UNIVERSIDAD DE MÁLAGA | uma.es

Dpto. de Lenguajes y Ciencias de la Computación

Programación de Sistemas y Concurrencia

3 de septiembre de 2019

APELLIDOS_____ NOMBRE_____

DNI _____ ORDENADOR _____ GRUPO/TITULACIÓN_____

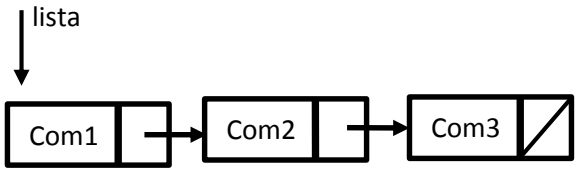
Programación C (3,5 pts.)

El objetivo del ejercicio es simular la gestión en memoria de los componentes que se almacenan en un almacén de reparto. Para ello, los componentes se almacenarán en una lista enlazada, tal y como aparece en el dibujo de abajo. La lista de componentes es un puntero a una estructura definida como sigue:

```
typedef struct elemLista {
        long codigoComponente;
        char textoFabricante[MAX_CADENA];
        struct elemLista * sig;
} Componente;

typedef Componente * Lista;
```

Observa que, en el dibujo, el puntero externo a la estructura apunta al primer nodo de la lista. En esta implementación, una lista vacía es un puntero a NULL, y en una lista con un único nodo, su campo siguiente apunta a NULL.



El ejercicio consiste en implementar el módulo componentes.h con las siguientes operaciones que se listan a continuación.

```
/*(0,5 puntos) La funcion Lista_Crear crea y devuelve una lista enlazada vacía de nodos de tipo Componente.*/
Lista Lista_Crear();

/*(1 punto) La rutina Adquirir_Componente se encarga de recibir los datos de un nuevo componente (codigo y texto)
que se introducen por teclado y devolverlos por los parametros pasados por referencia "codigo" y "texto". */
void Adquirir_Componente(long * codigo,char * texto);

/*(1 punto) La funcion Lista_Agregar toma como parametro un puntero a una lista,
el codigo y el texto de un componente y  anyade un nodo al final de la lista con
estos datos. */
void  Lista_Agregar(Lista *lista, long codigo, char* textoFabricante);

/*(1 punto) La funcion Lista_Imprimir se encarga de imprimir por pantalla la lista enlazada completa que se le pasa
como parametro.*/
void  Lista_Imprimir(Lista lista);
```

```
/*(0,5 puntos) La rutina Lista_Vacia devuelve 1 si la lista que se le pasa como parametro esta vacia y 0 si no lo esta.*/
int  Lista_Vacia(Lista lista);

/*(1 punto) Num_Elementos es una funcion a la que se le pasa una lista y devuelve el numero de elementos de dicha lista.*/
int  Num_Elementos(Lista lista);

/*(2 puntos) Lista_Extraer toma como parametro un puntero a una Lista y elimina el componente que se encuentra en su ultima posicion.*/
void  Lista_Extraer(Lista *lista);

/*(1 punto) Lista_Vaciar es una funcion que toma como parametro un puntero a una Lista y extrae todos sus Componentes.*/
void  Lista_Vaciar(Lista *lista);

/*(2 puntos) La funcion Lista_Salvar se encarga de guardar en el fichero binario
"examen.dat" la lista enlazada completa que se le pasa como parametro.
Para cada nodo de la lista, debe almacenarse en el fichero el código y el texto de la componente correspondiente.*/
void  Lista_Salvar(Lista lista);
```

Observa que para poder leer posteriormente el contenido del fichero binario “examen.dat” es necesario que se almacene la longitud de la cadena de texto.

INSTRUCCIONES: descarga el fichero componentes.h y Driver.c del campus virtual. En Eclipse, crea un nuevo proyecto de C, y copia ambos ficheros al proyecto.

Anexo.
Los prototipos de las funciones de escritura de caracteres en ficheros de la biblioteca `<stdio.h>` son los siguientes (se dan por conocidos los prototipos de las funciones de `<stdlib.h>` que necesites, como `free` o `malloc`):

`FILE *fopen(const char *path, const char *mode);`
Abre el fichero especificado en el modo indicado ("rb" para lectura binaria y "wb" para escritura binaria). Devuelve un puntero al manejador del fichero en caso de éxito y NULL en caso de error.

