



## 1. Introducción a C++

**Ejercicio 1.** Crear un archivo: “labo00.cpp” (con cualquier editor de texto) y escribir lo siguiente:

```
#include <iostream>

int f(int x){
    return x+1;
}

int main() {
    std::cout << "El resultado es: " << f(10) << std::endl;
    return 0;
}
```

Luego, compilar y ejecutar el código en la terminal:

```
g++ labo00.cpp -o labo00_ejecutable
./labo00_ejecutable
```

**Ejercicio 2.** Modificar el programa anterior para que  $f$  tome dos parámetros de tipo `int` y los sume.

**Ejercicio 3.** Modificar el programa anterior para que  $f$  tome dos parámetros  $x$  e  $y$  de tipo `int` y los sume sólo si  $x > y$ , en caso contrario el resultado será el producto.

**Ejercicio 4.** Crear un proyecto nuevo de C++ en **CLion** con el nombre labo00. Escribir el programa del ejercicio anterior y ejecutarlo.

**Ejercicio 5.** Escribir la función que dado  $n \in \mathbb{N}$  devuelve si es primo. Recuerden que un número es primo si los únicos divisores que tiene son 1 y el mismo.

## Iteración vs Recursión

Los siguientes ejercicios deben ser implementados primero en su versión **recursiva**, luego iterativa utilizando **while** y por último iterativa utilizando **for**.

**Ejercicio 6.** Escribir la función de Fibonacci que dado un entero  $n$  devuelve el  $n$ -ésimo número de Fibonacci. Los números de Fibonacci empiezan con  $F_0 = 0$  y  $F_1 = 1$ .  $F_n = F_{n-1} + F_{n-2}$

**Ejercicio 7.** Escribir la función que dado  $n \in \mathbb{N}$  devuelve la suma de todos los números impares menores que  $n$ .

**Ejercicio 8.** Escribir la función `sumaDivisores` que dado  $n \in \mathbb{N}$ , devuelve la suma de todos sus divisores entre  $[1, n]$ .

■ **Hint:** Recordar que para la versión recursiva es necesario implementar `divisoresHasta`

**Ejercicio 9.** Escribir una función que dados  $n, k \in \mathbb{N}$  compute el combinatorio:  $\binom{n}{k}$ . Hacerlo usando la igualdad  $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$   
¿Qué pasa si tuvieran que escribir la versión iterativa?

**Ejercicio 10.** ¿Es mejor programar utilizando algoritmos recursivos ó iterativos? ¿Es mejor usar **while** o **for**?

## 2. Entrada/Salida + Pasaje de parámetros

**Ejercicio 11.** Escribir un programa en el que se ingrese un número por teclado (entrada estándar), calcule si es primo y muestre por pantalla (salida estándar) “El número ingresado es primo” si es primo. En caso contrario: “El número ingresado no es primo”

**Ejercicio 12.** Escribir una función `escribirArchivo` que escriba en un archivo `salida.txt` dos enteros `a` y `b` y luego dos reales `f` y `g` separados con coma en una única línea.

**Ejercicio 13.** Leer del archivo `entrada.txt` un valor entero y almacenarlo en una variable llamada `a` y luego leer un valor real y almacenarlo en una variable `f`. Mostrar los valores leídos en la salida estándar. Ambos valores están separados por un espacio y hay una única línea en el archivo (por ejemplo: “-234 1.7”)

**Ejercicio 14.** `numeros.txt` contiene una lista de números separados por espacios. Leerlos e imprimirlos por pantalla.

**Ejercicio 15.** ¿Cuál es el valor de `a` luego de la invocación `prueba(a,a)`?

```
int a = 10;
void prueba(int& x, int& y) {
    x = x + y;
    y = x - y;
    x = 1/y;
}
prueba(a, a);
```

En los siguientes ejercicios, ingresar los valores por entrada estándar, mostrar en la salida estándar los valores ingresados y los resultados de las funciones.

**Ejercicio 16.** Implementar la función `swap`: `void swap(int& a, int& b)`, que cumpla con la siguiente especificación:

```
proc swap (inout a:Z, inout b:Z) {
    Pre {a = a0 ∧ b = b0}
    Post {a = b0 ∧ b = a0}
}
```

**Ejercicio 17.** Implementar la función `division` que cumpla con la siguiente especificación:

```
proc division (in dividendo Z, in divisor Z, out cociente:Z, out resto:Z) {
    Pre {dividendo ≥ 0 ∧ divisor > 0}
    Post {dividendo = divisor * cociente + resto ∧ 0 ≤ resto < divisor}
}
```

Resolver este ejercicio en versiones iterativa y recursiva.

**Ejercicio 18.** `void collatz(int n, int& cantPasos)`

La conjetura de *Collatz* dice que dado un número natural  $n$  y el proceso que describimos a continuación, sin importar cuál sea el número original, provocará que la serie siempre termine en 1. El proceso:

- Si  $n$  es par lo dividimos por 2
- Si  $n$  es impar lo multiplicamos por 3 y le sumamos 1 al resultado

En este ejercicio, supondremos que la conjetura es cierta y se pide implementar una función que devuelva la cantidad de pasos que se realizan desde el número original hasta llegar a 1 y que muestre en la salida estándar la sucesión de números por la que pasa. Ejemplo: si calculamos `collatz` de 11, la cantidad de pasos es 15 y la sucesión es 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Resolver este ejercicio en versiones iterativa y recursiva.

