Resolución de problemas en *¡Acepta el reto!* y DOMjudge

Durante el curso, tendrás que resolver problemas en el portal *¡Acepta el reto!* (https://www.aceptaelreto.com) o en el DOMjudge de la asignatura.

Todos los problemas son aplicaciones *de consola* (línea de órdenes), que recibirán datos por la *entrada estándar* (`cin`) y tendrán que escribir la respuesta por la salida estándar (`cout`). Para poder saber si está bien, el juez probará tu solución *muchas veces*. Por ejemplo, si el problema pide decir si un número es par o impar, no lo probará con un único número, sino con muchos.

Para eso, tu programa *debe ayudarle*, y estar preparado para responder *múltiples veces* a la misma pregunta. En el ejemplo, en lugar de recibir un único número, decir si es par o impar y terminar, tu programa deberá leer del teclado muchos números, y responder, para cada uno, si es par o impar. A cada consulta lanzada a tu programa se le conoce con el nombre de *caso de prueba*.

Al necesitar responder muchas veces a la misma pregunta, con diferentes datos de entrada, surge la duda de cuando debería tu programa terminar. Dependiendo del problema, se realizará de un modo u otro de acuerdo a *tres esquemas* de la entrada:

1. Al principio de la ejecución el programa recibe el número de casos de prueba que tendrá que procesar.

2. El programa va leyendo casos de prueba hasta llegar a uno especial, que marca el final.

3. El programa va leyendo casos de prueba hasta consumir todos los datos de la entrada (algo así como si el teclado "se desconectara").

En este documento te proporcionamos *esqueletos* de código fuente que puedes usar para cada uno de los tres esquemas.

Ten en cuenta que los jueces on-line son *estrictos* con el formato de la entrada y salida. Por ejemplo, si el enunciado dice que la entrada siempre será un número positivo, *no* debes preocuparte por reaccionar dignamente si no lo es, porque ese código nunca se ejecutará. De la misma forma, debes ser estricto con la salida. *No escribas nada no pedido*. En particular, en estos problemas *no* seas amigable con el usuario (¡el usuario es un ordenador!). Si escribes cosas como "Dame el siguiente número", confundirás al juez automático y te dirá que tu solución es incorrecta.

1. Número de casos de prueba al principio

En este esquema de la entrada, el programa recibe un número inicial que indica cuantos *casos de prueba* deben procesarse. Es el esquema más sencillo. En el programa principal se lee ese número, y se realiza un bucle que repite el proceso tantas veces como se haya solicitado. Se usa, por ejemplo, en el problema *117 La fiesta aburrida*[1] y *216 Goteras*[2].

```
#include <iostream>
using namespace std;

void casoDePrueba() {

   // TU CÓDIGO AQUÍ

} // casoDePrueba

int main() {
   unsigned int numCasos, i;

   cin >> numCasos;

   for (i = 0; i < numCasos; ++i) {
      casoDePrueba();
   }

   return 0;

}
```

2. Caso de prueba especial que marca el final

En este esquema de la entrada, hay un caso de prueba especial que marca el fin, y le indica al programa que debe detenerse. Es algo equivalente a las marcas "XXX" que has utilizado (o utilizarás) con los ficheros.

En el esqueleto de solución que te proponemos, se implementa una *función* que procesa cada caso de prueba y que, además, devuelve si lo hizo o se llegó al final. La función lee el caso de prueba y si es el especial que marca el fin, termina devolviendo falso (`false`). En otro caso, procesa el caso de prueba y devuelve cierto (`true`).

Este esquema de la entrada lo utilizan, por ejemplo, los problemas *155 Perímetro de un rectángulo*[3] y *239 Pi. Pi. Pi. Pi. Pi. Piiiii*[4].

[1] https://www.aceptaelreto.com/problem/statement.php?id=117 : La fiesta aburrida
[2] https://www.aceptaelreto.com/problem/statement.php?id=216 : Goteras
[3] https://www.aceptaelreto.com/problem/statement.php?id=155 : Perímetro de un rectángulo
[4] https://www.aceptaelreto.com/problem/statement.php?id=239 : Pi. Pi. Pi. Pi. Pi. Piiiii.

```
#include <iostream>
using namespace std;

bool casoDePrueba() {

   Leer caso de prueba
   if (es el caso que marca el final)
      return false;
   else {
      // CÓDIGO PRINCIPAL AQUÍ
      return true;
   }

} // casoDePrueba

int main() {

   while(casoDePrueba()) {
   }

   return 0;

}
```

3. Fin de la entrada (EOF)

Este es el esquema menos habitual. En él, simplemente, deja de haber datos en la entrada, algo así como si el teclado se desconectara. Esto no es demasiado natural cuando la entrada es el teclado, pero sí lo es cuando es un fichero.

Para probar los problemas que siguen este esquema, cuando decidas que no quieres introducir más datos por teclado deberás pulsar Ctrl + Z e "Intro" si usas Windows, y Ctrl + D si usas GNU/Linux o Mac.

El esqueleto que te proponemos para las soluciones a los problemas que usan este esquema es similar al anterior. Implementaremos una *función* que procesa cada caso de prueba y que, además, indica si pudo hacerlo o no (se llegó al final).

Lo primero que hará será *intentar leer* los datos del siguiente caso de prueba. Si lo consiguió, continuará, y en otro caso parará. La comprobación de si tuvo o no éxito en la lectura se hace preguntándoselo a `cin`.

Este esquema lo usan por ejemplo los problemas *147 Las 15 cerillas*[5] o *149 San Fermines*[6].

[5] https://www.aceptaelreto.com/problem/statement.php?id=147 : Las 15 cerillas
[6] https://www.aceptaelreto.com/problem/statement.php?id=149 : San Fermines

```
#include <iostream>
using namespace std;

bool casoDePrueba() {

   Leer caso de prueba
   if (!cin)
      return false;
   else {
      // CÓDIGO PRINCIPAL AQUÍ
      return true;
   }

} // casoDePrueba

int main() {

   while(casoDePrueba()) {
   }

   return 0;

}
```

Si el caso de prueba supone la lectura de varios valores (por ejemplo los dos números del problema *147 Las 15 cerillas* mencionado antes), es preferible leer únicamente el primer dato, comprobar si se pudo leer, y continuar el procesamiento normal si todo fue bien. Es decir en la parte del esqueleto de *Leer caso de prueba*, leeríamos únicamente el primer número, y en la parte de `// CODIGO PRINCIPAL AQUÍ` leeríamos el segundo, que en ese punto siempre existirá.