`unsigned fread(void *ptr, unsigned size, unsigned nmemb, FILE *stream);`
Lee `nmemb` elementos de datos, cada uno de tamaño `size`, desde el fichero `stream`, y los almacena en la dirección apuntada por ptr. Devuelve el número de elementos leídos.

`unsigned fwrite(const void *ptr, unsigned size, unsigned nmemb, FILE *stream);`
Escribe `nmemb` elementos de datos, cada uno de tamaño `size`, al fichero `stream`, obteniéndolos desde la dirección apuntada por `ptr`. Devuelve el número de elementos escritos.

`int fclose(FILE *fp);`
Guarda el contenido del buffer y cierra el fichero especificado. Devuelve 0 en caso de éxito y -1 en caso de error.

Los prototipos de las funciones para manipular strings (están en <string.h>) son

`char* strcpy(char* s1, char*s2);`
Copia los caracteres de la cadena s2 (hasta el carácter ‘\0’, incluido) en la cadena s1. El valor devuelto es la cadena s1.
`size_t strlen(const char *s);`
Calcula el número de caracteres de la cadena apuntada por s.

Bloque de Concurrencia

El objetivo de este ejercicio es modelar un concurso entre **dos** jugadores para ver quien coge antes las vacaciones. El jugador ganador será el primero que gane **tres** juegos. Cada juego consiste en que cada jugador tira 10 veces una moneda del modo siguiente:

- Cada juego termina cuando los dos jugadores han tirado 10 veces su moneda.
- Si los dos jugadores sacan el mismo número de caras, el juego queda en empate.
- Si un jugador saca más caras que el otro, entonces es el ganador del juego.
- El concurso termina cuando uno de los dos jugadores gana **tres** juegos.

Resuelve este ejercicio utilizando semáforos binarios (3,25 pts), y Monitores/Locks (3,25 pts.). En el cv, puedes encontrar la plantilla del ejercicio, que tiene la siguiente estructura:

- Una clase Driver que crea el recurso compartido (el Concurso) y los dos jugadores.
- Una clase Jugador que hereda de Thread y simula el comportamiento de los jugadores en su método run().
- Una clase Concurso que representa el concurso, que es el recurso compartido. En esta clase hay que implementar las siguientes funciones:
  - **public void** tirarMoneda(**int** id,Boolean cara). El jugador id utiliza esta función para indicar si le ha salida cara o no durante la realización de un juego. El jugador que deja la última tirada del juego muestra por la pantalla quien ha sido el jugador ganador del juego, o de la partida, si ya ha habido un jugador que ha ganado tres juegos.
    Nota: Si un jugador tiene identificador id (0 o 1), el otro tiene identificador 1-id (1 o 0). Observa que en este método si un jugador ya ha tirado 10 veces su moneda, pero el otro no ha terminado, tiene que esperar a que el otro jugador termine la jugada, para volver a jugar, si ese es el caso.
  - **public boolean** concursoTerminado(). Devuelve true sii el concurso ha terminado, es decir, uno de los dos jugadores ha ganado tres juegos.

Los dos jugadores tiran 10 veces la moneda (hay 20 tiradas de moneda) -> el jugador que tira la última moneda decide si hay un ganador del juego y si la partida ha terminado. Si la partida no ha terminado, el proceso comienza de nuevo. Por ejemplo, una ejecución posible del programa sería:

```
Juego 1: Ha ganado la partida el jugador 1 con 6 caras
Juego 2: Ha ganado la partida el jugador 1 con 7 caras
Juego 3: Ha ganado la partida el jugador 0 con 7 caras
Juego 4: Ha ganado la partida el jugador 0 con 6 caras
Juego 5: El juego ha empatado
Juego 6: Ha ganado la partida el jugador 0 con 8 caras
Final del concurso. Ha ganado el jugador 0
```

Notas:

1. La única documentación que se puede utilizar es el API de Java disponible en Eclipse.
2. Hay que utilizar los ficheros fuentes/proyecto que se proporcionan.
3. Hay que subir al campus virtual (del grupo) la implementación de la clase Concurso.