Programa-Me 2011
Cómo resolver los problemas de Programa-Me

Patrocinado por

Universidad Complutense Madrid

Facultad de Informática

Vicerrectorado de Informatica y Comunicaciones

Realizado en

IES Antonio de Nebrija. Móstoles

Cómo resolver los problemas de Programa-Me

1 ¿Por dónde empezar?

Cuando comienza el concurso se da a los participantes el enunciado de un número fijo de problemas (entre 8 y 10) de distinta dificultad. La competición consiste en intentar resolver el mayor número de problemas en el menor tiempo posible. Ganará el equipo que tenga más problemas resueltos correctamente cuando termine el tiempo.

Es importante destacar que en caso de empate (dos o más equipos han resuelto el mismo número de problemas) se utiliza *el tiempo total* para desempatar (también aquí hay que sumar el tiempo por las penalizaciones de envíos incorrectos). Eso se traduce en que es preferible resolver los problemas *cuanto antes*.

Teniendo en cuenta esto, los equipos experimentados en concursos de este tipo suelen comenzar *repartiendose los enunciados* de los ejercicios para leerselos rápidamente y evaluar de forma rápida cuáles son más fáciles y cuáles son más difíciles. Tras este periodo inicial de lectura, y una vez que se tienen más o menos ordenados los problemas por orden de dificultad, se empieza por el más fácil. De esta forma se pretende tener ejercicios resueltos lo antes posible para, en caso de empate, llevarse la victoria.

2 ¿Qué hacer durante el concurso?

Durante el concurso sólo hay un ordenador por equipo, por lo que sólo una persona puede estar tecleando. Es habitual, no obstante, que al menos dos personas estén pensando y programando el ejercicio en cuestión, para evitar errores.

Mientras tanto, el otro miembro del equipo puede estar pensando en la solución al siguiente problema, e incluso plantear el programa en un papel para poder trasladarlo más rápidamente al fichero de código fuente cuando el ordenador "quede libre".

Tened en cuenta que, una vez enviada la solución de un problema, la respuesta de los jueces puede tardar algún tiempo. Mientras tanto el ordenador queda libre y esa tercera persona que estaba pensando en el otro ejercicio puede comenzar a resolverlo (con la ayuda de otro compañero).

Es muy normal durante el concurso tener dos o tres problemas "abiertos" a medio programar. Por ejemplo, si la respuesta de los jueces es negativa y hay algún error, uno de los miembros del equipo puede dedicarse a intentar encontrar el fallo (sobre el papel) mientras los otros dos siguen adelante con la programación de otro ejercicio.

3 Cómo son los problemas

Los problemas del concurso son de muy distinta índole. Habrá ejercicios cuya única dificultad es la programación mientras que habrá otros donde lo más complicado es saber *cómo se resuelve* el ejercicio.

Por ejemplo, el ejercicio de muestra de los cuadrados diabólicos y esotéricos tiene una solución relativamente sencilla: guardar en una matriz de enteros el contenido del cuadrado y sumar casillas con cuidado para ver si se cumplen las condiciones indicadas.

Otros problemas pueden tener una solución corta (desde el punto de vista de programación) pero que es difícil de llegar a ella. El problema de ejemplo de los móviles entra en esta categoría. Una vez entendido el enunciado lo complicado es saber *cómo* se puede programar para que funcione. Este es un claro ejemplo de enunciado que puede dedicarse a pensar un miembro del equipo mientras los otros dos están utilizando el ordenador resolviendo otro.

Nota: Es importante resaltar el hecho de que el juez on-line exige que el main finalice con un return 0. De no hacerse así no acepta el problema y nos notificara que se ha producido un error

4 Entrada de datos

Todos los problemas utilizan el mismo esquema: dado un caso de entrada (un cuadrado mágico, la configuración de un móvil, un mensaje encriptado) hay que escribir algo sobre él (el tipo de cuadrado, si está o no equilibrado, el número de vocales).

Para que se pueda probar si el programa funciona, éste tendrá que leer numerosos casos de entrada, uno detrás de otro, y dar la respuesta para todos ellos. Para hacerlo, hay dos alternativas:

- Al principio de la ejecución, el programa lee *el número de casos de prueba* que se utilizan.
- El programa va leyendo casos de prueba hasta que se encuentra con un caso de prueba *especial* (por ejemplo un cuadrado mágico de dimensiones 0,0).

Dependiendo de si es una u otra alternativa el esquema general del programa será distinto. Como consejo recomendamos que se utilice una función/método que se encargue de resolver un caso de prueba y que sea llamado desde el programa principal tantas veces como sea necesario.

