Ejercicios de implementación y uso de TDAs

1. Implementa el TDA descrito a continuación:

```
// El tipo Punto representa un punto en el plano real
typedef struct PuntoRep * Punto;

// Devuelve un nuevo Punto con las coordenadas (x,y)
Punto crea_punto( double x, double y );

// Libera la memoria asociada al Punto p
void libera_punto( Punto p );

// Muestra el Punto p por la pantalla
void muestra_punto( Punto p );

// Devuelve la coordenada x del Punto p
double recupera_x_punto( Punto p );

// Devuelve la coordenada y del Punto p
double recupera_y_punto( Punto p );

// Modifica las coordenadas del Punto p
void modifica_punto( Punto p, double x, double y );
```

2. Implementa el TDA descrito a continuación:

```
// El tipo Rectangulo representa un rectángulo en el plano
typedef struct RectanguloRep * Rectangulo;

// Devuelve un nuevo Rectangulo cuya esquina superior izquierda
// estará situada en las coordenadas (x,y) y tendrá la anchura y
// altura indicadas. Todos los valores serán de tipo entero.
Rectangulo crea_rectangulo( int x, int y, int ancho, int alto );

// Libera la memoria asociada al Rectangulo r
void libera_rectangulo( Rectangulo r );

// Muestra el Rectangulo r por la pantalla
void muestra_rectangulo( Rectangulo r );

// Devuelve la coordenada x del Rectangulo r
int recupera_x_rectangulo( Rectangulo r );

// Devuelve la coordenada y del Rectangulo r
int recupera_y_rectangulo( Rectangulo r );

// Modifica las coordenadas del Rectangulo r
void modifica_rectangulo( Rectangulo r, int x, int y );

// Devuelve 1 si el punto (x,y) está dentro del rectángulo r
int dentro_rectangulo( Rectangulo r, int x, int y );
```

3. Implementa el TDA descrito a continuación:

```
// El tipo Complejo representa un número complejo mediante dos
// números reales: la parte real y la parte imaginaria
typedef struct ComplejoRep * Complejo;

// Devuelve un nuevo Complejo con las partes real e imaginaria
// pasadas como parámetro
Complejo crea_complejo( double r, double i );

// Devuelve un nuevo Complejo a partir de su módulo (r) y su
// ángulo (o)
Complejo crea_polares_complejo( double r, double o );

// Libera la memoria asociada al Complejo c
void libera_complejo( Complejo c );

// Muestra el Complejo c por la pantalla
void muestra_complejo( Complejo c );

// Devuelve la parte real del Complejo c
double recupera_parte_real_complejo( Complejo c );

// Devuelve la parte imaginaria del Complejo c
double recupera_parte_imaginaria_complejo( Complejo c );

// Devuelve el módulo del número complejo c
double modulo_complejo( Complejo c );

// Devuelve un nuevo número complejo resultado de sumar c y d
Complejo suma_complejo( Complejo c, Complejo d );

// Devuelve un nuevo número complejo resultado de restar c y d
Complejo resta_complejo( Complejo c, Complejo d );

// Devuelve un nuevo número complejo resultado de multiplicar c
// y d
Complejo multiplica_complejo( Complejo c, Complejo d );

// Devuelve un nuevo número complejo resultado de dividir c
// entre d
Complejo divide_complejo( Complejo c, Complejo d );

// Devuelve 1 si c == d y 0 en caso contrario
int compara_complejo( Complejo c, Complejo d );
```

