PRÁCTICA 3

Tipo Abstracto de Datos: Listas, Listas Doblemente Enlazadas, Listas Circulares. Nota: para implementar los TADs utilice archivos .c y .h

IMPORTANTE: para cada función pedida en los TADs analizar, diseñar y programar todos los casos:

- **Cada función debe estar correctamente documentada (comentario especificando *qué hace la función*.)**
- **Si la operación se puede llevar a cabo, determinar qué retorna la función y el estado de las distintas variables y/o parámetros.**
- **Casos en que la operación no se pueda llevar a cabo: a modo de ejemplo, eliminar el nodo de la posición 10 siendo 5 la longitud de la lista. Determinar precondiciones y postcondiciones, decidir qué hace la función en estos casos, programar y documentar. Las funciones deben funcionar correctamente para todos los casos.**
- **Analizar y justificar los parámetros de las funciones. Detectar para cada parámetro si está pasado por valor o por referencia y justificar el por qué.**

1. LISTAS

1) Implementar el TAD lista de enteros utilizando punteros. Evaluar si se usará directamente puntero al primer elemento de la lista o una solución del tipo:

```
struct Nodo
{
  int dato;
  struct Nodo *sig;
};

typedef struct
{
  int n;
  struct Nodo *lista;
}Lista_T;
```

El TAD debe disponer de las siguientes funciones:

1. // Crea una lista del tipo `Lista_T` con 0 elementos y la retorna (lista vacía).
   `Lista_T crearLista();`

2. // Inserta el elemento x como primer elemento de la lista. Retorna 1 si la operación se realiza con éxito, -1 caso contrario.
   `int InsertarPrimero(Lista_T *l, int x);`

3. // Inserta elemento x como último elemento de la lista. Retorna 1 si la operación se realiza con éxito, -1 caso contrario.
   `int InsertarUltimo(Lista_T *l, int x);`

4. // Recorre la lista "l" imprimiendo sus elementos. Retorna 1 si la operación se realiza con éxito, -1 caso contrario.
   `int ImprimirLista(Lista_T l );`

5. // Retorna Verdadero si la lista está vacía, Falso caso contrario.
   `int EstaVacia(Lista_T l);`

6. // Libera la memoria ocupada por la lista. La lista queda vacía. Retorna 1 si la operación se realiza con éxito, -1 caso contrario.
   `int VaciarLista(Lista_T *l);`

7. // Elimina el dato x de la lista. Completar con los casos que x existe en la lista o que no existe
   `int SuprimirDato(Lista_T *l, int x);`

8. // Elimina el dato con posición `pos` de la lista. Determinar la acción a realizar en caso que la posición no exista en la lista.
   `int SuprimirDatoPosicion(Lista_T *l, int pos);`

9. // Retorna la longitud de la lista `l`.
   `int LongitudLista(Lista_T l);`

10. // Devuelve el dato de la posición `pos`. Determinar la acción a realizar en caso que la posición no exista en la lista.
    `int DevolverDatoPosicion(Lista_T l, int pos);`

11. // Retorna Verdadero si el dato *x* existe en la lista, Falso caso contrario.
    `int ExisteDato(Lista_T l, int x);`

2) Implemente el TAD Lista Simplemente Enlazada con arreglos, con las operaciones incluidas en 1). Compare ambas implementaciones desde el punto de vista de memoria utilizada. Para cada función implementada, defina si la operación es más simple o más costosa a nivel de cantidad de operaciones que debe realizar (comparaciones, corrimientos, etc). Defina una longitud máxima de la lista, y para cada operación de inserción verifique si

es posible insertar.

3) Responda:

(a) ¿Cuál de las 2 implementaciones anteriores es más conveniente? ¿Qué aspectos tendría en cuenta para optar por una de las dos implementaciones?

(b) Cuál de las 2 implementaciones elije si:

(I) No se sabe la cantidad de elementos de la lista. No es posible acotar una posible longitud de la lista.

(II) Se realizan muchas operaciones `InsertarPrimero()`, `InsertarUltimo()` y `SuprimirDato()`.

(III) Sobre todo, se realizan operaciones `devolverDatoPosicion()` sobre la lista.

4) Se desea implementar una lista de enteros. Se pide obtener la cantidad de espacio de memoria utilizado para las implementaciones:

1. La lista tiene 100 elementos y se implementa con punteros.
2. La lista tiene 100 elementos y se tiene como precondición que la cota superior de elementos de la lista es de 100 elementos. Se implementa con arreglos.

Indique ventajas y desventajas de cada implementación. Indique qué implementación utilizaría y justifique su respuesta.

5) Utilizando el TAD implementado en 1) escriba un programa que agregue los números del 1 al 20, usando la función InsertarPrimero(). Imprima los datos de la lista. Ingrese un dato desde el teclado y verifique si el dato existe en la lista. Programe una función para eliminar los valores pares de la lista. Imprima nuevamente verificando correctitud.

