UNIVERSIDAD AUTÓNOMA DE MADRID
INGENIERÍA DE TELECOMUNICACIÓN
Estructura de Datos y Algoritmos (Curso 2007/2008)

EXAMEN DE PRÁCTICAS (Junio 2008)

1. Siguiendo el estilo de programación modular, escribir el cuerpo del siguiente procedimiento del **programa principal**, suponiendo que se han incluido los ficheros de cabecera de los TADs ModeloCoche y Lista indicados en el apéndice. No se debe hacer comprobación de errores. (*4 puntos*)

```
/* ========================================================= */
/* =                Imprimir_Euros_Siguiente               = */
/* ========================================================= */

/* Descripción: Imprime el valor en euros almacenado en el nodo de la lista que
    está a continuación del nodo dado. La información de la lista son punteros
    a nodos del tipo tModeloCoche.

   Entrada: Puntero a un objeto del tipo tNodoLista.

   Valor de retorno: ninguno.
*/

void Imprimir_Euros_Siguiente( tNodoLista *nodolista )
{
     tNodoLista *siguiente;
     tModeloCoche *modelococoche;
     float euros;

     siguiente = Lista_Siguiente( nodolista );
     modelococoche = (tModeloCoche*) Lista_Informacion( siguiente );
     ModeloCoche_Leer_Euros( modelococoche, &euros );

     printf("euros %f\n", euros);
}
```

2. Siguiendo el estilo de programación modular, escribir el cuerpo del siguiente procedimiento del **programa principal**, suponiendo que se han incluido los ficheros de cabecera de los TADs ModeloCoche y Lista indicados en el apéndice. No se debe hacer comprobación de errores. (*2 puntos*)

```
/* ========================================================= */
/* =                Liberar_Modelo_Coche                   = */
/* ========================================================= */

/* Descripción: Libera toda la memoria dinámica ocupada por el modelo
    de coche indicado.

   Entrada: Puntero a un objeto del tipo tModeloCoche que está almacenado en
            memoria dinámica.

   Valor de retorno: ninguno.
*/

void Liberar_Modelo_Coche( tModeloCoche *modelococoche )
{
     ModeloCoche_Suprimir( modelococoche );
     free( modelococoche );
}
```

3. Siguiendo el estilo de programación modular, escribir el cuerpo del siguiente procedimiento del **programa principal**, suponiendo que se han incluido los ficheros de cabecera de los TADs ABBModeloCoche, ModeloCoche y Lista indicados en el apéndice. No se debe hacer comprobación de errores. (4 *puntos*)

```
/* ======================================================== */
/* =                 Arbol_Contiene                       = */
/* ======================================================== */

/* Descripción: Devuelve si el árbol contiene algún tModeloCoche con el valor
    en euros indicado.

   Entrada:  Puntero a un objeto del tipo tABBModeloCoche
             Valor en euros buscado

   Valor de retorno:       0: no encontrado
                           Cualquier otro valor: Encontrado
*/

char Arbol_Contiene( tABBModeloCoche arbol, float valor )
{
     tLista lista=NULL;
     char result;

     Lista_Inicializar( &lista );

     ABBModeloCoche_Buscar_Valor( arbol, valor, &lista );
     result= !Lista_Vacia( &lista );

     VaciarLista( &lista, 0 );
     Lista_Finalizar( &lista );

     return result;
}
```

Se supone además que existe y que, por lo tanto, no hay que implementar, la función VaciarLista, con la siguiente descripción:

```
/* Descripción: Vacía la lista pasada por argumento, liberando además la
   memoria del campo info de los nodos cuando liberarInfo es verdadero
   (distinto de cero)

   Entrada:  Puntero a un objeto del tipo Lista
             Booleano que indica si se debe liberar el campo info

   Valor de retorno:       Ninguno
*/

void VaciarLista( tLista *lista, char liberarInfo );
```

