Listas enlazadas

Ing José Luis MARTÍNEZ

27 de mayo de 2019

1. Estructuras y funciones

- Una estructura puede ser pasada como argumento a una función de tres formas distintas
  - Pasando los miembros en forma individual
  - Pasando la estructura completa
  - Pasando la dirección de la estructura

1.1. Pasar los miembros en forma individual

```
    #include <stdio.h>
typedef struct
{
int x;
int y;
}Punto;
void display(int, int);
int main()
{
Punto p1 = 2, 3;
display(p1.x, p1.y);
return 0;
} void display(int a, int b)
{
    printf("Las coordenadas del punto son: %d %d", a, b);
    }
```

1.2. Pasar una estructura completa a una función

```
    #include <stdio.h>
typedef struct
{
int x;
int y;
}Punto;
void display(Punto);

    int main()
{
Punto p1 = 2, 3;
display(p1);
return 0;
}
```

```
void display(Punto p)
{
printf("Las coordenadas del punto son: %d %d", p.x, p.y);
}
```

1.3. Pasar una estructura a una función mediante punteros

- Pasar por valor estructuras grandes puede ser muy ineficiente
- Por eso es preferible pasarlas por referencia es decir mediante un puntero, aprovechando que C permite crear punteros a cualquier tipo de valor inclusive los definidos por el usuario.
- Como en los otros tipos de datos, un puntero a una estructura no es una estructura en sí mismo sino que es una variable que contiene la dirección de la estructura.
- La sintaxis para declarar un puntero a una estructura es:

```
struct nombre
{
tipoDeDato miembro_1;
tipoDeDato miembro_2;
.....
}*ptr;
```

o también

```
struct nombre *ptr;
```

Por ejemplo tenemos una estructura llamada **alumno** podemos declarar una variable de tipo puntero
struct **alumno *ptr_alumno , alumn**;
A continuación se le asigna la dirección de **alumn** al puntero utilizando el operador de dirección:
ptr_alumno = &alumn
Para acceder a los miembros de la estructura podemos escribir:
(*ptr_alumno).legajo;
Como el paréntesis tiene una precedencia mayor que el asterisco, la instrucción funciona bien.

La declaración con el paréntesis puede que presente dificultades para utilizarlas especialmente cuando se está iniciando en el manejo de C. Por eso el lenguaje introduce un nuevo operador que realiza el mismo trabajo, y se lo conoce como operador flecha (u operador 'ponting to') **-¿**.
El operador puede ser utilizado de la siguiente forma:
//* El legajo en la estructura ptr_alumno apunta a *//
ptr_alumno $->$ **legajo** = 212121;

Esta declaración es más simple que utilizando los paréntesis.

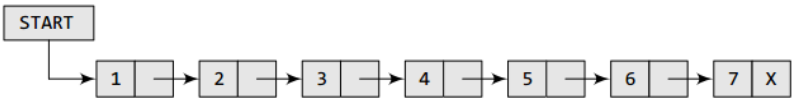
2. Estructuras autoreferenciadas

- Se vio que una estructura puede anidar a otras estructuras.
- Una estructura no puede anidarse a sí misma, es decir no se puede tenerse a sí misma como un miembro, como es el caso de una función recursiva.
- Sin embargo una estructura puede tener como miembro un puntero que apunte a una estructura de su tipo.

- Por ejemplo la estructura siguiente:
  struct nodo
  {
  int val;
  struct nodo *siguiente;
  };
  La estructura tiene dos tipos de miembros, un entero **val** y un puntero **siguiente**

3. Listas enlazadas

- Una lista enlazada es una colección lineal de elementos.
- A diferencia de los arreglos los elementos no se encuentran almacenados en posiciones contiguas de memoria.
- También se diferencian en que no es necesario declarar previamente el tamaño, creciendo y decreciendo de acuerdo a la necesidad.
- Como desventaja con respecto a los arreglos podemos decir que no se pueden acceder de forma aleatoria a sus elementos, sino en forma secuencial.
- Al igual que los arreglos sus elementos se pueden agregar o quitar en cualquier punto.
- Una lista enlazada se puede representar como una cadena de elementos, donde cada elemento se denomina *Nodo*, y los eslabones de la cadena son los apuntadores al siguiente elemento.
- Cada nodo tiene uno o varios miembros.
- El primer nodo es la *cabeza* de la lista y el último es la *cola* de la lista
- El último nodo no tiene un eslabón conectado a él, por lo que se le carga el valor **NULL**



En memoria una lista enlazada podría quedar representada de la siguiente forma:

START 1

	Data	Next
1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	O	-1

- Las listas enlazadas se clasifican en:
  - **Simplemente enlazadas:** son las que cada nodo se conecta únicamente con el sucesor
  - **Doblemente enlazadas:** cada nodo se liga doblemente con su antecesor y con su sucesor
  - **Listas circulares:** el último nodo (*cola*) se conecta al primer elemento (*cabeza*), permite que la lista sea recorrida en forma circular. Dentro de las listas circulares tenemos:
    - **Simplemente enlazadas.**
    - **Doblemente enlazadas.**

3.1. Operaciones con las listas enlazadas

Las operaciones que se pueden realizar con listas enlazadas contemplan

- Declaración de los tipos nodo y puntero a nodo.
- Inicialización o creación.
- Desplegar los valores.
- Insertar elementos en una lista.
- Eliminar elementos de una lista.
- Buscar elementos de una lista (comprobar la existencia de elementos en una lista).
- Recorrer una lista enlazada (visitar cada nodo de la lista).
- Comprobar si la lista está vacía.
- Ordenar una lista.

