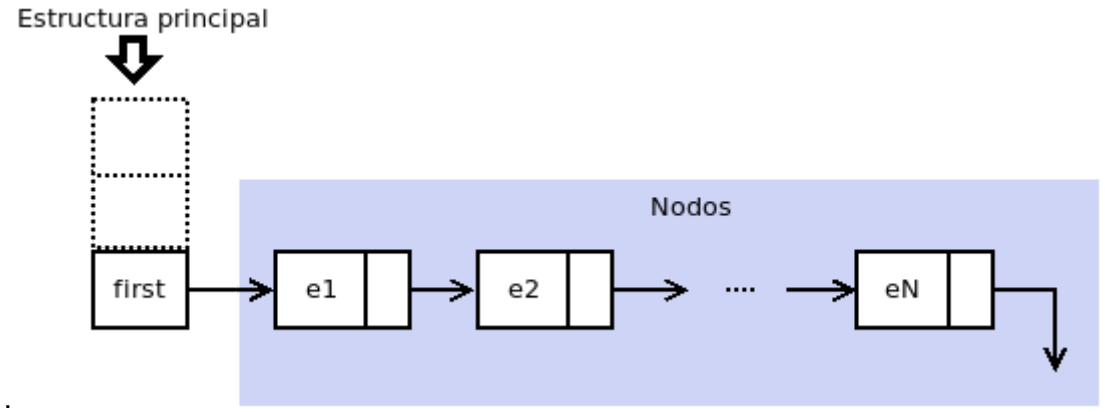
Córdoba 2 de Junio del 2023

Segundo Parcial de Laboratorio

Algoritmos y Estructura de Datos II

Tema C - TAD Conjunto

Implementar el TAD **set** que representa un conjunto de valores numéricos. El TAD almacena números, los cuales dada la naturaleza de los conjuntos en matemática no pueden repetirse. Se debe representar el conjunto mediante nodos (simplemente) enlazados, usando una estructura principal que tenga un puntero al primer nodo. A modo de ilustración se puede ver el siguiente diagrama:



La implementación **debe mantener ordenada la cadena de nodos** logrando que en todo momento `e1 < e2 < ... < eN`. Esto debe ser tenido en cuenta al implementar `set_add()`.

Representar de esta forma a los conjuntos permite que la búsqueda de un elemento `e` pueda realizarse sin recorrer todos los nodos, ya que al momento en que se encuentra un elemento mayor en la cadena de nodos, si `e` no apareció todavía, nunca lo hará debido a que los posteriores serán todos mayores.

Las operaciones del TAD se listan a continuación:

Función	Descripción
`set set_empty(void)`	Crea un conjunto vacío
`set set_add(set s, set_elem e)`	Agrega el elemento `e` al conjunto. Si ya estaba presente, el conjunto queda intacto.
`bool set_is_empty(set s)`	Indica si el conjunto está vacío
`unsigned int set_cardinal(set s)`	Devuelve la cantidad de elementos que hay en el conjunto
`bool set_member(set_elem e, set s)`	Indica si el elemento `e` se encuentra en el conjunto

`set set_elim(set_elem e, set s)`	Elimina el elemento e del conjunto. Si **e** no estaba el conjunto queda intacto.
`set_elem set_get(set s);`	Devuelve algún elemento del conjunto
`set_elem * set_to_array(set s);`	Devuelve un arreglo en memoria dinámica con el contendio del conjunto
`set set_destroy(set s)`	Destruye el conjunto y libera toda la memoria usada

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

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

Una vez compilado el programa puede probarse ejecutando:

```
$ ./setload input/example-easy-005.in
```

Obteniendo como resultado:

```
{1, 2, 3, 4, 5}

El archivo 'input/example-easy-005.in' tiene 5 elementos únicos
```

El nombre del archivo indica cuántos valores enteros están listados dentro de él. Puesto que pueden haber valores repetidos, la cantidad de elementos únicos no coincidirá con la cantidad de datos en el archivo. Otro ejemplo de ejecución:

```
$ ./setload input/example-unsorted-005.in
{-1, 0, 2, 3}

El archivo 'input/example-unsorted-005.in' tiene 4 elementos únicos
```

Los valores esperados para los ejemplo son

Archivo	Elementos únicos
example-easy-005.in	5
example-sorted-005.in	4
example-unsorted-005.in	4
example-unsorted-015.in	7
example-unsorted-100.in	44

Consideraciones:

- Solo se debe modificar el archivo **`set.c`**
- Se incluyen signaturas de un par de funciones **`static`** que creemos pueden ser útiles a la hora de completar el TAD. No es necesario que las implementen, pero hacerlo puede facilitar la tarea.
- Se provee el archivo **`Makefile`** para facilitar la compilación.
- Se recomienda usar las herramientas **`valgrind`** y **`gdb`**.
- Usando **`make test`** se corren todos los ejemplos usando **`valgrind`**
- Para que se sigan probando los ejemplos aunque falle alguno se puede usar:
  `$ make test -k`
- Si el programa no compila, no se aprueba el parcial.
- Los *memory leaks* bajan puntos
- Entregar código muy improlijo puede restar puntos
- Si **`set_cardinal()`** no es de orden constante *O(1)* baja muchísimos puntos
- Para promocionar **se debe** hacer una invariante que chequee la propiedad fundamental de la representación.
- **No modificar los .h puesto que solo se entregará set.c**