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 _____

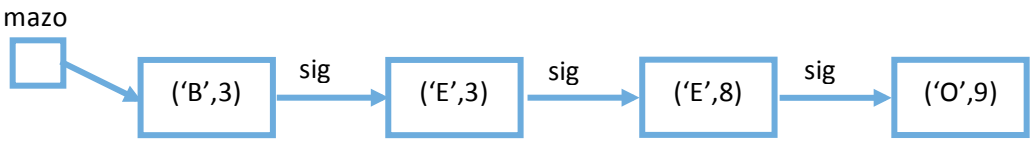
DNI _____ ORDENADOR _____ GRUPO/TITULACIÓN _____

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 cada turno cada jugador se descarta de todas las carta que puede de su mano, insertándolas en el mazo de una forma determinada que se explica más adelante. Cuando los dos jugadores no pueden descartar más 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 valor y palo diferente. Por ejemplo, para que un jugador pueda descartar el 6 de copas de su mano, en el mazo tiene que estar el 6 de oros, o el 6 de espadas o el 6 de bastos.

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 valor de forma ascendente, y para un mismo valor, se ordenan por palo de forma ascendente ('B' < 'C' < 'E' < 'O'), 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 es vacía muestra el mensaje "Lista vacía..."
 *   - 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 valor de forma
 * ascendente y para el mismo valor, por el palo 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 la 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 descartar una carta de la lista2 se hará lo siguiente:
 *   - se comprueba si en la lista1 existe una carta con el mismo valor y
 *     diferente palo.
 *   - 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 de alguna carta
 * y 0 en otro caso.
 *
 * 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` para crear el mazo y las manos de los jugadores a partir de un fichero binario con el siguiente formato para cada una de las listas:

<num_total> <num_B> <valorB1>...<valorBn> <num_C><valorC1>...<valorCm> <num_E> <valorE1>...<valorEx> <num_O> <valorO1>...<valorOy>

En el fichero aparecerán primero las cartas de bastos, luego las de copas, luego las de espadas y por último las de oros. Para cada palo hay primero un número con la cantidad de cartas que hay de ese palo. A continuación se mostrará una lista de números indicando los valores de esas cartas. Si no hay ninguna carta de un palo determinado en ese caso habrá un 0 en el fichero en el lugar donde irían las cartas de ese palo.

Por ejemplo, si la lista tiene las 4 cartas 7:B 7:E 8:E 3:O

En el fichero binario aparecerán los números 4 1 7 0 2 7 8 1 3, donde 4 significa que tenemos 4 cartas en la lista, el 1 que tenemos una carta con palo 'B', el 7 simboliza el 7:B, el 0 indica que no hay cartas con del palo 'C', el 2 indica que hay 2 cartas del palo 'C' que son el 7:E y el 8:E y finalmente el 1 indica que hay una carta de palo 'O' que es el 3:O.

```
/* Dado un fichero f, ya abierto en modo binario, leer una línea de ese
 * fichero y guardar las cartas en la lista que se pasa como parámetro.
 *
 * El formato de cada línea es:
 * <num_total> <num_B> <valorB1>...<valorBn> <num_C>...<num_E>...<num_O>
 * <valorO1>...<valorOn>
 *
 * Ejemplo de lista con las 4 cartas 7:B 7:E 8:E 3:O
 *  4 1 7 0 2 7 8 1 3
 * Debe suponerse que la lista se ha creado previamente. Los datos se
 * guardaran en la lista de forma ordenada, primero por el valor ascendente y
 * luego, para el mismo valor, por el palo ascendente también.
 * En los dos casos el orden es ascendente.
 */
void leerLinea(FILE *f, TLista *lista);

/* Dado el nombre de un fichero binario (parámetro nombre) que contiene
 * tres líneas con cartas, lee del fichero esas tres líneas. El contenido
 * de la primera línea lo guarda en mazo, el de la segunda en jugador1 y
 * el de la tercera en jugador2.
 *
 * El formato de cada línea es:
 * <num_total> <num_B> <valorB1>...<valorBn> <num_C>...<num_E>...<num_O>
 * <valorO1>...<valorOn>
 *
 * Ejemplo de lista con las 4 cartas 7:B 7:E 8:E 3:O
 *  4 1 7 0 2 7 8 1 3
 *
 * Las líneas se almacenarán en las listas correspondiente de forma ordenada,
 * primero por el valor de forma ascendente y luego por el palo de forma
ascendente también.
 */
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.