Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Computación
Estructuras de Datos y Algoritmos I

Práctica 2 - Listas

1. Lea la implementación provista de listas simplemente enlazadas, y el ejemplo presentado en el archivo `main.c`. Asegúrese comprenderlo.

2. Extienda la implementación de listas enlazadas dada en clase.
Diseñe y programe las siguientes funciones:

- **slist_longitud** que devuelve la longitud de una lista.
- **slist_concatenar** que devuelve la concatenación de dos listas (modificándolas).
- **slist_insertar** que inserta un dato en una lista en una posición arbitraria.
- **slist_eliminar** que borra de una lista un dato apuntado en una posición arbitraria.
- **slist_contiene** que determina si un elemento está en una lista dada.
- **slist_indice(SList lista, int dato)** que devuelve la posición de la primera ocurrencia del elemento dato si el mismo está en la lista dada, -1 en caso que no esté.
- **slist_intersecar** que devuelve una nueva lista con los elementos comunes (independientemente de la posición) de dos listas dadas por parámetro. Las listas originales no se modifican.
- **slist_intersecar_custom** que trabaja como la anterior pero recibe un parámetro extra que es un puntero a una función de comparación que permite definir la noción de igualdad a ser usada al comparar elementos por igualdad.
- **slist_ordenar** que ordena una lista de acuerdo al criterio dado por una función de comparación (que usa los mismos valores de retorno que *strcmp()*) pasada por parámetro.
- **slist_reverso** que obtenga el reverso de una lista.
- **slist_intercalar** que dadas dos listas, intercale sus elementos en la lista resultante. Por ejemplo, dadas las listas [1, 2, 3, 4] y [5, 6], debe obtener la lista [1, 5, 2, 6, 3, 4].

Indique cuáles son las operaciones que piensa que más tiempo consumen en ejecutarse. ¿Cuáles de ellas dependen del tamaño de sus argumentos?

3. Las listas enlazadas implementadas con punteros tienen la flexibilidad de poder insertar elementos en cualquier parte de ellas sin mover los elementos anteriores ni posteriores, mientras que los arreglos no cuentan con esta flexibilidad.
Proponga una implementación similar a listas enlazadas, pero de longitud fija, utilizando arreglos, y que provea esta flexibilidad.
Nota: Ver sección *"Implantación con arreglo de listas"*, página 195, *"Estructuras de Datos con C"* Tenenbaum.

4. Implemente listas enlazadas circulares, siguiendo el formato hecho para listas enlazadas. Llame a los ficheros como `cslist.h` y `cslist.c`. Implemente `cslist_recorrer` de manera que solo ejecute una pasada sobre cada elemento.

5. Implemente listas doblemente enlazadas, de manera circular o con una estructura auxiliar que lleve registro del primer y último nodo en la lista. Llame a los ficheros como `dlist.h` y `dlist.c`. Implemente `dlist_recorrer` de manera que se pueda elegir si se avanza o retrocede en el recorrido, utilizando el tipo:

```
typedef enum {
  DLIST_RECORRIDO_HACIA_ADELANTE,
  DLIST_RECORRIDO_HACIA_ATRAS
} DListOrdenDeRecorrido;
```

6. Suponga que está implementando una lista doblemente enlazada. Si en lugar de guardar punteros a los nodos previo y siguiente, guarda un xor de ambos punteros: ¿puede recorrer la lista en ambas direcciones? ¿Cómo? Defina en C la estructura correspondiente, ¿qué problemas puede encontrar?
Enuncie posibles ventajas y desventajas de esta implementación de listas.

7. Implemente listas utilizando arreglos. Originalmente utilice un arreglo de 4 enteros. Luego, si se intenta realizar una operación que supera la longitud disponible, realoque la información en un arreglo con el doble de tamaño (puede ser de utilidad la función `realloc`).

8. Implemente listas generales. Para ello deberá reimplementar todas las funciones del ejercicio 2 usando las siguientes declaraciones.

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

typedef GNodo *GList;
```

9. Utilizando las funciones implementadas en el ejercicio anterior, cree un programa que manipule listas de enteros y que acepte comandos desde la entrada estándar de la siguiente forma:

Comando	Argumentos	Resultado	Ejemplo
crear	lista	Crea una lista	crear 1
destruir	lista	Destruye una lista	destruir 1
imprimir	lista	Imprime el contenido actual de la lista	imprimir 1
agregar_final	lista, elem	agrega elem al final de la lista	agregar_final 1 42
agregar_inicio	lista, elem,	agrega elem al principio de la lista	agregar_inicio 1 42
agregar_pos	lista, elem, pos	agrega elem a la lista en la posición pos	agregar_pos 1 42 3
longitud	lista	imprime la longitud de la lista	longitud 1
concatenar	l1, l2, l3	concatena l1 y l2 y crea l3 con el resultado	concatenar 1 2 3
eliminar	lista, pos	elimina de lista el elemento en la posición pos	eliminar 1 5
contiene	lista, elem	imprime “SI” si la lista contiene a elem, “NO” sino	contiene 1 42
indice	lista, elem	imprime las posiciones en las que está elem	indice 1 42
intersecar	l1, l2, l3	crea l3 con la intersección de l1 y l2	intersecar 1 2 3
ordenar	lista	ordena los elementos de la lista de menor a mayor	ordenar 1

Aclaración: Cada línea tendrá sólo un comando.