**Ejercicio 19.** Dados dos archivos que contienen números separados por espacios (ambos archivos tienen la misma cantidad de números), se pide que se sumen los valores de los archivos y se genere uno nuevo con la suma de los mismos. Ejemplo: “`numeros.txt`” contiene 1 25 6 y “`numeros1.txt`” contiene 45 5 4 debe crear el archivo “`suma.txt`” que contenga 46 30 10.

**Ejercicio 20.** `void primosGemelos(int n, int& res1, int& res2)` Decimos que  $a$  y  $b$  son primos gemelos, si ambos son primos y además  $a=b-2$ . Queremos obtener los  $i$ -ésimos primos gemelos. Por ejemplo, son primos gemelos 3 y 5, 5 y 7, 11 y 13, 17 y 19, 29 y 31, 41 y 43 ... , los 4-ésimos primos gemelos son 17 y 19. Además se debe escribir en un archivo la secuencia de primos gemelos hasta llegar al  $i$ -ésimo. Para el ejemplo el archivo debe contener: (3,5) (5,7) (11,13) (17,19)

### 3. Vectores

Importar el proyecto de la clase. En CLion seleccionar File → Import Project y seleccionar la carpeta con el nombre Labo02-Vectores que se encuentra en el archivo Laboratorio 02.zip

**Ejercicio 21.** Especificar y luego implementar en su versión **recursiva** e **iterativa**:

1. `bool divide(vector<int> v, int n)`  
Dados un vector  $v$  y un entero  $n$ , decide si  $n$  divide a todos los números de  $v$ .
2. `int maximo(vector<int> v)`  
Dado un vector, devuelve el valor máximo.
3. `bool pertenece(int elem, vector<int> v)`  
Dado un entero, indica si pertenece o no al vector.

**Ejercicio 22.** Implementar en su versión **iterativa**:

1. `void mostrarVector(vector<int> v)`  
Dado un vector de enteros muestra por la salida estándar (cout), el vector  
Ejemplo: si el vector es  $\langle 1, 2, 5, 65 \rangle$  se debe mostrar en pantalla  $[1, 2, 5, 65]$
2. `vector<int> limpiarDuplicados(vector<int> v)`  
Dado un vector de enteros, devuelve un vector de enteros con los elementos del vector sin duplicados. Ejemplo  $v = \langle 1, 1, 2, 1, 1, 2, 3, 2, 3, 3 \rangle$  el resultado es  $\langle 1, 2, 3 \rangle$
3. `vector<int> rotar(vector<int> v, int k)`  
Dado un vector  $v$  y un entero  $k$ , rotar  $k$  posiciones los elementos de  $v$ .  
 $[1, 2, 3, 4, 5, 6]$  rotado 2, debería dar  $[3, 4, 5, 6, 1, 2]$ .
4. `vector<int> reverso(vector<int> v)`  
Dado un vector  $v$ , devuelve el reverso. Implementar también la versión recursiva de este problema.
5. `vector<int> factoresPrimos(int n)`  
Dado un entero devuelve un vector con los factores primos del mismo. Los factores primos de un número entero son los números primos divisores exactos de ese número entero. Ejemplos: los factores primos de 6 son 2 y 3. Factores primos de 7 es 7
6. `bool estaOrdenado(vector<int> v)`  
Dado un vector  $v$  de int, dice si es monótonamente creciente o monótonamente decreciente.
7. `void negadorDeBooleanos(vector<bool>& booleanos)`  
Modifica un vector de booleanos negando todos sus elementos.
8. `void sinImpares(vector<int>& v)`  
Dado un vector de enteros, devuelve el mismo vector colocando un 0 en las posiciones en las que haya un número impar.
9. `vector<pair<int, int> > cantidadCaracteres(vector<int> v)`  
Dado un vector de enteros, devuelve una tupla que contine por cada entero la cantidad de apariciones del mismo.  
Ejemplo  $v = \langle 1, 1, 2, 1, 1, 2, 3, 2, 3, 3 \rangle$  el resultado es  $\langle (1, 4), (2, 3), (3, 3) \rangle$   
Tip: para aprender a usar las tuplas entrar a <http://www.cplusplus.com/> y buscar (en donde dice search) pair.

**Ejercicio 23.** *Integradores.* Implementar las siguientes funciones

1. `void palindromos(string rutaArchivoIn, string rutaArchivoOut)`  
Este procedimiento debe leer un archivo que contiene una lista de strings y crear uno nuevo dejando sólo los palíndromos. Además, debe transformar las palabras a mayúscula. **Ayuda:** Buscar la función `toupper` definida en `cctype`. Utilizar como ejemplo el archivo `palindromos.txt`.
2. `void promedios(string rutaArchivoIn1, string rutaArchivoIn2, string rutaArchivoOut)`  
Dados dos archivos en los que cada uno contiene una secuencia de enteros (ambas con la misma longitud), guardar el promedio de cada par de números que se encuentran en la misma posición en el archivo de salida. Por ejemplo: si tenemos dos secuencias "1 2 3 4" y "1 25 3 12" el resultado debe ser "1 13.5 3 8"

3. `void promediosHasta(string rutaArchivoIn, string rutaArchivoOut)`

Dado un archivo `rutaArchivoIn`, que contiene una lista de números separados por espacios, calcular una lista de promedios de la misma longitud donde cada elemento corresponde al promedio de todos los anteriores de la lista original (incluyendo dicho elemento). La lista de promedios debe ser guardada en el archivo indicado por `rutaArchivoOut`. Por ejemplo, dada la secuencia “1 2 3 2 1”, el resultado debe ser “1 1.5 2 2 1.8”.

