Ejercicios entrenamiento para el concurso regional Programming.class 2013 que se celebrará en el IES Augustóbriga de Navalmoral de la Mata el 18 de Abril del 2013

Con el apoyo de:

IES AUGUSTÓBRIGA

I.E.S. ALBARREGAS

Instituto de Educación Secundaria Ágora Cáceres

ah

I.E.S. DONOSO CORTÉS DON BENITO

INSTITUTO CASTELAR BADAJOZ

En este documento se especifican todos los detalles de cada uno de los ejercicios de la ***TERCERA ENTREGA*** que están subidos al juez online.

Los ejercicios no están colocados por orden de dificultad, es decir, puede ser que el último ejercicio de este documento sea más sencillo de solucionar que el primero.

Cada ejercicio tiene un nombre y, precediéndole y entre paréntesis, un identificador. Este identificador es el que utilizaremos en el juez online para encontrarlo.

Para acceder al juez utilizaremos la siguiente dirección:

http://80.36.53.96/domjudge/team/

Los usuarios y contraseñas para poder subir ejercicios al juez se proporcionarán por correo electrónico.

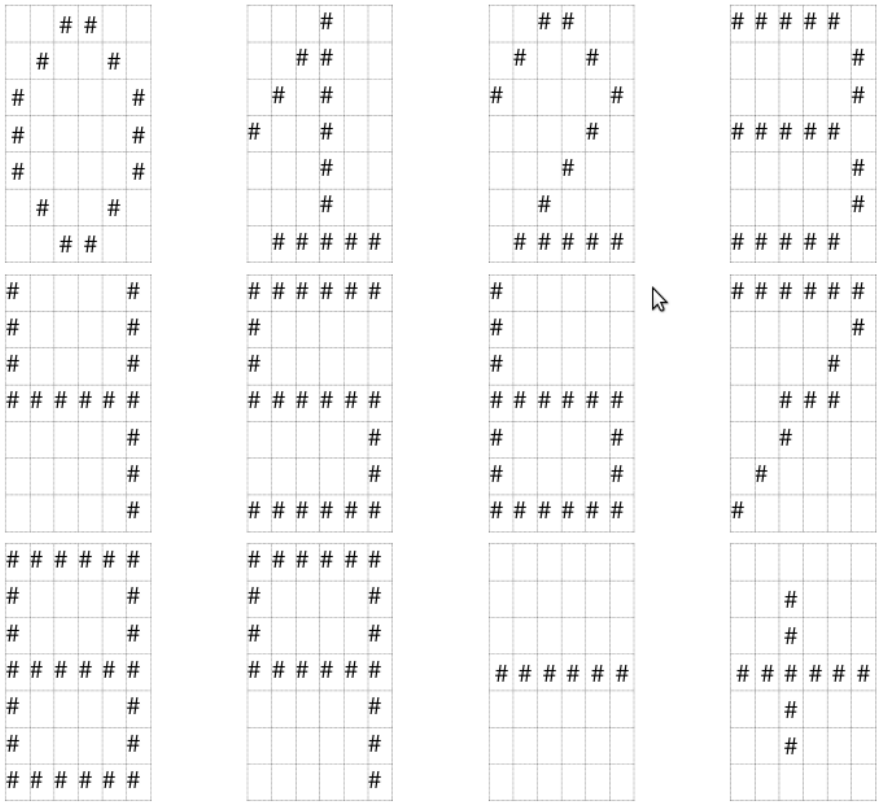
TABLA DE EJERCICIOS

(BANNERS) Banners

Descripción del problema

En el colegio Madrileño de *El Sol* se tiene una serie de alumnos con deficiencia visual. Se encuentran con el problema de que en clase de matemáticas no pueden ver las operaciones que el profesor les pone en sus cuadernos, de forma que se plantean realizar un programa informático para que los alumnos puedan ver mejor los números utilizando el ordenador en lugar del cuaderno. Se quiere que el programa muestre las operaciones que se le indique pero transformándolas en símbolos numéricos utilizando para ello el carácter #. Las operaciones permitidas serán la suma y la resta. Se podrán introducir todas las operaciones que se quiera terminando con una línea donde el primer operando sea 0.