APÉNDICE

ModeloCoche.h

```
typedef struct tModeloCoche
{
	char *marca;		// Marca
	char *modelo;		// Modelo
	int ano1, ano2;		// Rango de años de fabricación
	int cilindrada;		// Centímetros cúbicos
	int cilindros;		// Número de cilindros
	char combustible;	// (G)asolina o (D)iesel
	int CV1, CV2;		// Rango de potencias en Caballos de Vapor
	int kW1, kW2;		// Rango de potencias en kilowatios
	float CVf;		// Potencia en Caballos de Vapor fiscales
	float euros;		// Precio oficial en euros
} tModeloCoche;

int ModeloCoche_Inicializar( tModeloCoche *ModeloCoche );
int ModeloCoche_Suprimir( tModeloCoche *ModeloCoche );

int ModeloCoche_Guardar_Marca( tModeloCoche *ModeloCoche, char *marca );
int ModeloCoche_Guardar_Modelo( tModeloCoche *ModeloCoche, char *modelo );
int ModeloCoche_Guardar_Anos( tModeloCoche *ModeloCoche, int ano1, int ano2 );
int ModeloCoche_Guardar_Cilindrada( tModeloCoche *ModeloCoche, int cilindrada );
int ModeloCoche_Guardar_Cilindros( tModeloCoche *ModeloCoche, int cilindros );
int ModeloCoche_Guardar_Combustible( tModeloCoche *ModeloCoche, char combustible );
int ModeloCoche_Guardar_CVs( tModeloCoche *ModeloCoche, int CV1, int CV2 );
int ModeloCoche_Guardar_kWs( tModeloCoche *ModeloCoche, int kW1, int kW2 );
int ModeloCoche_Guardar_CVf( tModeloCoche *ModeloCoche, float CVf );
int ModeloCoche_Guardar_Euros( tModeloCoche *ModeloCoche, float euros );

int ModeloCoche_Leer_Marca( tModeloCoche *ModeloCoche, char* *marca );
int ModeloCoche_Leer_Modelo( tModeloCoche *ModeloCoche, char* *modelo );
int ModeloCoche_Leer_Anos( tModeloCoche *ModeloCoche, int *ano1, int *ano2 );
int ModeloCoche_Leer_Cilindrada( tModeloCoche *ModeloCoche, int *cilindrada );
int ModeloCoche_Leer_Cilindros( tModeloCoche *ModeloCoche, int *cilindros );
int ModeloCoche_Leer_Combustible( tModeloCoche *ModeloCoche, char *combustible );
int ModeloCoche_Leer_CVs( tModeloCoche *ModeloCoche, int *CV1, int *CV2 );
int ModeloCoche_Leer_kWs( tModeloCoche *ModeloCoche, int *kW1, int *kW2 );
int ModeloCoche_Leer_CVf( tModeloCoche *ModeloCoche, float *CVf );
int ModeloCoche_Leer_Euros( tModeloCoche *ModeloCoche, float *euros );
```

Lista.h

```
typedef void* tInfoNodo;  // Puntero a objeto genérico de información

typedef struct tNodoLista
{
	tInfoNodo info;			// Información asociada al nodo
	struct tNodoLista* siguiente;	// Puntero al nodo siguiente de la lista
	struct tNodoLista* anterior;	// Puntero al nodo anterior de la lista

} tNodoLista;

typedef tNodoLista* tLista;

int Lista_Inicializar( tLista *lista );
int Lista_Finalizar( tLista *lista );
int Lista_Insertar( tNodoLista *nodolista, tInfoNodo info );
int Lista_Borrar( tNodoLista *nodolista, tInfoNodo *info );

char Lista_Vacia( tLista *lista );
tNodoLista* Lista_Primero( tLista *lista );
tNodoLista* Lista_Siguiente( tNodoLista *nodolista );
tNodoLista* Lista_Anterior( tNodoLista *nodolista );
tInfoNodo Lista_Informacion( tNodoLista *nodolista );
```

ABBModeloCoche.h

```
typedef struct tNodoABBModeloCoche
{
	tModeloCoche* info;                         // Puntero a modelo de coche
	struct tNodoABBModeloCoche* hi;      // Puntero al hijo izquierdo
	struct tNodoABBModeloCoche* hd;      // Puntero al hijo derecho

} tNodoABBModeloCoche;

typedef tNodoABBModeloCoche* tABBModeloCoche;

int ABBModeloCoche_Inicializar( tABBModeloCoche *ABBModeloCoche );
int ABBModeloCoche_Borrar( tABBModeloCoche *ABBModeloCoche, char LiberarInfo );
int ABBModeloCoche_Insertar( tABBModeloCoche *ABBModeloCoche,
                             tModeloCoche *ModeloCoche );
int ABBModeloCoche_Buscar_Valor( tABBModeloCoche ABBModeloCoche, float valor,
                                                          tLista *resultado );
int ABBModeloCoche_Buscar_Valores( tABBModeloCoche ABBModeloCoche, float valor1,
	float valor2,  tLista *resultado );
int ABBModeloCoche_Inorden( tABBModeloCoche ABBModeloCoche, tLista *resultado );
```