4. `void cantidadApariciones(string rutaArchivoIn, string rutaArchivoOut)`

Dado un archivo `rutaArchivoIn`, que contiene una lista de números separados por espacios, contar la cantidad de apariciones de cada uno y escribe `rutaArchivoOut` con una línea por cada número encontrado, un espacio y la cantidad de apariciones. Por ejemplo: si el “1” aparece 44 veces y el “2” 20 veces, la salida debería contener dos líneas: “1 44” y “2 20”.

5. `int cantidadAparicionesDePalabra(string rutaArchivo, string palabra)`

Dada una palabra y un archivo de texto devuelve la cantidad de apariciones de la palabra en el archivo.

6. `void estadisticas(string rutaArchivo)`

Dado un archivo de texto, mostrar por pantalla las estadísticas de cantidad de palabras con longitud 1, 2, 3... hasta el máximo. Por ejemplo:

```
Palabras de longitud 1: 100
Palabras de longitud 2: 12
Palabras de longitud 3: 6
...
```

7. `void interseccion()`

Procedimiento que pide al usuario que se ingresen dos nombres de archivos que contengan sólo números enteros separados por espacios, luego calcula la intersección (números que se encuentran en ambos archivos) e imprime por pantalla el resultado.

## 4. Herramientas: Git + L<sup>A</sup>T<sub>E</sub>X

### Ejercicio 24.

Crear un repositorio en `git.exactas.uba.ar` con el nombre *repo-taller-latex* y darle permisos a todos los miembros del grupo.

### Ejercicio 25.

Clonar el repositorio para conseguir una copia local: `git clone URL` (buscar la URL en la página del repositorio)

### Ejercicio 26.

Crear un documento llamado *ejemplo1.tex* que contenga el texto que está a continuación en lenguaje L<sup>A</sup>T<sub>E</sub>X utilizando cualquier editor (ya sea local u online):

*El factorial de un entero positivo  $n$  se define como:*

$$5! = \prod_{i=1}^5 i! = 1 \times 2 \times 3 \times 4 \times 5 = 120$$

Guardar dicho documento en la copia local del repositorio.

### Ejercicio 27.

Crear un nuevo documento *ejemplo2.tex* con título **Especificación** con el contenido siguiente utilizando las macros de algo I. Las macros deben incluirlas después de `documentclass` y antes de `\begin{document}` de la siguiente manera:

`\input{Algo1Macros.tex}` (si las macros se encuentran en una carpeta distinta, tendrán que incluir la ruta completa).

Para ver cómo utilizar los comandos de las macros, revisen el archivo `latexsheet.pdf`

```
proc factorial (in n:  $\mathbb{Z}$ , out result:  $\mathbb{Z}$ ) {  
    Pre { $n \geq 0$ }  
    Post {( $n = 0 \rightarrow result = 1$ )  $\wedge$  ( $n > 0 \rightarrow result = \prod_{k=1}^n k$ )}  
}
```

Guardar dicho documento en la copia local del repositorio.

### Ejercicio 28.

1. Ejecutar el comando `git status` en la carpeta donde se encuentra el repositorio e interpretar el mensaje.
2. Crear un commit que contenga los archivos *ejemplo1.tex* y *ejemplo2.tex* con un mensaje que describa los archivos que se están agregando. Para agregar archivos al commit, deberán utilizar `git add ejemplo1.tex` y `git add ejemplo2.tex`. Luego, para ejecutar el commit con un mensaje, pueden utilizar el comando `git commit -m "mensaje que quieran"`
3. Ejecutar el comando `git status` en la carpeta donde se encuentra el repositorio e interpretar el mensaje.
4. Cambiar el título de los archivos por *Ejemplo1: Factorial* y *Ejemplo2: MacrosAlgo1*
5. Ejecutar el comando `git status` en la carpeta donde se encuentra el repositorio e interpretar el mensaje.
6. Crear un commit que contenga los cambios en el archivo *ejemplo2.tex* y **no contenga** los cambios en el archivo *ejemplo1.tex* con un mensaje descriptivo de los cambios realizados.
7. Ejecutar el comando `git status` en la carpeta donde se encuentra el repositorio e interpretar el mensaje.
8. Pushear los cambios al repositorio.
9. Ejecutar el comando `git status` en la carpeta donde se encuentra el repositorio e interpretar el mensaje.
10. Revisar que los archivos en la página del repositorio fueron actualizados correctamente.

### Ejercicio 29.

Agregar en el archivo *ejemplo1.tex* la siguiente fórmula:

$$\hat{R}(\hat{f}, \bar{D}_n^m) = \frac{1}{|\bar{D}_n^m|} \sum_{i: (X_i, Y_i) \in \bar{D}_n^m} (Y_i - \hat{f}_m, \hat{D}_n^m)$$

.

1. Crear un commit que describa incluya los cambios realizados en el archivo *ejemplo1.tex* (recordar que además de la fórmula el título cambió).
2. Pushear los cambios.

### Ejercicio 30.

1. Realizar una nueva copia del repositorio (en una carpeta separada u otro compañero del grupo).
2. En las dos copias del repositorio, modificar el archivo *ejemplo1.tex* agregando nuevas secciones (sección *cambio1* y *cambio2*).
3. En cada copia, crear un commit que incluya los cambios en el archivo con un mensaje asociado.
4. Pushear los cambios en una de las copias locales.
5. Intentar pushear desde la otra copia. En este punto, deberán actualizar su copia con la versión más reciente del repositorio. Para ello, utilizar *git pull*. Al hacer pull, deberían ver que el archivo contiene las dos secciones y git requerirá un mensaje para el commit que genere la mezcla (merge).
6. Repetir desde el segundo punto de este ejercicio pero esta vez, en el momento de hacer pull, agregar la opción *-rebase*.
7. Pushear los cambios
8. Revisar el grafo de commits en la página del repositorio.