Se considerará que cada digito del 0 al 9 y el signo + y - , se representan con una altura de 7 caracteres y un ancho de 6 caracteres. Seguidamente se muestra como se deben representar cada uno de ellos.



Entrada

La entrada del programa serán operaciones de suma o resta de la forma *numero1 + numero2* o *numero1 - numero2*. Entre el *numero1* y el *operador* irá un espacio y entre el operador y el *numero2* otro espacio. El *numero1* será un número del 0 al 999999 y el *número2* será un número del 0 al 99999.

Salida

Como salida el programa debe mostrar la operación introducida (un numero en cada fila), pero convirtiendo los números según lo indicado anteriormente (números con caracteres ASCII, en este caso se utilizará el carácter #). En el segundo numero a su izquierda se deberá poner el signo + o -, que también se pondrá en ASCII. Se tendrá en cuenta que cada número se representa exactamente como se ha indicado en el tercer párrafo (todos tendrán 6 caracteres de ancho x 7 caracteres de alto). Además entre cada cifra del mismo número se dejará un espacio en blanco. Entre el primer operando y el segundo también se dejará una línea en blanco. Al finalizar una operación se dejarán dos líneas en blanco, incluyendo la última operación introducida. Por último, indicar que los números deben aparecer alineados a la derecha (se debe suponer que en cada fila caben 6 números). Si la entrada es incorrecta, es decir, los números están fuera del rango especificado o el signo introducido es diferente del signo + o -, se mostrará el mensaje no valido

Entrada de ejemplo

```
123 + 34
543 - 458
8989898 - 87878787
2345 + 34
0
```

Salida de ejemplo

```
                          #     ##   #####
                         ##    #  #       #
                        # #    #    #      #
                       #  #       #  #####
                          #      #        #
                          #     #         #
                         #####  ##### #####

                               #####  #    #
                          #         # #    #
                          #         # #    #
                       ###### #####  ######
                          #         #       #
                          #         #       #
                               #####       #


                       ###### #    # #####
                       #      #    #       #
                       #      #    #       #
                       ###### ###### #####
                            #      #       #
                            #      #       #
                       ######      # #####

                       #    # ###### ######
                       #    # #      #    #
                       #    # #      #    #
                  ###### ###### ###### ######
                            #      # #    #
                            #      # #    #
                            # ###### ######


no valido
                    ##   #####  #    # ######
                   #  #       # #    # #
                  #    #      # #    # #
                      #  #####  ###### ######
                     #        #      #       #
                    #        #       #       #
                   ##### #####       # ######

                               #####  #    #
                          #         # #    #
                          #         # #    #
                       ###### #####  ######
                          #         #       #
                          #         #       #
                               #####       #
```

(FECHAS) Fechas correctas

Descripción del problema

El actual calendario gregoriano está establecido en 12 meses y con 30, 31, 28/29 días.

Recordad que un año es bisiesto si es divisible entre 4, excepto el último de cada siglo (el divisible por 100), salvo que este sea divisible por 400.

Confeccionar un programa que nos sirva de filtro para comprobar si una fecha que se introduce por teclado es correcta.

Entrada

Formato

En la entrada se introducirán 3 número naturales y positivos que corresponderán a día, mes y año. Todas las fechas estarán entre 1800 y 9999.

El programa finalizará cuando se introduzca un 0 en cualquiera de los tres números

Salida

La salida reflejará un "`Fecha Correcta`" si es correcta y un "`Fecha Incorrecta`" si no lo es.

