Programación II. Examen final. 22 de mayo 2017. Parcial 1.

Apellidos (en mayúsculas): _____ Nombre: _____

Ejercicio 1 (2,5 puntos). Se desea crear una aplicación para gestionar canciones de música. Cada canción tiene un título, un nombre y una dirección IP asociad. Además, se desea re-utilizar el código de la librería "colección" que gestiona una colección de objetos. Se muestra el fichero colección.h de la librería.

a) ¿Cuantos TAD gestionará la aplicación? ¿Qué ficheros se deberán crear, modificar y/o lincar? ¿Qué hay en cada uno de ellos? Justifique brevemente su respuesta.
b) Proporcione el código c de una función main (con control de errores) que gestione correctamente la memoria y realice las siguientes acciones:
Crea una colección, crea una canción con el título "19 días y 500 noches" del autor "Joaquín Sabina" e IP "https://youtu.be/hx8LrtN", inserta la canción en la colección, extrae la canción de la colección, imprime la información asociada a la canción en la salida estándar y libera los recursos.
c) Proporcione el prototipo de las primitivas que haya utilizado en el ejercicio anterior (y que no pertenezcan a la interfaz de la colección).
d) Proporcione una posible estructura `struct _COLECCIÖN` en C que utilice memoria contigua para almacenar las referencias a los objetos de la colección compatible con el fichero colección.h.
e) Proporcione el código C de la primitiva `coleccion_extract` cuando se utiliza la estructura de datos del apartado anterior. Para simplificar el código suponga que dispone de una función privada de prototipo `int coleccion_get(const COLECCION* col, const void *obj)` que devuelve el índice de la posición del objeto almacenado en la colección. En caso de no estar el objeto en la colección devuelve -1.

```
#ifndef COLECCION_H
#define COLECCION_H
#include "tipos.h"

typedef struct _COLECCION COLECCION;

/* ------ Tipos de los punteros a función soportados por la coleccion -----*/
typedef void (*destroy_object_function_type) (void*);
typedef void *(*copy_object_function_type) (const void*);
typedef int (*print_object_function_type) (FILE *, const void*);
typedef int (*cmp_object_function_type)(const void*, const void*);

/* ------------- Primitivas ---------------*/
/* Inicializa la coleccion reservando memoria */
COLECCION* coleccion_ini (destroy_object_function_type f1, copy_object_function_type f2,
print_object_function_type f3, cmp_object_function_type f4);
/* Libera la coleccion y todos sus objetos */
void coleccion_free (COLECCION* list);
/* Comprueba si una colección está vacía */
Bool coleccion_isEmpty(const COLECCION* col);
/* Comprueba si una colección está llena */
Bool coleccion_isFull(const COLECCION* col);
/* Devuelve el tamaño de una colección. */
int coleccion_size(const COLECCION* col);
/*Extrae un objeto. Devuelve FALSE si el objeto no se encuentra en collección*/
Bool coleccion_extract(COLECCION *col, void *pObj);
/* Inserta un objeto en la colección realizando una copia del objeto*/
COLECCION* coleccion_insert(COLECCION *col, const void *pObj);
#endif  /* COLECCION_H */
```

Ejercicio 2 (2.5 punto)

1. Traduce la siguiente expresión a posfijo aplicando el algoritmo correspondiente estudiado en la asignatura. Dibuja la evolución del algoritmo sobre la expresión paso a paso.

```
2 * ( 5 – 3 / 4 ) + 1 ;
```

2. Evalúa la siguiente expresión posfijo aplicando el algoritmo correspondiente estudiado en la asignatura. Dibuja la evolución del algoritmo sobre la expresión paso a paso.

```
1 3 + 2 * 9 1 – – ;
```

Ejercicio 3 (2.5 puntos). Escribe el pseudocódigo de una función derivada que, dada una pila de elementos de tipo entero, devuelva la media (entera) de los elementos de la misma, dejando la pila como se encontraba inicialmente.

Asumir que en el TAD se han definido las siguientes primitivas:

```
Pila pila_crear();
void pila_liberar(Pila pila);
status pila_push(Pila pila, entero elemento);
entero pila_pop(Pila pila);
boolean pila_vacia(Pila pila);
```

Puedes usar las funciones:
entero suma (entero e1, entero e2);
entero divide (entero e1, entero e2);

a) Escribe una primera versión sin control de errores

b) Marca en el código anterior qué puntos serían susceptibles de error y escribe, para cada uno de ellos, la versión con control de errores (no hace falta copiar todo el pseudocódigo de nuevo, solo lo que cambia).

Ejercicio 4 (2.5 puntos). Considera las siguientes primitivas de una implementación de colas en C:

```
Cola *cola_crear();

void cola_liberar(Cola *pq);

boolean cola_vacia(const Cola *pq);

boolean cola_llena(const Cola *pq);

status cola_insertar(Cola *pq, const Elemento *pe);

Elemento *cola_extraer(Cola *pq);

int cola_tamanyo(Cola *pq); // devuelve el tamaño de la cola
```

Supongamos que el TAD Elemento tiene, entre otras, la primitiva:

```
double elemento_valor(Elemento *pe); // devuelve el valor numérico del elemento
```

a. Dar el código C de una **función derivada** de prototipo

   ```
   boolean cola_ordenada(Cola *pq)
   ```

   que chequea si los elementos de la cola están dispuestos de manera ordenada de menor a mayor valor. Si la cola está vacía se devolverá TRUE. Se deberán controlar todos los posibles errores que puedan surgir, devolviendo en esas situaciones FALSE. En cualquier caso, el contenido de la cola pasada como argumento no debe cambiar, tanto si se devuelve FALSE como si no.

b. Dar ahora el código C de otra versión de la anterior función que ahora sea una **primitiva básica**, teniendo en cuenta la siguiente estructura de datos para implementar el TAD Cola:

   ```
   struct _Cola {
        Elemento *datos[COLA_MAX];
        int front, rear;
   };
   ```

c. Discuta las ventajas e inconvenientes de las dos implementaciones (como función derivada / como función primitiva).