

# Trabajo Práctico # de C++

Estructuras de Datos, Universidad Nacional de Quilmes

7 de mayo de 2016

## 1. Introducción

### Ejercicio 1

Definir el tipo abstracto de datos Pokemon como un número entero que representa la vida y un nombre. Su interfaz es la siguiente:

- `Pokemon crearPokemon(string nombre)`
- `void restarVida(Pokemon& p)`
- `void cambiarNombre(Pokemon& p, string nombre)`
- `bool estaVivo(Pokemon p)`
- `string getNombre(Pokemon p)`
- `int getVida(Pokemon p)`
- `void luchar(Pokemon& p, Pokemon& r)`
- `void lucharN(int n, Pokemon& p, Pokemon& r)`
- `void destruir(Pokemon& p)`

Implementar esta interfaz destructiva, utilizando memoria de Heap.

### Ejercicio 2

Definir un tipo abstracto para representar fracciones, con la siguiente interfaz:

- `Fraccion fraccion(int x, int y)`
- `Fraccion sumar(Fraccion f1, Fraccion f2)`
- `Fraccion restar(Fraccion f1, Fraccion f2)`
- `Fraccion multiplicar(Fraccion f1, Fraccion f2)`
- `Fraccion dividir(Fraccion f1, Fraccion f2)`
- `void invertir(Fraccion& f1)`
- `void sumplificar(Fraccion& f1)`
- `void destruir(Fraccion& f1)`

### Ejercicio 3

Definir el tipo abstracto Maybe utilizando una interfaz destructiva (operaciones nothing, just, fromJust e isNothing).

### Ejercicio 4

Definir el tipo abstracto Stack con la siguiente interfaz destructiva:

- `Stack emptyS()`
- `bool isEmptyS(Stack s)`
- `void push(int x, Stack& s)`
- `int top(Stack s)`
- `void pop(Stack& s)`
- `int size(Stack s)`
- `void destroyS(Stack& s)`

Todas las operaciones deben costar  $O(1)$ , a excepción de `destroyS`. Luego como usuario de esta estructura definir las funciones del 2.1 de la Práctica 4.

### Ejercicio 5

Definir el tipo abstracto Queue con la siguiente interfaz destructiva :

- `Queue emptyQ()`
- `bool isEmptyQ(Queue q)`
- `void queue(Queue& q, int x)`
- `int firstQ(Queue q)`
- `void dequeue(Queue& q)`
- `void destroyQ(Queue& q)`

Todas las operaciones deben costar  $O(1)$ , a excepción de `destroyQ`. ¿Esto era posible en Haskell? Luego como usuario de esta estructura definir las funciones del 1.1 de la Práctica 4.

## 2. Listas

### Ejercicio 6

Implementar como usuario de `LinkedList` funcionales las siguientes funciones:

- `int sumar(List xs)`
- `int length(List xs)`
- `List mapSucc(List xs)`
- `List take(int n, List xs)`
- `List drop(int n, List xs)`

**Ejercicio 7**

Implementar como usuario de `LinkedList` destructivas las siguientes funciones:

- `int sumar(List xs)`
- `int length(List xs)`
- `void mapSucc(List& xs)`
- `void take(int n, List& xs)`
- `void drop(int n, List& xs)`

**Ejercicio 8**

Definir como usuario de listas iterables las primeras 12 funciones del punto 2.1. de la Práctica 1.

### 3. Árboles

**Ejercicio 9**

Definir una representación de árbol binario en C++ e implementar los primeros 7 items del ejercicio 1 de la Práctica 3, utilizando recursión.

**Ejercicio 10**

Definir las mismas funciones del apartado anterior pero de forma iterativa (sin recursión).

**Ejercicio 11**

Implementar el ejercicio de expresiones aritméticas en C++, utilizando una interfaz funcional como en Haskell.

### 4. Arrays

**Ejercicio 12**

Implementar las primeras 15 funciones del punto 2.1 de la Práctica 1 pero utilizando arrays en lugar de listas.

**Ejercicio 13**

Definir el tipo abstracto `ArrayList`, una lista que posee la misma interfaz que las listas recorribles pero cuya representación es un array.

**Ejercicio 14**

Definir el tipo abstracto `Set` utilizando un array de booleanos. Luego como usuario de esta estructura definir las funciones del 3.1 de la Práctica 4.

**Ejercicio 15**

Definir el tipo abstracto `Multiset` utilizando una técnica muy similar a la del ejercicio anterior.

**Ejercicio 16**

Implementar una versión eficiente del ejercicio de `MiniTablero` de la práctica 5, pero esta vez implementando un tablero de 2 dimensiones.

## 5. Heaps

### Ejercicio 17

Definir el tipo abstracto BinaryHeap visto en la teórica.

## 6. Hashing

### Ejercicio 18

Definir el tipo abstracto HashMap, que es un Map cuya implementación utiliza una tabla de Hash.

### Ejercicio 19

Definir el tipo abstracto HashSet, que es un Set cuya implementación utiliza una tabla de Hash.

## 7. Sorting

### Ejercicio 20

Implementar los distintos sortings vistos en la teórica, incluyendo heapsort.