ListaLigadaSimple

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

struct nodo{
        int dato;
        struct nodo * siguiente;
};

struct listaLigada{
   struct nodo * cabecera;
};

struct nodo * crearNodo(int);
struct listaLigada * crearLista();
void desplegar(struct listaLigada *);
void insertar_inicio(struct listaLigada *, int);
void insertar_final(struct listaLigada *, int);
void insertar_intermedio(struct listaLigada *, int, int);
void borrar_inicio(struct listaLigada *);
void borrar_final(struct listaLigada *);
void borrar_intermedio(struct listaLigada *, int);
int contar_nodos(struct listaLigada * cabecera);

int main(){

        // Desplegando nodos en la lista
        struct listaLigada * l1 = crearLista(); // l1 es una instancia de Lista
Ligada
        // Probando las funciones implementadas
        printf("\nOperaciones de insercion: \n\n");
        // Insertando nodos al principio
        insertar_inicio(l1, 23);
        insertar_inicio(l1, 12);
        insertar_inicio(l1, 76);
        insertar_inicio(l1, 93);
```

```
        insertar_inicio(l1, 8);
        desplegar(l1);
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
        // Insertando nodos al final
        insertar_final(l1, 9);
        insertar_final(l1, 7);
        insertar_final(l1, -5);
        insertar_final(l1, 16);
        desplegar(l1);
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
        // Insertando nodos en una posicion intermedia
        insertar_intermedio(l1, 0, 14);
        insertar_intermedio(l1, 5, 25);
        insertar_intermedio(l1, 2, 17);
        desplegar(l1);
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));

        /*printf("\n\nOperaciones de borrado: \n\n");
        // Borrando nodos al principio
        cabecera = borrar_inicio(cabecera);
        cabecera = borrar_inicio(cabecera);
        cabecera = borrar_inicio(cabecera);
        desplegar(cabecera);
        printf("Numero de Nodos en la Lista: %d\n",
contar_nodos(cabecera));
        // Borrando nodos al final
        cabecera = borrar_final(cabecera);
        cabecera = borrar_final(cabecera);
        desplegar(cabecera);
        printf("Numero de Nodos en la Lista: %d\n",
contar_nodos(cabecera));
        // Borrando nodos en una posicion intermedia
        cabecera = borrar_intermedio(cabecera, 1);
        cabecera = borrar_intermedio(cabecera, 4);
        cabecera = borrar_intermedio(cabecera, 0);
        desplegar(cabecera);*/
```

```
	return 0;
}

struct nodo * crearNodo(int x){
	struct nodo * nuevo = NULL;
	nuevo = (struct nodo *) malloc(sizeof(struct nodo));
	if(nuevo == NULL) exit(0);
	nuevo->dato = x;
	nuevo->siguiente = NULL;
	return nuevo;
}

struct listaLigada * crearLista(){
   struct listaLigada * nuevaLista = NULL;
   nuevaLista = (struct listaLigada *) malloc(sizeof(struct listaLigada));
   if(nuevaLista == NULL) return NULL;
   nuevaLista->cabecera = NULL;
   return nuevaLista;
}

void insertar_inicio(struct listaLigada * lista, int x){
	struct nodo * n = crearNodo(x);
	if(lista->cabecera == NULL){
		lista->cabecera = n;
	} else{
		n->siguiente = lista->cabecera;
		lista->cabecera = n;
	}
}

void insertar_final(struct listaLigada * lista, int x){
	struct nodo * n = crearNodo(x);
	if(lista->cabecera == NULL){
		lista->cabecera = n;
	} else{
		struct nodo * temp = lista->cabecera;
		while(temp->siguiente != NULL){
```

```
                temp = temp->siguiente;
            }
            temp->siguiente = n;
        }
}

void borrar_inicio(struct listaLigada * lista){
        if(lista->cabecera == NULL){
            printf("\nLista Vacia");
        } else{
            struct nodo * aux = lista->cabecera;
            lista->cabecera = lista->cabecera->siguiente;
            free(aux);
        }
}

void borrar_final(struct listaLigada * lista){
        struct nodo * temp = NULL, * prev = NULL;
        if(lista->cabecera == NULL){
            printf("\nLista Vacia");
        } else{
            temp = lista->cabecera;
            while(temp->siguiente != NULL){
                prev = temp;
                temp = temp->siguiente;
            }
            if(temp == lista->cabecera){
                // Si solo queda un nodo en la lista
                lista->cabecera = NULL;
            } else{
                prev->siguiente = NULL;
            }
            free(temp);
        }
}

int contar_nodos(struct listaLigada * lista){
```

```
        if(lista->cabecera == NULL){
                return 0;
        } else{
                int cont = 0;
                struct nodo * temp = lista->cabecera;
                while(temp != NULL){
                        cont++;
                        temp = temp->siguiente;
                }
                return cont;
        }
}

