1. `EsPico`: Devuelve verdadero si la parcela en unas coordenadas, (x, y), es un pico.
2. `ObtenerPicos`: Construye y devuelve un objeto de la `clase SecuenciaPuntos` con las coordenadas de las parcelas que son picos.
3. `Fusion`: Involucra dos objetos de la `clase Relieve` para construir y devolver un nuevo objeto de esta clase con las siguientes características:
   - El largo/ancho del nuevo objeto será el mínimo entre los largos y anchos de los objetos involucrados en la operación.
   - Cada parcela (x, y) del nuevo objeto tendrá como altura el valor máximo entre las alturas de dicha parcela en los objetos involucrados en la operación.

Para implementar estos métodos, puede incorporar los métodos adicionales que considere oportunos. Si lo hace, justifique si deben ser públicos o privados. Además, considere que dispone de la implementación ya terminada de las clases `Punto2D` y `SecuenciaPuntos`, mostradas en la Tabla 2.

Punto2D
- int posX - int posY
+ Punto2D(int x, int y) + Punto2D() + int ObtenerPosX() + int ObtenerPosY() + double Distancia(Punto2D otro)

SecuenciaPuntos
- static const int MAX = 100 - Punto2D lista[MAX] - int utilizados
+ SecuenciaPuntos() + void Aniade(Punto2D pto) + Punto2D ObtenerPunto(int i) + int Utilizados()

Table 2: Clases disponibles y métodos que **NO hace falta implementar**.

Suponga que, en la función `main` dispone de objetos `r1` y `r2` ya creados de la clase Relieve. Especifique cómo haría para:

- Mostrar las coordenadas y la altura del pico más alto de `r1`.
- Calcular la longitud de cable requerida para unir los picos de `r1` (en el orden dado por el método `ObtenerPicos`).
- Mostrar las coordenadas de los picos del relieve que se obtendría al fusionar los objetos `r1` y `r2`.

◁ Ejercicio 3 ▷ Rima asonante [3.5 puntos]

Suponga que dispone de la implementación del código de la clase `SecuenciaCaracteres` mostrada en la Tabla 3. Sobre dicha clase, implemente un método que compruebe si una secuencia `s1` rima de forma asonante con otra secuencia `s2` con grado `k`. Esto significa que las últimas `k` vocales de `s1` y `s2` deberán coincidir. El método devolverá `true` si las dos secuencias `s1` y `s2` riman de forma asonante, y `false` en caso contrario (si alguna secuencia tiene menos de k vocales, devolverá `false`). Implemente todos los métodos auxiliares que estime oportuno. No hace falta construir el programa principal, pero incluya al menos la línea en la que se realizaría la llamada al método pedido.

SecuenciaCaracteres
- static const int TAMANIO = 100 - char vector_privado[TAMANIO] - int total_utilizados
+ SecuenciaCaracteres() + int TotalUtilizados() + int Capacidad() + void Aniade(char nuevo) + char Elemento(int indice)

Table 3: Clase disponible y métodos que **NO hace falta implementar**.

Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

Fundamentos de Programación (2019/2020)
1º Doble Grado en Matemáticas e Informática
Examen de Teoría
10 de Enero de 2020

DECSAI

Normas para la realización del examen: **Duración: 2.5 horas**

- **El único material permitido durante la realización del examen es un bolígrafo azul o negro.**
- **Debe disponer de un documento oficial que acredite su identidad a disposición del profesor.**
- **No olvide escribir su nombre completo y grupo en todos y cada uno de los folios que entregue.**

◁ Ejercicio 1 ▷ Robot [3 puntos]

Se dispone de un robot que se mueve en una línea, utilizando pasos de longitud fija. Inicialmente, el robot se ubica en la posición `pos`, siendo `pos` un valor entero positivo con $1 \leq \texttt{pos} \leq 100$. Luego, el robot ejecuta una serie de órdenes, indicadas mediante un array `ord` de tipo char, con longitud `lon`. Cada orden es una letra 'I' o 'D', indicando si el robot se mueve a la izquierda (decrementando la posición actual `pos` en 1 unidad) o a la derecha (incrementando la posición actual `pos` en 1 unidad). Las posiciones válidas del robot cumplen $\texttt{pos} \in \{1, 100\}$. Se dice que **una serie de órdenes es correcta si el robot nunca se sale de las posiciones válidas**.

Se pide implementar un programa (directamente en la función `main`) que, dada una posición inicial `pos`, una lista de órdenes `ord` y la longitud `lon` de dicha lista, haga lo siguiente:

- Si la serie de órdenes es correcta, muestre cuántas veces se visitó cada posición.
- Si la serie de órdenes NO es correcta, el programa terminará indicando cuántas órdenes se pudieron ejecutar.

No hace falta que escriba el código de programa que lee los valores de `pos`, `lon` y `ord`.

Ejemplos

`Posición inicial pos: 10`
`Longitud lon: 6`
`Órdenes ord: DDIIII`
 	
Posiciones visitadas: 10→11→12→11→10→9→8
Salida del programa
Serie de órdenes: correcta
Frecuencia de visitas por posición: (8,1), (9,1), (10,2), (11,2), (12,1).
	`Posición inicial pos: 1`
	`Longitud lon: 4`
	`Órdenes ord: DIID`
	
	Posiciones visitadas: 1→2→1
	Salida del programa
	Serie de órdenes: incorrecta.
	Se ejecutaron 2 órdenes.

◁ Ejercicio 2 ▷ Relieve [3.5 puntos]

El relieve de una región geográfica se puede representar mediante una tabla rectangular t, donde cada elemento $t[x][y]$ representa la altura (sobre el nivel del mar), de la parcela (x, y) del terreno (x e y son enteros). Una parcela dada se define como "pico" si sus 8 parcelas vecinas tienen una altura menor. Por simplicidad, se supone que las parcelas en los bordes de la región no pueden ser picos. Se propone la representación para la clase Relieve mostrada en la Tabla 1. Remarcar que **todos los objetos de esta clase estarán registrados respecto a la coordenada** $(0, 0)$.

Relieve
- static const int MAX_LARGO = 100 - static const int MAX_ANCHO = 100 - int t[MAX_LARGO][MAX_ANCHO] - int largo, ancho
+ Relieve(int la, int an) + int ObtenerLargo() + int ObtenerAncho() + int ObtenerAltura(int x, int y) + void ModificarAltura(int x, int y, int alt)

Table 1: Propuesta para la clase Relieve

Se pide implementar los siguientes métodos miembro: