

---

# Relación de ejercicios tema 4: Clases en C++ (ampliación) y tema 5: Sobrecarga de operadores

---

## Contenido:

1	Introducción	1
2	Clases, constructores y destructores, métodos	1
3	Clases, constructor de copia y sobrecarga de operadores	3

---

## 1 Introducción

Los ejercicios propuestos están relacionados con los conceptos de constructor, destructor, métodos de acceso a datos miembro, modularización, etc. Los ejercicios deben implementarse de forma completa, lo que implica que para cada uno de ellos debe existir:

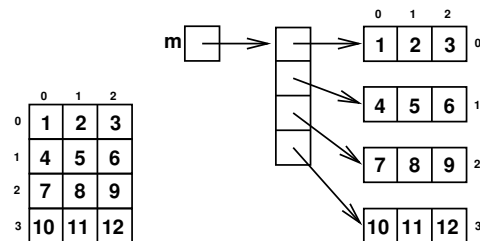
- Un archivo **.h** con las declaraciones.
- Un archivo **.cpp** con la implementación.
- Un archivo **makefile** para generar el ejecutable.
- Una estructura de directorios similar a la usada en las prácticas, para que todos los elementos que constituyen el programa queden organizados de forma clara.

## 2 Clases, constructores y destructores, métodos

1. Implementad la clase **Racional**, destinada a soportar el trabajo con números fraccionarios de la forma **a/b**, siendo **a** y **b** números enteros. Proponed una representación para la clase e incluid los siguientes métodos:
  - (a) Constructor sin argumentos, para construir un objeto que represente al valor 0.
  - (b) Constructor para crear un racional a partir de un número entero.
  - (c) Constructor para crear un racional a partir de dos enteros: numerador y denominador.
  - (d) Destructor.
  - (e) Métodos para devolver los valores de numerador y denominador.
2. Implementad la clase **VectorDinamico** para trabajar con vectores de enteros de tamaño arbitrario y no definido a priori. La clase debe contar con los siguientes elementos:
  - (a) Constructor sin argumentos, que crea un vector vacío.
  - (b) Constructor con un argumento, que indica el número de posiciones deseadas en el vector, e inicializa todos los elementos a 0.
  - (c) Constructor con dos argumentos: número de casillas y valor para inicializar todas las casillas. Fíjese que podemos implementar todos los constructores con un único método si usamos valores por defecto.

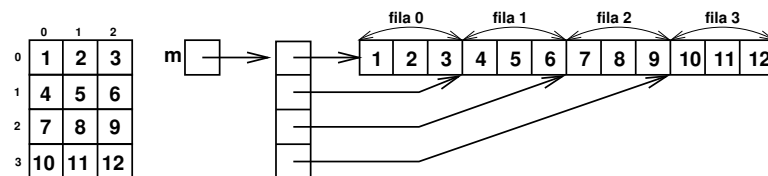
- (d) Destructor.
- (e) Método para consultar el tamaño del vector.
- (f) Método para obtener el valor en la posición  $i$ .
- (g) Método para modificar el valor en la posición  $i$ .
- (h) Método para aumentar el tamaño del vector en un determinado número de posiciones (pasado como argumento). Los nuevos elementos se inicializan a 0, pero el método se encarga de preservar todos los que hubiese.

3. Implementad la clase **Matriz2D\_1**, cuya organización interna es la siguiente:



dotándola de los siguientes métodos (debería realizarse la implementación de la forma más modularizada posible, introduciendo métodos auxiliares siempre que sea necesario):

- (a) Constructor sin argumentos, que crea una matriz vacía.
  - (b) Constructor con dos argumentos: número de filas y columnas. Todos los valores se inicializan a 0.
  - (c) Constructor con tres argumentos: número de filas, columnas y valor inicial de todas las casillas.
  - (d) Destructor.
  - (e) Métodos para modificar y devolver el valor de una celda concreta.
  - (f) Métodos para obtener el número de filas y el número de columnas.
4. Igual que en el ejercicio anterior, pero con otra estructura interna para almacenar los valores de la matriz (en este caso la clase será **Matriz2D\_2**), tal y como se indica a continuación:



5. Implementad la clase **Lista** para trabajar con listas dinámicas (de tamaño arbitrario, y no definido a priori) de datos de tipo **TipoBase**. Cada nodo de la **Lista** estará enlazado con el siguiente. Se debe aportar la siguiente funcionalidad:

- (a) Constructor sin argumentos para crear una lista vacía.
- (b) Destructor.
- (c) Método para devolver el número de elementos en la lista.
- (d) Método para devolver el elemento de una posición dada.
- (e) Método para modificar el elemento de una posición dada.
- (f) Método para insertar un elemento al principio.
- (g) Método para insertar un elemento en una posición dada.
- (h) Método para insertar un elemento al final.

