# CS2023 - Aula de Ejercicios Nº 4

**Angel Napa**
**ACL: Joaquín Jordán**
Semestre 2024-1

Se sugiere que cada estudiante trate de resolver los ejercicios de forma **individual** y luego los discuta en grupo.

**IMPORTANTE:**

- Enviar en canvas un **único archivo .cpp** (colocar las 3 soluciones en el mismo archivo .cpp).
- **No se revisaran archivos .zip** subidos a canvas.
- **Colocar nombres y apellidos** de los integrantes **al inicio** del archivo .cpp.
- La evaluación continua es grupal (mínimo 2 alumnos y máximo 3 alumnos).

## Ejercicios

1. (6 pts) Implemente una pila de último en entrar, primero en salir (LIFO) utilizando solo dos colas. La pila implementada debe admitir todas las funciones de una pila normal (empujar, colocar arriba, abrir y vaciar).

   Implemente la clase MyStack:

   - void push(int x) Empuja el elemento x a la parte superior de la pila.
   - int pop() Elimina el elemento en la parte superior de la pila y lo devuelve.
   - int top() Devuelve el elemento en la parte superior de la pila. boolean vacío() Devuelve verdadero si la pila está vacía, falso en caso contrario.

```
class MyStack {
    public:
        MyStack() {

        }

        void push(int x) {

        }

        int pop() {

        }

        int top() {

        }

        bool empty() {

        }
    };

    /**
     * Your MyStack object will be instantiated and called as such:
     * MyStack* obj = new MyStack();
```

```
 * obj->push(x);
 * int param_2 = obj->pop();
 * int param_3 = obj->top();
 * bool param_4 = obj->empty();
 */
```

Notas:

Debe utilizar solo operaciones estándar de una cola, lo que significa que solo son válidas las operaciones de empujar hacia atrás, mirar/sacar desde el frente, tamaño y está vacío. Dependiendo de su idioma, es posible que la cola no sea compatible de forma nativa. Puede simular una cola usando una lista o deque (cola de dos extremos) siempre que use solo las operaciones estándar de una cola.

- **Ejemplo 1:**
  **Input:**

```
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
```

  **Output:** `[null, null, null, 2, 2, false]`
  **Explicación:**

```
MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
myStack.top(); // return 2
myStack.pop(); // return 2
myStack.empty(); // return False
```

2. (7 pts) Diseñe su implementación de la cola circular de doble extremo (deque).

   Implemente la clase MyCircularDeque:

   - MyCircularDeque(int k) Inicializa la deque con un tamaño máximo de k.
   - boolean insertFront() Agrega un elemento al frente de Deque. Devuelve verdadero si la operación es exitosa o falso en caso contrario.
   - boolean insertLast() Agrega un elemento en la parte posterior de Deque. Devuelve verdadero si la operación es exitosa o falso en caso contrario.
   - boolean deleteFront() Elimina un elemento del frente de Deque. Devuelve verdadero si la operación es exitosa o falso en caso contrario.
   - boolean deleteLast() Elimina un elemento de la parte posterior de Deque. Devuelve verdadero si la operación es exitosa o falso en caso contrario.
   - int getFront() Devuelve el elemento frontal del Deque. Devuelve -1 si el deque está vacío.
   - int getRear() Devuelve el último elemento de Deque. Devuelve -1 si el deque está vacío.
   - boolean isEmpty() Devuelve verdadero si el deque está vacío, o falso en caso contrario.
   - boolean isFull() Devuelve verdadero si el deque está lleno, o falso en caso contrario.

```
class MyCircularDeque {
    public:
        MyCircularDeque(int k) {

        }
```

```
    bool insertFront(int value) {

    }

    bool insertLast(int value) {

    }

    bool deleteFront() {

    }

    bool deleteLast() {

    }

    int getFront() {

    }

    int getRear() {

    }

    bool isEmpty() {

    }

    bool isFull() {

    }
};

/**
 * Your MyCircularDeque object will be instantiated and called as such:
 * MyCircularDeque* obj = new MyCircularDeque(k);
 * bool param_1 = obj->insertFront(value);
 * bool param_2 = obj->insertLast(value);
 * bool param_3 = obj->deleteFront();
 * bool param_4 = obj->deleteLast();
 * int param_5 = obj->getFront();
 * int param_6 = obj->getRear();
 * bool param_7 = obj->isEmpty();
 * bool param_8 = obj->isFull();
 */
```

- **Ejemplo 1:**
  **Input:**

```
["MyCircularDeque", "insertLast", "insertLast",
"insertFront", "insertFront",
"getRear", "isFull", "deleteLast", "insertFront", "getFront"]
[[3], [1], [2], [3], [4], [], [], [], [4], []]
```

**Output:** `[null, true, true, true, false, 2, true, true, true, 4]`

**Explicación:**

```
MyCircularDeque myCircularDeque = new MyCircularDeque(3);
myCircularDeque.insertLast(1);  // return True
myCircularDeque.insertLast(2);  // return True
myCircularDeque.insertFront(3); // return True
myCircularDeque.insertFront(4); // return False, the queue is full.
myCircularDeque.getRear();      // return 2
myCircularDeque.isFull();       // return True
myCircularDeque.deleteLast();   // return True
myCircularDeque.insertFront(4); // return True
myCircularDeque.getFront();     // return 4
```

3. (7 pts) Diseñe una pila que admita push, pop, top y recuperación del elemento mínimo en tiempo constante.

   Implemente la clase MinStack:

   - MinStack() inicializa el objeto de la pila.
   - void push(int val) empuja el elemento val a la pila.
   - void pop() elimina el elemento en la parte superior de la pila.
   - int top() obtiene el elemento superior de la pila.
   - int getMin() recupera el elemento mínimo en la pila.

   Debe implementar una solución con complejidad temporal O(1) para cada función.

```
class MinStack {
    public:
        MinStack() {

        }

        void push(int val) {

        }

        void pop() {

        }

        int top() {

        }

        int getMin() {

        }
    };

    /**
     * Your MinStack object will be instantiated and called as such:
     * MinStack* obj = new MinStack();
     * obj->push(val);
     * obj->pop();
```

```
 * int param_3 = obj->top();
 * int param_4 = obj->getMin();
 */
```

- **Ejemplo 1:**
  textbfInput:

  ```
  ["MinStack","push","push","push","getMin","pop","top","getMin"]
  [[],[-2],[0],[-3],[],[],[],[]]
  ```

  **Output:** `[null,null,null,null,-3,null,0,-2]`
  **Explicación:**

  ```
  MinStack minStack = new MinStack();
  minStack.push(-2);
  minStack.push(0);
  minStack.push(-3);
  minStack.getMin(); // return -3
  minStack.pop();
  minStack.top();    // return 0
  minStack.getMin(); // return -2
  ```