3.2. Declaración del nodo

```
typedef int Item;
typedef struct Elemento
{
    Item dato;
    struct Elemento *siguiente;
}Nodo;
```

3.3. Crear un nodo

Tendrá un algoritmo similar al siguiente:

1 Declarar el tipo de dato y el puntero de cabeza o primero.

2 Asignar memoria para un elemento del tipo definido anteriormente utilizando alguna de las funciones de asignación de memoria (malloc( ), calloc( ), realloc( ))y un cast para la conversión de void* al tipo puntero a nodo; la dirección del nuevo elemento es *siguiente*.

3 Comprobar que se haya reservado la memoria necesaria.

4 Si es el primer nodo que se va a insertar en la lista, el puntero siguiente debe apuntar a NULL, y el nodo pasa a ser el inicio

5 Si la lista ya tiene nodos, los nuevos nodos se incorporan al final, para ello se recorre la lista hasta el último nodo, el puntero de siguiente de este nodo deja de apuntar a NULL para apuntar al nuevo nodo creado, y el puntero del nuevo nodo apunta a NULL porque pasa a ser el nodo final.

5 Repetir hasta que no haya más entradas para el elemento.

```
Nodo *crear(Nodo* inicio)
{
    Nodo *nuevoNodo, *ptr;
    int num;
    printf("\nIngrese -1 para finalizar.");
    printf("\nIngrese el dato.");
    scanf("%d", &num);
```

```
    while(num != -1)
        {
            nuevoNodo = (Nodo*)malloc(sizeof(Nodo));
            nuevoNodo->dato = num;
            if(inicio==NULL)
                {
                    nuevoNodo -> siguiente = NULL;
                    inicio = nuevoNodo;
                }
            else
                {
                    ptr=inicio;
                    while(ptr->siguiente != NULL)
                        ptr = ptr->siguiente;
                    ptr->siguiente = nuevoNodo;
                    nuevoNodo->siguiente=NULL;
                }
            printf("\n Ingrese el valor del nodo : ");
            scanf("%d", &num);
        }
return inicio;
}
```

3.4. Desplegar una lista

Algoritmo:

1 Crear un puntero ptr y cargarlo con inicio

2 Recorrer la lista hasta el último nodo

3 Imprimir en cada iteracción el valor del nodo

```
Nodo *muestraLista(Nodo *inicio)
{
Nodo *ptr;
ptr = inicio;
while(ptr != NULL)
{
 printf("\t %d", ptr -> dato);
 ptr = ptr -> siguiente;
}
return inicio;
}
```

3.5. Insertar cabecera

Algoritmo:

1 Verificar si hay memoria, de lo contrario salir

2 Reservar espacio para nuevoNodo

3 Cargar el dato en el nuevoNodo

4 Apuntar nuevoNodo− >siguiente a cabecera

5 Hacer que el nuevoNodo sea la cabecera

```
Nodo *insertaCabeza(Nodo *inicio)
{
    Nodo *nuevoNodo;
    int num;
    printf("\n Ingrese el valor del nodo : ");
    scanf("%d", &num);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    nuevoNodo -> siguiente = inicio;
    inicio = nuevoNodo;
    return inicio;
}
```

3.6. Insertar nodo al final de la lista (cola)

Algoritmo:

1 Verificar si hay memoria, de lo contrario salir

2 Reservar espacio para nuevoNodo

3 Cargar el dato en el nuevoNodo

4 Apuntar nuevoNodo− >siguiente a NULL

5 Apuntar ptr a inicio

6 Recorrer la lista hasta llegar al último nodo

7 Apuntar ptr− >siguiente a nuevoNodo

```
Nodo *insertaCola(Nodo *inicio)
{
    Nodo *ptr, *nuevoNodo;
    int num;
    printf("\n Ingrese el valor del nodo : ");
    scanf("%d", &num);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    nuevoNodo -> siguiente = NULL;
    ptr = inicio;
    while(ptr -> siguiente != NULL)
        ptr = ptr -> siguiente;
    ptr -> siguiente = nuevoNodo;
    return inicio;
}
```

3.7. Insertar un nodo en una posición anterior a un nodo de la lista

Algoritmo:

1 Declarar punteros para el nodo a insertar, el valor del puntero previo y el valor del puntero actual

2 Verificar si hay memoria, de lo contrario salir

3 Reservar espacio para nuevoNodo

4 Cargar el dato en el nuevoNodo

5 Apuntar ptr a inicio

6 Recorrer la lista hasta llegar al nodo en el cual se debe ingresar el nodo anterior. En cada iteracción actualizar el puntero actual (ptr) que va a detectar la posicion del nodo y el puntero con la posicion anterior(preptr)

7 Apuntar preptr− >siguiente a nuevoNodo

8 Apuntar nuevoNodo− >siguiente a ptr

```
Nodo *insertaAntes(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr, *preptr;
    int num, val;
    printf("\n Ingrese el valor del nodo que quiere agregar : ");
    scanf("%d", &num);
    printf("\n Ingrese el valor del nodo de la posicion posterior  : ");
    scanf("%d", &val);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    ptr = inicio;
    while(ptr -> dato != val)
        {
            preptr = ptr;
            ptr = ptr -> siguiente;
        }
    preptr -> siguiente = nuevoNodo;
    nuevoNodo -> siguiente = ptr;
    return inicio;
}
```

