), the code needs to be absolutely safe for *any possible* input.

The contractor's function is *supposed* to work as follows:

1. Read an input string (stored in a global variable)
2. Verify that the string contains only numerals 0 through 9 (if it does not, exit and return 1)
3. Extract each digit, and put them into a list
4. Compute the mathematical mean and average of the list of digits
5. Display the results

When James got the contractor's code back, he noticed something a little funny about it. Not only were the computed average and median rounded badly, it was possible to cause the code to crash. When he told the contractor about the flaws, the contractor denied them and demanded proof.

## 3.2 Your Task

Your task is to help James prove the contractor wrong, by writing a program that crashes when calling the function in Listing 3.4. Your program will be given no input. Instead, your code should set the contents of the global variable `mean_and_median_input` to a string that will crash the function `compute_the_mean_and_median()`, and then call that function. Your program will be judged correct if it crashes while still executing the suspect function `compute_the_mean_and_median()`.

While writing your code, you do *not* have to type in the source of the function `compute_the_mean_and_median()`.<sup>1</sup> It will automatically be put into your code. Similarly, you do not have to declare the input buffer. Your code should basically look just like the code given in the Sample Input section below.

**Special note:** For this problem, the contest environment will ask for an input file (as it does for every other problem). However, there is no input for this contest, so just pick any file to make it happy.

## 3.3 Constraints and Assumptions

You may assume the following:

- The vulnerability does not rely on obscure nuances of the C++ language, its standard function library, or the particular platform on which the code is run.

---

<sup>1</sup>In fact, if you try to do so, your program won't compile.

## 3.4 Code Listing

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <numeric>
5  #include <string.h>
6  using namespace std;
7
8  char mean_and_median_input[100];
9  int compute_the_mean_and_median()
10 {
11     int num_digits, value, sum;
12     vector<int> Numbers;
13     char *input = mean_and_median_input;
14
15     // Security: ensure the input is a string
16     input[100] = '\0';
17
18     // compute number of characters in input
19     num_digits = strlen(input);
20
21     // put each digit in the vector
22     for (int i = 0; i < num_digits; ++i) {
23         // convert char to int
24         int digit = input[i] - '0';
25
26         // Security check: can only input 0-9
27         if (digit < 0 || digit > 9)
28             return 1;
29
30         Numbers.push_back(digit);
31     }
32
33     // compute the sum of the numbers in the list
34     sum = accumulate(Numbers.begin(), Numbers.end(), 0);
35
36     // sort the list
37     sort(Numbers.begin(), Numbers.end());
38
39     cout << "Mean: " << (sum / num_digits) <<endl;
40     cout << "Median: " << Numbers[num_digits/2] <<endl;
41     return 0;
42 }
```

### 3.5 Sample Input

```
1 //
2 // This is how your source code should look.
3 // The only bit you change is the string "01234".
4 //
5
6 #include <string.h>
7 #include <iostream>
8
9 using namespace std;
10 int main()
11 {
12     strcpy(mean_and_median_input,
13           "01234");
14
15     cout << "Get ready...\n";
16     flush(cout);
17     compute_the_mean_and_median();
18     cout << "Oops, didn't crash!\n";
19
20     return 0;
21 }
```

### 3.6 Correct Output

```
1 Get ready...
```

# # 4 : How Many Fingers Am I Holding Up?

## 4.1 The Story So Far

As it turns out, the communists *were* putting fluoride into our water supplies. One unfortunate result of the mutagenous characteristics of fluoride is that, by the year 2525, if man is still alive (and woman can survive), humans will have mutated to have only  $n$  fingers on their hands (where  $2 \leq n \leq 10$ ). This won't pose a major problem for most day-to-day activities, but counting will be somewhat challenging. Our current decimal system assumes that each person has exactly ten fingers at his or her disposal. In light of this development, the future space-government mandated that all numbers be converted from their "old" base-10 form to their new base- $n$  form.

## 4.2 Base Conversion

We are most familiar with the base-10 way of representing numbers. In base 10, each numeral of a number has value  $10^k$ , where  $k$  is the numeral's position from the right. For example

$$\begin{aligned} 234 &= 2 \times 10^2 \\ &\quad + 3 \times 10^1 \\ &\quad + 4 \times 10^0 \end{aligned}$$

In a different base, like base 7, you can only work with the digits 0 through 6 (in base  $n$ , the digits 0 through  $n - 1$ , when  $n < 10$ ). To convert a base-10 representation to a base 7 representation, you must figure what to multiply

powers of 7 with. For example, 234 in base 7 is 453, because

$$\begin{aligned} 234 &= 4 \times 7^2 \\ &+ 5 \times 7^1 \\ &+ 3 \times 7^0 \end{aligned}$$

(get 453 by reading the numerals 4, 5, and 3 down the column).

As another example, 1024 in base 4 is 100000, because

$$\begin{aligned} 1024 &= 1 \times 4^5 \\ &+ 0 \times 4^4 \\ &+ 0 \times 4^3 \\ &+ 0 \times 4^2 \\ &+ 0 \times 4^1 \\ &+ 0 \times 4^0 \end{aligned}$$

### 4.3 Your Task

Your task is to write a program that performs number conversion. Your program will be given lines of input, with each line containing  $x$  and  $n$ . A correct program will convert  $x$  from base 10 to base  $n$ , and then print the result.

The last line of input will be start with the number 0, which indicates that your program should exit.

### 4.4 Constraints and Assumptions

You may assume the following:

- $1 \leq x \leq 1000000$ . Each input number will be between 1 and 1000000.
- $2 \leq n \leq 10$ . The target base will be between 2 and 10.

## 4.5 Sample Input

