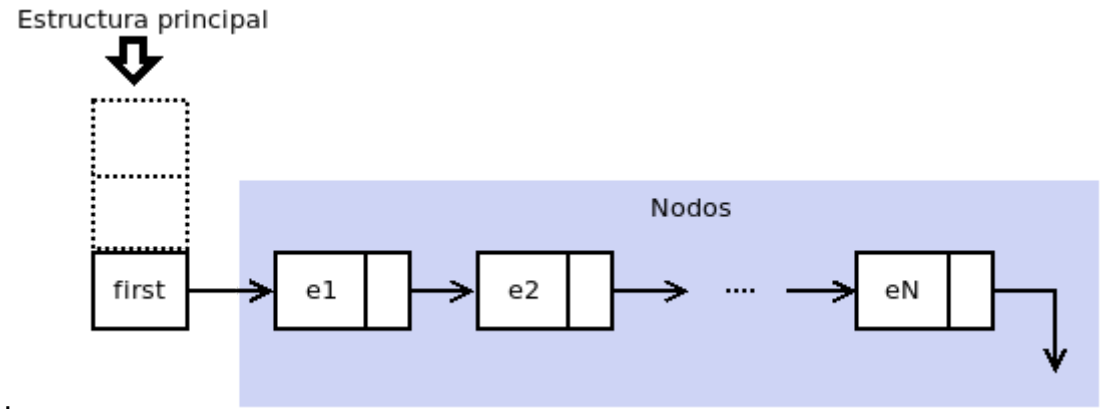
Córdoba 8 de Julio del 2024

Examen Final de Laboratorio

Algoritmos y Estructura de Datos II

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 contenido del conjunto
`void set_dump(set s);`	Muestra el contenido del conjunto por pantalla
`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 usando los archivos de la carpeta `input`. El nombre de cada archivo de esta carpeta 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. Un ejemplo de ejecución:

```
$ ./setload input/example-easy-005.in
Set of elements read in the file: {1, 2, 3, 4, 5}

The file 'input/example-easy-005.in' has 5 unique elements
1 [OK], 2 [OK], 3 [OK], 4 [OK], 5 [OK]
```

La salida del programa muestra al principio el conjunto construido (usando `set_dump()`) y luego indica la cantidad de elementos únicos encontrados usando `set_cardinal()`. Por último para cada elemento del arreglo devuelto por `set_to_array()` verifica que dicho elemento pertenezca al conjunto usando `set_member()` indicando con `OK` si el elemento fue encontrado o `FAIL` en caso contrario (si esto último sucede evidentemente hay un problema en la implementación del TAD). Otro ejemplo de ejecución:

```
$ ./setload input/example-unsorted-005.in
Set of elements read in the file: {-1, 0, 2, 3}

The file 'input/example-unsorted-005.in' has 4 unique elements
-1 [OK], 0 [OK], 2 [OK], 3 [OK]
```

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

Por último, cuando al programa se le dan más parámetros, luego de cargar los datos del archivo procederá a eliminar los elementos que se especificaron en los parámetros extra usando la función **`set_elim()`** y luego mostrará el conjunto resultante. Algunos ejemplos:

```
$ ./setload input/example-unsorted-005.in -1
Set of elements read in the file: {-1, 0, 2, 3}

The file 'input/example-unsorted-005.in' has 4 unique elements
-1 [OK], 0 [OK], 2 [OK], 3 [OK]

Removings elements: <-1> DONE

Resulting set with 3 elements (not empty): {0, 2, 3}
```

```
$ ./setload input/example-unsorted-005.in 0
Set of elements read in the file: {-1, 0, 2, 3}

The file 'input/example-unsorted-005.in' has 4 unique elements
-1 [OK], 0 [OK], 2 [OK], 3 [OK]

Removings elements: <0> DONE

Resulting set with 3 elements (not empty): {-1, 2, 3}
```

```
$ ./setload input/example-unsorted-005.in 2
Set of elements read in the file: {-1, 0, 2, 3}

The file 'input/example-unsorted-005.in' has 4 unique elements
-1 [OK], 0 [OK], 2 [OK], 3 [OK]

Removings elements: <2> DONE

Resulting set with 3 elements (not empty): {-1, 0, 3}
```

```
$ ./setload input/example-unsorted-005.in 3
Set of elements read in the file: {-1, 0, 2, 3}

The file 'input/example-unsorted-005.in' has 4 unique elements
-1 [OK], 0 [OK], 2 [OK], 3 [OK]

Removings elements: <3> DONE

Resulting set with 3 elements (not empty): {-1, 0, 2}
```

```
$ ./setload input/example-unsorted-005.in 2 0 3 -1
Set of elements read in the file: {-1, 0, 2, 3}

The file 'input/example-unsorted-005.in' has 4 unique elements
-1 [OK], 0 [OK], 2 [OK], 3 [OK]

Removings elements: <2> <0> <3> <-1> DONE

Resulting set with 0 elements (empty): {}
```

Consideraciones:

- Solo se debe modificar el archivo **`set.c`**
- Una invariante de representación completa debería verificar la propiedad fundamental de la representación.
- Lo mínimo que se pide de la invariante es que verifique consistencia entre los campos de la estructura principal.
- La implementación debe verificar las pre y post condiciones especificadas en **`set.h`**.
- 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 ejecuta el programa cargando todos los archivos de input junto con los ejemplos mostrados donde se elimina elementos.
- Usando **`make valgrind`** es equivalente a **`make test`** con la diferencia que se ejecuta el programa 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 examen.**
- 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
- **No se deben modificar los .h**