Olimpiada Nacional de Informática
Problemas para la Tercera Ronda 2013

Problema 1

Una biblioteca cuenta una cantidad suficiente de libreros para mantener ordenados todos sus libros, cada librero tiene capacidad para 50 libros como máximo. Un estudiante de Ingeniería obsesionado por la estética y el orden entra en pánico al entrar a esta biblioteca y notar que entre todos los libros, con solo 4 alturas diferentes, no existe ningún criterio de ordenamiento para ubicarlos en los libreros. Si bien no sabe cuantos libros hay, sabe que son menos de 300, ante esto el estudiante toma el desafío de ordenarlos por tamaño, ordenando de más alto a menos alto.

Realice un programa que reciba como entrada un fichero **libros.in** que contenga la cantidad N de libros en la primera línea y en la segunda los N tamaños de libros separados por un espacio, ejemplo:

```
7
30 15 23 17 23 30 30
```

El programa debe validar que se cumplan las exigencias del problema y escribir sus resultados en un fichero **libros.out** que contenga el número de librero en una línea y en la siguiente los tamaños de libros que se colocarán allí separados por un espacio, ejemplo:

```
1
30 30 30 23 23 17 15
```

Cuando exista más de un librero, se pondrán en las siguientes lineas nuevamente el número de librero y en una siguiente línea los tamaños de los libros ordenados, igual que en el ejemplo anterior, ejemplo:

Número de libero >
50 libros del primer librero en una sola línea >
Número de libero >
Todos los libros del librero en una sola línea >

```
1
30 30 30 23 23 17 15 ...
2
7 6 3 2 2 1
```

Problema 2

Queremos implementar una solución para un popular juego de carreras, "Carrera Aguarandú". Consiste en una carrera de obstáculos en la que el jugador avanza automáticamente hacia delante sin tener que presionar tecla alguna para acelerar, por tanto no puede detenerse o frenar, es así que para poder esquivar los obstáculos sólo puede presionar izquierda o derecha para cambiar de carril, y llegar hasta uno donde no tenga obstáculos al frente, o en último caso mantenerse en su carril actual.

La pista de carrera cuenta con 3 carriles y además de la línea de posición actual se muestra al jugador las siguientes 4 posiciones, la solución que se pide sea implementada debe leer de un archivo **carrera.in** una matriz de 3 filas con 5 columnas, donde para cada lugar libre hay un guión bajo y para cada obstáculo una X. Cada elemento está separado de otro por un espacio, ejemplo:

```
X _ _
_ _ X
_ X _
_ X _
_ _ _
```

La posición inicial en la carrera será siempre la posición (5,2) de la matriz, y como se dijo antes el jugador avanza automáticamente hacia delante, o lo que es lo mismo en este gráfico va "hacia arriba" (de la fila 5 a la 4, de la 4 a la 3 y así hasta llegar a la primera fila).

- Con la orden **izquierda**, la posición actual cambia por la misma fila pero una columna menos que la actual, si estoy en la primera columna simplemente y realizo la orden de ir a la izquierda simplemente sigo en el carril actual.
- Con la orden **derecha**, la posición actual cambia por la misma fila pero una columna mas que la actual, si estoy en la última columna y realizo la orden de ir a la derecha, simplemente sigo en el carril actual.
- También existe la orden "**no cambiar**" que me mantiene en mi carril o columna actual.
- Tras ejecutarse el resultado de cada orden, automáticamente avanzo una fila hasta llegar a la primera.

