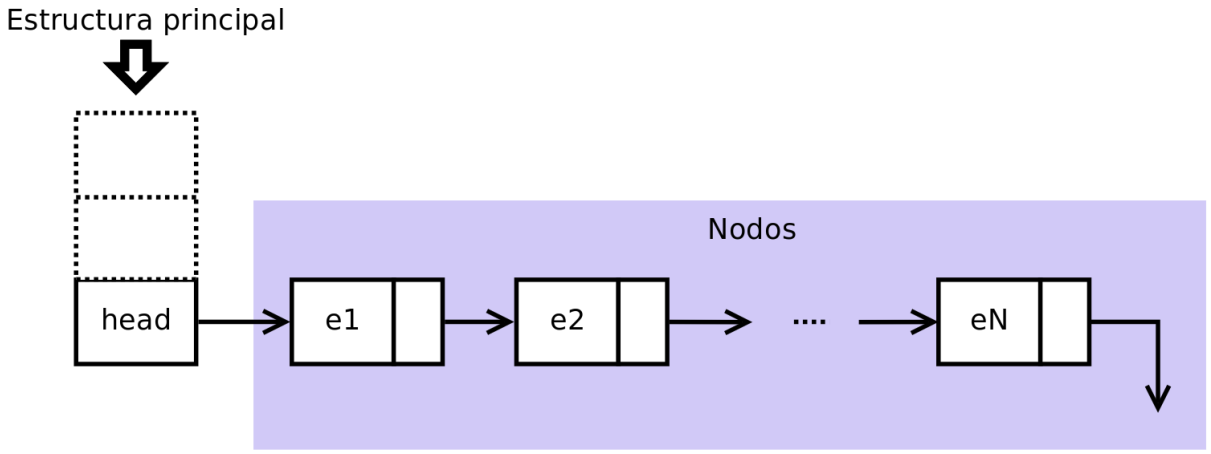
Córdoba 22 de Junio del 2023

Algoritmos y Estructuras de Datos II

Recuperatorio Tema B - TAD List

Implementar el TAD **list** que representa una lista de valores numéricos. El TAD almacena números en nodos (simplemente) enlazados, usando una estructura principal que tiene entre otras cosas un puntero al primer nodo. A modo de ilustración se puede ver el siguiente diagrama:



Las operaciones del TAD se listan a continuación:

Función	Descripción
`list list_empty()`	Crea un lista vacía
`list list_addl(list l, elem e)`	Agrega el elemento **e** a la lista por izquierda
`bool list_is_empty(list l)`	Indica si la lista está vacía
`elem list_head(list l)`	Devuelve el primer elemento de la lista
`list list_tail(list l)`	Elimina el primer elemento de la lista
`list list_addr(list l, elem e)`	Agrega el elemento **e** a la lista por derecha
`ListSize list_length(list l);`	Devuelve el tamaño de la lista
`void list_print(list l);`	Muestra el contenido de la lista por pantalla
`list list_destroy(list l)`	Destruye la lista **l** liberando toda la memoria utilizada
`list list_greater_than(list l, unsigned int n)`	Devuelve una nueva lista con los elementos de **l** que son estrictamente mayores que **n**.

`unsigned int list_greater_than_count(list l, unsigned int n)`	Devuelve la cantidad de elementos de **`l`** que son estrictamente mayores que **`n`**.
`list list_insert_at(list l, unsigned int position, elem e)`	Inserta el elemento **`e`** en la lista **`l`** en la posición indicada por **`position`**

Se debe completar la estructura principal para lograr que la función **`list_length()`** sea de orden constante *O(1)*.

El programa resultante no debe dejar *memory leaks ni lecturas/escrituras inválidas*.

Se provee un archivo `main.c` que contiene ya un caso de prueba implementado. Deben completarlo incluyendo:

- Un caso de prueba para la función **`list_greater_than()`**: Debe filtrar la lista por el mayor que y luego deberá llamar a **`list_greater_than_count()`** para comparar el largo de la lista devuelta por **`list_greater_than()`** con el resultado de **`list_greater_than_count()`**
- Un caso de prueba para la función **`list_insert_at()`**: Se debe insertar un elemento en la posición **`0`** de la lista validar que se comporte como **`list_addl()`**
- Un caso de prueba para la función **`list_insert_at()`**: Se debe insertar un elemento en la posición **`N-1`** de la lista validar que se comporte como **`list_addr()`**. La lista debe tener más de cuatro elementos.
- Un caso de prueba para la función **`list_insert_at()`**: Se debe insertar un elemento en alguna posición `0 <` **`position`** `< N-1` la lista debe tener al menos cuatro elementos. Imprimir la lista.

Una vez compilado el programa puede probarse ejecutando:

```
$ ./listrun
```

Consideraciones:

- Solo se deben modificar el archivo **`main.c`** y **`list.c`**
- Se provee el archivo **`Makefile`** para facilitar la compilación.
- Se recomienda usar las herramientas **`valgrind`** y **`gdb`**.
- Usando **`make test`** se compila y ejecuta el programa
- Con **`make valgrind`** se compila y luego se ejecuta el programa usando **`valgrind`**
- Si el programa no compila, no se aprueba el parcial.
- Los *memory leaks* bajan puntos
- Entregar código muy improlijo puede restar puntos
- Si **`list_length()`** no es de orden constante *O(1)* baja muchísimos puntos
- **Se debe hacer una invariante** que chequee consistencia entre los campos de la estructura principal.