# Trabajo Práctico I

## 1. Motivación del problema

El objetivo del trabajo es implementar algunos algoritmos de ordenación sobre listas enlazadas generales. Un par de ellos ya hemos visto en **Programación II**. En particular nos referimos a los algoritmos: **Insertion Sort** (Ordenación por Inserción), **Selection Sort** (Ordenación por Selección) y **Merge Sort** (Ordenamiento por Mezcla).

Ante todo, acá hay una referencia de cada algoritmo:

- Selection Sort: `https://es.wikipedia.org/wiki/Ordenamiento_por_selecci%C3%B3n`
- Insertion Sort: `https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n`
- Merge Sort: `https://es.wikipedia.org/wiki/Ordenamiento_por_mezcla`

Analicemos ahora cada una de las funciones y cómo sería el prototipo que deberíamos darle en *C*.

Tomemos, por ejemplo, el algoritmo **Selection Sort** que toma una lista de elementos y una función comparación, es decir una función que dados dos elementos retorna un entero indicando si el primero es menor que el segundo. Pasando esto en limpio, nuestro prototipo debería ser:

```
GList glist_selectionSort(GList lista, Compara c);
```

donde *GList* es una lista genérica (simple, doble, circular o cualquier otra variante) donde, por ejemplo, una implementación posible de una lista genérica simplemente enlazada sería:

```
typedef struct _GNodo {
  void* dato;
  struct _GNodo *sig;
} GNodo;

typedef GNodo *GList;
```

y *Compara* no es otra cosa que:

```
typedef int (*Compara) (void* dato1, void* dato2);
```

de forma tal que retorna un entero menor que, igual a, o mayor que 0 , si el primer argumento es considerado, respectivamente, menor que, igual a ó mayor que el segundo.

Para poder hacer un uso adecuado de la gestión de la memoria, debemos liberarla, para eso definimos una función:

```
void gList_destruir(GList lista, Destruir d);
```

donde *Destruir* es:

```
typedef void (*Destruir) (void* dato);
```

esta función recibe un puntero a un dato y, sabe cómo liberar la memoria que ocupa.

## 2. Juego de datos de prueba

Para hacer una prueba de nuestra implementación, definamos la siguiente estructura:

```
typedef struct  {
  char*  nombre;
  int edad;
  char* lugarDeNacimiento; //pais o capital
} Persona;
```

Junto con este documento va a encontrar dos archivos: *nombres.txt* y *paises.txt*. El objetivo es generar un nuevo archivo que almacene, datos generados en forma aleatoria (con la información dada en los archivos, siendo la edad un número de 1 a 100) con el formato:

```
Juan Perez, 19, Villa General Belgrano
Francisco Jose Maria Olviares, 23, Rosario
```

La cantidad de datos a generar se pasará como argumento.

## 3. Implementación

Se deben entregar dos programas:

1. Generación de los datos de prueba. Este programa debería generar el archivo, mencionado en la sección inmediata anterior.
2. El segundo programa:
   *a*) Elegir **una** implementación de listas sobre la cual realizar el TP.

*b*) Implementar los 3 algoritmos de ordenación sobre la implementación de listas elegida. La implementación de los mismos **no** debe usar una lista auxiliar para copiar datos.

*c*) Toma como entrada el archivo del punto anterior generando una *GList* con esos datos.

*d*) Se debe ejecutar cada algoritmo con 2 funciones de comparación que actúen sobre difentes atributos de *Persona*. Las listas obtenidas como resultado de esas funciones deberían volcarse a un archivo junto con el tiempo que se tardó en ejecutar dicho algoritmo.

Sugerencia: Se podría hacer una función que encapsule este comportamiento, es decir, que tome un nombre de archivo, un método de ordenación, una función comparación y una lista; invoque al método correspondiente y, vuelque la lista resultante a un archivo de salida.

## 4. Características del Código a Entregar

Se pide que escriba un programa que cumpla con los siguientes requisitos:

- cumplir con las Convenciones marcadas por la cátedra en el sitio de Comunidades;
- no se pueden usar variables globales, es decir, definidas fuera de funciones;
- los nombres de los archivos de entrada y salida deben ser pasados como argumentos al programa;
- tengan en cuenta que hay letras acentuadas y caracteres especiales que pueden llegar a generar problemas dependiendo de la codificación del sistema operativo; explique de qué forma se resuelve esto en la documentación.
- Se debe entregar un informe explicando:
  - la elección de la estructura de datos elegida;
  - cómo se compila y usa el programa, en caso de no entregar un makefile;
  - compare cada método con los resultados obtenidos al correrlo con juegos de datos de diferente tamaño. Consideramos 20.000 un tamaño significativo;
  - bibliografía usada, en caso de haberla.