## 5. Debugging

**Ejercicio 31.** Dados los códigos de los siguientes programas, utilizar los test para verificar si cumplen con su objetivo. Si los test fallan, utilizar el debugger de CLion para encontrar los posibles errores y arreglar el código.

1. `bool estaOrdenado(vector<int> v)`  
Dado un vector de enteros  $v$ , indica si está ordenado tanto ascendente como descendentemente.
2. `bool esPrimo(int numero)`  
Dado un entero, decide si el mismo es un número primo o no.
3. `bool pertenece(int elem, vector<int> v)`  
Dado un entero  $elem$ , indica si pertenece o no al vector  $v$ .
4. `float desvioEstandar(vector<float> v)`  
Dado un vector  $v$ , calcula su desvío estandar.
5. `long fibonacci(int k)`  
Este procedimiento debe calcular el  $K$ -esimo número de la serie de fibonacci.
6. `int maximoComunDivisor(int x, int y)`  
Dados dos enteros  $X$  e  $Y$ , calcula el máximo común divisor de los mismos.
7. `int sumaDoble(vector<int> v)`  
Dado un vector de enteros  $v$ , debe calcular la sumatoria del doble de los elementos que son positivos y pares.
8. `int cantPalabras(string filename)`  
Dado un archivo de texto, debe contar la cantidad de palabras que hay en el mismo.

## 6. Ciclos a partir de invariantes

A continuación se presentan una serie de ejercicios. Cada uno contiene un invariante asociado. Resolver cada problema respetando, para el ciclo principal del programa, el invariante dado.

### Ejercicio 32. *Existe Pico*

Una secuencia tiene picos si en alguna posición el elemento es mayor tanto del anterior como del siguiente. Decidir si una secuencia dada (de al menos tres elementos) tiene picos.

$$I \equiv 1 \leq i < |s| \wedge_L res = (\exists k : \mathbb{Z}) \ 1 \leq k < i \wedge_L s[k] > s[k-1] \wedge s[k] > s[k+1]$$

1. Agregar asserts en las posiciones del código en las que debe cumplirse el invariante para verificar que  $(1 \leq i < |s|)$ .
2. Demostrar que el ciclo preserva el invariante:  $\{I \wedge B\}S\{I\}$

### Ejercicio 33. *Mínimo de una subsecuencia*

Devolver el índice del mínimo valor de una subsecuencia.

```
proc indice_min_subsec (in s:seq<Z>, in i,j:Z, out res:Z) {
  Pre { |s| > 0 ∧ 0 ≤ i, j < |s| ∧ i ≤ j }
  Post { i ≤ res ≤ j ∧ (∀k : Z) i ≤ k ≤ j →L s[k] ≥ s[res] }
}
```

$$I \equiv i - 1 \leq l \leq j \wedge i \leq res \leq j \wedge (\forall k : \mathbb{Z}) \ l < k \leq j \rightarrow_L s[k] \geq s[res]$$

### Ejercicio 34. *Sumatoria de los elementos de una secuencia*

Calcular la suma de todos los elementos de una secuencia  $s$ .

$$Pc \equiv |s| \bmod 2 = 1$$

$$I \equiv 1 \leq i \leq (|s| \div 2) + 1 \wedge_L suma = s[(|s| \div 2)] + \sum_{k=1}^{i-1} s[(|s| \div 2) - k] + s[(|s| \div 2) + k]$$

1. ¿Qué habría que cambiar en el invariante si la precondition fuese  $Pc : |s| > 0$ .

### Ejercicio 35. *Máximo Común Divisor*

Encontrar el máximo común divisor entre dos enteros positivos  $m$  y  $n$ . La post condición del ciclo es

$$Q_c \equiv a = b = \text{mcd}(n, m).^1$$

$$I \equiv 0 \leq a \leq m \wedge 0 \leq b \leq n \wedge \text{mcd}(a, b) = \text{mcd}(m, n)$$

### Ejercicio 36. *División*

Dados dos enteros positivos  $n$  y  $d$  calcular el cociente  $q$  y el resto  $r$ . El resultado debe ser de tipo *pair*  $\langle int, int \rangle$ , donde el primer elemento de la tupla es  $q$  y el segundo es  $r$  y cumple  $Q_c \equiv 0 \leq r < d \wedge 0 \leq q \leq n \wedge n = q \times d + r$ .

$$I \equiv (n \bmod d) \leq r \leq n \wedge 0 \leq q \leq n \wedge n = q \times d + r$$

### Ejercicio 37. *Ordenar*

Ordenar ascendentemente una secuencia (no vacía) de enteros.

$$I \equiv 0 \leq i \leq |s| \wedge_L |s| = |s_0| \wedge \text{mismos}(s, s_0) \wedge_L \text{ordenada}(\text{subseq}(s, 0, i)) \wedge$$

$$(\forall k : \mathbb{Z}) \ 0 \leq k < |s| \wedge i > 0 \rightarrow_L ((k < i \wedge s[k] \leq s[i-1]) \vee (k \geq i \wedge s[k] \geq s[i-1]))$$

**fun** mismos( $s, s_0 : \text{seq}\langle \mathbb{Z} \rangle$ ) : **Bool** =  $(\forall i : \mathbb{Z}) \ \#apariciones(s, i) = \#apariciones(s_0, i)$

**fun** ordenada( $s : \text{seq}\langle \mathbb{Z} \rangle$ ) : **Bool** =  $(\forall i : \mathbb{Z}) \ 0 \leq i < |s| - 1 \rightarrow_L s[i] \leq s[i+1]$

*Hint:* Utilizar la función *indice\_min\_subsec* resuelta en el ejercicio 33.

