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

Práctica 1

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

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

- **slist_has_next** que diga si un nodo tiene siguiente elemento.
- **slist_length** que devuelva la longitud de una lista.
- **slist_concat** que devuelva la concatenación de dos listas.
- **slist_insert** que inserta un dato en una lista en una posición arbitraria.
- **slist_remove** que borra de una lista un dato apuntado en una posición arbitraria.
- **slist_contains** que diga si un elemento está en una lista dada.
- **unsigned int slist_index(SList *list, const int data)** que devuelva la posición del elemento data si el mismo está en la lista list.
- **slist_intersect** que devuelva una nueva lista con los elementos comunes (independientemente de la posición) de dos listas dadas por parámetro. Las listas originales no se modifican.
- **slist_intersect_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_sort** 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. Puede usar su propio método de ordenación o es aceptable usar qsort de la biblioteca estándar.

Indique cuáles son las operaciones que piensa que más tiempo consumen en ejecutarse.

3. 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
create	lista	Crea lista	create 1
destroy	lista	Destruye lista	destroy 1
print	lista	Imprime el contenido actual de la lista	print 1
add_end	lista, elem	agrega elem al final de lista	add_end 1 42
add_beg	lista, elem,	agrega elem al principio de lista	add_beg 1 42
add_pos	lista, elem, pos	agrega a elem a lista en la posición pos	add_pos 1 42 3
length	lista	imprime la longitud de la lista	length 1
concat	l1, l2, l3	concatena l1 y l2, crea l3 con el resultado	concat 1 2 3
remove	lista, pos	elimina de lista el elemento en la posición pos	remove 1 5
contais	lista, elem	imprime “SI” si lista contiene a elem, “NO” sino	contains 1 42
index	lista, elem	imprime las posiciones en las que está elem	index 1 42
intersec	l1, l2, l3	crea l3 con la intersección de l1 y l2	intersec 1 2 3
sort	lista	ordena los elementos de lista de menor a mayor	sort 1

Aclaración: Cada línea tendrá sólo un comando.

4. Implemente listas doblemente enlazadas, siguiendo el formato hecho para las listas enlazadas, considere si agregaría nuevas operaciones. Llame a los ficheros como `DList.h` y `DList.c`. Implemente `dlist_foreach` de manera que se pueda elegir si se avanza o retrocede en el recorrido. Utilice el tipo:

```
typedef enum {
  DLIST_TRAVERSAL_ORDER_FORWARD,
  DLIST_TRAVERSAL_ORDER_REWARD
} DListTraversalOrder;
```

5. 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 de similar a listas enlazadas, pero de longitud fija, utilizando arreglos, y que provea esta flexibilidad.

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

7. 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.

8. 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`).