

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 ▷ Nuevo Orden Enteros

[3.5 puntos]

Se pretende establecer un nuevo orden para comparar valores enteros positivos. Un valor entero a será mayor que otro b si el número de dígitos nueve es mayor en a que en b . Si el número de nueves es igual en los dos, el mayor será el que tiene más ochos. Si hay empate también con este dígito, se considera el siete. Así hasta llegar al cero, si fuese necesario. Si la frecuencia de todos los dígitos es igual en ambos valores, se les considera iguales, bajo este orden. Se pide lo siguiente:

- a) Diseñar una función que reciba dos enteros positivos, a y b , y devuelva `true` si a es mayor estricto a b , atendiendo al orden especificado y `false` en caso contrario.
- b) Realizar una función `main` que ordene de forma ascendente `v`, un array de `util` enteros siguiendo este nuevo criterio. La ordenación se realizará en el mismo vector `v`. Solo hay que implementar la parte de cálculo. Suponga declarados

```
const int TAM = 100;
int v[TAM];
int util; // util <= TAM
```

Por ejemplo, si `v` contiene los valores 5000, 2244, 9000, 99, 550, tras la ordenación quedará 2244, 5000, 550, 9000, 99.

◁ Ejercicio 2 ▷ Clase Multiconjunto

[3.5 puntos]

En matemáticas, un *multiconjunto* es una modificación del concepto de conjunto que, a diferencia de éste, permite múltiples instancias para cada uno de sus elementos. El número entero positivo de instancias de cada elemento se llama *multiplicidad* de este elemento en el multiconjunto. Por ejemplo, en el multiconjunto $\{5, 5, 5, 3, 3, 2\}$, 5 tiene multiplicidad 3, 3 tiene multiplicidad 2 y 2 tiene multiplicidad 1.

Los elementos de un multiconjunto se toman en un conjunto fijo U (*universo*). Dado un multiconjunto A en un universo U , para cada elemento $x \in U$, definimos $m_A(x)$ como la multiplicidad de x en A . Entonces, se pueden definir las siguientes operaciones sobre multiconjuntos:

- El **soporte** de un multiconjunto A en un universo U , se define como $Sop(A) = \{x \in U \mid m_A(x) > 0\}$.
- A está **incluido** en B , denotado como $A \subseteq B$, si $\forall x \in U, m_A(x) \leq m_B(x)$.
- La **intersección** de A y B es un multiconjunto C tal que $m_C(x) = \min\{m_A(x), m_B(x)\} \forall x \in U$.

Se requiere implementar la clase `Multiconjunto` con $U = \{1, 2, \dots, 1000\}$. Observe que no es necesario representar explícitamente todas las repeticiones de cada valor. Así pues, en vez de representar el multiconjunto $\{5, 5, 5, 3, 3, 2\}$ como una secuencia de enteros repetidos, resulta más conveniente trabajar con la multiplicidad de cada valor del universo U . Por lo tanto, para representar el anterior multiconjunto basta con considerar cada valor del dominio (5, 3, 2) y sus multiplicidades asociadas (3, 2, 1).

Se pide lo siguiente. Suponga definida la clase `SecuenciaEnteros` (véase final ejercicio).

- a) Defina los datos miembro de la clase y los constructores que estime oportunos.

- b) Implemente un método para obtener la multiplicidad de un número perteneciente a U en un multiconjunto y un método para añadir un elemento x al multiconjunto con cierta multiplicidad k . Si el elemento ya existe en el multiconjunto, se incrementará su multiplicidad en k .
- c) Diseñe métodos que implementen las tres operaciones definidas con anterioridad (soporte, inclusión e intersección). El soporte devolverá un objeto de la clase `SecuenciaEnteros`, la inclusión un `bool` y la intersección un objeto de la clase `Multiconjunto`.
- d) Implemente un método que devuelva un objeto de la clase `SecuenciaEnteros` con **todos** los elementos de un multiconjunto. No se requiere que estén ordenados. Para el ejemplo anterior, se devolvería $\{5, 5, 5, 3, 3, 2\}$.

SecuenciaEnteros
<i>Datos miembro privados:</i>
- static const int TAMANIO = 100; - int vector_privado[TAMANIO]; - int total_utilizados;
<i>Métodos públicos que NO hay que implementar y se pueden usar:</i>
+ SecuenciaEnteros(); + int TotalUtilizados(); + int Capacidad(); + void Aniade(int nuevo); + int Elemento (int indice);

◁ Ejercicio 3 ▷ Clase RedSocial

[3 puntos]

Implementar una clase `RedSocial` para gestionar una red social. La red debe almacenar los nombres de los usuarios y las relaciones de amistad. Se propone la siguiente representación:

```
class RedSocial{
private:
    static const int MAXIMO_USUARIOS = 100;
    string Usuarios[MAXIMO_USUARIOS];
    bool RelacionesAmistad[MAXIMO_USUARIOS][MAXIMO_USUARIOS];
    int usuarios_utiles;
public:
    void HacerAmigos(string usr1, string usr2);    // Métodos que NO hay que implementar
    void DeshacerAmigos(string usr1, string usr2); // y se pueden usar
    bool SonAmigos(string usr1, string usr2);
    bool Existe(string usr);
    ...
};
```

donde `RelacionesAmistad` es una matriz binaria tal que la componente (i, j) es `true`, si el usuario i es amigo del usuario j , y `false`, en otro caso. Asumiremos que las relaciones de amistad son simétricas, lo que implica que la matriz binaria también lo es.

Se pide implementar, al menos, los siguientes métodos (los usuarios vienen especificados por sus nombres):

- a) Un método que, dado un usuario A de la red, sugiera un amigo potencial para A . Un amigo potencial es aquel usuario de la red (no amigo de A) que tiene más amigos en común con A .
- b) Un método que decida si dos usuarios son *amigos circunstanciales*, es decir, si son amigos pero no tienen amigos en común.

Implemente cualquier otro método adicional, público o privado, que considere útil.