Homework 2

1. Write a program that prompts for and reads a floating-point value. The program prints the whole part on one line and the decimal (fraction) part on a second line.

For example, if the input value is 123.456, it would print the output:

the input value is 123.456

the whole part is 123

the decimal(fraction) part is 0.456

2. Write a program that asks the user to enter two numbers, obtains the two numbers from the user and prints the sum, product, difference, and quotient of the two numbers.

|  |  |  |
| --- | --- | --- |
|  | Number1 | Number2 |
| Case I | int | int |
| Case II | float | float |
| Case III | double | double |
| Case VI | int | float |

Consider and discuss the results.

3. Write a program that accepts an integer between 7 and 9 digits long.

a. Extracts and prints the third-rightmost digit of the input data.

b. Writes the integer with commas between every third digit starting from the right.

Example:

Input: 12345678

Output:

The third-rightmost digit of the input data is 6

The input data with commas between every third digit is 12,345,678

4. Write a program to calculate the diameter, the circumference, and the area of a circle with a radius of 6.75.

Assign the radius of float variable, and then output the radius with an appropriate message. Declare a named const PI with the value 3.14159.

The program should output the diameter, the circumference, and the area, each on a separate line. **Print each value to five decimal places within a total field width(欄位) of 10**.

Note1: **when the compiler reads this specifier: %10.5f**

1. **The compiler prepares 10 columns to output this real number with the five right-most columns for the fraction part.**
2. **If the real number has less than five digits in the fraction part, the compilers pads the remaining columns with zero.**
3. **The 6th column from the right is the decimal point.**
4. **The remaining four columns are the integer part. If the real number has less than four digits in the integer part, the output is padded with blank on the left.**

**%10.5f 的意思: 這個format對應的data其輸出格式如下:**

**留十個欄位來輸出實數，其中後五個欄位為小數點後面的位數，不足五位則在後方補「0」，後面數來第六位數是小數點，剩下四位放整數，整數不足四位則在前方插入空白。**

Note2: Be sure to include appropriate comments in your program, choose meaningful identifier.

5. The effective resistance of a parallel circuit with five parallel resistances

is given by:

*R* = 

Read these five resistances from the keyboard and calculate the effective

resistance *R*.

6. Solve a set of simultaneous equations:

*ax* + *by* = *c*

*dx* + *ey* = *f*

Input data: six real numbers.

Formulas for the solution:

*x* = 

*y* = 

Output all the input values *a, b, c, d, e,* and *f* and the computed values for

*x* and *y.*

*7.* 拈 (Nim) 這種遊戲遊戲**(沒有程式設計基礎的同學, 以說明方式來 解決. 有程式設計基礎的同學請儘量以程式解決. 同學們可以互相玩玩看!!)**

規則

1. 堆石子，石子的個數分別為, , …, ;

2. 遊戲者為A和B兩位，A和B輪流取石子，但由A先取；

3. 取石子的時候只能從任何一堆的石子中取1個或多個石子；

4. 拿到最後一個石子的遊戲者為勝，並且遊戲過程中雙方都採取最好的策略。

有一個名詞稱為”**輸的狀態**(losing position)”，其意思是說遊戲者從此狀態玩下去，一定會輸，假如雙方都採取最好的策略。另外一個名詞稱為”**贏的移動**(winning move)”，其意思是說這一步走下去，將留給對手**輸的狀態**。你的任務就是設計一個程式決定任一給予的狀態有多少**贏的移動。**

接著提供一個定理，此定理可作為程式設計的參考。

假設有一個狀態，其堆石子的個數分別為, , …,，並以**二進制來表示**，則此遊戲處於**輸的狀態**，若且唯若，每一位元都包含偶數個 “1”，亦即所有, , 的xor為0。

例如某一狀態有3堆石子，, , 。將, ,和 表示成二進制，如下所示

0111

1011

1101

很明顯，上面的數值，左邊3個位元，其 “1” 的個數均為偶數，但是最右邊的位元，其 “1” 的個數為奇數，因此此遊戲不是處於**輸的狀態**。

若從第3堆石子取走1個石子，從13變為12，那麼每一位元都包含偶數個 “1”，此遊戲便處於**輸的狀態**。所謂**贏的移動**，便是將**輸的狀態**留給對手。上面的例子除了從第3堆石子取走1個石子之外，也可以從第1堆或第2堆取走1個石子，均可使此遊戲處於**輸的狀態**。因此此狀態有3個**贏的移動**。

為了測試程式，輸入的資料將包含多組狀態，每一組狀態的第一列為石子的堆數, 。下一列將包含個正整數，,，代表堆石子的個數，並以空格格開。最後一列為0，亦即，表示輸入結束。

對應每一組輸入狀態，請輸出其**贏的移動**的個數。

**輸入範例**

3

7 11 13

2

1000000000 1000000000

0

**輸出範例**

3

0

*7.* Nim:

**In this problem, you may write out the program or just describe the method to solve the game. You may play this game each other.**

Nim is a 2-player game featuring several piles of stones. Players alternate turns, and on his/her turn, a player’s move consists of removing *one or more stones* from any single pile. Play ends when all the stones have been removed, at which point the last player to have moved is declared the winner. Given a position in Nim, your task is to determine how many winning moves there are in that position.

A position in Nim is called “losing” if the first player to move from that position would lose if both sides played perfectly. A “winning move,” then, is a move that leaves the game in a losing position. There is a famous theorem that classifies all losing positions. Suppose a Nim position contains *n* piles having *k*1, *k*2, …, *kn* stones respectively; in such a position, there are *k*1 + *k*2 + … + *kn* possible moves. We write each *ki* in binary (base 2). Then, the Nim position is losing if and only if, among all the *ki*’s, there are an even number of 1’s in each digit position. In other words, the Nim position is losing if and only if the *xor* of the *ki*’s is 0.

Consider the position with three piles given by *k*1 = 7, *k*2 = 11, and *k*3 = 13. In binary, these values are as follows:

111  
1011  
1101

There are an odd number of 1’s among the rightmost digits, so this position is not losing. However, suppose *k*3 were changed to be 12. Then, there would be exactly two 1’s in each digit position, and thus, the Nim position would become losing. Since a winning move is any move that leaves the game in a losing position, it follows that removing one stone from the third pile is a winning move when *k*1 = 7, *k*2 = 11, and *k*3 = 13. In fact, there are exactly three winning moves from this position: namely removing one stone from any of the three piles.

**Input**

The input test file will contain multiple test cases, each of which begins with a line indicating the number of piles, 1 ≤ *n* ≤ 1000. On the next line, there are n positive integers, 1 ≤ *ki* ≤ 1, 000, 000, 000, indicating the number of stones in each pile. The end-of-file is marked by a test case with *n* = 0 and should not be processed.

**Output**

For each test case, write a single line with an integer indicating the number of winning moves from the given Nim position.

**Sample Input**

3

7 11 13

2

1000000000 1000000000

0

**Sample Output**

3

0