void insertar_intermedio(struct listaLigada * lista, int pos, int x){
   struct nodo * nuevo = crearNodo(x);
   // Se toma el 0 como posicion valida
   int noNodos = contar_nodos(lista);
   if(pos<-1 || pos>noNodos+1){
      printf("Posicion invalida\n");
   } else if(pos == 0){
                if(lista->cabecera == NULL) {
       lista->cabecera = nuevo;
       } else{
                        nuevo->siguiente = lista->cabecera;
       lista->cabecera = nuevo;
                }
   } else{
      struct nodo * temp = lista->cabecera;
      int i=0;
      while(i<pos-1){ // <-- Linea corregida
         temp = temp->siguiente; // (pos-1)-th nodo
         i++;
      }
      nuevo->siguiente = temp->siguiente;
      temp->siguiente = nuevo;
   }
}
```

```
void borrar_intermedio(struct listaLigada * lista, int pos){
    if(lista->cabecera == NULL) {
        printf("Lista Vacia!!\n");
    } else {
        // Se toma el 0 como posicion valida
        int noNodos = contar_nodos(lista);
        if(pos<-1 || pos>noNodos){
            printf("Posicion invalida\n");
        } else {
            struct nodo * temp = NULL;
            if(pos == 0){
                temp = lista->cabecera;
                lista->cabecera = lista->cabecera->siguiente;
                free(temp);
            } else{
                int i=0;
                temp = lista->cabecera;
                while(i<pos-1){ // <-- Linea corregida
                    temp = temp->siguiente; // (pos-1)-th nodo
                    i++;
                }
                struct nodo * borrado = temp->siguiente; // (pos)-th nodo
                temp->siguiente = borrado->siguiente; // (pos+1)-th nodo
                free(borrado);
            }
        }
    }
}

void desplegar(struct listaLigada * lista){
        struct nodo * temp = lista->cabecera;
        if(temp == NULL){
                printf("Lista Vacia");
        } else{
                while(temp != NULL){
                        printf("%d->",temp->dato);
```