Entrada de ejemplo

```
30 11 1971
6 4 1971
4 8 2001
29 2 2001
32 11 2005
30 11 2004
-20 15 2000
0 0 0
```

Salida de ejemplo

```
Fecha Correcta
Fecha Correcta
Fecha Correcta
Fecha Incorrecta
Fecha Incorrecta
Fecha Correcta
Fecha Incorrecta
```

(FERCUT) Codificación Fercut

Descripción del problema

Un alumno de la universidad de Fercut quiere realizar como trabajo final de la asignatura de computación un programa del que se pueda extraer posteriormente información acerca de la cantidad de unos y ceros que se utilizan en un texto normal, una vez convertido este a binario. Para ello se quiere realizar un programa que dado una cantidad de líneas de texto devuelva dichas líneas de texto convertidas a binarios, pero mostrándolo en agrupaciones de ceros y unos.

Entrada

Se recibirá como entrada líneas de texto (máximo 80 caracteres por línea), utilizando únicamente los caracteres imprimibles del código ASCII de 8 bits, incluido el espacio en blanco, es decir desde el código (en decimal) 32 al 126 (ambos inclusive). Se considerará que todos los caracteres que se introduzcan serán correctos, es decir, son todos caracteres imprimibles.

Se terminará cuando se escriba una línea con un único carácter imprimible. Esta línea no deberá ser convertida a binario. Es decir, se escribirán todas las líneas de entrada que se quieran y se terminará la última línea escribiendo un único caracter imprimible para indicar la finalización de las líneas de entrada.

Salida

La salida consistirá en, una vez convertida cada línea de texto a binario carácter a carácter, mostrar la cuenta de bits iguales consecutivos que nos vamos encontrando en dicha línea. Para ello, se mostrará el primer bit de la línea seguido de dos puntos, un espacio y el número de bits consecutivos iguales al primero. A continuación mostraremos para cada agrupación de ceros o unos consecutivos, el número de ellos que aparecen, separando cada agrupación de la anterior por un espacio.

Por ejemplo, si hemos escrito la línea `hola`, el texto convertido a binario sería `01101000011011110110110001100001` y se mostrará en pantalla;

```
0: 1 2 1 1 4 2 1 4 1 2 1 2 3 2 4 1
```

Se observa del ejemplo anterior que el primer bit de la cadena es un 0 que aparece una única vez. A continuación, tenemos 2 unos, seguido de 1 cero, seguido de 1 uno, seguido de 4 ceros, y así consecutivamente

Por lo tanto, el formato de salida será: `primer bit: agrupacion1 agrupacion2 agrupacion3 ... agrupacionN`

.

Entrada de ejemplo

```
WebMatrix
basada en
.Net 4.0
integra
software
para
realizar
una web
a
```

Salida de ejemplo

```
0: 1 1 1 1 1 3 1 2 2 1 1 1 1 2 3 1 2 1 2 2 1 1 1 2 4 1 1 3 1 1 3 3 2 1 2 2 1 1 2 1 1 4 3
0: 1 2 3 1 2 2 4 1 1 3 2 2 1 2 4 1 1 2 2 1 3 2 4 1 2 1 6 2 2 1 1 1 1 2 1 3 1
0: 2 1 1 3 2 1 2 3 2 2 2 1 1 1 1 3 1 1 4 1 7 2 1 1 4 1 1 3 3 2 4
0: 1 2 1 1 2 1 1 2 1 3 2 3 1 1 3 2 2 1 1 1 1 2 2 3 1 3 2 1 2 2 4 1
0: 1 3 2 2 1 2 1 4 1 2 2 2 2 3 1 1 3 3 1 3 1 2 4 1 1 3 2 1 2 2 2 1 1 1
0: 1 3 5 2 4 1 1 3 2 1 2 2 4 1
0: 1 3 2 1 2 2 2 1 1 1 1 2 4 1 1 2 1 2 3 2 1 1 2 1 1 4 1 1 2 2 4 1 1 3 2 1 1
0: 1 3 1 1 1 1 1 2 1 3 2 2 4 1 2 1 6 3 1 3 1 2 2 1 1 1 1 2 3 1 1
```

