Memoria (¡otra vez!)

El objetivo de este control es implementar un TAD genérico sencillo, así como familiarizarse con la gestión de memoria dinámica en clases C++

1) Trabajo a realizar

Debe implementarse un TAD genérico *Memoria*<T> que simule una memoria para una máquina virtual en la que cada una de sus celdas puede estar en dos de los estados siguientes: (i) no inicializada; o (ii) almacenando un valor de tipo T. Dicho TAD proporcionará las siguientes operaciones:

- Un constructor `Memoria(c)` que construye una memoria con *c* celdas, todas ellas no inicializadas. Las celdas comienzan a direccionarse desde 0, por lo que el espacio de direcciones válido de esta memoria será el rango [0..*c*) (es decir, de 0 a *c*-1).
- Una operación observadora `valor` que toma como argumento una dirección *d*, y devuelve el valor almacenado en dicha dirección. La dirección debe estar en el espacio de direcciones válido de la memoria (en caso contrario, se levanta la excepción `EDireccionNoValida`). Así mismo, la celda direccionada debe haber sido inicializada (en caso contrario, se levanta la excepción `ECeldaSinInicializar`)
- Una operación mutadora `almacena` que toma como argumentos una dirección *d* y un valor *v*, y almacena *v* en la dirección *d*. La dirección debe estar en el espacio de direcciones válido de la memoria (en caso contrario, se levanta la excepción `EDireccionNoValida`).
- Una operación observadora `inicializada` que devuelve `true` si la dirección de memoria ha sido inicializada, y `false` en caso contrario. La dirección debe estar en el espacio de direcciones válido de la memoria (en caso contrario, se levanta la excepción `EDireccionNoValida`).

Consideraciones sobre la representación

Dado que la capacidad de la memoria es un parámetro del constructor, como representación de la memoria debe utilizarse un array dinámico (no se permitirán soluciones que utilicen las clases y plantillas de la STL -tales como `vector`-, implementaciones de otros TADS, etc.). Así mismo, ***no*** deberá suponerse que el tipo T utilizado para instanciar T soporta necesariamente un constructor por defecto. Por tanto, las celdas del array no podrán ser directamente de tipo T, debiéndose utilizar, en su lugar, punteros a T. Una celda estará inicializada cuando apunte a algún valor, y no inicializada cuando valga **nullptr**.

Dado que la implementación estará basada en el manejo de memoria dinámica, deberán incluirse los constructores y métodos adicionales necesarios para garantizar el correcto funcionamiento, así como para evitar cualquier pérdida de memoria.

2) Código de apoyo

Se proporcionan los siguientes archivos:

- `Memoria.h`. Este archivo debe completarse con la implementación del TAD genérico pedido.
- `main.cpp`. Programa de prueba. Este archivo no debe modificarse. El programa mantiene una memoria en la que los valores almacenados son pares de enteros (la clase que representa dichos pares *no* incluye constructor por defecto). El programa comienza leyendo la capacidad de la memoria. A continuación, lee y ejecuta *comandos*. Los comandos leídos son de los siguientes tipos:
  - `almacena` *d x y*: Almacena el par de enteros (*x*,*y*) en la dirección *d*. Imprime OK en caso de que la dirección sea válida, DIRECCION_INVALIDA en otro caso.
  - `valor` *d*: Recupera el valor almacenado en la posición *d*. Imprime el valor recuperado en caso de que la dirección sea válida, y la celda esté inicializada. En caso contrario, imprime ERROR_DE_LECTURA.
  - `inicializada` *d*: Si la dirección es válida, imprime SI si la celda en la dirección *d* ha sido inicializada, y NO si no lo ha sido. Si la dirección no es válida, imprime DIRECCION_INVALIDA.
  - `termina`. Termina la ejecución.

A continuación, se muestra un ejemplo de entrada / salida:

Entrada	**Salida**
10 almacena 5 5 6 inicializada 5 inicializada 8 almacena 8 10 20 inicializada 8 valor 8 valor 5 valor 4 valor 11 almacena 10 1 1 inicializada 10 termina	OK SI NO OK SI 10 20 5 6 ERROR_DE_LECTURA ERROR_DE_LECTURA DIRECCION_INVALIDA DIRECCION_INVALIDA