- (i) Método para borrar un elemento en una posición dada.
- 6. Implementad la clase **Pila**: estructura de datos donde los elementos se acceden siguiendo una política **LIFO** (last in, first out). La clase debe proporcionar la siguiente funcionalidad:
  - (a) Constructor sin argumentos, creando una pila vacía.
  - (b) Destructor.
  - (c) Método para añadir un valor a la pila.
  - (d) Método para extraer un valor de la pila (consultará el valor del elemento ubicado en la cima y lo elimina de la pila).
  - (e) Método para consultar si la pila está vacía.
  - (f) Método para consultar el valor del elemento ubicado en la cima de la pila sin borrarlo.
- 7. Igual que en el ejercicio anterior, pero en este caso para la clase **Cola** (la política de acceso es ahora **FIFO** (first in, first out)).

### 3 Clases, constructor de copia y sobrecarga de operadores

Los ejercicios propuestos están relacionados con los anteriores, pero centrados ahora en los conceptos de constructor de copia y sobrecarga de operadores. Se trata aquí de añadir funcionalidad adicional a los ejercicios incluidos en la primera parte de la relación.

1. Ampliad la clase **Racional** con las siguientes operaciones:
  - (a) Sobrecarga de los operadores unarios  $+$  y  $-$ .
  - (b) Sobrecarga de los operadores aritméticos binarios  $+$ ,  $-$ ,  $*$ ,  $/$ , con el objeto de poder operar entre dos racionales y racionales con enteros (en cualquier orden).
  - (c) Sobrecarga de los operadores aritméticos binarios  $+=$ ,  $-=$ ,  $*=$  y  $/=$ .
  - (d) Sobrecarga de los operadores relacionales binarios  $==$ ,  $!=$ ,  $<$ ,  $>$ ,  $>=$  y  $<=$  para poder comparar racionales con racionales y racionales con enteros en cualquier orden.
  - (e) Sobrecarga de los operadores  $<<$  y  $>>$  para insertar un número racional en un flujo y extraer un número racional de un flujo. La inserción/extracción se realiza de la siguiente forma: sea **r** un dato de la clase **Racional**:
    - Si **r** contiene el valor  $3/5$  entonces **cout** mostrará  $3/5$ .
    - La ejecución de **cin >> r** hará que se lea una cadena de caracteres y se procese adecuadamente para separar numerador y denominador.
2. Ampliad la clase **VectorDinamico** de la sección anterior, agregando:
  - (a) Constructor de copia (empleando código reutilizable).
  - (b) Sobrecarga del operador de asignación (empleando código reutilizable).
  - (c) Reescribid el destructor empleando código reutilizable.
  - (d) Sobrecarga del operador `[]` para que sirva de operador de acceso a los elementos del vector, de forma que pueda actuar tanto como **lvalue** como **rvalue**.
  - (e) Sobrecarga de los operadores relaciones binarios  $==$  y  $!=$  para comparar dos vectores dinámicos. Dos vectores serán iguales si tienen el mismo número de elementos y sus contenidos son idénticos (y ocupan las mismas posiciones).
  - (f) Sobrecarga de los operadores relacionales binarios  $>$ ,  $<$ ,  $>=$  y  $<=$  para poder comparar dos vectores. Usar un criterio similar al que se sigue en la comparación de dos cadenas de caracteres clásicas (orden lexicográfico).
  - (g) Considerad una implementación nueva para redimensionar un vector: emplear los operadores binarios  $+$ ,  $-$ ,  $+=$  y  $-=$  de manera que, por ejemplo:

- Si **v** es un vector, la instrucción **v = v+1** crea un nuevo vector con un elemento más, generado a partir del último sumando 1.
  - Si **v** es un vector, la instrucción **v2 = v+1** crea un vector dinámico con un elemento más que **v**, lo rellena a partir de **v** y agrega un elemento más tal y como se ha indicado en el punto anterior.
  - Si **v** es un vector, la instrucción **v -= 10** crea uno nuevo con 10 casillas menos que **v** (descarta las 10 últimas).
- (h) Sobrecarga los operadores **<<** y **>>** para leer/escribir un vector dinámico. Para la implementación del operador **>>** leerá una secuencia indefinida de valores (separados por espacios), hasta que se introduzca el valor **\***. Los valores se leerán en una cadena de caracteres, y sólo se convertirán al tipo **TipoBase** cuando se verifique que son válidos para su almacenamiento (no se ha introducido el terminador **(\*)**).
3. Ampliad la clase **Matriz2D\_1** con los siguientes métodos:
- (a) Constructor de copia y sobrecarga del operador de asignación, empleando código reutilizable.
  - (b) Reescribe el destructor en base a la estrategia anterior.
  - (c) Sobrecarga alternativa del operador de asignación, que recibe como argumento un dato de **TipoBase** e inicia toda la matriz al valor especificado.
  - (d) Sobrecarga del operador **()** para que sirva de operador de acceso a los elementos de la matriz y pueda actuar como **lvalue** y **rvalue**.
  - (e) Sobrecarga los operadores unarios **+** y **-**.
  - (f) Sobrecarga los operadores relacionales binarios **==** y **!=** para poder comparar matrices dinámicas: para la igualdad han de contener el mismo número de filas y columnas y mismos valores en cada posición.
  - (g) Sobrecarga el operador **<<** para mostrar el contenido de la matriz.
4. Igual que en el ejercicio anterior, pero para la clase **Matriz2D\_2**.
5. Ampliad la clase **Lista** (de datos **TipoBase**) con los siguientes métodos:
- (a) Constructor de copia y sobrecarga del operador de asignación, empleando código reutilizable. Reescribid el destructor en base a esta estrategia.
  - (b) Sobrecarga del operador **[]** para que sirva de operador de acceso a los elementos de la lista y pueda actuar tanto como **lvalue** como **rvalue**. El índice hace referencia a la posición, de tal manera que 1 indica el primer nodo, 2 el segundo, etc.
  - (c) Sobrecarga de los operadores **<<** y **>>** para leer/escribir una lista.
    - Para la implementación del operador **>>** leerá una secuencia indefinida de valores, hasta que se introduzca el valor **\***. Los valores se leerán en una cadena de caracteres, y sólo se convertirán al tipo **TipoBase** cuando se verifique que son válidos para su almacenamiento (no se ha introducido el terminador **\***).
    - El nuevo valor siempre se guardará al final.
6. Ampliad la clase **Pila** (de datos **TipoBase**) con los siguientes métodos:
- (a) Constructor de copia y sobrecarga del operador de asignación, mediante código reutilizable.
  - (b) Reescribe el destructor en base a la estrategia anterior.
  - (c) Sobrecarga el operador **<<**.
7. Ampliad la clase **Cola** (de datos **TipoBase**) con los siguientes métodos:
- (a) Constructor de copia y sobrecarga del operador de asignación, mediante código reutilizable.
  - (b) Reescribe el destructor en base a la estrategia anterior.
  - (c) Sobrecarga el operador **<<**.