4. Implementa el TDA descrito a continuación:

```
// El tipo Matriz representa una Matriz de las usadas en álgebra
// Los valores almacenados serán de tipo real
typedef struct MatrizRep * Matriz;

// Devuelve una nueva Matriz con f filas y c columnas
Matriz crea_matriz( int f, int c );

// Devuelve una nueva Matriz cuadrada con n filas y n columnas
// con todos los elementos de su diagonal iguales a 1.0
Matriz matriz_identidad( int n );

// Libera la memoria asociada a la Matriz m
void libera_matriz( Matriz m );

// Muestra la Matriz m por la pantalla
void muestra_matriz( Matriz m );

// Devuelve el valor que situado en la fila f y columna c
double recupera_valor_matriz( Matriz m, int f, int c );

// Asigna el valor r a la posición de la fila f y columna c
void asigna_valor_matriz( Matriz m, int f, int c, double r );

// Devuelve el número de filas de la Matriz m
int recupera_filas_matriz ( Matriz m );

// Devuelve el número de columnas de la Matriz m
int recupera_columnas_matriz( Matriz m );

// Devuelve una nueva Matriz resultado de multiplicar todos los
// valores de m por e.
Matriz escala_matriz( Matriz m, double e );

// Devuelve una nueva Matriz resultado de sumar m y n siempre
// que sea posible hacerlo. En caso contrario devuelve NULL
Matriz suma_matriz( Matriz m, Matriz n );

// Devuelve una nueva Matriz resultado de multiplicar m y n
// siempre que sea posible hacerlo. En caso contrario devuelve
// NULL
Matriz multiplica_matriz( Matriz m, Matriz n );

// Devuelve una nueva Matriz resultado de trasponer m
Matriz matriz_traspuesta( Matriz m );
```

5. Implementa el TDA descrito a continuación:

```
// El tipo Monomio representa una expresión algebraica de un
// solo término con la forma cx^n siendo c el coeficiente y n el
// exponente. C será un número real y n un entero
typedef struct MonomioRep * Monomio;

// Devuelve un nuevo monomio representando a coef * X^exp
Monomio crea_monomio( double coef, int exp );

// Libera la memoria asociada al monomio m
void libera_monomio( Monomio m );

// Muestra el monomio m por la pantalla
void muestra_monomio( Monomio m );

// Devuelve el coeficiente del monomio m
double recupera_coeficiente_monomio( Monomio m );

// Devuelve el exponente del monomio m
int recupera_exponente_monomio( Monomio m );

// Asigna el coef como coeficiente del monomio m
void modifica_coeficiente_monomio( Monomio m, double coef );

// Devuelve el resultado de hacer cx^n
double evalua_monomio ( Monomio m, double x );
```

6. Usando el TDA Monomio del ejercicio anterior, implementa el TDA descrito a continuación:

```
// El tipo Polinomio representa una expresión algebraica de una
// sola variable compuesta por la suma de un número
// indeterminado de Monomios
typedef struct PolinomioRep * Polinomio;

// Devuelve un nuevo polinomio representando a coef * X^exp
Polinomio crea_polinomio( double coef, int exp );

// Libera la memoria asociada al polinomio p
void libera_polinomio( Polinomio p );

// Muestra el polinomio p por la pantalla
void muestra_polinomio( Polinomio p );

// Devuelve un nuevo polinomio resultante de sumar p y q
Polinomio suma_polinomio( Polinomio p, Polinomio q );

// Devuelve el valor de sustituir x en el polinomio p
double evalua_polinomio( Polinomio p, double x );
```

7. Usando el TDA Punto del ejercicio 1, implementa el TDA descrito a continuación:

```
// Una Polilínea representa una secuencia ordenada de puntos en
// el plano bidimensional. Los puntos de la Polilínea se
// mantendrán en el orden en que sean añadidos
typedef struct PolilineaRep * Polilinea;

// Devuelve una nueva polilínea vacía
Polilinea crea_polilinea( );

// Libera la memoria asociada a la polilínea p
void libera_polilinea( Polilinea p );

// Muestra la polilínea p por la pantalla
void muestra_polilinea( Polilinea p );

// Añade el punto q al final de la secuencia de puntos de p
void añade_punto_polilinea( Polilinea p, Punto q );

// Devuelve el punto incluido en p que esté más cerca
// de las coordenadas (x,y) usando distancia euclídea
Punto selecciona_polilinea( Polilinea p, double x, double y );

// Elimina el punto q de la secuencia de puntos de p
void elimina_punto_polilinea( Polilinea p, Punto q );
```

