UNIVERSIDAD DE MÁLAGA | uma.es

Programación de Sistemas y Concurrencia

Dpto. de Lenguajes y Ciencias de la Computación

Examen 1ª Convocatoria Ordinaria
Curso 2019-2020

APELLIDOS_____NOMBRE_____

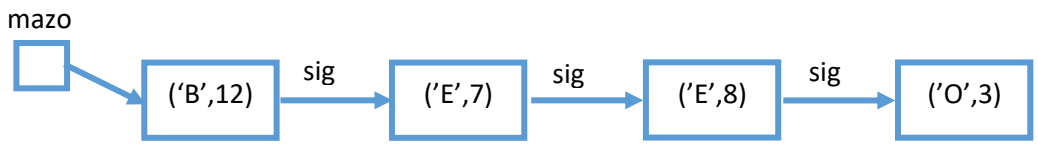
DNI _____ ORDENADOR _____GRUPO/TITULACIÓN_____

Bloque 1 – Programación de Sistemas

Este ejercicio consiste en implementar los módulos necesarios para que dos personas jueguen a las cartas. En el juego, existe un mazo de cartas y cada jugador tiene una mano, es decir, un conjunto de cartas con las que juega. El juego consiste en que cada jugador descarta todas las cartas que puede de su mano, insertándolas en el mazo de una forma determinada que se explica más adelante. Cuando los dos jugadores han terminado de descartar sus cartas, se suma el valor de las cartas que siguen teniendo en la mano, y gana el jugador cuya mano tiene menor valor.

Para poder descartar una carta de la mano del jugador, en el mazo tiene que existir una carta con el mismo palo y el valor inmediatamente superior o inferior. Por ejemplo, para que un jugador pueda descartar el 6 de copas de su mano, en el mazo tiene que estar el 5 de copas o el 7 de copas.

Una carta se representa por su palo (carácter 'B', 'C', 'E', 'O') y un valor (un número entre 1 y 12). El mazo y la mano de los dos jugadores se almacenan en tres listas enlazadas diferentes. Cada una de estas listas estará ordenada primero por el palo de manera ascendente ('B' < 'C' < 'E' < 'O') y, para las cartas de un mismo palo, ordenadas por el valor también de forma ascendente, como se muestra en la figura:



Para la implementación, vamos a utilizar la siguiente definición de tipos que se encuentra en el fichero Lista.h:

```
struct TCarta{
        char palo; //'B', 'C', 'E', 'O'
        int valor; //entre 1 y 12
};
typedef struct TCarta TCarta;

typedef struct TNodo *TLista;
struct TNodo{
        TCarta carta;
        TLista sig;
};
```

También se proporciona un fichero `Principal.c` con un programa principal que permite probar los diferentes métodos que se piden a continuación.

Parte 1 (obligatoria para aprobar)

Implementar los siguientes métodos en el módulo `Lista.c`:

```
/* Crea una lista vacía */
void crear(TLista *lista);

/** Muestra el contenido de la lista.
 *   - Si la lista está vacía muestra el mensaje "Lista vacia..."
 *   - Si la lista no está vacía la muestra en el formato:
 *       <valor>:<palo> <valor>:<palo> <valor>:<palo>
 */
void mostrar(TLista lista);

/* Inserta la carta en la lista ordenada primero por el palo y luego,
 * para las cartas del mismo palo, por el valor.
 * En ambos casos en orden ascendente.
 *
 * Podemos suponer que la carta a insertar no está en la lista.
 */
void insertarOrdenado(TLista *lista, TCarta carta);


/* Elimina toda la memoria dinámica reservada para la lista. */
void destruir(TLista *lista);
```

Parte 2 (opcional – permite llegar al notable)