(JUG_TUTE) Jugando al Tute

Descripción del problema

El juego del tute (con cuatro jugadores), se juega con las cartas de una baraja española normal de 40 cartas. Al inicio se reparten todas las cartas y cada jugador recibe 10 cartas boca abajo, dejando el jugador que reparte una carta de las suyas boca arriba. El palo de esa carta boca arriba indica el palo del triunfo. Un jugador puede cantar 20 si tiene entre las cartas que recibe su mano el rey y el caballo del mismo palo, y si además son del palo del triunfo, en vez de 20, se cantan 40.

El objetivo es a partir de las cartas que recibe cada jugador, determinar si alguno puede cantar, y en caso afirmativo, cuanto canta cada jugador.

Entrada

Las cartas se representan de la siguiente forma:

- Un número indicando la carta (`1,2,3,4,5,6,7,10,11,12`)
- El palo de cada carta (`O` para Oros, `B` para bastos, `C` para Copas y `E` para Espadas)

Por ejemplo `3C` indicaría *Tres de copas*, `10B` indicaría *Sota de Bastos* y `12O` indicaría *Rey de oros*.

En la primera línea de la entrada, el programa leerá el número de casos de prueba que se van a introducir.

A continuación para cada caso de prueba, la entrada se compondrá de 4 líneas indicando las 10 cartas de cada jugador. En cada línea, primero se introduce el nombre del jugador (puede contener espacios), seguido de `:` y a continuación las cartas de cada jugador. Por último se introducirá una última línea indicando el palo del triunfo (ver ejemplo de entrada)

Salida

En la salida del programa, obtendríamos lo siguiente:

- Indicaciones de que puede jugadores pueden cantar. Se indicarán las opciones de cante de cada jugador por orden secuencial de la siguiente forma: `<<Jugador>> puede cantar <<veinte|cuareta>> en <<Palo>>`
- Si más de un jugador puede cantar, se mostrarán siguiendo el orden de aparición en la entrada.
- Si un jugador puede cantar en varios palos, se seguirá el orden alfabético del nombre de los palos.
- Si ningún jugador puede cantar, se indicará en la salida: `Nadie puede cantar`

Al final de cada caso de prueba se introduce una nueva línea indicando `Fin`.

Entrada de ejemplo

```
3
Jugador 1:1C5C11C12C7B1E5E1O4O7O
Jugador2:2C6C1B4B10B2E6E11E12E10O
Jugador3:3C7C2B5B11B3E7E2O5O11O
Jugador4:4C10C3B6B12B4E10E3O6O12O
E
Jugador1:1C5C1B12C7B1E5E1O4O11E
Jugador2:2C6C11C4B10B2E6E7E12E10O
Jugador3:3C7C2B5B11B3E7E2O5O11O
Jugador4:4C10C3B6B12B4E10E3O6O12O
B
Jugador 1:1C5C11C12C7B1E5E1O11E12E
Jugador2:2C6C1B4B10B2E6E10O4O7O
Jugador3:3C7C2B5B11B3E7E2O5O11O
Jugador4:4C10C3B6B12B4E10E3O6O12O
E
```

Salida de ejemplo

```
Jugador 1 puede cantar veinte en Copas
Jugador2 puede cantar cuarenta en Espadas
Fin
Nadie puede cantar
Fin
Jugador 1 puede cantar veinte en Copas
Jugador 1 puede cantar cuarenta en Espadas
Fin
```

(MEN_SECR) El mensaje secreto

Descripción del problema

Durante la Segunda Guerra Mundial, los aliados enviaban mensajes codificados en una secuencia numérica para que el enemigo no pudiera descifrar el contenido del mensaje si lo interceptaba.