3.8. Insertar un nodo en una posición posterior a un nodo de la lista

El algoritmo es similar al anterior, en vez de recorrer la lista con el puntero actual, lo hago con el puntero anterior

```
Nodo *insertaDespues(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr, *preptr;
    int num, val;
    printf("\n Ingrese el valor del nodo que quiere agregar: ");
    scanf("%d", &num);
    printf("\n Ingrese el valor del nodo anterior : ");
    scanf("%d", &val);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    ptr = inicio;

    while(preptr -> dato != val)
        {
            preptr = ptr;
            ptr = ptr -> siguiente;
        }
    preptr -> siguiente=nuevoNodo;
    nuevoNodo -> siguiente = ptr;
    return inicio;
}
```

3.9. Borrar encabezado

```
Nodo *borraCabeza(Nodo *inicio)
{
     Nodo *ptr;
     ptr = inicio;
     inicio = inicio -> siguiente;
     free(ptr);
     return inicio;
}
```

3.10. Borrar último nodo(cola)

```
Nodo *borraCola(Nodo *inicio)
{
     Nodo *ptr, *preptr;
     ptr = inicio;
     while(ptr -> siguiente != NULL)
          {
               preptr = ptr;
               ptr = ptr -> siguiente;
          }
     preptr -> siguiente = NULL;
     free(ptr);
     return inicio;
}
```

3.11. Borrar un nodo específico

```
Nodo *borraNodo(Nodo *inicio)
{
    Nodo *ptr, *preptr;
    int val;
    printf("\n Ingrese el valor del nodo a borrar : ");
    scanf("%d", &val);
    ptr = inicio;
    if(ptr -> dato == val)
        {
            inicio = borraCabeza(inicio);
            return inicio;
        }
    else
        {
            while(ptr -> dato != val)
            {
                preptr = ptr;
                ptr = ptr -> siguiente;
            }
        preptr -> siguiente = ptr -> siguiente;
        free(ptr);
        return inicio;
}
}
```

3.12. Borrar un nodo posterior a uno dado

```
Nodo *borraDespues(Nodo *inicio)
{
    Nodo *ptr, *preptr;
    int val;
    printf("\n Ingrese el nodo anterior del que va a eliminar : ");
    scanf("%d", &val);
    ptr = inicio;
    while(preptr -> dato != val)
        {
            preptr = ptr;
            ptr = ptr -> siguiente;
        }
    preptr -> siguiente=ptr -> siguiente; // así tiene cargada la dirección
    free(ptr);
    return inicio;
}
```

3.13. Borrar lista

```
Nodo *borraLista(Nodo *inicio)
{
    Nodo *ptr;
    if(inicio!=NULL)  // verifica que la lista no está vacía
    {
        ptr=inicio;
        while(ptr != NULL)
            {
                printf("\n %d esta por ser borrado", ptr -> dato);
                inicio = borraCabeza(ptr);
                ptr = inicio;
            }
    }
    return inicio;
}
```

3.14. Ordenar lista

```
Nodo *ordenaLista(Nodo *inicio)
{
    Nodo *ptr1, *ptr2;
    int temp;
    ptr1 = inicio;
    while(ptr1 -> siguiente != NULL)  // ordenamiento burbuja
        {
            ptr2 = ptr1 -> siguiente;
            while(ptr2 != NULL)
                {
                    if(ptr1 -> dato > ptr2 -> dato)
                    {
                        temp = ptr1 -> dato;
                        ptr1 -> dato = ptr2 -> dato;
                        ptr2 -> dato = temp;
                    }
                    ptr2 = ptr2 -> siguiente;
                }
            ptr1 = ptr1 -> siguiente;
        }
    return inicio;
}
```

4. Problemas

1 Modifique el programa ejemploLista.c para que evite cargar datos duplicados.

2 Modifique el programa ejemploListaCircular.c para multiplicar todos los datos por un número ingresado por teclado.

3 Modifique el programa ejemploLista.c para que invierta el orden de los nodos utilizando recursión.

4 Modifique el programa ejemploLista.c para que imprima el valor máximo y mínimo de los nodos.

5 El programa para ubicar un disco de una colección de música que se encuentra ordenada en estantes debe tener los siguientes datos:

- a Nombre el disco
- b Grupo musical
- c Soporte (vinilo, CD, cassette)
- d Año de lanzamiento
- e Categoría (clásica, jazz, folclore, tango, etc)
- f Ubicación (estante en el cual se encuentra)

5.1 La base de datos debe crecer y decrecer dinámicamente.

5.2 Debe permitir buscar y mostrar la búsqueda de cualquiera de los datos miembro.

5.3 Debe permitir el ordenamiento.

NOTA: Se envía por autogestión los programas de ejemplo.

