Ejercitación – Punteros

Se recomienda resolver los ejercicios en orden. En CLion se encuentran disponibles los siguientes targets:

- ej$N$ — para $1 \leq N \leq 5$, ejecuta los tests.
- ej$N$_correrValgrind — para $N \in \{3, 4, 5\}$, ejecuta el test indicado con la herramienta `valgrind`, verificando que no haya problemas con el manejo de la memoria dinámica.

Los targets también pueden compilarse y ejecutarse sin usar CLion. Para ello:

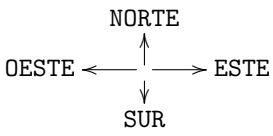
1. En una consola pararse en el directorio raíz del projecto. En este debería haber un archivo CMakeLists.txt.
2. Ejecutar el comando `$ cmake .` (incluyendo el punto). Esto generará el archivo `Makefile`.
3. Ejecutar el comando `$ make TARGET` donde `TARGET` es uno de los targets mencionados anteriormente. Esto creará un ejecutable con el nombre del target en el directorio actual.
4. Ejecutar el comando `$ ./TARGET` siendo `TARGET` el nombre del target utilizado anteriormente. Esto correrá el ejecutable.

La herramienta `valgrind` también puede ejecutarse manualmente, con el siguiente comando: `valgrind --leak-check=full ./TARGET`

Introducción

Las instancias de la clase `Mapa<T>` representan **mapas** que almacenan elementos de tipo `T`. Un mapa es una grilla de tamaño `ancho` × `alto` donde `ancho` y `alto` son números naturales (estrictamente positivos). Las coordenadas del mapa son de la forma (x, y) donde $0 \leq x <$ `ancho` y $0 \leq y <$ `alto`. Un mapa cuenta con un **cursor** que está posicionado sobre una coordenada. El mapa permite mover el cursor hacia cualquiera de las cuatro direcciones (`NORTE`, `ESTE`, `SUR` y `OESTE`).

$(0, \texttt{alto})$	$(1, \texttt{alto})$	...	$(\texttt{ancho}, \texttt{alto})$
...	...	...	...
$(0, 1)$	$(1, 1)$	...	$(\texttt{ancho}, 1)$
$(0, 0)$	$(1, 0)$	...	$(\texttt{ancho}, 0)$



En todo momento se puede consultar cuál es el valor almacenado en la coordenada donde se encuentra actualmente el cursor. El valor se obtiene por referencia, de manera que el usuario puede utilizarlo como **lvalue** para modificar el contenido del mapa.

Por ejemplo, el siguiente código:

```
Mapa<int> mapa(3, 2, 0);
mapa.valor() = 9;
mapa.mover(ESTE);
mapa.valor()++;
mapa.mover(NORTE);
mapa.valor() = 2;
```

Crea un mapa de números enteros, de ancho 3 y alto 2:

0	2	0
9	1	0

Ejercicio 1

Modificar el archivo `Mapa.hpp`, eligiendo una representación privada para la clase `Mapa<T>`. Definir el constructor `Mapa(int ancho, int alto, const T& def)` que inicializa un mapa del tamaño indicado usando `def` como valor inicial para todas las casillas. El cursor empieza ubicado sobre la coordenada $(0, 0)$.

Nota: en este ejercicio, **no está permitido** usar la clase `std::vector` ni ninguna otra clase de la STL de C++. El objetivo es representar la matriz manualmente por medio punteros, reservando memoria dinámicamente.

Sugerencia: incluir un campo `T** grilla`. Recordar que un puntero `T* p` se puede entender como un arreglo cuyos elementos son `p[0]`, ..., `p[n]` de tipo `T`. De este modo, el campo `T** grilla` se puede entender como un arreglo de arreglos, de tal modo que `grilla[i][j]` es el contenido de la grilla en la coordenada `(i, j)`.

Ejercicio 2

Definir los métodos:

- `int ancho() const` — Devuelve el ancho del mapa.
- `int alto() const` — Devuelve el alto del mapa.
- `T& valor()` — Devuelve una referencia al valor ubicado en la coordenada donde se encuentra el cursor.
- `void mover(Direccion direccion)` — Mueve el cursor hacia la dirección indicada. Notar que el resultado es el de aplicar el siguiente desplazamiento a la posición actual del cursor:

  $$\begin{aligned} \texttt{NORTE} &\mapsto (0, +1) \\ \texttt{ESTE} &\mapsto (+1, 0) \\ \texttt{SUR} &\mapsto (0, -1) \\ \texttt{OESTE} &\mapsto (-1, 0) \end{aligned}$$

  Si la coordenada queda fuera de los límites del mapa, se debe ajustar utilizando aritmética modular. Por ejemplo, si el mapa es de tamaño 100×100, la coordenada que se encuentra al este de (99, 3) es (0, 3).

Ejercicio 3

Ejecutar el programa con la herramienta `valgrind`. Dado que el constructor reserva memoria dinámicamente pero aún no se implementó el destructor, la herramienta debería reportar que el programa pierde memoria.

Implementar el destructor `Mapa<T>::~Mapa()`. Debe liberar toda la memoria que se haya reservado. Verificar con la herramienta `valgrind` que el programa no tenga problemas en el manejo de memoria dinámica.

Ejercicio 4

Implementar el constructor por copia `Mapa<T>::Mapa(const Mapa& otro)`. Verificar con la herramienta `valgrind` que el programa no tenga problemas en el manejo de memoria dinámica.

Ejercicio 5

Implementar el operador de asignación `Mapa& Mapa<T>::operator=(const Mapa& otro)`. Verificar con la herramienta `valgrind` que el programa no tenga problemas en el manejo de memoria dinámica.