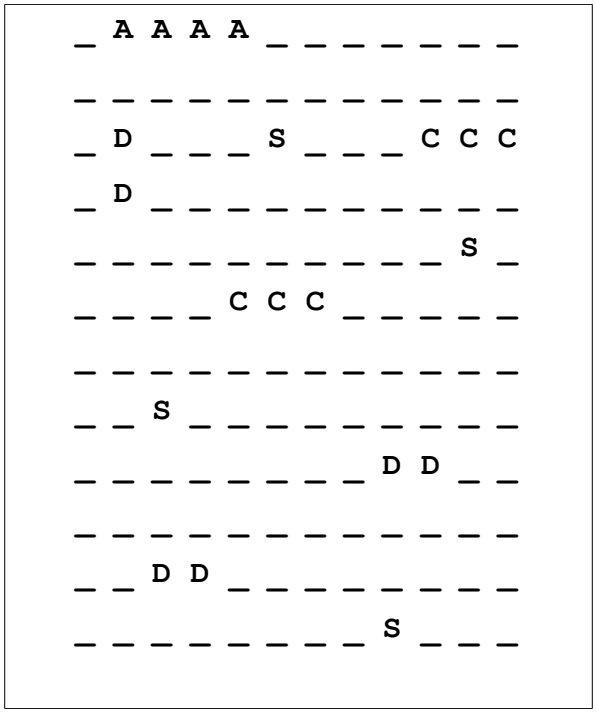
Tras leer el archivo, en caso de no cumplir con las indicaciones anteriores el sistema retornará un mensaje de error, sin embargo si la validación fue exitosa la misión de la implementación será escribir en otro fichero **carrera.out** la serie de combinaciones izquierda o derecha que permiten ganar la carrera. Solo debes escribir una dirección por línea, ejemplo de salida que soluciona al ejemplo anterior:

```
izquierda
no cambiar
no cambiar
derecha
```

Problema 3

Se desea implementar un juego de "Batalla Naval", el cual consiste en un tablero cuadrado de dimensión dinámica (sólo puede tener desde 10 x 10 hasta 12 x 12 casilla). En el tablero se colocan diferentes cantidades de 4 tipos de piezas, según la siguiente disposición:

- 4 submarinos (cada submarino ocupa solo una casilla del tablero)
- 3 destructores (cada destructor ocupa dos casillas consecutivas del tablero, **no** pueden ser en diagonal)
- 2 cruceros (cada crucero ocupa tres casillas consecutivas del tablero, **no** pueden ser en diagonal)
- 1 acorazado (cada acorazado ocupa cuatro casillas consecutivas del tablero, **no** pueden ser en diagonal)



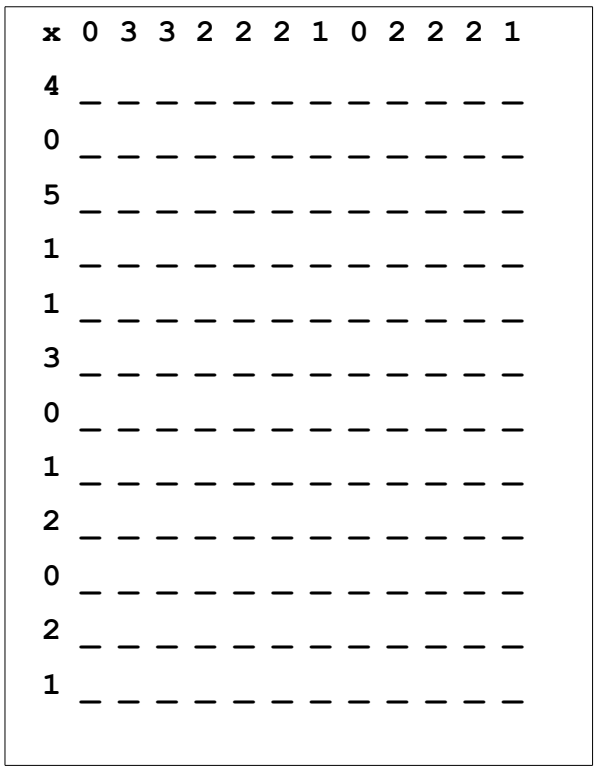
El programa debe ser capaz de leer de un archivo **batalla.txt** una matriz donde el contenido de cada casilla se denotará por la inicial de las piezas recien citadas, y si la casilla está vacía se colorá el símbolo guión bajo _. Cada casilla estará separada de otra por un espacio. Un ejemplo del formato de batalla.txt es el siguiente:

Si la matriz ingresada no cumple las condiciones de dimensión mencionadas al principio, o las de formato según lo expuesto en el ejemplo anterior, el programa debe arrojar en pantalla un mensaje de error.

En caso que la validación sea exitosa, comienza el juego, tras leer el archivo batalla.txt se solicitará en pantalla que se seleccione un nivel de dificultad:

- Fácil: Permite hasta 20 errores.
- Medio: Permite hasta 15 errores.
- Difícil: Permite hasta 10 errores.