Como ejemplo, una solución en C++ de un problema que comienza con el número de casos de prueba podría tener el siguiente esquema:

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

// Resuelve un caso de prueba, leyendo de la entrada la
//  configuracion, y escribiendo la respuesta.
```

```
void casoDePrueba() {
   // ...
}

int main() {

   int numCasos;
   cin >> numCasos;

   for (int i = 0; i < numCasos; i++)
      casoDePrueba();

   return 0;
}
```

Si la entrada del problema en vez de empezar con el número de casos de prueba termina con un *caso de prueba especial*, podemos utilizar el siguiente:

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

// Resuelve un caso de prueba, leyendo de la entrada la
// configuracion, y escribiendo la respuesta. Si el caso
// de prueba leido es el que marca el final de la ejecucion,
// la funcion devuelve false para indicar al main que hay
// que terminar.
bool casoDePrueba() {
   // ...
   // Leemos de la entrada
   cin >> ...

   if (caso_especial)
      return false;

   // ... resolvemos ...

   return true;
}

int main() {

   bool seguir;

   do {
```

```
        seguir = casoDePrueba();
    } while (seguir);

    // Tambien se puede resumir en una unica linea:
    // while (casoDePrueba()) ;
}
```

5 Ejemplo de ejercicio con casos de prueba limitados

Para completar, veamos un ejemplo muy sencillo. Imaginemos que nos piden un problema tan simple como éste[1]:

Entrada

La primera línea de la entrada contendrá el número de casos de prueba que el programa debe leer. A continuación vendrá uno detraś de otro todos esos casos. Cada uno de ellos consiste en una única línea con un número entero.

Salida

Para cada caso de prueba el programa escribirá `PAR` si el caso de prueba es un número par y escribirá `IMPAR` si el número es impar.

Las soluciones en C, C++ y Java aparecen a continuación. Como se ve, en todas ellas se sigue el mismo esquema descrito en la sección anterior.

```
// SOLUCION EN C
#include <stdio.h>

// Resuelve un caso de prueba, leyendo de la entrada la
//  configuracion, y escribiendo la respuesta.
void casoDePrueba() {
    int num;

    scanf("%d\n", &num);

    if ((num % 2) == 0)
        printf("PAR\n");
    else
        printf("IMPAR\n");
}

int main() {
```

[1] Los problemas del concurso serán más complicados. Este lo utilizamos como ejemplo para ver cómo sería el esquema de la solución.

```
    int numCasos;
    int i;
    scanf("%d\n", &numCasos);

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

  return 0;
}
```

```
// SOLUCION EN C++
#include <iostream>
using namespace std;

// Resuelve un caso de prueba, leyendo de la entrada la
//  configuracion, y escribiendo la respuesta.
void casoDePrueba() {
    int num;

    cin >> num;

    if ((num % 2) == 0)
        cout << "PAR\n";
    else
        cout << "IMPAR\n";

}

int main() {

    int numCasos;
    cin >> numCasos;

    for (int i = 0; i < numCasos; i++)
        casoDePrueba();

    return 0;
}
```

```
// SOLUCION EN Java
class solution {
```

```
    static java.util.Scanner in;

    public static void casoDePrueba() {
       int n;
       n = in.nextInt();

       if ((n % 2) == 0)
          System.out.println("PAR");
       else
          System.out.println("IMPAR");
    }

    public static void main(String args[]) {

        in = new java.util.Scanner(System.in);

        int numCasos;
        numCasos = in.nextInt();
        for (int i = 0; i < numCasos; i++)
           casoDePrueba();
    }
}
```

6 Ejemplo de ejercicio con casos de prueba ilimitados

Los jueces podrían haber puesto el mismo problema pero cambiando el formato de los casos de entrada. En ese caso el enunciado tendría la forma siguiente:

Entrada

La entrada consistirá en un número indeterminado de casos de prueba. Cada caso de prueba consistirá en un número entero. Los casos de prueba terminarán con el número `-1`, que marcará el final de la entrada y que no será procesado.

Salida

Para cada caso de prueba el programa escribirá `PAR` si el caso de prueba es un número par y escribirá `IMPAR` si el número es impar.

Las soluciones en C, C++ y Java aparecen a continuación. Como se ve, en todas ellas se sigue el mismo esquema descrito más arriba en el documento.

```
// SOLUCION EN C
#include <stdio.h>
```

```
int casoDePrueba() {
    int num;

    scanf("%d\n", &num);

    if (num == -1)
        // Marca de fin de entrada
        return 0;

    if ((num % 2) == 0)
        printf("PAR\n");
    else
        printf("IMPAR\n");

    return 1;
}

int main() {

    while (casoDePrueba())
        ;
}
```

```
// SOLUCION EN C++
#include <iostream>
using namespace std;

bool casoDePrueba() {
    int num;

    cin >> num;

    if (num == -1)
        return false;

    if ((num % 2) == 0)
        cout << "PAR\n";
    else
        cout << "IMPAR\n";

    return true;
}
```

```
int main() {

    while (casoDePrueba())
       ;
}
```

```
// SOLUCION EN Java
class solution {

     static java.util.Scanner in;

     public static boolean casoDePrueba() {
        int n;
        n = in.nextInt();

        if (n == -1)
           return false;

        if ((n % 2) == 0)
           System.out.println("PAR");
        else
           System.out.println("IMPAR");

        return true;
     }

     public static void main(String args[]) {

         in = new java.util.Scanner(System.in);

         while (casoDePrueba())
            ;
     }
}
```