Implementar los siguientes métodos en el módulo `Lista.c`

```
/* Borra una carta de la lista, teniendo en cuenta que la lista está ordenada.
 * Se puede suponer que la carta a borrar estará en la lista.
 */
void borrar(TLista *lista, TCarta carta);

/* Descarta la primera carta de la lista2 que pueda ser insertada en la lista1
 * siguiendo el siguiente criterio:
 *
 * Para poder descartar la carta (palo, valor) de la lista2 se hará lo siguiente:
 *   - se comprueba si en la lista1 existe una carta con el mismo palo y un valor
 *     inmediatamente anterior o posterior. Es decir, la carta (palo, valor-1) o (palo, valor + 1)
 *   - si existe, la carta se elimina de lista2 y se inserta en lista1, de forma que lista1
       siga estando ordenada.
 *
 *   La función devuelve 1 si el jugador ha podido descartarse una carta y 0 en otro caso.
 *   En cada llamada a esta función el jugador se descartará una sola carta, aunque haya más
 *   cartas descartables en la mano del jugador.
 *   Se puede suponer que las cartas de lista2 no están en lista1.
 */
int descartar(TLista * lista1, TLista *lista2);

/* Suma y devuelve el valor de todas las cartas de la lista,
 * independientemente de su palo.
```

```
 */
int sumar(TLista lista);
```

Parte 3 (opcional – permite llegar a sobresaliente)

Implementar los siguientes métodos en el fichero `Principal.c`

```
/* Dado un fichero con descriptor f, ya abierto en modo texto, lee una línea de ese fichero
 * y guarda las cartas en la lista que se pasa como parámetro.
 *
 * El formato de cada línea es: <num_cartas> <carta1> <carta2> ... <cartan>
 *
 * donde el primer valor es el número de cartas que se almacenarán en la lista.
 * Este valor no se almacena en la lista.
 *
 * Cada carta se representa por el valor de la carta, un espacio y
 * la letra 'B', 'C', 'E' o 'O', representando el palo.
 *
 * Ejemplo de línea con 5 cartas:      5 2 C 1 E 1 O 6 B 4 B
 *
 * Para el ejemplo, se crearía una lista con las cartas 2:C 1:E 1:O 6:B y 4:B
 *
 * Debe suponerse que la lista se ha creado previamente. Los datos se guardarán en lista
 * de forma ordenada, primero por el palo y luego, para el mismo palo, por el valor.
 * En los dos casos en orden ascendente.
 */
void leerLinea(FILE *f, TLista *lista);


/* Dado el nombre de un fichero de texto que contiene tres líneas, cada una
 * con el formato descrito anteriormente:  <num_cartas> <carta1> <carta2> ... <cartan>
 *
 * Leer la información del fichero y generar tres listas de cartas del siguiente modo:
 * El contenido de la primera línea se almacena en mazo, el de la segunda en
 * jugador1 y el de la tercera en jugador2.
 *
 * Debe suponerse que las listas se han creado previamente. Las cartas se almacenarán
 * en las listas correspondientes de forma ordenada, primero por el palo y luego,
 * para el mismo palo, por el valor.
 */
void crearDesdeFichero(char *nombre, TLista *mazo, TLista *jugador1, TLista *jugador2);
```

Anexo. Los prototipos de las funciones de lectura y escritura 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"/ y "wb" para lectura/escritura binaria y "rt"/"wt" para lectura/escritura de texto). Devuelve un puntero al manejador del fichero en caso de éxito y NULL en caso de error.

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.

LECTURA / ESCRITURA BINARIA

unsigned fread(void *ptr, unsigned size, unsigned nmemb, FILE *stream): Lee nmemb elementos de datos, cada uno de tamaño size bytes, 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.
LECTURA/ESCRITURA TEXTO

int fscanf(FILE *stream, const char *format, ...): Lee del fichero stream los datos con el formato especificado en el parámetro format, el resto de parámetros son las variables en las que se almacenan los datos leídos en el formato correspondiente. La función devuelve el número de variables que se han leído con éxito.

int fprintf(FILE *stream, const char *format, …): Escribe en el fichero stream los datos con el formato especificado en el parámetro format. El resto de parámetros son las variables en las que se almacenan los datos que hay que escribir. La función devuelve el número de variables que se han escrito con éxito.