<sup>1</sup>Recordar que  $\text{mcd}(a, b) = \text{mcd}(a-b, b) = \text{mcd}(a, b-a)$



## 7. Matrices y Tableros

**Ejercicio 38.** Escribir una función que imprima por pantalla una matriz de enteros dejando tabs (`\t`) entre columnas y enters (`\n`) entre filas.

**Ejercicio 39.**

Escribir un algoritmo que dadas una matriz  $A$  de  $N \times M$ , y números  $n_2$  y  $m_2$  devuelva la matriz resultante de re-dimensionar la matriz  $A$  con la nueva dimensión  $n_2 \times m_2$ . ¿Cuál es la precondition del problema?

Ejemplo, suponiendo  $n_2 = 6$  y  $m_2 = 2$ :

A =						A' =					
1	2	3	4			1	2	3	4	5	6
5	6	7	8			7	8	9	10	11	12
9	10	11	12								

**Ejercicio 40.** Transponer in-place

Implementar un programa que cumpla con la siguiente especificación:

```
proc trasponer (inout m: seq<seq<Z>>) {  
  Pre {m = m0 ∧ (∀i : Z)(0 ≤ i < |m| →L |m[i]| = |m|)}  
  Post {|m| = |m0| ∧L (∀i : Z)(0 ≤ i < |m| →L (|m[i]| = |m0| ∧L (∀j : Z)(0 ≤ j ≤ |m| →L m[i][j] = m0[j][i])))}}  
}
```

**Ejercicio 41.** Picos

Implementar un programa que cumpla con la siguiente especificación:

```
proc contarPicos (in m: seq<seq<Z>>, out res: Z) {  
  Pre {|m| ≥ 2 ∧ |m[0]| ≥ 2 ∧L (∀i : Z)(0 ≤ i < |m| →L |m[i]| = |m[0]|)}  
  Post {res = ∑i=0|m| ∑j=0|m[i]| if esPico(m, i, j) then 1 else 0 fi}  
}  
  
pred esPico (m : seq<seq<Z>>, i: Z, j: Z) {  
  (∀a : Z)(i - 1 ≤ a ≤ i + 1 ∧ 0 ≤ a < |m| → (∀b : Z)(j - 1 ≤ b ≤ j + 1 ∧ 0 ≤ b < |m[a]| → (m[i][j] > m[a][b] ∨ (i == a ∧ j == b))))  
}
```

### 7.1. Opcionales (pero recomendados)

**Ejercicio 42.**

Para los siguientes ejercicios, recomendamos escribir una versión iterativa y una versión recursiva de la misma:

- Implementar una función que calcule el producto punto entre dos vectores.
- Dada una matriz  $A$ , escribir una función que retorne  $B = AA^t$  siendo  $A^t$  la matriz transpuesta de  $A$ .

**Ejercicio 43.**

Una matriz dispersa es una matriz de gran tamaño en la que la mayor parte de sus elementos son cero. Es común cambiar la forma en que representamos estas matrices para mejorar su procesamiento.

- Implementar la función `vector<tuple<int, int, int>> aTriplas(vector<vector<int>> m)` que: dada una matriz  $A$  formada por un vector de vectores, devuelva un vector de triplas  $(i, j, v)$  que contenga las posiciones de la matriz cuyos elementos sean distintos a cero junto a su valor.
- Implementar la función `vector<vector<int>> aMatriz(vector<tuple<int, int, int>> m)` que, dada una matriz representada como vector de triplas, genere la matriz asociada representada como vector de vectores.
- Implementar la función `void transponerDispersa(vector<tuple<int, int, int>>& m)` que transpone la matriz

#### Ejercicio 44.

Dada un vector multidimensional (tensor) de dimensiones  $(N \times M \times T)$  que representa la medición de temperatura en distintos puntos de una zona en el tiempo ( $N$  y  $M$  dimensiones de una grilla que cubren el terreno, y  $T$  el tiempo), se pide:

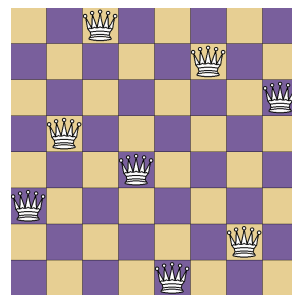
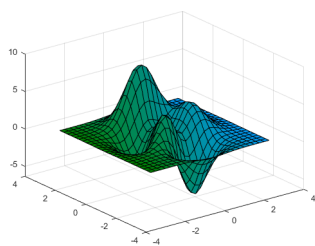
- Calcular la temperatura promedio de cada punto de la zona (el resultado debe ser una matriz de dimensión  $N \times M$ ).
- Calcular el promedio en la zona de la temperatura en cada momento (el resultado debe ser un vector de dimensión  $T$ ).