5. Apéndice Programas

5.1.

```
/* Diseun programa que utilice las posibles operaciones con listas enlazadas*/
#include <stdio.h>
#include <stdlib.h>

typedef int Item;
typedef struct Elemento
{
Item dato;  /* De esta forma si en vez de un int
        se requiere un float solo se cambia en la definicion de Item*/
struct Elemento *siguiente;
}Nodo;
Nodo *inicio = NULL;  // inicializa la lista vac
Nodo *creaLista(Nodo*); // funciue crea la lista
Nodo *muestraLista(Nodo*);
Nodo *insertaCabeza(Nodo*);
Nodo *insertaCola(Nodo*);
Nodo *insertaAntes(Nodo*);
Nodo *insertaDespues(Nodo*);
Nodo *borraCabeza(Nodo*);
Nodo *borraCola(Nodo*);
Nodo *borraNodo(Nodo*);
Nodo *borraDespues(Nodo *);
Nodo *borraLista(Nodo *);
Nodo *ordenaLista(Nodo *);
int main() {
int opcion;
do
{
 printf("\n\n ***** MENU PRINCIPAL *****");
 printf("\n 1: Crear una lista enlazada");
 printf("\n 2: Mostrar la lista");
 printf("\n 3: Agregar el nodo cabecera");
 printf("\n 4: Agregar el nodo cola");
 printf("\n 5: Agregar un nodo antes de un nodo dado");
 printf("\n 6: Agregar un nodo a continuacion de un nodo dado");
 printf("\n 7: Borrar nodo cabecera");
 printf("\n 8: Borrar nodo cola");
 printf("\n 9: Borrar un nodo dado");
 printf("\n 10: Borrar nodo siguiente a un nodo dado");
 printf("\n 11: Borrar la lista completa");
 printf("\n 12: Ordenar la lista");
 printf("\n 13: Salir");
 printf("\n\n Ingrese su opcion : ");
 scanf("%d", &opcion);
 switch(opcion)
 {
 case 1: inicio = creaLista(inicio);
 printf("\n LISTA ENLAZADA CREADA ");
 break;
 case 2: inicio = muestraLista(inicio);
 break;
 case 3: inicio = insertaCabeza(inicio);
 break;
 case 4: inicio = insertaCola(inicio);
 break;
 case 5: inicio = insertaAntes(inicio);
 break;
 case 6: inicio = insertaDespues(inicio);
 break;
 case 7: inicio = borraCabeza(inicio);
 break;
 case 8: inicio = borraCola(inicio);
 break;
 case 9: inicio = borraNodo(inicio);
 break;
 case 10: inicio = borraDespues(inicio);
 break;
 case 11: inicio = borraLista(inicio);
 printf("\n LISTA ENLAZADA BORRADA ");
 break;
 case 12: inicio = ordenaLista(inicio);
 break;
 }
}while(opcion !=13);
getchar();
return 0;
}

/* Funciue crea una lista enlazada */
Nodo *creaLista(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr; // punteros de tipo Nodo
    int num;
    printf("\n Ingrese -1 para finalizar\n");
    printf("\n Ingrese el valor del nodo : ");
    scanf("%d", &num); // carga el valor a guardar en dato en la estructura
    while(num != -1)
        {
            nuevoNodo = (Nodo*)malloc(sizeof(Nodo)); // reserva la memoria para el nodo
            nuevoNodo->dato = num; // carga el valor en dato
            if(inicio==NULL)  // pregunta si la lista esta vacia
                {
                    nuevoNodo -> siguiente = NULL; // el primer nodo ingresado es el ultimo de la lista
                    inicio = nuevoNodo;  // ocupa la cabecera tambien al ser el unico nodo
                }
            else  // la lista no esta vacia
                {
                    ptr=inicio; //apunta a la direccion del primer nodo
```

```
                    while(ptr->siguiente != NULL) // mientras no llegue a la cola
                        ptr = ptr->siguiente; // recorre la lista hasta llegar a la cola
                    ptr->siguiente = nuevoNodo; // el nuevo nodo se coloca entre el nodo anterior y NULL
                    nuevoNodo->siguiente=NULL; // el ultimo nodo ingresado apunta a nodo
                }
            printf("\n Ingrese el valor del nodo : ");
            scanf("%d", &num);
        }
return inicio;  // devuelve la direccion de la nueva cabecera
}

/* Funcion que despliega los nodos ingresados a la lista */
Nodo *muestraLista(Nodo *inicio)
{
Nodo *ptr;                      // puntero de tipo nodo
ptr = inicio;                   // apunta a la cabecera
while(ptr != NULL)              // recorre la lista de la cabecera a la cola
{
 printf("\t %d", ptr -> dato);  // al ingresar por primera vez imprime la cabecera
 ptr = ptr -> siguiente;        // actualiza al siguiente elemento
}
return inicio;                  // devuelve la direccion del nodo cabecera
}

/* Funciue inserta un nodo cabecera */
Nodo *insertaCabeza(Nodo *inicio)
{
    Nodo *nuevoNodo;                            // puntero tipo Nodo
    int num;
    printf("\n Ingrese el valor del nodo : ");
    scanf("%d", &num);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));   // reserva memoria
    nuevoNodo -> dato = num;                    // dato que va a ocupar la cabecera
    nuevoNodo -> siguiente = inicio;            // la cabecera actual pasa a ser el segundo nodo
    inicio = nuevoNodo;                         // la cabecera ahora es la direccion de nuevoNodo
    return inicio;                              // devuelve la direccion de la nueva cabecera
}

/* Funciue inserta un nodo al final*/
Nodo *insertaCola(Nodo *inicio)
{
    Nodo *ptr, *nuevoNodo;                      // punteros tipo nodo
    int num;
    printf("\n Ingrese el valor del nodo : ");
    scanf("%d", &num);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));   // reserva memoria
    nuevoNodo -> dato = num;                    //carga datos
