Examen Final del Taller - Regulares

Algoritmos y Estructuras de Datos II

Tarea

El alumno deberá implementar el tipo abstracto de dato *Set* (ver especificación abajo) en el lenguaje de programación C, utilizando la técnica de ocultamiento de información visto en el taller de la materia.

Es requisito mínimo para aprobar lo siguiente:

- Implementar en C el TAD *set* (utilizando punteros a estructuras y manejo dinámico de memoria).
- Implementar en C una interfaz de línea de comando para que usuarios finales puedan usarlo (ver detalles más abajo).
- El programa resultado no debe tener *memory leaks* ni accesos inválidos a memoria, se chequeará tal condición usando `valgrind`.

Especificaciones

Se define el TAD *set*, que representa los conjuntos finitos de números enteros. Ejemplos:

$\{1, 3, 5, 6, 9, 11\}$
$\{23, 5, 3\}$
$\{\}$

El TAD deberá tener 2 constructores:

- uno para crear el conjunto vacío: $\{\}$
- otro para agregar un entero a un conjunto existente: 11 x $\{1, 3, 5, 6, 9\}$. Este último constructor tiene por argumento un entero a y un set H.

TAD conjunto
constructores
vacío : conjunto
agregar : entero $\times$ conjunto $\rightarrow$ conjunto
operaciones
es_vacío : conjunto $\rightarrow$ booleano
pertenece : entero $\times$ conjunto $\rightarrow$ booleano
cardinalidad : conjunto $\rightarrow$ natural
unión: conjunto $\times$ conjunto $\rightarrow$ conjunto
intersección : conjunto $\times$ conjunto $\rightarrow$ conjunto
ecuaciones
agregar(n,agregar(n,c)) = agregar(n,c)
agregar(n,agregar(m,c)) = agregar(m,agregar(n,c))

es_vacío(vacío) = verdadero
es_vacío(agregar(n,c)) = falso

pertenece(n,vacío) = falso

pertenece(n,agregar(n,c)) = verdadero
n $\neq$ m $\implies$ pertenece(n,agregar(m,c)) = pertenece(n,c)

cardinalidad(vacío) = 0
pertenece(n,c) $\implies$ cardinalidad(agregar(n,c)) = cardinalidad(c)
$\neg$ pertenece(n,c) $\implies$ cardinalidad(agregar(n,c)) = 1 + cardinalidad(c)

unión(vacío,d) = d
unión(agregar(n,c),d) = agregar(n,unión(c,d))

intersección(vacío,d) = vacío
pertenece(n,d) $\implies$ intersección(agregar(n,c),d) = agregar(n,intersección(c,d))
$\neg$ pertenece(n,d) $\implies$ intersección(agregar(n,c),d) = intersección(c,d)

Así, el conjunto $\{7, 2, 6, 0, 9, 3\}$, por ejemplo, puede construirse como:

c = agregar(3,agregar(7,agregar(2,agregar(0,agregar(9,agregar(6,vacío))))))

y de numerosas (infinitas) maneras más gracias a las primeras dos ecuaciones.

El objetivo del examen es implementar el TAD set con una lista enlazada ordenada sin repeticiones. Seguir el esquema dado en el teórico.

La implementación de la operación agregar debe modificar el conjunto que recibe como argumento, mientras que las de las operaciones unión e intersección deben dejar sus dos argumentos intactos y generar un nuevo conjunto que devuelven como resultado.

Para implementarlas puede convenirte implementar también una operación para clonar (es decir, generar una copia de) un conjunto.

Implementación

Los siguentes son los módulos a implementar:

- `set.h` y `set.c`, archivos de cabecera e implementación del TAD set.
- `main.c`, interfaz de línea de comandos.
- `Makefile`, para compilar el ejecutable.

Las siguiente tabla indica las funciones a implementar junto con las operaciones del respectivo TAD:

Signaturas de set_t	**Operación asociada**
`set_t set_empty()`	vacío
`set_t set_add(set_t set, int element)`	agregar
`bool set_is_empty(set_t set)`	es_vacío
`bool set_belongs(set_t set, int element)`	pertenece
`unsigned int set_size(set_t set)`	cardinalidad
`set_t set_union(set_t set1, set_t set2)`	unión
`set_t set_intersect(set_t set1, set_t set2)`	intersección
`set_t set_destroy(set_t set)`	destructor (no especificado)
`void set_dump(set_t set, FILE *fd)`	impresor a archivo (no especificado)

La interfaz de línea de comandos

En el archivo `main.c` se debe implementar la interfaz de línea de comando para manipular conjuntos. La interfaz tiene que ser capaz de mantener hasta 2 conjuntos a la vez para que el usuario final pueda hacer intersecciones y uniones de los mismos.

Cada opción del menú que se refiere a un conjunto, debe pedirle al usuario que elija si quiere operar sobre el conjunto 1 o el conjunto 2.

Esta interfaz debe proveer las siguientes operaciones:

```
c -> create a new set from a number (up to 2 sets total can be created)
e -> create a new empty set (up to 2 sets total can be created)
a -> add a number to a set
r -> empty (reset) set
p -> print set to stdout
s -> show set size
Choose an option:
```

A modo de ejemplo, se muestra a continuación la interacción con un usuario final esperada de la intefaz de línea de comandos para, por ejemplo, agregar un número al conjunto 2:

- Usuario elige opción `a`
- La interfaz le pide al usuario que elija si quiere agregar un elemento al conjunto 1 o 2
- El usuario elige 1 o 2
- La interfaz le pide al usuario que ingrese el elemento a agregar
- El usuario ingresa el número entero

El Makefile

Se deberá proveer un archivo `Makefile` que tenga dos reglas:

- La de compilación del ejecutable
- La de limpieza de los .o y otros archivos generados

Recordar

- Se debe resolver el ejercicio en tres horas (o menos).
- Se debe compilar pasando todos los flags usados en los proyectos.
- Comentar e indentar el código apropiadamente, siguiendo el estilo de código ya indicado por la cátedra (indentar con 4 espacios, no pasarse de las 80 columnas, inicializar todas las variables, etc).
- Todo el código tiene que usar la librería estándar de C, y no se puede usar extensiones GNU de la misma.
- El programa resultante **no** debe tener *memory leaks* **ni** accesos (read o write) inválidos a la memoria.