La interpretación de la secuencia es la siguiente:

- El primer número indica cuántas palabras tiene el mensaje.
- Cada palabra del mensaje se representa con una secuencia numérica donde:
  - El primer número indica cuántas letras tiene la palabra.
  - El resto de la secuencia corresponde a las letras de la palabra. Cada letra está formada por dos números:
    - El primer número indica la posición de la letra en el alfabeto inglés.
    - El segundo número indica si la posición de la letra es relativa al principio del alfabeto (si el número es par) o al final (si el número es impar).

La secuencia numérica de entrada se supone correcta.

Se pide realizar un programa para descodificar mensajes.

Entrada

La entrada del programa es una secuencia numérica correspondiente a la codificación del mensaje.

NOTA: *La secuencia numérica de entrada en la **Entrada de ejemplo** se muestra en diferentes líneas para una mejor interpretación.*

Salida

La representación numérica del mensaje descodificado.

NOTA: *La secuencia alfabética de salida en la **Salida de ejemplo** se muestra en diferentes líneas para poderla relacionar más fácilmente con la secuencia numérica de entrada. En realidad sería una única línea con el mensaje de salida: **El enemigo atacara al amanecer.***

Entrada de ejemplo

```
5
2
5
-10
12
194
7
22
-11
13
197
5
40
14
-13
9
222
20
13
12
-77
7
1
-34
7
-7
26
99
24
777
1
214
9
9
1
98
2
26
49
15
7
8
26
25
13
100
26
77
14
200
22
7
3
-38
5
98
9
3
```

Salida de ejemplo

```
el enemigo atacara al amanecer
```

(MET_RUSO) Multiplicación por duplicación: método ruso

Descripción del problema

La multiplicación de dos números por duplicación utilizando el método ruso, consiste en realizar operaciones sucesivas de multiplicar por 2 uno de ellos, que llamaremos multiplicando (M), y al mismo tiempo, dividir por 2 el segundo número, que será el multiplicador (m), hasta que el multiplicador alcance el valor 1.

En cada iteración se comprueba si el multiplicador (m) es impar, en cuyo caso, se acumulará el valor correspondiente del multiplicando (M). La suma final es el resultado de la multiplicación.

Ejemplo: 82 x 27:

- `M ------ m ------- suma`
- `82 ----- 27 ------ 82`
- `164 ---- 13 ------ 246 (82+164)`
- `328 ---- 6`
- `656 ---- 3 ------- 902 (246+656)`
- `1312 --- 1 ------- 2214 (902+1312)`

Construir un programa que simule a una calculadora rusa que permita visualizar los resultados parciales de la multiplicación.

El proceso concluirá al introducir un valor 0 en cualquiera de los dos operadores

Entrada

La entrada será una secuencia de parejas de números enteros que llamaremos multiplicando al primero y multiplicador al segundo

82 27

Salida

La salida reflejará en una secuencia cada uno de los multiplicandos que correspondan a multiplicadores impares separados por el signo mas (+) y sin dejar espacios en blanco entre ellos

```
82+164+656+1312
```

Entrada de ejemplo

```
82 27
37 12
134 -65
-45 154
18 125
-25 -15
548 97
0 1
```

Salida de ejemplo

```
82+164+656+1312
148+296
-134+-8576
-90+-360+-720+-5760
18+72+144+288+576+1152
25+50+100+200
548+17536+35072
```

(DOM_RESU) Domingo de Resurrección

Descripción del problema

El Concilio de Nicea (año 325) estableció a la Cristiandad que la fiesta de Pascua de Resurrección debía celebrarse cada año "el Domingo siguiente al primer plenilunio tras el equinoccio de Primavera", fijado el 21 de Marzo. Es decir, buscamos el día en el que comienza la primavera (momento en el que el Sol atraviesa el plano del ecuador), vemos cuando es la siguiente Luna llena, y el siguiente domingo a dicha Luna llena será Domingo de Resurrección. Esta decisión se tomó para relacionar la Semana Santa con la Luna llena, de forma que durante ella haya una Luna llena, pues Jesucristo murió en la Cruz un viernes con Luna llena.

