PLU February 2013 Programming Contest

Novice Problems

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 10.

2. Problems will have either no input or will read input from standard input (stdin, cin, System.in -- the keyboard). All output should be to standard output (the monitor).

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.

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

Number	**Name**
Problem 1	Welcome
Problem 2	Microsoft Logo
Problem 3	Call For Help
Problem 4	Secret Location
Problem 5	Pyramid
Problem 6	Billing
Problem 7	Parity Bit
Problem 8	Good Versus Evil
Problem 9	GCD
Problem 10	PLU Count
Problem 11	Frodo Sequence

Good luck!

1. Welcome

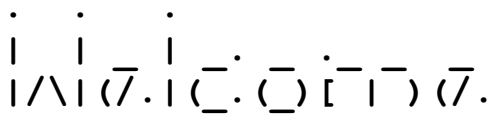
Problem Description: Print out the “Welcome” sign shown below.

Input: none

Output: Print out the picture as shown below.

Sample Input: none

Sample Output:



2. Microsoft Logo

Problem Description: Print the Microsoft logo as shown below.

Input: none

Output: Print out the picture as shown below.

Sample Input: none

Sample Output:

```
         _.-;;-._
'-..-'|   ||   |
'-..-'|_.-;;-._|
'-..-'|   ||   |
'-..-'|_.-''-._|
```

3. Call For Help

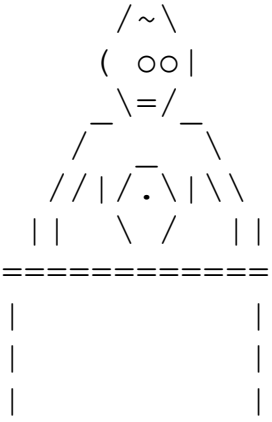
Problem Description: C3PO is in trouble. Your only chance to save him is to broadcast this ASCII picture depicting his current predicament.

Input: none

Output: ASCII picture exactly has shown below. The last 3 rows have 10 spaces between the two '|' characters.

Sample Input: none

Sample Output:



4. Secret Location

Problem Description: Someone is waiting at a secret location and has sent a text message with the coordinates of their location. The message contains the latitude and longitude coordinates in degrees, minutes, and seconds. Of course, the message is encoded so you must write a program to decode the message. The message is 6 lines and the number of characters in each line corresponds to the coordinates of the secret location.

Input: There are six lines of input.

Output: There should be two lines of output. The first line will start with the word Latitude followed by three positive integers and the integers will be separated by colons. The second line will start with the word Longitude followed by three positive integers and the integers will be separated by colons. The output should have the following format.

```
Latitude x1:x2:x3
Longitude x4:x5:x6
```

There should be one space between the first word on each line and the first positive integer on the line. The variable `xi` ($1 \leq i \leq 6$) is the number of characters in the i^{th} line of input.

Sample Input:

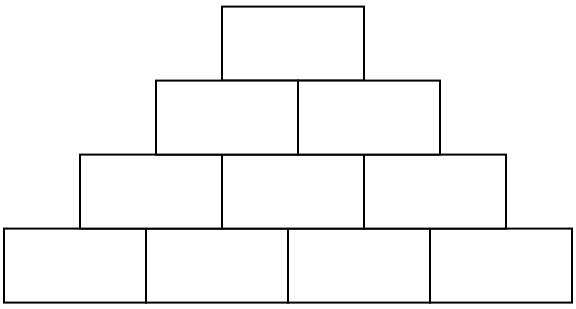
```
This is a simple message.
Each line of the input,
represents some positive integer.
The first
three lines are for latitude and
the last three lines are for longitude.
```

Sample Output:

```
Latitude 25:23:33
Longitude 9:32:39
```

5. Pyramids

Problem Description: A pyramid of blocks is constructed by first building a base layer of **n** blocks and then adding **n-1** blocks to the next layer. This process is repeated until the top layer only has one block.



You must calculate the number of blocks needed to construct a pyramid given the size of the base. For example, a pyramid that has a base of size 4 will need a total of 10 blocks.

Input: The input will be a sequence of integers, one per line. The end of input will be signaled by the integer 0, and does not represent the base of a pyramid. All integers, other than the last (zero), are positive.

Output: For each positive integer print the total number of blocks needed to build the pyramid with the specified base.

Sample Input:

```
4
6
0
```

Sample Output:

```
10
21
```

6. Billing

Problem Description: The accounting office is having troubling with billing each department for the supplies purchased last semester. Below is a chart of all the supplies and how much they cost. You must write a program that reads a list of the supplies and computes the total cost.

Item	Cost
Paper	57.99
Printer	120.50
Planners	31.25
Binders	22.50
Calendar	10.95
Notebooks	11.20
Ink	66.95

Input: The input is a list of office supplies with one item per line. End-of- input is denoted by the string "EOI".

Output: Print the total cost of all the items in the list. The output must start with a dollar sign, "$" and the total printed with exactly two decimal digits.

Sample Input:
```
Binders
Calendar
Ink
Notebooks
Binders
Ink
EOI
```

Sample Output:
```
$201.05
```

7. Parity Bit

Problem Description: A parity bit, or check bit, is a bit that is added to the end of a sequence of bits for error detection. In telecommunications and computing all data is transformed into a sequence of zeros and ones. For this problem you may assume that all information is coded using 8 bits, with the first 7 bits used for data and the last bit used as the parity bit. The parity bit is set to one if the number of ones in the preceding 7 bits is odd, and the parity bit is set to zero if the number of ones in the preceding 7 bits is even.