    nuevoNodo -> siguiente = NULL;              // al ser el ltimo debe apuntar a NULL
    ptr = inicio;                               // carga la direccion de la cabecera
    while(ptr -> siguiente != NULL)
        ptr = ptr -> siguiente;                 // recorre la lista hasta encontrar la cola
    ptr -> siguiente = nuevoNodo;               // carga la direccion de la cola con la del nuevoNodo
    return inicio;                              // devuelve el nodo cabecera
}

/* Funciue inserta un nodo previo a un nodo dado*/
Nodo *insertaAntes(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr, *preptr;     // inicializa punteros
    int num, val;
    printf("\n Ingrese el valor del nodo que quiere agregar : ");
    scanf("%d", &num);  // nodo a agregar
    printf("\n Ingrese el valor del nodo de la posicion posterior  : ");
    scanf("%d", &val);  // nodo posterior
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo)); // reserva memoria para el nodo
    nuevoNodo -> dato = num;  // carga el nuevo dato
    ptr = inicio;  // apunta a la cabecera
    while(ptr -> dato != val)  // busca el nodo de la lista cuya posicion quiero
        {
            preptr = ptr;  // actualizo el puntero previo al valor
            ptr = ptr -> siguiente; // actualizo el puntero actual al valor siguiente
        }
    preptr -> siguiente = nuevoNodo;  // el valor del puntero previo pasa a ser el del nodo insertado
    nuevoNodo -> siguiente = ptr;   // el nodo insertado apunta al nodo posterior que se quer
    return inicio;
}

/* Insertar nodo en la posicion posterior a un nodo dado*/
Nodo *insertaDespues(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr, *preptr;
    int num, val;
    printf("\n Ingrese el valor del nodo que quiere agregar: ");
    scanf("%d", &num);
    printf("\n Ingrese el valor del nodo anterior : ");
    scanf("%d", &val);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    ptr = inicio;

    while(preptr -> dato != val)    // En vez de recorrer con el puntero actual lo hago con el puntero previo
        {
            preptr = ptr;
            ptr = ptr -> siguiente;
        }
    preptr -> siguiente=nuevoNodo;
    nuevoNodo -> siguiente = ptr;
    return inicio;
}

/* Funciue borra el encabezado */
Nodo *borraCabeza(Nodo *inicio)
{
```

```
    Nodo *ptr;
    ptr = inicio;                   //apunta al nodo cabeza
    inicio = inicio -> siguiente;   // la cabeza pasa a ser el segundo nodo
    free(ptr);                      //borra la direccion
    return inicio;                  // devuelve nuevo encabezadp
}

/* Funciue borra el ltimo nodo (cola)*/
Nodo *borraCola(Nodo *inicio)
{
    Nodo *ptr, *preptr;
    ptr = inicio;
    while(ptr -> siguiente != NULL)
        {
            preptr = ptr;
            ptr = ptr -> siguiente;
        }
    preptr -> siguiente = NULL;         //puntero penltimo lo hago apuntar a Null para cerrar la lista
    free(ptr);
    return inicio;
}

/*Funciue borra un nodo especco*/
Nodo *borraNodo(Nodo *inicio)
{
    Nodo *ptr, *preptr;
    int val;
    printf("\n Ingrese el valor del nodo a borrar : ");
    scanf("%d", &val);
    ptr = inicio;
    if(ptr -> dato == val)                  // verifica si es el encabezado y lo borra
        {
            inicio = borraCabeza(inicio);
            return inicio;
        }
    else
        {
            while(ptr -> dato != val)     // busca el nodo
            {
                preptr = ptr;
                ptr = ptr -> siguiente;
            }
        preptr -> siguiente = ptr -> siguiente;  // debe enlazar el dato anterior con el posterior
        free(ptr);  // borra el nodo
        return inicio;
}
}

/* Funciue borra un nodo posterior a uno dado*/
Nodo *borraDespues(Nodo *inicio)
{
    Nodo *ptr, *preptr;
    int val;
    printf("\n Ingrese el nodo anterior del que va a eliminar : ");
    scanf("%d", &val);
    ptr = inicio;
    while(preptr -> dato != val)
        {
            preptr = ptr;
            ptr = ptr -> siguiente;
        }
    preptr -> siguiente=ptr -> siguiente; // ptr tiene cargada la direcciel nodo a eliminar
    free(ptr);
    return inicio;
}

/* Funciue elimina toda la lista */
Nodo *borraLista(Nodo *inicio)
{
    Nodo *ptr;
    if(inicio!=NULL)  // verifica que la lista no estac
    {
        ptr=inicio;
        while(ptr != NULL)
            {
                printf("\n %d esta por ser borrado", ptr -> dato);
                inicio = borraCabeza(ptr);
                ptr = inicio;
            }
    }
    return inicio;
}

/* Funciue ordena la lista de menor a mayor */
Nodo *ordenaLista(Nodo *inicio)
{
    Nodo *ptr1, *ptr2;
    int temp;
    ptr1 = inicio;
    while(ptr1 -> siguiente != NULL)  // ordenamiento burbuja
        {
            ptr2 = ptr1 -> siguiente;
            while(ptr2 != NULL)
                {
                    if(ptr1 -> dato > ptr2 -> dato)
                    {
                        temp = ptr1 -> dato;
                        ptr1 -> dato = ptr2 -> dato;
                        ptr2 -> dato = temp;
                    }
                    ptr2 = ptr2 -> siguiente;
                }
            ptr1 = ptr1 -> siguiente;
```

```
        }
    return inicio;
}
```