Karl F. Gauss (1777-1855), Príncipe de la Matemática, ideó un método para calcular la fecha exacta en la que celebrar la Pascua de Resurrección en base a dos incógnitas generales (A y B) y tres auxiliares (c,d,e). Según la fórmula de Gauss la fecha de Pascua debe ser una de las dos siguientes (la única que exista de las dos):

- El (22 + A + B) de Marzo,
- o el (A + B - 9) de Abril.

Diseñar un programa que nos indique los días que se correspondieron con el Domingo de Resurrección durante el siglo XX y en los 30 primeros años del XXI, es decir entre 1900 y 2030 a partir de la introducción de una cifra que se corresponde con el año.

Entrada

Se introducirá una secuencia de números que se corresponderá con un año comprendido entre 1900 y 2030 ambos inclusive. El proceso concluirá cuando se introduzca cualquier valor fuera de ese rango.

Los cálculos son los siguientes:

- c: es el resto de la división de la cifra del año / 19
- d: es el resto de la división de la cifra del año / 4
- e: es el resto de la división de la cifra del año / 7
- A: es el resto de la división de (19*c+24) / 30
- B: es el resto de la división de (2*d+4*e+6*A+5) / 7

Salida

La salida será una fecha que indique el día concreto que se corresponde con el Domingo de Resurrección.

El formato será un formato numérico del tipo "día/mes/año", por ejemplo `15/4/1990, 8/4/2012`.

Entrada de ejemplo

```
2000
1954
2005
1990
2012
1908
2011
1500
1
```

Salida de ejemplo

```
23/4/2000
25/4/1954
27/3/2005
15/4/1990
8/4/2012
19/4/1908
24/4/2011
```

(TAR_LLUV) Tarde de Lluvia

Descripción del problema

Una tarde de lluvia Javier y tres de sus amigos se han reunido en casa de este. Como están muy aburridos y no saben qué hacer, su abuelo les ha propuesto jugar a los cálculos matemáticos. El juego consiste en que cada uno de los chavales indica al abuelo un número entre 0 y 150. Javier le indica una operación y el abuelo calcula el valor que se obtiene al aplicar dicha operación sobre ellos. Las operaciones que entran en juego son la suma, el producto y la media aritmética, eso sí, esta última será sin decimales, porque al abuelo le cuesta demasiado obtenerlos.

El abuelo realiza las operaciones de una manera asombrosa, tanto que Javier se ha convertido en un héroe para sus amigos. El problema es que tienen que fiarse de lo que dice el abuelo, porque aún no saben usar la calculadora. Para ayudarles a comprobar si el resultado obtenido es correcto, realiza un programa que, dados 4 valores muestre su producto, suma y media aritmética.

Entrada

El programa recibirá una cantidad determinada de grupos de *4 números*, todos ellos *positivos* y *menores de 150*. El número de ocurrencias de la lista vendrá determinado por el primer valor introducido.

Salida

El programa devolverá la suma, el producto y la media aritmética de los valores introducidos. Dicha salida tendrá el siguiente formato:

```
suma:valor->producto:valor->media:valor
```

Donde valor es el obtenido al realizar la operación correspondiente sobre el grupo de valores. El valor correspondiente a la media se obtendrá sin decimales.

Nota: Observad que no existe espacio entre ninguno de los caracteres de la salida.