8. Implementa el TDA descrito a continuación:

```
// El tipo Contacto representa la información de contacto de una
// persona conocida. En concreto su nombre y su teléfono.
// Tanto el nombre como el teléfono serán cadenas de caracteres
// de longitud máxima no limitada.
typedef struct ContactoRep * Contacto;

// Devuelve un nuevo Contacto para la persona llamada nombre
// El teléfono asignado por defecto será “000 000 000”
Contacto crea_contacto( char * nombre );

// Libera la memoria asociada al Contacto c
void libera_contacto( Contacto c );

// Muestra el Contacto c por la pantalla
void muestra_contacto( Contacto c );

// Devuelve una copia en memoria dinámica del nombre de c
char * recupera_nombre_contacto( Contacto c );

// Compara el nombre de c con nombre y devuelve -1, 0 o 1 según
sea menor, igual o mayor lexicográficamente hablando
int compara_nombre_contacto( Contacto c, char * nombre );

// Devuelve una copia en memoria dinámica del tlf de c
char * recupera_tlf_contacto( Contacto c );

// Modifica el teléfono del Contacto c pasando a ser tlf
void modifica_tlf_contacto( Contacto c, char * tlf );
```

9. Usando el TDA Contacto del ejercicio anterior, implementa el TDA descrito a continuación:

```
// Una Agenda representa una secuencia ordenada de Contactos.
// Los Contactos se mantienen en orden lexicográfico por nombre
// y sin repeticiones.
typedef struct AgendaRep * Agenda;

// Devuelve una nueva Agenda vacía
Agenda crea_agenda( );

// Libera la memoria asociada a la agenda a
void libera_agenda( Agenda a );

// Muestra la lista de contactos de la agenda a por la pantalla
void muestra_agenda( Agenda a );

// Añade un contacto para la persona llamada nombre a la lista
// de Contactos de a si y sólo si dicha persona aún no existe en
// la lista. Además, le asigna el teléfono tlf
void añade_contacto_agenda( Agenda a, char * nombre, char * tlf );

// Devuelve una copia en memoria dinámica del teléfono de nombre
// si es que éste existe en la lista de contactos de la agenda
// a. Si no devuelve NULL.
char * busca_tlf_agenda( Agenda a, char * nombre );

// Elimina el Contacto asociado a nombre si es que hay alguno
// que coincida
void elimina_contacto_agenda( Agenda a, char * nombre );
```

Dada la siguiente especificación del TDA Lista:

```
typedef int Elemento
typedef struct ListaRep * Lista;
typedef struct PosicionRep * Posicion;

// Devuelve una nueva lista vacía
Lista crea_lista( );

// Libera la memoria asociada a la lista l
void libera_lista( Lista l );

// Inserta un nuevo nodo en la posición p con el dato e
void inserta_lista( Lista l, Posicion p, Elemento e );

// Suprime el elemento que ocupa la posición p
// La posición p debe ser distinta del final de la lista
void suprime_lista( Lista l, Posicion p );

// Devuelve el elemento que ocupa la posición p
Elemento recupera_lista( Lista l, Posicion p );

// Sustituye el elemento que ocupa la posición p por e
void asigna_lista( Lista l, Posicion p, Elemento e );

// Devuelve la longitud de la lista
int longitud_lista( Lista l );

// Devuelve la posición que ocupa el primer elemento
Posicion inicio_lista( Lista l );

// Devuelve la siguiente posición a la ocupada por el último de
// la lista
Posicion final_lista( Lista l );

// Devuelve la siguiente posición a p siempre que pe no sea Fin
Posicion siguiente_posicion( Lista l, Posicion p );

// Devuelve la posición anterior a p siempre que p no sea Inicio
Posicion anterior_posicion(Lista l, Posicion p);
```

10. Completa el código de la siguiente función para que muestre todos sus elementos en el orden en que estén almacenados.