6) ¿Cómo haría para recorrer la lista en orden inverso?

7) Se desea formar una lista enlazada de números aleatorios. El programa que realiza esta tarea inserta los valores al final de la lista. Una vez creada la lista, se recorre la misma para mostrar los números pares. Implementar con funciones.

9) Un conjunto es una colección de elementos del mismo tipo sin duplicidades. Escribir un programa para representar un conjunto de enteros. El programa debe contemplar las operaciones:
- Crear un conjunto.
- Cardinal del conjunto.
- Pertenencia de un elemento al conjunto.
- Añadir un elemento al conjunto.
- Escribir en pantalla los elementos del conjunto.

10) El TAD Lista Ordenada, es similar a la lista simplemente enlazada, pero cada elemento que se agrega a la lista se agrega de forma ordenada, según un criterio de orden. Este criterio puede ser un campo clave o cualquier otra comparación de sus elementos. Cada vez que se agrega un dato a la lista ordenada, se busca su posición correcta y se inserta. Es una post condición que luego de insertar un dato a una lista ordenada, la misma sigue siendo una lista ordenada. Implementar este TAD para elementos del tipo complejo donde se pide que queden ordenados por su magnitud (magnitud de z = $\sqrt{(re(z)^2 + im(z)^2)}$).

2. LISTAS DOBLEMENTE ENLAZADAS

11) Implementar el TAD lista doblemente enlazada de enteros utilizando punteros. Una posible implementación del tipo sería:

```
struct NodoDE
{
  int dato;
  struct NodoDE *sig;
  struct NodoDE *ant;
};

typedef struct
{
  int n;
  struct Nodo *primero;
  struct Nodo *ultimo;
}ListaDE_T;
```

El TAD debe disponer de las siguientes funciones:

1. // crea una lista del tipo `ListaDE_T` con 0 elementos y la retorna.
   `ListaDE_T crearLista();`

2. // inserta el valor x como primer elemento de la lista. Retorna 1 si la operación se realiza con éxito, -1 caso contrario. `int InsertarPrimero(ListaDE_T *l, int x );`

3. // inserta el valor x como último elemento de la lista. Retorna 1 si la operación se realiza con éxito, -1 caso contrario.
   `int InsertarUltimo(ListaDE_T *l, int x );`

4. // Recorre la lista l imprimiendo sus elementos, del primer elemento al último. Retorna 1 si la operación se realiza con éxito, -1 caso contrario.
   `int ImprimirLista(ListaDE_T l );`

5. // Recorre la lista l imprimiendo sus elementos, del último al primero. Retorna 1 si la operación se realiza con éxito, -1 caso contrario.
   `int RecorrerListaEnOrdenInverso(ListaDE_T l);`

6. // retorna Verdadero si la lista está vacía, Falso caso contrario.
   `int EstaVacia(ListaDE_T l);`

7. // Elimina el dato x de la lista.
   `int SuprimirDato(ListaDE_T *l, int x);`

8. // desaloca la memoria ocupada por la lista.
   `int VaciarLista(ListaDE_T *l);`

9. // retorna la longitud de la lista `int LongitudLista(ListaDE_T l);`

10. // devuelve el dato de la posición pos.
    `int DevolverDatoPosicion(ListaDE_T l, int pos);`

11) Realizar un programa que inserte en una lista los enteros: 4, 3, 7, 10, 11 y 12. Luego, imprima la lista de dos formas: en el orden que tiene la lista y en el orden inverso. Agregar los datos 45, 67, 87 y luego eliminar los datos 11, 12 y mostrar la lista en orden inverso.

3. Listas Circulares

12) Implementar el TAD lista circular con punteros. Una posible implementación del tipo sería:

```
struct NodoT {
   int dato;
   struct NodoT *sig;
};
typedef struct NodoT Nodo;

typedef struct{
   int n;
   Nodo *lista;
}ListaC_T;
```

El TAD debe disponer de las siguientes funciones:

1. // crea una lista circular con 0 elementos y la retorna.
   `ListaC_T CrearLista();`

2. // inserta el elemento x al comienzo de la lista. Retorna 1 si la operación se realiza con éxito, -1 caso contrario.

3. `int InsertarDato(ListaC_T *l, int x );`

4. // imprime los elementos de la lista.
   ```
   int ImprimirLista(ListaC_T l);
   ```

5. // Retorna la longitud de la lista l.
   ```
   int LongitudLista(ListaC_T l );
   ```

13) Utilizando el TAD lista circular, implemente un programa que genere una lista con los datos 1, 2, 3, 4, 5 y 6 y luego imprima todos los datos de la lista.