```
1 1      4
2 2      4
3 3      4
4 4      4
5 1023   4
6 1024   4
7 1024   5
8 1024   6
9 654321 9
10 654321 10
11 0      0
```

## 4.6 Correct Output

```
1 1
2 2
3 3
4 10
5 33333
6 100000
7 13044
8 4424
9 1206503
10 654321
```



# # 5 : What's the Score?

## 5.1 The Story So Far

In basketball, points may be scored with either a free throw, a field goal, or a three-point field goal. A free throw is worth 1 point, the field goal 2 points, and the three-point field goal is (naturally) worth 3 points.

## 5.2 Your Task

Your task is to write a program that, when given the final scores (of both teams) from a basketball game, determines the number of possible ways those points could have been scored. (The order in which they were scored does not matter). For example, if the final scores are (a simple, but unlikely) 3 and 2, there are six possible ways that score could have been obtained:

- Team A: three free throws; Team B: two free throws
- Team A: three free throws; Team B: one field goal
- Team A: one free throw and a field goal; Team B: two free throws
- Team A: one free throw and a field goal; Team B: one field goal
- Team A: a three-point field goal; Team B: two free throws
- Team A: a three-point field goal; Team B: one field goal

Your program will be given a sequence of lines of input. Each line will have  $A$  and  $B$ , the scores of Team A and Team B, respectively. For each line, a correct program will output the number of possible ways those scores could have been obtained. The last line of input will contain zero, indicating that your program should exit.

### 5.3 Constraints and Assumptions

You may assume the following:

- $1 \leq A \leq 100$ . Team A scores between 1 and 100 points.
- $1 \leq B \leq 100$ . Team B scores between 1 and 100 points.
- The correct answer will fit in an `int`.

### 5.4 Sample Input

```
1 3 2
2 3 3
3 5 1
4 10 1
5 10 10
6 100 1
7 40 41
8 0 0
```

### 5.5 Correct Output

```
1 6
2 9
3 5
4 14
5 196
6 884
7 24794
```

# # 6 : Factorials!

## 6.1 The Story So Far

Mrs. Erlang is the principal of Function Call High School. The FCHS computer club recently completed the construction of their 500-node Beowulf cluster, and Mrs. Erlang decided to throw a celebration for all  $n$  students at FCHS. However, the school board recently enacted two new policies:

- Every student must be treated fairly. When served food, each student must get exactly the same portion.
- Taxpayer dollars must not be wasted: all purchased food must be served to the students, with none left over.

Because the budget is tight, Mrs. Erlang would really like to buy food on the cheap. Naturally, she turns to local retailer Wal★Math, with its famous “∀day Low Prices.” However, Wal★Math only sells items packaged in factorial quantities. For example, the store sells rice in sacks of 3628800 grains, because  $10! = 3628800$ .

## 6.2 A Mathematic Interlude

The factorial operator (written as a number followed by an exclamation mark, as in “7!”) is defined for any positive integer  $k$  such that

$$k! = \begin{cases} k \cdot (k - 1)! & k > 1 \\ 1 & k = 1 \end{cases}$$

So, for example,

$$\begin{aligned}
 1! &= & &= 1 \\
 2! &= 2 \cdot 1 & &= 2 \\
 3! &= 3 \cdot 2! = 3 \cdot 2 & &= 6 \\
 4! &= 4 \cdot 3! = 4 \cdot 6 & &= 24 \\
 5! &= 5 \cdot 4! = 5 \cdot 24 & &= 120
 \end{aligned}$$

## 6.3 Your Task

Your task is to write a program that determines whether Mrs. Erlang may purchase a particular item. Your program will be given a sequence of input lines, containing two integers  $k$  and  $n$ . It must print “divisible” if  $k!$  is divisible by  $n$ , or “not divisible” otherwise. The last line of input will have  $k = 0$ , which indicates your program should quit.

## 6.4 Constraints and Assumptions

You may assume the following:

- $0 \leq k \leq 500$ . See the special warning below.  $k = 0$  only when your program should exit.
- $1 \leq n \leq 2^{30} \approx 1000000000$ .  $n$  will fit in an integer, but it may not be small.

## 6.5 A Special Warning

The largest value that will fit into C’s `unsigned long` is (depending on your platform) usually about  $2^{32}$ , a 9-digit number. Even with 64-bit arithmetic, the largest representable value has only 19 digits. This problem allows  $k$  to be as large as 500, so you will be unable to store  $k!$  directly, since  $500!$  has 1135 digits.

In general, you should be very careful when dealing with large numbers in C or C++. In these languages, if you accidentally make the value of an `unsigned int`<sup>2</sup> larger than `UINT_MAX`, it will “wrap around” to zero without

---

<sup>2</sup>`signed ints` are even worse—their overflow behavior is undefined.

warning you.<sup>3</sup>

## 6.6 Sample Input