#### Ejercicio 45.

Dada una matriz de enteros que representa un terreno en el cual cada celda contiene el “nivel” del terreno (medido en metros): valores mayores o iguales a 0 se encuentran elevados sobre el agua y valores menores a 0 valores sub-acuáticos. Escribir funciones en c++ para los siguientes problemas:

- `void elevar(vector<vector<int>> &terreno, int x)` que eleve el terreno en  $x$  metros. Por ejemplo, si el terreno está todo a 0m salvo una celda con valor 4m, elevar en 2m dejaría todo el terreno en valor 2m salvo esa celda que contendría 6m. Aclaración:  $x$  podría ser negativo (en ese caso es como hundir el terreno). También podría ser 0, en ese caso el terreno no se modifica.
- `bool sobresalen(vector<vector<int>> terreno, int n, int& mts)` Que determine si es posible elevar o hundir el terreno una cierta cantidad de metros de manera tal que la cantidad de celdas bajo el agua sea exactamente  $n$ . En caso de ser posible, la función deberá devolver `true` y la variable `mts` deberá tomar como valor la cantidad de metros a elevar tal que se cumpla la condición. En caso que no sea posible, no hace falta asignar un valor a `mts` y el resultado deberá ser `false`. **Aclaración:** la variable `mts` no contiene ningún valor al comenzar.
- `int islas(vector<vector<int>> terreno)` Que calcule la cantidad de islas. Es decir, cantidad de zonas totalmente rodeadas por agua.

## 7.2. Para pensar (opcionales también)



#### Ejercicio 46.

**Valles:** Dada una matriz cuadrada de enteros sin repetidos que representa la elevación de un terreno, un valle es una celda tal que ninguna de las celdas adyacentes tiene un valor menor (mínimo local).

**Especificar** y escribir un algoritmo ...

- ... **iterativo** que busque algún valle a partir de la celda  $\langle i, j \rangle$  de la matriz.
- ... **recursivo** que encuentre la posición del valle más profundo alcanzable desde un punto dado  $\langle i, j \rangle$ .
- ... **recursivo** que encuentre todos los caminos que conducen a valles desde un punto dado  $\langle i, j \rangle$ .
- ... **iterativo** que resuelva los items anteriores (DIFICIL).

**Ejercicio 47.** Escribir un programa que dado un tablero de TaTeTi determine el estado de la partida. La función debe devolver “circulo” si el ganador fue circulo, “cruz” si el ganador fue cruz, “empate” si fue empate, “in progress” si el juego no concluyó o “invalido” si el tablero no es alcanzable. Precondiciones: el tablero es una matriz de  $3 \times 3$  que sólo contiene los chars 'X' (cruz), ' ' (espacio) o 'O' (circulo). Empiezan las cruces.

#### Ejercicio 48. N Reinas

El problema de las  $N$  reinas consiste en colocar  $N$  reinas en un tablero de  $N \times N$  sin que se amenacen entre ellas. Las reinas en el ajedrez se pueden atacar horizontalmente, verticalmente y en diagonal.

Utilizaremos una matriz de Char de dimensión  $N \times N$  en donde cada casillero será 'r' si hay una reina en el casillero, ' ' (espacio) si no.

- Implementar un función que dado un tablero de ajedrez determine si hay dos reinas que se encuentran en amenaza.
- Escribir un algoritmo que, dado un tablero vacío de  $N \times N$ , decida si se pueden ubicar las  $N$  reinas.

## 8. Testing

**Ejercicio 49.** Sin mirar el código, correr los casos de test provistos para el programa `esPrimo(in: int n, out: bool res)` visto en clase. Nuevamente sin mirar el código, y en base a los tests, discutir qué grado de confianza podemos tener en que la implementación sea correcta.

**Ejercicio 50.** *puntaje* 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.

El programa debe devolver: ¿Cuántos puntos ganó Sofía?

Armaz casos de test de caja blanca para el programa `int puntaje(int n)`, los test deben cubrir todas las decisiones (branches) del código.

**Ejercicio 51.** `bool sandia(int peso)`

Sara y Flor consiguieron una sandía para el postre que pesa una cantidad entera de kg. Quisieran dividirla en dos partes tales que cada una coma una cantidad par de kg (no necesariamente la misma cantidad cada una). Decidir si es posible realizar el corte en función del peso de la sandía.<sup>2</sup>

1. Programar en C++ una solución al problema.
2. Crear un conjunto de casos de test de caja negra que contenga un caso por cada partición del dominio del problema mostrado arriba.
3. Crear un conjunto de casos de test de caja blanca que cubra todas las líneas del programa.
4. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas las ramas de decisiones (branches) del programa.

Pensar ejemplos en donde se explote la siguiente partición de dominio:

1. Sandía con peso par.
2. Sandía con peso impar.

**Ejercicio 52.** *llenarTaxis*

Un grupo de chicos quiere viajar a un cumpleaños en varios taxis, que tienen capacidad para 4 personas cada uno. Los chicos están divididos en grupos de amigos de entre 1 y 3 personas y cada grupo de amigos quiere viajar en el mismo taxi. Cada taxi puede llevar a más de un grupo (por ejemplo, puede llevar dos grupos de dos personas). Se quiere contestar: ¿Cuál es la mínima cantidad de taxis que necesitan para poder llegar todos?<sup>3</sup>

La entrada son 3 números: la cantidad de grupos de un chico ( $n_1$ ), de dos ( $n_2$ ) y de tres ( $n_3$ ), respectivamente.

La salida es un número: la mínima cantidad de taxis que necesitan.

