Trabajo Práctico I

1. Motivación del problema

El objetivo del trabajo es implementar un par de funciones que están disponibles en cualquier lenguaje funcional de forma tal de agregarlas a nuestra implementación de listas. En particular estamos pensando en la función **filter** y la función **map**.
Analicemos cada una de las funciones y cómo sería el prototipo que deberíamos darle en *C*.

La función **filter** toma una lista de elementos y un predicado, es decir una función que retorna true/false, como argumentos y retorna una *nueva lista* formada por los elementos de la lista original que cumplen con el predicado. Pasando esto en limpio, nuestro prototipo debería ser:

```
GList filter(GList lista, Predicado p, Copia 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;
```

Predicado no es otra cosa que:

```
typedef int (*Predicado) (void* dato);
```

y *Copia* es una función que recibe un puntero y retorna un puntero a una copia física (real, en memoria) de la información a la que referencia el argumento, es decir:

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

Por otro lado, la función **map**, toma como argumentos una lista y una función y retorna una *nueva lista* formada por las imágenes de cada elemento de la lista original luego de aplicarle la función. Con esta idea en mente, el prototipo debería ser:

```
GList map(GList lista, Funcion f, Copia c);
```

donde *Funcion* es:

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

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 el documento va a encontrar dos archivos: *nombres.txt* y *paises.txt*. El objetivo es generar un nuevo archivo que almacene, al menos, 2000 datos generados en forma aleatoria (con la información dada en los archivos, siendo la edad un número de 1 a 100) con el formato:

```
nombre, edad, lugarDeNacimiento
```

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, con la información aleatoria.
2. El segundo programa:
   a) Toma como entrada el archivo del punto anterior generando una *GList* con esos datos.
   b) Se debe ejecutar la función *filter* con al menos 2 funciones que actuen sobre difentes atributos de *Persona*. Las listas obtenidas como resultado de esas funciones deberían volcarse a un archivo.
   Observación: se podría hacer una función que encapsule este comportamiento, es decir, que tome un nombre de archivo, una función y una lista. Invoque a filter y vuelque la lista resultante al archivo.

c) Se debe ejecutar la función *map* con al menos 2 funciones que actuen sobre difentes atributos de *Persona*. Las listas obtenidas como resultado de esas funciones deberían volcarse a un archivo.
Observación: se puede usar la misma idea mencionada anteriormente.

4. Caracerísticas 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, definidas fuera de funciones;
- la forma en la que se leen las palabras de entrada y, la forma en la que se genera el archivo de salida queda librado a la decisión de cada grupo pudiendo hacer uso de funciones propias de archivo o, redirigiendo la entrada/salida. Esta información debe formar parte de la documentación a entregar. 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 haga uso adecuado de la memoria dinámica (pedido y liberación de la misma).