Programación II. Parcial 1. 14 de Marzo 2017

Apellidos: _____Nombre:_____

Ejercicio 1 (3 puntos en total). Una empresa que se dedica a organizar celebraciones (bodas, bautizos, comuniones) quiere desarrollar un software para gestionar los eventos. Nuestro cometido es diseñar los TAD sobre los que se va a construir la herramienta. Para ello, tenemos que tener en cuenta que:

- La empresa tiene un edificio de 10 plantas, cada una de las cuales puede estar vacía o acoger una celebración. Por eficiencia, nunca va a haber plantas intermedias que no acojan celebración (es decir, si hay 3 celebraciones, serán en las 3 primeras plantas).
- Cada planta tiene asociada una descripción, por ejemplo "Comunión de Luis Pérez" (longitud máxima de 50 caracteres), y acoge un número máximo de 30 mesas.
- Cada mesa tiene asociada una cadena que la describe, por ejemplo "hermanos de la novia + parejas" (longitud máxima de 30 letras), y tiene capacidad para 10 personas como mucho.
- Cada persona vendrá descrita por su nombre (máximo 20 letras), una cadena con su relación con la celebración (por ejemplo, "tía del novio"), con un máximo de 20 letras también, y su DNI (sin letra).

a) **(1 punto)** ¿Cuántos Tipos Abstractos de Datos necesitas para representar lo descrito anteriormente, y qué nombres les darías? Los tipos Status y Boolean se encuentran ya definidos (no se consideran aquí)

Nº TADs: ___ Nombres TADs: _____

Escribe el código en C de las Estructuras de Datos (EdD) necesarias para implementar estos TADs y di en qué ficheros estarían esos códigos.

b) **(1 punto)** Escribe el código C con control de errores de la primitiva:

Status imprimeRelaciónPersonaMesa(Mesa *pm, long int DNI, FILE *f)

que imprime en el fichero la relación que tiene con la celebración la persona con el DNI dado que está en la mesa apuntada por pm. ¿En qué TAD estaría implementada esta primitiva?

¿Qué primitivas necesitas usar para su implementación y a qué TADs pertenecen? No hace falta que implementes estas últimas, sólo dar sus prototipos.

c) **(1 punto)** Escribe el código C con control de errores de la primitiva:

int npersonasPlanta (Planta *p)

que devuelve el número total de personas que hay en una planta. ¿En qué TAD estaría implementada esta primitiva?

¿Qué primitivas necesitas usar para su implementación y a qué TADs pertenecen? No hace falta que implementes estas últimas, sólo dar sus prototipos.

Programación II. Primer examen parcial. Marzo 2017

Apellidos: _____Nombre:_____

Ejercicio 2 (2 puntos).

a) **(1 punto)** Traduce la siguiente expresión a sufijo (posfijo) utilizando el algoritmo de traducción estudiado en clase. Dibuja la evolución del algoritmo sobre la expresión paso a paso:

A * (B + C – D / F) + G ;

b) **(1 punto)** Traduce la siguiente expresión a prefijo, aplicando el algoritmo correspondiente y mostrando su evolución paso a paso:

A B C + D / E F - * + ;

Programación II. Examen parcial de evaluación continua. Marzo 2017

Apellidos: _____ Nombre: _____ Grupo: ____

Ejercicio 3 (2.5 puntos). XML es un lenguaje de marcas que se utiliza para almacenar datos de forma estructurada y legible para una persona.

En XML los datos se componen de elementos explícitamente identificados y delimitados, pudiendo cada uno de esos elementos estar compuesto a su vez de otros elementos. Por ejemplo, los datos almacenados sobre un libro podrían ser su título y autor, y los datos almacenados sobre un autor podrían ser su nombre y sus apellidos.

Cada una de esas partes se delimita mediante dos etiquetas, una de apertura/inicio de elemento `<tag>` y otra de cierre/fin `</tag>`, donde `tag` se debe sustituir por el nombre de etiqueta que se elija, p.e. `<libro>` y `</libro>`, `<titulo>` y `</titulo>`. Así, entre una etiqueta de apertura y la correspondiente de cierre puede haber o bien otro(s) par(es) de etiquetas diferentes anidadas o bien valores literales, p.e. `Hamlet`, `William`, `Shakespeare`.
Considérese el siguiente ejemplo de fichero XML:

```
<libro>
    <titulo>Hamlet</titulo>
    <autor>
      <nombre>William</nombre>
      <apellidos>Shakespeare</apellidos>
    </autor>
</libro>
```