Armaz casos de test de caja negra según la partición de dominio que se encuentra al final del problema. Correr los tests sobre las diferentes implementaciones y encontrar **el error de cada una** de las implementaciones (no hace falta arreglarlas). Pensar ejemplos en donde se explote la siguiente partición de dominio:

1. La misma cantidad de grupos para cada tamaño.
2. Con cantidades distintas para tamaños de grupos distintos.
  - a)  $n_2$  par.
  - b)  $n_2$  impar...
    - 1) y  $n_1 - n_3 = 1$  ó  $2$
    - 2) y  $n_1 - n_3 \neq 1$  ó  $2$

---

<sup>2</sup>Adaptación de <http://codeforces.com/problemset/problem/4/A>

<sup>3</sup>Adaptación de <http://codeforces.com/problemset/problem/158/B>

**Ejercicio 53.** *baldosasDelPiso* Trabajar sobre el problema `int baldosasDelPiso(int N int M int B)`<sup>4</sup>:

La facultad quiere cambiar las baldosas del piso de un aula rectangular (de dimensiones  $M \times N$ ) sin huecos ni superposiciones. Las baldosas son cuadradas (de  $B \times B$ ).

Las baldosas se pueden cortar para ponerlas en el piso, pero a lo sumo se puede usar un pedazo de cada una (es decir, no vale cortarlas a la mitad para tener dos más chicas).

¿Cuál es la mínima cantidad de baldosas que se necesita para cubrir el aula?

1. Programar en C++ una solución al problema.
2. Crear un conjunto de casos de test de caja negra que contenga un caso por cada partición del dominio del problema mostrado arriba.
3. Crear un conjunto de casos de test de caja blanca que cubra todas las líneas del programa.
4. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas las ramas de decisiones (branches) del programa.
5. Utilizar Lcov para revisar que los tests cubran todas las líneas de esta función.

Pensar ejemplos en donde se explote la siguiente partición de dominio:

1.  $M$  y  $N$  divisibles por  $B$ .
2. Sólo  $M$  divisible por  $B$ .
3. Sólo  $N$  divisible por  $B$ .
4. Ninguno divisible por  $B$ .

**Ejercicio 54.** `int contandoDragones(int t, int d1, int d2, int d3)`

Ana Paula tiene insomnio, y cuenta dragones para dormir porque las ovejas la aburrieron. Una noche se aburrió también de contar dragones, entonces se empezó a imaginar que a algunos les hacía maldades.

Una noche, contó  $T$  dragones (en algún orden). Luego, respetando el orden a uno de cada  $D_1$  dragones le tiró gas pimienta en la nariz, a uno de cada  $D_2$  dragones lo roció con un extintor, y a uno de cada  $D_3$  le puso demasiado picante en la comida. Sabemos que  $D_1 < D_2 < D_3$  (como puede verse, algunos dragones podrían recibir más de un escarmiento).

¿Cuántos dragones no fueron víctimas de ninguna de sus maldades?<sup>5</sup>

Pista: sale sin usar un ciclo que chequee los dragones uno por uno.

1. Programar en C++ una solución al problema.
2. Crear un conjunto de casos de test de caja negra que contenga un caso por cada partición del dominio del problema mostrado arriba.
3. Crear un conjunto de casos de test de caja blanca que cubra todas las líneas del programa.
4. Extender (si es necesario) el conjunto de casos del ítem anterior para que cubra todas las ramas de decisiones (branches) del programa.

Pensar ejemplos en donde se explote la siguiente partición de dominio:

1.  $D_1$ ,  $D_2$  y  $D_3$  coprimos. Se divide en:
  - a) Coprimos de a pares
  - b) No coprimos de a pares
2.  $D_1$ ,  $D_2$  y  $D_3$  no coprimos. Se divide en:
  - a)  $D_2$  múltiplo de  $D_1$  y  $D_3$  múltiplo de  $D_2$ .
  - b) No ocurre la condición de arriba.

---

<sup>4</sup><http://codeforces.com/problemset/problem/1/A>

<sup>5</sup>Adaptación de <http://codeforces.com/problemset/problem/148/A>

## 9. Tiempo de Ejecución de Peor Caso de un Programa

Para los siguientes ejercicios recomendamos utilizar `contruir_vector`, que recibe un entero  $n$  y un `string disposicion` y devuelve un vector de  $n$  elementos en la disposición especificada (opciones para disposición: “asc”, “desc”, “azar”, “iguales”)

### Ejercicio 55. Medición de tiempos

En el archivo `ejercicios.cpp` encontrarán la implementación de `hayDuplicados(vector<int> v)` que dado un vector  $v$  demora aproximadamente  $c \cdot n^2$  segundos en terminar en el peor caso (siendo  $n$  el tamaño del vector de entrada y  $c$  una constante que no depende de  $n$ ). Es decir, su tiempo de ejecución de peor caso pertenece a  $O(n^2)$ .

1. Estimar el valor de  $c$  mediante simulaciones. ¿Cambia el valor encontrado dependiendo el valor de los elementos del vector? ¿Cambia el valor encontrado según desde qué  $n$  se comienza a medir?
2. Calcular cuántos segundos demoraría (sin correrlo!) aproximadamente en el peor caso si  $n = 2200000000$  (# de usuarios de Facebook)

### Ejercicio 56. Graficar

Escribir los tiempos calculados en un archivo con  $n$  desde 1 hasta 10000 (paso 500). Este archivo deberá contener una columna para  $n$  (con encabezado “n”) y una columna (con cualquier encabezado declarativo) para el tiempo para dicho  $n$  en segundos.

Generar un gráfico con eje  $x$ :  $n$  y eje  $y$ : tiempo de ejecución en segundos. Para graficar, pueden utilizar cualquier programa que deseen o el script provisto por la cátedra. Para ver el modo de uso del script: `python3 graficador.py --help`. Si se desea visualizar más de una curva a la vez, pueden agregar columnas al archivo con nombres declarativos como encabezado de la columna.

### Ejercicio 57. Opcional

Modificar las guardas del programa del punto 1 para que los ciclos terminen en caso de haber encontrado un duplicado. Medir nuevamente. ¿Cambió la complejidad?