5.2.

```
/* Disen programa que construya una lista enlazada circular
y ejecute sus posibles operaciones */
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
 int dato;
 struct node *siguiente;
}Nodo;

Nodo *inicio = NULL;
Nodo *crearCircular(Nodo *);
Nodo *muestraLista(Nodo *);
Nodo *insertaCabeza(Nodo *);
Nodo *insertaCola(Nodo *);
Nodo *borraCabeza(Nodo *);
Nodo *borraCola(Nodo *);
Nodo *borraDespues(Nodo *);
Nodo *borraLista(Nodo *);

int main()
{
 int opcion;

 do
    {
        printf("\n\n *****MAIN MENU *****");
        printf("\n 1: Crear una lista circular");
        printf("\n 2: Desplegar la lista");
        printf("\n 3: Agregar un nodo al principio");
        printf("\n 4: Agregar un nodo al final");
        printf("\n 5: Borrar el primer nodo");
        printf("\n 6: Borrar el ultimo nodo");
        printf("\n 7: Borrar el nodo siguiente a uno dado");
        printf("\n 8: Borrar la lista");
        printf("\n 9: SALIR");
        printf("\n\n Ingrese Opcion  : ");
        scanf("%d", &opcion);
        switch(opcion)
        {
            case 1:
                inicio = crearCircular(inicio);
                printf("\n LISTA CIRCULAR CREADA");
                break;
            case 2:
                inicio = muestraLista(inicio);
                break;
            case 3:
                inicio = insertaCabeza(inicio);
                break;
            case 4:
                inicio = insertaCola(inicio);
                break;
            case 5:
                inicio = borraCabeza(inicio);
                break;
            case 6:
                inicio = borraCola(inicio);
                break;
            case 7:
                inicio = borraDespues(inicio);
                break;
            case 8:
                inicio = borraLista(inicio);
                printf("\n LISTA CIRCULAR BORRADA");
                break;
            }
        }while(opcion !=9);
 getchar();
 return 0;
}

Nodo *crearCircular(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr;
    int num;
    printf("\n Ingrese -1 para finalizar");
    printf("\n Ingrese el dato : ");
    scanf("%d", &num);

    while(num != -1)
        {
            nuevoNodo = (Nodo*)malloc(sizeof(Nodo));
            nuevoNodo -> dato = num;
            if(inicio == NULL)
                {
                    nuevoNodo -> siguiente = nuevoNodo;
                    inicio = nuevoNodo;
                }
            else
                {
                    ptr = inicio;
                    while(ptr -> siguiente != inicio)
                        ptr = ptr -> siguiente;
```

```
                    ptr -> siguiente = nuevoNodo;
                    nuevoNodo -> siguiente = inicio; // Al ser circular ya no apunta a NULL
                }
                printf("\n Enter the dato : ");
                scanf("%d", &num);
            }
        return inicio;
}

Nodo *muestraLista(Nodo *inicio)
{
    Nodo *ptr;
    ptr=inicio;
    while(ptr -> siguiente != inicio) // No busca NULL al ser circular
        {
            printf("\t %d", ptr -> dato);
            ptr = ptr -> siguiente;
        }
    printf("\t %d", ptr -> dato);
    return inicio;
}

Nodo *insertaCabeza(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr;
    int num;
    printf("\n Enter the dato : ");
    scanf("%d", &num);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    ptr = inicio;
    while(ptr -> siguiente != inicio)
        ptr = ptr -> siguiente;
    ptr -> siguiente = nuevoNodo;
    nuevoNodo -> siguiente = inicio;
    inicio = nuevoNodo;
    return inicio;
}

Nodo *insertaCola(Nodo *inicio)
{
    Nodo *ptr, *nuevoNodo;
    int num;
    printf("\n Ingrese el dato : ");
    scanf("%d", &num);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    ptr = inicio;
    while(ptr -> siguiente != inicio)
        ptr = ptr -> siguiente;
    ptr -> siguiente = nuevoNodo;
    nuevoNodo -> siguiente = inicio;
    return inicio;
}

Nodo *borraCabeza(Nodo *inicio)
{
    Nodo *ptr;
    ptr = inicio;
    while(ptr -> siguiente != inicio)
        ptr = ptr -> siguiente;
    ptr -> siguiente = inicio -> siguiente;
    free(inicio);
    inicio = ptr -> siguiente;
    return inicio;
}

Nodo *borraCola(Nodo *inicio)
{
    Nodo *ptr, *preptr;
    ptr = inicio;
    while(ptr -> siguiente != inicio)
        {
            preptr = ptr;
            ptr = ptr -> siguiente;
        }
    preptr -> siguiente = ptr -> siguiente;
    free(ptr);
    return inicio;
}

Nodo *borraDespues(Nodo *inicio)
{
    Nodo *ptr, *preptr;
    int val;
    printf("\n Ingrese el valor anterior del nodo a eliminar : ");
    scanf("%d", &val);
    ptr = inicio;
    preptr = ptr;
    while(preptr -> dato != val)
        {
            preptr = ptr;
            ptr = ptr -> siguiente;
        }
    preptr -> siguiente = ptr -> siguiente;
    if(ptr == inicio)
        inicio = preptr -> siguiente;
    free(ptr);
    return inicio;
}

Nodo *borraLista(Nodo *inicio)
{
```

```
    Nodo *ptr;
    ptr = inicio;
    while(ptr -> siguiente != inicio)
        inicio = borraCola(inicio);
    free(inicio);
    return inicio;
}
```