Tras eso se mostrará en pantalla la cantidad de filas y columnas ocupadas, para el ejemplo anterior se debe mostrar en pantalla algo similar a la siguiente tabla, donde en la parte superior se puede ver cuantas casillas de cada columna están ocupadas por alguna pieza, con los números de cada columna separados por espacios. De la misma forma se muestra cuantas casillas están ocupadas en cada fila.



Tras desplegar la tabla anterior en pantalla, se debe solicitar al usuario que ingrese una posición donde desea lanzar un misil (fila,columna).

Si esa posición no existe para la matriz, se debe mostrar un mensaje de error , sin embargo si la posición existe para la matriz de entrada, entonces se debe retornar una "tabla resultado" que consiste en pantalla igual a la anterior, pero en lugar de guión bajo _ en esa posición debe mostrarse **O** si en la posición solo había agua, y **X** si en la casilla existía alguna pieza o parte de una pieza.

La siguiente tabla muestra la tabla tras 2 jugadas, primero el usuario ingreso (1,2) y luego de haber visto una tabla resultado ingreso otra posición, en este ejemplo (10,5). Entonces se muestra en pantalla una nueva tabla resultado para esta segunda jugada, la cual se verá similar al siguiente ejemplo:

x	0	3	3	2	2	2	1	0	2	2	2	1
4	_	X	_	_	_	_	_	_	_	_	_	_
0	_	_	_	_	_	_	_	_	_	_	_	_
5	_	_	_	_	_	_	_	_	_	_	_	_
1	_	_	_	_	_	_	_	_	_	_	_	_
1	_	_	_	_	_	_	_	_	_	_	_	_
3	_	_	_	_	_	_	_	_	_	_	_	_
0	_	_	_	_	_	_	_	_	_	_	_	_
1	_	_	_	_	_	_	_	_	_	_	_	_
2	_	_	_	_	_	_	_	_	_	_	_	_
0	_	_	_	_	O	_	_	_	_	_	_	_
2	_	_	_	_	_	_	_	_	_	_	_	_
1	_	_	_	_	_	_	_	_	_	_	_	_

El juego termina una vez que el usuario haya alcanzo su límite de fallos (definido por el nivel de dificultad antes mencionado) y por tanto pierde la partida, o cuando logra acertar todas las piezas del tablero.

Cuando el usuario gana el juego, se debe imprimir un mensaje de felicitaciones. Cuando pierde se debe desplegar en pantalla una tabla con las posiciones de todas las piezas que no pudo descubrir, manteniendo las **X** y **O** de las posiciones que sí pudo descubrir.

Problema 4

Existe una esquina de Asunción con complicaciones de tránsito, para ello debemos implementar un semáforo que solucione la vida tanto a los conductores como a los peatones. En la esquina se cruzan una Avenida preferencial y una Calle secundaria, ambas calles son unidireccionales. En un instante dado se tiene registro de la cantidad de autos que están en esperando en cada calle y en base a esa información el semáforo debe decidir a cuál de las dos calles dará el color verde, y la cantidad de **pasos**, es decir la cantidad de autos que podrán pasar ese semáforo hasta que ese semáforo cambie nuevamente de color.

Existen además dos restricciones adicionales:

- En la Avenida preferencial no puede esperar más de **X** pasos en la calle secundaria.
- Cualquier auto en la Calle secundaria no puede esperar más de **Y** pasos en la Avenida preferencial, siendo **Y > X**.

El fichero de entrada debe llamarse **transito.txt** y seguir exactamente el formato del siguiente cuadro. En la primera línea se ubica la cantidad de autos esperando en la Avenida preferencial, en la segunda línea está la cantidad de autos esperando en la calle secundaria. En las siguientes líneas están los valores de X e Y que se utilizan en las restricciones antes mencionadas.

```
Avenida 15
Calle 9
X 7
Y 10
```

Si existen problemas en el formato, el sistema deberá arrojar un error en pantalla. Si todo está en orden, se creará un fichero **verde.txt** donde se registrará cuál será la planificación según la cual cada calle se pondrá de color verde y la cantidad de pasos que durará en ese color, esto hasta que el último auto del archivo de entrada logre cruzar la esquina. Para el ejemplo anterior, el resultado escrito en verde.txt será el siguiente:

```
Avenida 10
Calle 7
Avenida 5
Calle 2
```