### Ejercicio 58.

Escribir programas con tiempo de ejecución de peor caso pertenecientes a  $O(1)$ ,  $O(n)$ ,  $O(n^3)$  y  $O(n \cdot \log(n))$ . Utilizar si se desea la función provista en el ejercicio 1 y la función `busqueda_binaria(vector<int> v, int e)` incluida en los archivos de la clase.

1. Medir el tiempo en segundos y completar la siguiente tabla

	$O(1)$	$O(n)$	$O(n^3)$	$O(n \cdot \log(n))$
n=100				
n=1000				
n=10000				
n=100000				

### Ejercicio 59. STL

Realizar simulaciones para estimar la complejidad temporal de las siguientes operaciones sobre vectores en C++.

1. `v.size()`
2. `v.push_back(e)`  
Tip: Analizar que pasa cuando se supera el tamaño del vector en memoria  
<http://www.cplusplus.com/reference/vector/vector/capacity/>  
<http://www.cplusplus.com/reference/vector/vector/reserve/>
3. `v.pop_back()`
4. `v[i]` (¿cambia la complejidad estimada según el valor de  $i$ ?)
5. `v[i] = e;` (¿cambia la complejidad estimada según  $i$ ?)
6. `v.flip()` (sólo para  $v$  de tipo `vector<bool>`)
7. `v.clear()`

¿Por qué sucede que las mediciones con  $O(1)$  contienen dos lineas marcadas? ¿Por qué el `push_back` tiene saltos?

### Ejercicio 60.

Realizar simulaciones y gráficos para determinar el tiempo de ejecución de **peor caso** del programa `algunSubconjSuma` (se encuentra en `ejercicios.cpp`).

## 10. Búsqueda y Ordenamiento

Para todos los ejercicios de esta guía, calcular el tiempo de ejecución en el peor caso de los algoritmos implementados.

**Ejercicio 61.** Dada una secuencia de palabras (strings) de longitud  $n$  en donde cada palabra tiene a lo sumo 40 caracteres, escribir un programa que devuelva la secuencia ordenada según la longitud de cada string y ante una misma longitud, en el orden en que aparecía originalmente en la secuencia. El programa debe garantizar tiempo de ejecución de peor caso perteneciente a  $O(n)$ . Por ejemplo, `ordenarPorFrec({"hola", "esto", "es", "una", "prueba"})` debe devolver `{"es", "una", "hola", "esto", "prueba"}`

### Ejercicio 62. Anagramas

Una palabra es anagrama de otra si las dos tienen las mismas letras, con el mismo número de apariciones. Por ejemplo la palabra "delira" es anagrama de la palabra "lidera". Se pide implementar un programa que tome dos palabras con letras minúsculas de la 'a' a la 'z' y calcule si son anagramas o no siguiendo las siguientes técnicas:

- Usando ordenamiento** para ver si  $p_1$  y  $p_2$  son anagramas basta con ordenar los caracteres de cada una y comparar si el string resultante es el mismo.
- Usando números primos:**
  - Asignamos a cada letra del alfabeto un número primo distinto.
  - Convertimos a la palabra en número a través de multiplicar los valores asignados a cada letra.
  - Por el teorema fundamental del álgebra cada combinación de letras tiene un número que identifica unívocamente a todos los anagramas. Entonces alcanza con comparar si los números obtenidos son iguales.
- Pensar otro** método que garantice tiempo de ejecución de peor caso sea  $O(|p_1| + |p_2|)$ .

**Ejercicio 63.** Dados dos secuencias de enteros de tamaño  $n$  en donde cada una de ellas está ordenada. Escribir un programa que encuentre la mediana<sup>6</sup> de la secuencia combinada.














- Usando ordenamiento** luego de concatenar las dos secuencias previo a calcular la mediana.
- Usando apareamiento** para combinar las dos secuencias previo a calcular la mediana.
- Sin ordenar**, pensando otra técnica que garantice tiempo de ejecución de peor caso perteneciente a  $O(\log(n))$ .

**Ejercicio 64.** El "ordenamiento natural" de strings es una forma de ordenar de manera alfabética pero considerando **números enteros** entre las palabras. Por ejemplo, si tenemos:

`{'version-1.9', 'version-2.0', 'version-1.11', 'version-1.10'}`, ordenar de manera convencional devuelve `{'version-1.10', 'version-1.11', 'version-1.9', 'version-2.0'}`, en cambio, uno esperaría un orden que considere que la versión 1.9 es anterior a la versión 1.11.<sup>7</sup> Para ver más ejemplos sobre ordenamiento natural, crear archivos en un mismo directorio y ver cómo los ordena sistema operativo como se muestra en la figura. Se pide entonces:

- Escribir un programa que determine cuando un `string` es menor a otro según el orden natural. Utilizar si es necesario la función `stoi` o `stof` que convierte `string` a `int` o `float` respectivamente.

#### Name

 Ejemplo00
 Ejemplo000
 Ejemplo0000
 Ejemplo1
 Ejemplo2
 Ejemplo12
 Ejemplo12.4
 Ejemplo12a
 Ejemplo12a2
 Ejemplo12a12
 Ejemplo12a12.3
 Ejemplo12a123
 Ejemplos0000

- Escribir un programa `natSorted` que ordene una secuencia de `string` de manera natural.

<sup>6</sup>[https://es.wikipedia.org/wiki/mediana\\_\(estadística\)](https://es.wikipedia.org/wiki/mediana_(estadística))

<sup>7</sup>en este caso, ya que  $1 = 1$ , el algoritmo compara 9 contra 11.