Entrada de ejemplo

```
4
2 7 3 9
15 23 18 19
13 14 16 7
8 8 8 8
```

Salida de ejemplo

```
suma:21->producto:378->media:5
suma:75->producto:117990->media:18
suma:50->producto:20384->media:12
suma:32->producto:4096->media:8
```

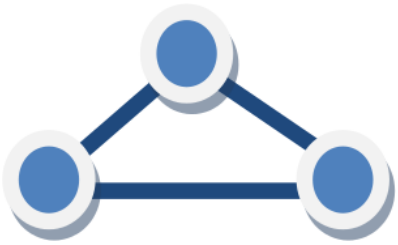
(TOP_MALL) Topología en malla completa

Descripción del problema

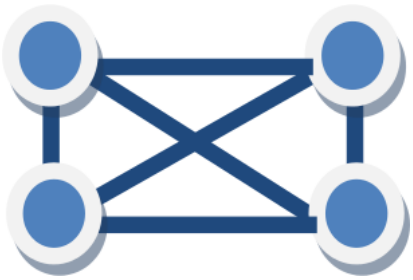
En la topología en malla completa cada nodo de una red se conecta con todos los demás, de forma que los datos pueden viajar del nodo origen al destino directamente o siguiendo distintas rutas.

Para 2 nodos, necesitamos 1 enlace:

Para 3 nodos, necesitamos 3 enlaces:



Para 4 nodos, necesitamos 6 enlaces:



Entrada

El programa recibirá una cantidad determinada de números positivos, menores o iguales a 25.000, que serán el *número de nodos* de la topología en malla. El programa termina cuando la entrada sea -1.

Salida

El programa devolverá el *número de enlaces* necesario para cada número de nodos recibido a la entrada.

Entrada de ejemplo

```
6
25
158
322
-1
```

Salida de ejemplo

```
15
300
12403
51681
```

(TRACKMAN) Trackman

Descripción del problema

La crisis nos obliga a pluriemplearnos, es por ello que hemos ingresado como DJ's en un prestigioso local de ocio nocturno. Aún y así su rudimentario sistema informático hace necesaria la aplicación de nuestros conocimientos para automatizar el cálculo de la duración de nuestras sesiones dance, trance y techno house.

Diseñar un programa que, dada una serie de duraciones de canciones de varios discos, sea capaz de calcular la duración final de cada disco.

Entrada

Se recibirá una primera fila con un solo número que indicará cuántas filas o discos se introducirán a continuación.

El primer número de cada fila nos indicará cuantas canciones le suceden. Cada canción está separada por un espacio y su duración tiene el formato de minutos:segundos. Todos los valores inferiores a diez llevan un cero delante (por ejemplo: 09 en lugar de 9). Pese a ser poco probable, en el caso de que una canción durara más de una hora también debe aceptarse el formato horas:minutos:segundos. El número máximo de canciones por línea es de catorce y no puede superarse dicho valor.

La entrada no debe contener errores como números negativos, números decimales o letras extrañas presentes de forma errónea. Sí puede darse el caso de pistas con duraciones anómalas como minutos o segundos con valores superiores a 59, en cuyo caso se describe en el apartado siguiente cómo deben ser tratados.

Salida

Para cada línea de la entrada con canciones se informará con una sola línea indicando la duración final en horas:minutos:segundos de todas las canciones de esa línea. En el poco probable caso de superar las 24 horas NO se informará con un nuevo campo correspondiente al dia. Se dejará con el cómputo total de horas sea cual sea este. En el caso de que se detectara un minuto o segundo superior a 59 en la duración de alguna de las canciones, se procederá a evaluar la siguiente línea dando para la presente el mensaje de *ERROR EN DISCO*.

En el caso de que haya cómputos de horas, minutos o segundos inferiores a 10 debe de colocarse un cero delante para forzar una longitud mínima de 2 dígitos por campo. Si no se supera la hora de duración debe mostrarse igualmente el valor 00 en la posición correspondiente a las horas.

Entrada de ejemplo

```
6
2 02:34 59:43
5 01:01:23 04:45 03:23 08:32 05:29
4 43:54 29:32 61:09 23:29
5 03:43 02:35 09:21 06:19 04:32
0
8 02:23 04:24 05:53 02:54 09:29 18:07 08:07 12:12
```

Salida de ejemplo

```
01:02:17
01:23:32
ERROR EN DISCO
00:26:30
00:00:00
01:03:29
```