**Algoritmos y Estructuras de Datos I**
Primer Cuatrimestre 2020

Guía Práctica
**EJERCICIOS DE TALLER**

*Departamento de Computación*
*Facultad de Ciencias Exactas y Naturales*
*Universidad de Buenos Aires*

# 1. Testing

**Ejercicio 1.** `int puntaje(int n)`
Sofía está jugando un juego. Tiene una bolsa con bolitas y mete la mano para sacar un puñado. Gana puntos según la cantidad de bolitas con las siguientes reglas:

- Si la cantidad de bolitas es menor que 10, gana dos puntos por cada bolita que sacó. Si no, un punto por cada una.
- Además, si la cantidad de bolitas que sacó es múltiplo de 3, gana 10 puntos. Si no, pierde 10 puntos.

Dado el programa que calcula la cantidad de puntos qué ganó Sofía. Armar casos de test estructurales para el programa, los test deben cubrir todas las decisiones (branches) del código.

**Ejercicio 2.** Armar casos de test estructurales para el programa rotar, implementado en el taller de Vectores,
`vector<int> rotar(vector<int> v, int k)`
Dada una secuencia v y un entero k, rotar k posiciones los elementos de v. [1, 2, 3, 4, 5, 6] rotado 2, deberia dar [3, 4, 5, 6, 1, 2].
Los test deben cubrir todas las decisiones (branches) del código tanto en funciones principales como en auxiliares que hayan definido.

**Ejercicio 3.** `bool sandia(int porciones)`
Sara y Flor consiguieron un cantidad de porciones de sandía para el postre. Quisieran dividir las porciones en dos partes tales que cada una coma una cantidad par de porciones (no necesariamente la misma cantidad cada una). El objetivo es decidir si es posible realizar la división dada la cantidad de porciones.

1. Pensar y escribir los tests que crean necesarios antes de programar la solución. ¿Cuáles son las particiones de dominio?
2. Programar en C++ una solución al problema que pase los tests antes creados.
3. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas líneas de código.
4. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas las ramas de decisiones (branches) del programa.

**Ejercicio 4.** *subSecuencia que suma*
`bool subSecunciaQueSuma(vector<int> s, int n)`, que dada una secuencia de enteros y un número devuelve verdadero si existe una subSecuencia de números consecuntivos que suman exactamente $n$

1. Pensar y escribir los tests que crean necesarios antes de programar la solución. ¿Cuáles son las particiones de dominio?
2. Programar en C++ una solución al problema que pase los tests antes creados.
3. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas líneas de código.

**Ejercicio 5.** *agrupar Anagramas*
`vector<vector<string> > agruparAnagramas(vector<string> v);`
Dado un vector de strings devolver un `vector<vector<string> >` que cumpla:

- Todos los strings del vector de entrada deben estar en exactamente un subvector resultado (y el resultado no puede tener strings que no estén en la entrada).
- Cada elemento (vector) del resultado debe contener solo anagramas.
- Se debe mantener el orden relativo de cada palabra dentro de cada vector y el orden de cada vector responde a la primera aparición de un nuevo anagrama.

Por ejemplo, `agruparAnagramas({"ab", "cd", "ef", "ba", "ab", "dc"})` debe devolver `{{"ab", "ba", "ab"}, {"cd", "dc"}, {"ef"}}`

1. Pensar y escribir los tests que crean necesarios antes de programar la solución. ¿Cuáles son las particiones de dominio?
2. Programar en C++ una solución al problema que pase los tests antes creados.
3. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas líneas de código.