    ```
    void muestra( Lista l )
    ```

11. Completa el código de la siguiente función para que muestre todos sus elementos en el orden contrario al que estén almacenados.

    ```
    void muestra( Lista l )
    ```

12. Completa el código de la siguiente función para que devuelva 1 si la lista contiene, al menos, un elemento repetido y 0 en caso contrario.

```
int existe( Lista l )
```

13. Completa el código de la siguiente función para que sustituya por `b` los elementos cuyo valor coincida con `a`.

```
void sustituye( Lista l, Elemento a, Elemento b )
```

14. Completa el código de la siguiente función para que devuelva una nueva Lista en la que estén todos los elementos de la lista cuyo valor sea distinto a `d`.

```
Lista elimina( Lista l, Elemento d )
```

15. Completa el código de la siguiente función para que construya y devuelva una nueva lista conteniendo los elementos de `l` ordenados de menor a mayor.

```
Lista recupera( Lista l )
```

16. Completa el código de la siguiente función para que construya y devuelva una nueva lista conteniendo los elementos de `l` sin ninguna repetición. Por ejemplo, dada la lista {1,1,2,3,3,4,5,5,6} devolvería la lista {1,2,3,4,5,6}.

```
Lista recupera( Lista l )
```

17. Completa el código de la siguiente función para que devuelva una nueva lista conteniendo todos los elementos de `l` que no estén repetidos.

```
Lista elimina( Lista l )
```

18. Completa el código de la siguiente función para que construya y devuelva una nueva lista resultado de concatenar las dos pasadas como parámetro.

```
Lista concatena( Lista l, Lista m )
```

19. Completa el código de la siguiente función para que construya y devuelva una nueva lista con todos los elementos de l cuyo valor esté en el intervalo [`a`, `b`].

```
Lista selecciona( Lista l, Elemento a, Elemento b )
```

20. Completa el código de la siguiente función para que construya y devuelva una nueva lista que contenga únicamente los elementos que estén en las dos listas pasadas como parámetro.

```
Lista interseccion( Lista l, Lista m )
```

21. Completa el código de la siguiente función para que construya y devuelva una nueva lista resultado que contenga los elementos que estén en cualquiera de las dos listas pasadas como parámetro.

```
Lista union( Lista l, Lista m )
```

Dada la siguiente especificación del TDA Pila

```
typedef int Elemento;

typedef struct PilaRep * Pila;

// Devuelve una nueva Pila vacía
Pila crea_pila();

// Libera la memoria asociada a la pila p
void libera_pila( Pila p );

// Añade un nuevo elemento al tope de la pila p
void inserta_pila( Pila p, Elemento dato );

// Devuelve el elemento situado en el tope de la pila p
Elemento recupera_pila( Pila p );

// Elimina el elemento situado en el tope de la pila p
void suprime_pila( Pila p );

// Devuelve 1 si la pila p está vacía y 0 en caso contrario
int vacia_pila( Pila p );
```

22. Usando el TDA Pila, completa el código de la siguiente función para que muestre los elementos de la lista `l` en orden inverso sin usar la operación `anterior`.

    ```
    void muestra( Lista l )
    ```

23. Completa el código de la siguiente función para que muestre todos los elementos de la pila `p`.

    ```
    void muestra( Pila p )
    ```

24. Usando el TDA Pila, completa el código de la siguiente función para que muestre la codificación en binario de `n`.

    ```
    void decimal2binario( int n )
    ```

25. Usando el TDA Pila, completa el código de la siguiente función para que devuelva 1 si los paréntesis de la cadena de caracteres están equilibrados y 0 en caso contrario.