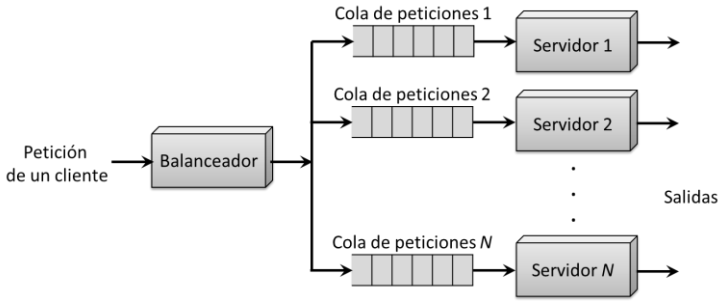
Asúmase que existe una rutina que procesa el fichero como una cadena de "tokens" (etiquetas o valores) extrayendo iterativamente token a token. Para el ejemplo anterior: `<libro>`, `<titulo>`, `Hamlet`, `</titulo>`, `<autor>`, etc.
Usando esa rutina, se desea desarrollar un programa que valide la sintaxis de un fichero XML comprobando la corrección de las etiquetas de apertura y cierre, y teniendo en cuenta la anidación de pares de etiquetas.

a) ¿Qué TAD y algoritmo de los estudiados en la asignatura se podrían aplicar para tal programa? Razona brevemente la respuesta.
b) Indica gráficamente, token a token, el estado del TAD y evolución del algoritmo anteriores sobre el fichero XML de arriba.
a) ¿Qué casos de sintaxis errónea, atendiendo a apertura-cierre de etiquetas, pueden darse. Propón un ejemplo de cada uno de esos errores con el fichero XML de arriba. ¿Cómo se detectarían esos casos de error con el TAD y algoritmo usados?

Programación II. Primer examen parcial. Marzo 2017

Apellidos: _____Nombre:_____

Ejercicio 4. (2.5 puntos). Un **balanceador de carga** es un dispositivo software o hardware que se sitúa antes de un conjunto de servidores que atienden peticiones de una aplicación, y asigna o balancea las peticiones que llegan de los clientes a los servidores usando algún algoritmo de asignación. Como se muestra en la figura de abajo, una situación típica es que cada servidor tiene una cola donde almacena las peticiones que le llegan para procesarlas según va teniendo disponibilidad.



Suponer que se dispone de las siguientes estructuras de datos para almacenar los datos de los servidores y sus colas.

```
typedef struct {
    Cola *colaPeticiones;
    char *direccion;
} Servidor;
typedef struct {
    Servidor *servidores;
    int numServidores;
    int servidorActual;
} Balanceador;
```

Al llegar una petición al balanceador, éste comprobará las colas de los servidores según el algoritmo de asignación implementado. Suponer que el algoritmo de asignación es el siguiente.

- El balanceador tiene un índice "servidorActual" que indica cuál de sus servidores (0, 1, 2,..., numServidores-1) es el siguiente cuya disponibilidad hay que comprobar.
- En caso de que la cola de ese servidor no esté llena, la petición se almacena en esa cola y "servidorActual" se incrementa adecuadamente.
- En caso contrario, se prueba con el siguiente en el array "servidores" incrementando "servidorActual" adecuadamente, y así sucesivamente.
- Si todos los servidores tienen sus colas llenas, se rechaza la petición.

Con todo lo anterior, se pide implementar la función:

```
status balanceador_asignarPeticion(Balanceador *balanceador, Peticion *peticion);
```

que inserta la petición de entrada en la cola del servidor actual o, en caso de que esté llena, en la cola del siguiente servidor gestionado por el balanceador que tenga hueco (de haber alguno, tal y como se ha explicado en la descripción anterior).

La función ha de tener los siguientes valores de retorno:

- `OK`, si la petición se pudo insertar en una de las colas de los servidores
- `ERROR_ARGUMENTOS`, si alguno de los argumentos de entrada es incorrecto
- `ERROR_SOBRECARGA`, si las colas de todos los servidores están llenas
- `ERROR_INTERNO`, si se produjo algún error durante la ejecución

Asumir que existen las primitivas de los TAD Cola y Elemento vistas en clase (sigue por detrás):

```
Cola *cola_crear();
void cola_liberar(Cola *pq);
Bool cola_vacia(const Cola *pq);
Bool cola_llena(const Cola *pq);
Status cola_insertar(Cola *pq, const Elemento *pe);
Elemento *cola_extraer(Cola *pq);
void elemento_liberar(Elemento *);
Elemento *elemento_copiar(const Elemento *);
```

y la siguiente función de creación de un elemento de cola a partir de una petición:

```
Elemento *elemento_crear(Peticion *);
```