8. Implementad la clase **Conjunto**, de forma que permita manipular un conjunto de elementos de tipo **TipoBase**. Para la representación interna de los elementos del conjunto, usad una lista de celdas enlazadas, que debería mantenerse ordenada para facilitar su tratamiento. Las operaciones con que cuenta son:
- (a) Constructor sin argumentos que crea un conjunto vacío.
  - (b) Constructor con un argumento, de tipo **TipoBase**: crea un conjunto con el elemento proporcionado como argumento.
  - (c) Constructor de copia (empleando código reutilizable).
  - (d) Destructor (empleando código reutilizable).
  - (e) Método para consultar si el conjunto está vacío.
  - (f) Sobrecarga del operador de asignación (empleando código reutilizable).
  - (g) Método que devuelva el número de elementos en el conjunto.
  - (h) Método para determinar si un valor **TipoBase** pertenece al conjunto.
  - (i) Sobrecarga de los operadores  $\gg$  y  $\ll$ .
    - Para  $\ll$  se leerá una secuencia indefinida de valores hasta la introducción del valor  $*$ . Los valores se leerán en una cadena de caracteres y sólo se convierten a **TipoBase** cuando se haya comprobado su validez (no se ha introducido el terminador  $*$ ).
    - No se permiten elementos repetidos.
  - (j) Sobrecarga de los operadores  $==$  y  $!=$  para comparar conjuntos. Dos conjuntos son iguales si tienen el mismo número de elementos y éstos son idénticos (independientemente de su posición).
  - (k) Sobrecarga del operador binario  $+$  para calcular la unión de dos conjuntos. La operación responde a los siguientes criterios:
    - Si  $a$  y  $b$  son conjuntos,  $a+b$  será otro dato de tipo **Conjunto** que contendrá todos los elementos de ambos, sin los elementos repetidos, o sea  $a \cup b$ .
    - Si  $a$  es un conjunto y  $v$  es un valor de **TipoBase**,  $a+v$  será otro dato de tipo **Conjunto** que contendrá  $a \cup \{v\}$ .
    - Si  $a$  es un conjunto y  $v$  es un valor de **TipoBase**,  $v+a$  será otro dato de tipo **Conjunto** que contendrá  $\{v\} \cup a$ .
  - (l) Sobrecarga el operador binario  $-$  para calcular la diferencia de conjuntos de forma similar al anterior:
    - Si  $a$  y  $b$  son conjuntos,  $a-b$  será otro dato de tipo **Conjunto** que contendrá el resultado de quitar de  $a$  los elementos que están en  $b$ , o sea  $a - b$ .
    - Si  $a$  es un conjunto y  $v$  es un valor de **TipoBase**,  $a-v$  será otro dato de tipo **Conjunto** que contendrá  $a - \{v\}$ .
  - (m) Sobrecarga el operador binario  $*$  para calcular la intersección de dos conjuntos. La operación responde a los siguientes criterios:
    - Si  $a$  y  $b$  son conjuntos,  $a*b$  será otro dato de tipo **Conjunto** que contendrá  $a \cap b$ .
    - Si  $a$  es un conjunto y  $v$  es un valor de **TipoBase**,  $a*v$  será otro dato de tipo **Conjunto** que contendrá  $a \cap \{v\}$ .
    - Si  $a$  es un conjunto y  $v$  es un valor de **TipoBase**,  $v*a$  será otro dato de tipo **Conjunto** que contendrá  $\{v\} \cap a$ .