    ```
    int comprueba( char * cadena )
    ```

Dada la siguiente especificación del TDA Cola

```
typedef int Elemento;

typedef struct ColaRep * Cola;

// Devuelve una nueva Cola vacía
Pila crea_cola();

// Libera la memoria asociada a la cola c
void libera_cola( Cola c );

// Añade un nuevo elemento al final de la cola c
void inserta_cola( Cola c, Elemento dato );

// Devuelve el elemento situado en el inicio de cola c
Elemento recupera_cola( Cola c );

// Elimina el elemento situado en el inicio de la cola c
void suprime_cola( Cola c  );

// Devuelve 1 si la cola c está vacía y 0 en caso contrario
int vacia_cola( Cola c );
```

26. Completa el código de la siguiente función para que devuelva una nueva cola conteniendo sólo los números pares de la cola c pero en el mismo orden.

    ```
    Cola filtra( Cola c )
    ```

27. Completa el código de la siguiente función para que muestre los elementos que quedarían en una cola inicialmente vacía a la que se le realizan las operaciones descritas por las letras de la cadena `comandos`. Interpreta cualquier letra como instrucciones `inserta_cola` en donde el elemento a insertar sería la letra y los asteriscos como instrucciones `suprime_cola`. Por ejemplo, si comandos fuera la cadena "FIFO**" deberían mostrarse las letras F y O.

    ```
    void procesa( char comandos[] )
    ```

28. Completa el código de la siguiente función para que determine qué jugador ganaría una guerra de cartas. En este juego, cada jugador tiene un mazo de cartas inicial. En cada turno, se retira la primera carta de cada jugador y se comparan, el ganador se queda con las dos cartas y las coloca al final de su propio mazo. El juego termina después de un máximo de `n` rondas o cuando un jugador se queda sin cartas. El ganador será el que tenga más puntos sumando los números de las cartas de su mazo. Los parámetros `a` y `b` representan los mazos de cartas iniciales como arrays de números enteros en el intervalo 1..12, su contenido no debe ser alterado. El resultado del juego se mostrará usando printf.

    ```
    void guerra_de_cartas( int n, int a[], int b[] )
    ```

Dada la siguiente especificación del TDA Conjunto

```
typedef int Elemento;

typedef struct ConjuntoRep * Conjunto;

// Devuelve un nuevo conjunto vacío
Conjunto crea_conjunto();

// Libera la memoria asociada al conjunto c
void libera_conjunto(Conjunto c);

// Añade e al conjunto siempre que e no pertenezca a c
void inserta_conjunto(Conjunto c, Elemento e);

// Elimina e del conjunto c siempre que e sí pertenezca a c
void suprime_conjunto(Conjunto c, Elemento e); ç

// Devuelve 1 si e pertenece a c y 0 en caso contrario
int pertenece_conjunto(Conjunto c, Elemento e);

// Devuelve el número de elemntos incluidos en el conjunto c
int cardinalidad_conjunto(Conjunto c);

// Devuelve una lista conteniendo todos los elementos de c
Lista enumera_conjunto(Conjunto c);
```

29. Completa el código de la siguiente función para que muestre todos los elementos del conjunto c.

    ```
    void muestra( Conjunto c )
    ```

30. Completa el código de la siguiente función para que devuelva un nuevo conjunto conteniendo todos los elementos pares incluidos en a y en b.

    ```
    Conjunto union( Conjunto a, Conjunto b )
    ```

31. Completa el código de la siguiente función para que devuelva un nuevo conjunto conteniendo todos los elementos impares incluidos tanto en a como en b.

    ```
    Conjunto interseccion( Conjunto a, Conjunto b )
    ```

32. Completa el código de la siguiente función para que devuelva un nuevo conjunto conteniendo todos los elementos incluidos en a que no estén en b.

    ```
    Conjunto diferencia( Conjunto a, Conjunto b )
    ```

33. Completa el código de la siguiente función para que devuelva 1 si ambos conjuntos son iguales (contienen los mismos elementos) y 0 si no lo son.

    ```
    int iguales( Conjunto a, Conjunto b )
    ```