7 bits of data (count of 1 bits)		8 bits including parity bit
0000000	(0)	00000000
0101001	(3)	01010011
0111001	(4)	01110010
0110111	(5)	01101111
1000100	(2)	10001000

The parity bit can be used to determine if the data was transmitted incorrectly. If the parity bit is incorrect (does not match the parity of the previous 7 bits), then a parity error occurred. Your job is to read lines of data and determine the number of parity errors on each line.

Input: The first line will be a positive integer indicating the number of lines of data transmitted. Each line of data will be a sequence of zeros and ones and the length of the line will be a multiple of 8 and no longer than 64.

Output: For each data transmission line print the number of parity errors found in the data transmission.

Sample Input:

```
3
00000000
01010010
0000000101010010
```

Sample Output:

```
0
1
2
```

8. Good Versus Evil

Problem Description: Middle Earth is about to go to war. The forces of good will have many battles with the forces of evil. Different races will certainly be involved. Each race has a certain `worth' when battling against others. On the side of good and evil we have the following races, with their associated worth.

On The Side of Good	On The Side of Evil
Hobbits - 1	Orcs - 1
Men - 2	Men - 2
Elves - 3	Wargs - 2
Dwarves - 3	Goblins - 2
Eagles - 4	Uruk Hai - 3
Wizards – 10	Trolls - 5
	Wizards - 11

Although weather, location, supplies and valor play a part in any battle, if you add up the worth of the side of good and compare it with the worth of the side of evil, the side with the larger worth will tend to win. Thus, given the count of each of the races on the side of good, followed by the count of each of the races on the side of evil, determine which side wins.

Input The first line of input will contain an integer greater than 0 signifying the number of battles to process. Information for each battle will consist of two lines of data as follows. First, there will be a line containing the count of each race on the side of good. Each entry will be separated by a single space. The values will be ordered as follows: Hobbits, Men, Elves, Dwarves, Eagles, Wizards.
The next line will contain the count of each race on the side of evil in the following order: Orcs, Men, Wargs, Goblins, Uruk Hai, Trolls, Wizards. All values are non-negative integers. The resulting sum of the worth for each side will not exceed the limit of a 32-bit integer.

Output: For each battle, print “Battle" followed by a single space, followed by the battle number starting at 1, followed by a “:", followed by a single space. Then print “Good triumphs over Evil" if good wins. Print “Evil eradicates all trace of Good" if evil wins. If there is a tie, then print “No victor on this battle field".

Sample Input:

```
3
1 1 1 1 1 1
1 1 1 1 1 1 1
0 0 0 0 0 10
0 1 1 1 1 0 0
1 0 0 0 0 0
1 0 0 0 0 0 0
```

Sample Output:

```
Battle 1: Evil eradicates all trace of Good
Battle 2: Good triumphs over Evil
Battle 3: No victor on this battle field
```

9. GCD

Problem Description: The greatest common divisor (GCD) of two integers is the largest positive integer that divides the numbers without a remainder. For example, GCD of 8 and 12 is 4. You will write a program that finds the gcd of two positive integers.

Input: The first line of input will be a positive integer, n, indicating the number of problem sets. Each problem set consist of two positive integers, separated by one or more spaces. There are no blank lines in the input.

Output: For each problem set print the gcd of the two positive integers. Print each gcd on a separate line.

Sample Input:

```
3
54 24
33 22
41 103
```

Sample Output:

```
6
11
1
```

10. PLU Count

Problem Description: Given a text string you must find the number of non-interleaved occurrences of PLU in the string. The letters P, L, U must occur in order, but capitalization is not important and the letters need not be consecutive. For example the string "pppxLLLxuuu" has just one non-interleaved occurrence of PLU.

Input: The first line of input will be a positive integer, **n**, indicating the number of text strings that follow. Following the positive integer will be **n** text strings, one per line. Each text string will be at most 80 characters and there will be no blank lines.

Output: For each text string print the maximum number of non-interleaved occurrences of PLU as described above.

Sample Input:

```
3
Please find the number of parts listed under automotive.
The building was constructed for the public.
pppxLLLxuuu
```

Sample Output:

```
2
0
1
```

11. Frodo Sequence

Problem Description: The Fibonacci sequence is a famous integer sequence defined by Leonardo of Pisa in 1202. The sequence is defined as follows:

$Fib_1 = 1$
$Fib_2 = 1$
$Fib_3 = 2$
$Fib_4 = 3$
$Fib_5 = 5$
…
$Fib_n = Fib_{n-1} + Fib_{n-2}$ for all n>2

What you may not know is that Frodo of Bag End also defined an integer sequence called the Frodo sequence. Frodo's sequence is defined as follows:

$Fro_1 = 1$
$Fro_2 = 1$
$Fro_3 = 2$
$Fro_4 = 2$
$Fro_5 = 3$
…
$Fro_n = Fro_{n-1} + Fro_{n-2} - Fro_{n-3}$ for all n>3

You need to write a program that given **n** finds Fro_n.

Input: The input will be a sequence of integers, one per line. The end of input will be signaled by the integer 0. All integers, other than the last (zero), are positive and less than 2^{31}.

Output: For each positive integer, **n**, print Fro_n.

Sample Input:

```
2
4
5
6
0
```

Sample Output:

```
1
2
3
3
```