5.3.

```
/* Escriba un programa que crea una lista doblemente enlazada*/

#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
struct node *siguiente;  //    |anterior|<-|nodo|->|siguiente|
int dato;
struct node *anterior;
}Nodo;

Nodo *inicio = NULL;   // Inicaliza puntero nodo a uno vac
Nodo *creaEnlaceDoble(Nodo *);  // Crea puntero a Nodo
Nodo *muestraListaDoble(Nodo *);
Nodo *insertaCabeza(Nodo *);
Nodo *insertaFinal(Nodo *);
Nodo *insertaAntes(Nodo *);
Nodo *insertaDespues(Nodo *);
Nodo *borraCabeza(Nodo *);
Nodo *borraFinal(Nodo *);
Nodo *borraAntes(Nodo *);
Nodo *borraDespues(Nodo *);
Nodo *borraListaDoble(Nodo *);
int main()
{
    int opcion;
    do
        {
            printf("\n\n ***** MENU PRINCIPAL *****");
            printf("\n 1: Crear la lista doblemente enlazada");
            printf("\n 2: Mostrar la lista");
            printf("\n 3: Agregar un nodo al principio");
            printf("\n 4: Agregar un nodo al final");
            printf("\n 5: Agregar un nodo antes de un nodo dado");
            printf("\n 6: Agregar un nodo despues de un nodo dado");
            printf("\n 7: Borrar el nodo cabecera");
            printf("\n 8: Borrar el nodo final");
            printf("\n 9: Borrar un nodo previo a un nodo dado");
            printf("\n 10: Borrar un nodo posterior a un nodo dado");
            printf("\n 11: Borrar la lista completa");
            printf("\n 12: Salir del programa");
            printf("\n\n Ingrese la opcion : ");
            scanf("%d", &opcion);
            switch(opcion)
            {
                case 1:
                    inicio = creaEnlaceDoble(inicio);
                    printf("\n Se ha creado una lista doblemente enlazada");
                    break;
                case 2:
                    inicio = muestraListaDoble(inicio);
                    break;
                case 3:
                    inicio = insertaCabeza(inicio);
                    break;
                case 4:
                    inicio = insertaFinal(inicio);
                    break;
                case 5:
                    inicio = insertaAntes(inicio);
                    break;
                case 6:
                    inicio = insertaDespues(inicio);
                    break;
                case 7:
                    inicio = borraCabeza(inicio);
                    break;
                case 8:
                    inicio = borraFinal(inicio);
                    break;
                case 9:
                    inicio = borraAntes(inicio);
                    break;
                case 10:
                    inicio = borraDespues(inicio);
                    break;
                case 11:
                    inicio = borraListaDoble(inicio);
                    printf("\n Ha eliminado la lista doblemente enlazada");
                    break;
            }
        }while(opcion != 12);
        getchar();
        return 0;
}

/* Crea puntero a Nodo que tiene doble enlace*/
Nodo *creaEnlaceDoble(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr;
```

```
    int num;
    printf("\n Ingrese -1 para finalizar");
    printf("\n Ingrese el dato : ");
    scanf("%d", &num);
    while(num != -1)
        {
            if(inicio == NULL)  // La lista no an no ha sido creada
            {
                nuevoNodo = (Nodo*)malloc(sizeof(Nodo)); // reserva memoria para el nodo
                nuevoNodo -> anterior = NULL;  // no hay nodo anterior al cual apuntar
                nuevoNodo -> dato = num; // carga el dato
                nuevoNodo -> siguiente = NULL;  // no hay nodo siguiente al cual apuntar
                inicio = nuevoNodo;  // el nodo pasa a ser la cabecera
            }
            else  // Si ya tengo al menos un nodo en la lista
            {
                ptr=inicio;  // posiciona el puntero en la cabecera
                nuevoNodo = (Nodo*)malloc(sizeof(Nodo)); // reserva memoria para el nuevo nodo
                nuevoNodo->dato=num; // carga el dato en el nuevo nodo
                while(ptr->siguiente!=NULL) // recorre la lista hasta llegar al ltimo nodo
                    ptr = ptr->siguiente;
                ptr->siguiente = nuevoNodo; // inserta el nuevo nodo al final
                nuevoNodo->anterior=ptr;  // el nuevo nodo apunta a su predecesor
                nuevoNodo->siguiente=NULL; // el nuevo nodo al estar al final apunta a NULL
            }
            printf("\n Ingrese el dato : ");  // Si ingresa distinto a -1 crea un nuevo nodo
            scanf("%d", &num);
        }
    return inicio;  // devuelve la direcciel nodo puntero
}

/* Despliega los valores de la lista deblemente enlazada */
Nodo *muestraListaDoble(Nodo *inicio)
{
    if(inicio == NULL)
    {
        printf("\nLista vacia cree una nueva lista\n");