```
1 5 4
2 5 5
3 5 6
4 5 7
5 150 30000
6 150 30001
7 150 3022110
8 150 3022112
9 500 499
10 500 249000
11 500 249001
12 0 0
```

---

<sup>3</sup>This “feature” of C has been the culprit of several recent security vulnerabilities, called “integer overflow” attacks.

## 6.7 Correct Output

```
1 divisible
2 divisible
3 divisible
4 not divisible
5 divisible
6 not divisible
7 divisible
8 not divisible
9 divisible
10 divisible
11 not divisible
```

# # 7 : Save the Contest!

## 7.1 The Story So Far

Despite working industriously all week on setting up the Spring 2007 ND Programming Contest, the contest organizers still haven't finished writing one crucial component: the judging software itself! The judging software is responsible for checking if contestant submissions are correct. It does this by running the contestant's program against a predefined set of inputs, then comparing the contestant's program's output to known-good outputs. If the program output matches the known-good output, the judging software *accepts* the contestant's program. Otherwise, the judging software *rejects* the program.

In prior years, contestants would sometimes add too many "space" characters to their programs' output. For example, if the correct response is

```
The graph has 7 edge(s) and is not Hamiltonian.
```

a contestant might accidentally write a program that outputs

```
The graph has 7  edge(s) and is not Hamiltonian
```

(Notice the extra space after the number 7 and after the word *is*, as well as the missing period at the end. This kinda stuff actually happens.)

Since the contest is all about problem-solving, and not about mucking around with white space, the contest organizers decided that the judging software should ignore all white space and punctuation.

For example, in the above example, all the white space and punctuation would be removed from the output and compared against

```
Thegraphhas7edgesandisnotHamiltonian
```

which would then cause the program to be judged as correct.

## 7.2 The Definition of White space and Punctuation

For this contest, if a character is not a letter a through z, A through Z, or a numeral 0 through 9, it is considered white space or punctuation, and should be removed. Specifically, only the following characters should be examined when checking a program's output for correctness:

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

## 7.3 Your Task

Your task is to write a judging program that judges multiple contest entries, ignoring all white space and punctuation. The input to your program will be a contestant's program's output (possibly spanning several lines), and terminated by a line containing only the word `END`. The known-good answer will then follow, and again will be terminated by a line containing only the word `END`. If the next input line is `DONE`, your program should exit. Otherwise, your program should begin reading and judging the next entry.

For each entry, your program should output the string `"ACCEPT"` if the two outputs are identical (after white space and punctuation are removed) or else output the string `"REJECT"`.

## 7.4 Constraints and Assumptions

You may assume the following:

- No input will contain anywhere the strings `"END"` or `"DONE"`.
- No contestant's program's output or known-good answer will be longer than 1000 characters.
- Everything is encoded in 7-bit ASCII. (This just means that there are no weird characters in the inputs).

## 7.5 Sample Input

```
1 The graph has 7 edge(s)) and is not Hamiltonian
2 END
3 The graph has 7 edge(s) and is not Hamiltonian.
4 END
5 1.23 45/6!!!
6 END
7 12 34 56
8 END
9 The graph has 6 edge(s) and is not Hamiltonian.
10 END
11 The graph has 7 edge(s) and is not Hamiltonian.
12 END
13 1 <
14 ^ 2 >
15   ^ 3
16 END
17 123
18 END
19 AbC
20 END
21 ABC
22 END
23 DONE
```

## 7.6 Correct Output

1	ACCEPT
2	ACCEPT
3	REJECT
4	ACCEPT
5	REJECT