Nombre y Apellido: ..... N° Legajo: .....

Recuperatorio del Segundo Parcial de Programación Imperativa (72.31)

27/11/2018

	Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*	*Firma Docente*
Entregado				-----	
Calificación	2/	4/	4/		

- **<u>Condición mínima de aprobación: Sumar 5 puntos</u>**
- **Se tendrá en cuenta en la calificación el ESTILO y la EFICIENCIA de los algoritmos.**
- **Los ejercicios que no se ajusten estrictamente al enunciado, <u>no serán aceptados</u>.**
- **Puede entregarse en lápiz.**
- **No usar variables globales ni static.**
- **En caso de necesitar usar malloc o similar, no validar que retorne distinto de NULL.**
- **No es necesario escribir los #include.**
- **Escribir en cada hoja Apellido, Legajo, Número de hoja y Total de hojas entregadas.**
- **Realizar los ejercicios en hojas separadas.**

<u>Ejercicio 1:</u>

Dada la siguiente definición de lista de enteros

```
typedef struct node * TList;

typedef struct node {
     int elem;
     struct node * tail;
} TNode;
```

Una lista vacía se representa con el valor `NULL`.

Escribir una **función recursiva** que dada una lista de enteros que debería estar ordenada en forma ascendente, elimine de la misma los elementos que no cumplan con ese orden.

El primer elemento de la lista (si no está vacía) no cambia.

<u>Ejemplos:</u>

- si la lista tiene los elementos 1, 3, 4, 5, 8, 10 no cambia
- si la lista tiene los elementos 1, 1, 3, 2, 4, 5, 8, 10,1 pasa a ser 1, 3, 4, 5, 8,10
- si la lista tiene los elementos 12, 1, 3, 4, 5, 8, 10 pasa a ser la lista con el elemento 12

<u>No usar variables `static` ni funciones o macros auxiliares</u>

Ejercicio 2:

El siguiente es un TAD de "**multiSet**". Un multi-set es un conjunto de elementos sin orden pero donde cada elemento puede aparecer más de una vez.

multiSetADT.h

```
typedef struct multiSetCDT * multiSetADT;

typedef ... elemType;    // Tipo de elemento a insertar

/**
** Retorna 0 si los elementos son iguales, negativo si e1 es "menor" que e2 y positivo
** si e1 es "mayor" que e2
*/
static int compare (elemType e1, elemType e2) {
      ...
}

/* Retorna un nuevo multiSet de elementos genéricos. Al inicio está vacío */
multiSetADT newMultiSet();

/* Inserta un elemento. Retorna cuántas veces está elem en el conjunto
** luego de haberlo insertado (p.e. si es la primera inserción retorna 1).
*/
unsigned int add(multiSetADT multiSet, elemType elem);

/* Retorna cuántas veces aparece el elemento en el multiSet */
unsigned int count(const multiSetADT multiSet, elemType elem);

/* Retorna la cantidad de elementos distintos que hay en el multiSet */
unsigned int size(const multiSetADT multiSet);

/* Elimina una repetición del elemento. Retorna cuántas veces está elem el conjunto
** luego de haberlo borrado (si el elemento no estaba, retorna cero)
*/
int remove(multiSetADT multiSet, elemType elem);

/* Elimina todas las apariciones de un elemento. */
void removeAll(multiSetADT multiSet, elemType elem);

/* Retorna un vector con los elementos que menos aparecen en el conjunto
** Si el conjunto está vacío retorna NULL
** En el parámetro de salida dim almacena la cantidad de elementos del vector
*/
elemType * minElements(const multiSetADT multiSet, int * dim);
```

Implementar el TAD completo (el multiSetCDT.c).

Ejemplo

Si el multiset m fuera de enteros y se agregan los elementos 10, 10, 20, 10, 30, 40, 40, se obtendrían los siguientes resultados:

```
size(m)                  => retorna 4
removeAll(m, 60)         => no hace nada
remove(m, 10)            => retorna 2
remove(m, 50)            => retorna 0
remove(m, 40)            => retorna 1
minElements(m, &n)       => retorna un vector con los elementos 20,30,40
                         ( no necesariamente en ese orden), y n = 3
```

Ejercicio 3:

Se desea implementar un TAD para **listas de elementos no repetidos**, **que permita recorrerla con dos criterios: en forma ascendente o por el orden de inserción de los elementos**.

listADT.h

```
typedef struct listCDT * listADT;

typedef int elemType;    // Tipo de elemento a insertar, por defecto int
/**
** Retorna 0 si los elementos son iguales, negativo si e1 es "menor" que e2 y
** positivo si e1 es "mayor" que e2
*/
static int compare (elemType e1, elemType e2) {
      return e1 - e2;
}

/* Retorna una lista vacía.
*/
listADT newList();

/* Agrega un elemento. Si ya estaba no lo agrega */
void add(listADT list, elemType elem);

/* Elimina un elemento. */
void remove(listADT list, elemType elem);

/* Resetea el iterador que recorre la lista en el orden de inserción */
void toBegin(listADT list);

/* Retorna 1 si hay un elemento siguiente en el iterador que
** recorre la lista en el orden de inserción. Sino retorna 0 */
*/
int hasNext(listADT list);

/* Retorna el elemento siguiente del iterador que recorre la lista en el orden de inserción.
** Si no hay un elemento siguiente o no se invocó a toBegin aborta la ejecución.
*/
elemType next(listADT list);

/* Resetea el iterador que recorre la lista en forma ascendente */
void toBeginAsc(listADT list);

/* Retorna 1 si hay un elemento siguiente en el iterador que
** recorre la lista en forma ascendente. Sino retorna 0 */
*/
int hasNextAsc(listADT list);

/* Retorna el elemento siguiente del iterador que recorre la lista en forma ascendente.
** Si no hay un elemento siguiente o no se invocó a toBeginAsc aborta la ejecución.
*/
elemType nextAsc(listADT list);
```

Implementar todas las funciones del TAD, excepto la función `remove`, teniendo en cuenta que las funciones `toBeginAsc`, `hasNextAsc`, `nextAsc`, `toBegin`, `hasNext` y `next` deben ser O(1), es decir, ninguna de ellas debe recorrer la lista.

Ejemplo de uso:

```
#include "list.h"
int
main(void) {
   listADT c = newList(); // una lista, en este caso de int
   add(c, 3);   add(c, 1);   add(c, 5);   add(c, 2);
   toBegin(c);          // iterador por orden de inserción
   int n = next(c);     // n = 3
   n = next(c);         // n = 1
   toBeginAsc(c);       // iterador por orden ascendente
   n = nextAsc(c);      // n = 1
   n = next(c);         // n = 5
   n = next(c);         // n = 2
   hasNext(c);          // retorna 0 ( falso )
   n = nextAsc(c);      // n = 2
   hasNextAsc(c);       // retorna 1 ( true )
   n = nextAsc(c);      // n = 3
   return 0;
}
```