    }
    else
    {
        Nodo *ptr;
        ptr=inicio;
        while(ptr!=NULL)
        {
            printf("\t %d", ptr -> dato); // recorre la lista e imprime el dato
            ptr = ptr -> siguiente;
        }
    }
    return inicio;
}

/*Inserta una nueva cabecera*/
Nodo *insertaCabeza(Nodo *inicio)
{
    Nodo *nuevoNodo;
    int num;
    printf("\n Ingrese nueva cabecera : ");
    scanf("%d", &num);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    inicio -> anterior = nuevoNodo; // en vez de apuntar a NULL la cebecera actual apunta al nuevo Nodo
    nuevoNodo -> siguiente = inicio; // el nuevo nodo pasa a ser la cabecera, y apunta a la cabecera vieja
    nuevoNodo -> anterior = NULL;  // el puntero a la posicion anterior de la cebecera apunta a NULL
    inicio = nuevoNodo; // actualiza la direccion de la cabecera
    return inicio;
}

/*Inserta nodo al final*/
Nodo *insertaFinal(Nodo *inicio)
{
    Nodo *ptr, *nuevoNodo;
    int num;
    printf("\n Ingrese el dato final : ");
    scanf("%d", &num);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    ptr=inicio;
    while(ptr -> siguiente != NULL) // busca el nodo final
        ptr = ptr -> siguiente;
    ptr -> siguiente = nuevoNodo;
    nuevoNodo -> anterior = ptr;
    nuevoNodo -> siguiente = NULL;
    return inicio;
}

/* Inserta un nodo antes de otro nodo*/
Nodo *insertaAntes(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr;
    int num, val;
    printf("\n Ingrese el valor del nodo nuevo : ");
    scanf("%d", &num);
    printf("\n Ingrese el valor del nodo posterior : ");
    scanf("%d", &val);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    ptr = inicio;
    while(ptr -> dato != val)  // recorre la lista hasta encontrar el nodo
        ptr = ptr -> siguiente;
    nuevoNodo -> siguiente = ptr; // el puntero queda a continuacion del nuevo nodo
    nuevoNodo -> anterior = ptr-> anterior; // el enlace previo del nuevo nodo
                                            // apunta a el enlace previo del puntero
    ptr -> anterior -> siguiente = nuevoNodo; // el nodo anterior al puntero debe poner su puntero siguiente a nuevo nodo
```

```
    ptr -> anterior = nuevoNodo; // el puntero anterior debe ahora apuntar al nuevo nodo
    return inicio;
}

/* Inserta un nodo despude un nuevo nodo */
Nodo *insertaDespues(Nodo *inicio)
{
    Nodo *nuevoNodo, *ptr;
    int num, val;
    printf("\n Ingrese el valor del nuevo nodo: ");
    scanf("%d", &num);
    printf("\n Ingrese el valor del nodo anterior : ");
    scanf("%d", &val);
    nuevoNodo = (Nodo *)malloc(sizeof(Nodo));
    nuevoNodo -> dato = num;
    ptr = inicio;
    while(ptr -> dato != val)
        ptr = ptr -> siguiente;
    nuevoNodo -> anterior = ptr;
    nuevoNodo -> siguiente = ptr -> siguiente;
    ptr -> siguiente -> anterior = nuevoNodo;
    ptr -> siguiente = nuevoNodo;
    return inicio;
}

/* Borra la cabecera */
Nodo *borraCabeza(Nodo *inicio)
{
    Nodo *ptr;
    ptr = inicio;
    if(ptr->siguiente != NULL)
    {
        inicio = inicio -> siguiente; // el segundo nodo pasa a ser la cabecera
        inicio -> anterior = NULL;  // el segundo nodo pone su puntero anterior a NULL
        free(ptr);  // la antigua cabecera quedsligada y se puede eliminar
    }
    else
    {
        inicio = NULL;
        free(ptr);

    }
    return inicio;
}

/* Borra final */
Nodo *borraFinal(Nodo *inicio)
{
    Nodo *ptr;
    ptr = inicio;
    while(ptr -> siguiente != NULL)
        ptr = ptr -> siguiente;
    ptr -> anterior -> siguiente = NULL;
    free(ptr);
    return inicio;
}

Nodo *borraAntes(Nodo *inicio)
{
    Nodo *ptr, *temp;
    int val;
    printf("\n Ingrese el valor posterior al nodo a ser eliminado : ");
    scanf("%d", &val);
    ptr = inicio;
    while(ptr -> dato != val)
        ptr = ptr -> siguiente;
    temp = ptr -> anterior;
    if(temp == inicio)
        inicio = borraCabeza(inicio);
    else
        {
            ptr -> anterior = temp -> anterior;
            temp -> anterior -> siguiente = ptr;
        }
        free(temp);
        return inicio;
}

/*Borra el nodo a continuacion de un nodo dado*/
Nodo *borraDespues(Nodo *inicio)
{
    Nodo *ptr, *temp;
    int val;
    printf("\n Ingrese el valor previo del nodo a ser eliminado : ");
    scanf("%d", &val);
    ptr = inicio;
    while(ptr -> dato != val)
        ptr = ptr -> siguiente;
    temp = ptr -> siguiente;
    ptr -> siguiente = temp -> siguiente;
    temp -> siguiente -> anterior = ptr;
    free(temp);
    return inicio;
}

/* Elimina la lista completa */
Nodo *borraListaDoble(Nodo *inicio)
{
    while(inicio != NULL) // elimina las cabeceras a medida que recorre la lista
        inicio = borraCabeza(inicio);

    return inicio;
}
```