```
                temp = temp->siguiente;
            }
            printf("NULL\n");
        }
}
```

ListaLigadaSimple2Apuntadores

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

struct nodo{
        int dato;
        struct nodo * siguiente;
};

struct listaLigada{
    struct nodo * cabecera;
    struct nodo * final;
};

struct nodo * crearNodo(int);
struct listaLigada * crearLista();
void desplegar(struct listaLigada *);
void insertar_inicio(struct listaLigada *, int);
void insertar_final(struct listaLigada *, int);
void insertar_intermedio(struct listaLigada *, int, int);
void borrar_inicio(struct listaLigada *);
void borrar_final(struct listaLigada *);
void borrar_intermedio(struct listaLigada *, int);
int contar_nodos(struct listaLigada * cabecera);
int frente(struct listaLigada *);
int ultimo(struct listaLigada *);

int main(){
        // Desplegando nodos en la lista
```

```
        struct listaLigada * l1 = crearLista(); // l1 es una instancia de Lista
Ligada
        // Probando las funciones implementadas
        printf("\nOperaciones de insercion: \n\n");
        // Insertando nodos al principio
        insertar_inicio(l1, 23);
        insertar_inicio(l1, 12);
        insertar_inicio(l1, 76);
        insertar_inicio(l1, 93);
        insertar_inicio(l1, 8);
        desplegar(l1);
   if(frente(l1) != 0){
      printf("Primer nodo de la lista: %d\n", frente(l1));
   }
   if(ultimo(l1) != 0){
      printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
   }
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
        // Insertando nodos al final
        insertar_final(l1, 9);
        insertar_final(l1, 7);
        insertar_final(l1, -5);
        insertar_final(l1, 16);
        desplegar(l1);
   if(frente(l1) != 0){
      printf("Primer nodo de la lista: %d\n", frente(l1));
   }
   if(ultimo(l1) != 0){
      printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
   }
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
        // Insertando nodos en una posicion intermedia
        insertar_intermedio(l1, 0, 14);
        insertar_intermedio(l1, 5, 25);
        insertar_intermedio(l1, 2, 17);
   insertar_intermedio(l1, 12, 89);
        desplegar(l1);
```

```
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));

    printf("\n\nOperaciones de borrado: \n\n");
    // Borrando nodos al principio
borrar_inicio(l1);
    borrar_inicio(l1);
    borrar_inicio(l1);
    desplegar(l1);
if(frente(l1) != 0){
   printf("Primer nodo de la lista: %d\n", frente(l1));
}
if(ultimo(l1) != 0){
   printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
}
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
    // Borrando nodos al final
    borrar_final(l1);
    borrar_final(l1);
borrar_final(l1);
    borrar_final(l1);
borrar_final(l1);
    borrar_final(l1);
insertar_final(l1, -4);
    desplegar(l1);
if(frente(l1) != 0){
   printf("Primer nodo de la lista: %d\n", frente(l1));
}
if(ultimo(l1) != 0){
   printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
}
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
    // Borrando nodos en una posicion intermedia
borrar_intermedio(l1, 0);
borrar_intermedio(l1, 4);
    borrar_intermedio(l1, 1);
borrar_intermedio(l1, 2);
borrar_intermedio(l1, 0);
```

```
    borrar_intermedio(l1, 1);
    borrar_intermedio(l1, 0);
    insertar_inicio(l1, 1);
    borrar_intermedio(l1, 0);
        desplegar(l1);
    if(frente(l1) != 0){
       printf("Primer nodo de la lista: %d\n", frente(l1));
    }
    if(ultimo(l1) != 0){
       printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
    }
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));

        return 0;
}

struct nodo * crearNodo(int x){
        struct nodo * nuevo = NULL;
        nuevo = (struct nodo *) malloc(sizeof(struct nodo));
        if(nuevo == NULL) exit(0);
        nuevo->dato = x;
        nuevo->siguiente = NULL;
        return nuevo;
}

struct listaLigada * crearLista(){
    struct listaLigada * nuevaLista = NULL;
    nuevaLista = (struct listaLigada *) malloc(sizeof(struct listaLigada));
    if(nuevaLista == NULL) return NULL;
    nuevaLista->cabecera = NULL;
    nuevaLista->final = NULL;
    return nuevaLista;
}

void insertar_inicio(struct listaLigada * lista, int x){
        struct nodo * n = crearNodo(x);
        if(lista->cabecera == NULL){
```

```
        // Insertamos el primer nodo
                lista->cabecera = lista->final = n;
         } else{
                n->siguiente = lista->cabecera;
                lista->cabecera = n;
         }
}

void insertar_final(struct listaLigada * lista, int x){
         struct nodo * n = crearNodo(x);
         if(lista->cabecera == NULL){
                lista->cabecera = lista->final = n;
         } else{
        lista->final->siguiente = n;
        lista->final = n;
         }
}

void borrar_inicio(struct listaLigada * lista){
         if(lista->cabecera == NULL){
                printf("\nLista Vacia");
         } else{
                struct nodo * temp = lista->cabecera;
        if(lista->cabecera == lista->final){
           // Solo queda un nodo en la lista
           lista->cabecera = lista->final = NULL;
        } else{
           lista->cabecera = lista->cabecera->siguiente;
        }
                free(temp);
         }
}

void borrar_final(struct listaLigada * lista){
         if(lista->cabecera == NULL){
                printf("\nLista Vacia");
         } else{
```

```
                struct nodo * temp = lista->cabecera;
        if(lista->cabecera == lista->final){
           // Solo queda un nodo en la lista
           free(temp);
           lista->cabecera = lista->final = NULL;
        } else{
           while(temp->siguiente != lista->final){
                           temp = temp->siguiente;
                   }
           free(lista->final);
           lista->final = temp;
           lista->final->siguiente = NULL;
        }
         }
}

int contar_nodos(struct listaLigada * lista){
         if(lista->cabecera == NULL){
                return 0;
         } else{
                int cont = 0;
                struct nodo * temp = lista->cabecera;
                while(temp != NULL){
                        cont++;
                        temp = temp->siguiente;
                }
                return cont;
         }
}

void insertar_intermedio(struct listaLigada * lista, int pos, int x){
   struct nodo * nuevo = crearNodo(x);
   // Se toma el 0 como posicion valida
   int noNodos = contar_nodos(lista);
   if(pos<-1 || pos>noNodos+1){
      // Rango valido de pos: [0, pos]
      printf("Posicion invalida\n");
```

```
    } else if(pos == 0){
        if(lista->cabecera == NULL) {
            // Apenas vamos a insertar el primer nodo
            lista->cabecera = lista->final = nuevo;
        }  else{
            // El nuevo nodo será el primero
            nuevo->siguiente = lista->cabecera;
            lista->cabecera = nuevo;
        }
    } else if(pos == noNodos){
        // Si el valor de pos es igual al numero de nodoss
        // el nuevo nodo será el último de la lista
        lista->final->siguiente = nuevo;
        lista->final = nuevo;
    }else{
        // Insertamos un nodo en cualquier posición
        // que no sea la primera y la última
        struct nodo * temp = lista->cabecera;
        int i=0;
        while(i<pos-1){ // <-- Linea corregida
            temp = temp->siguiente; // (pos-1)-th nodo
            i++;
        }
        nuevo->siguiente = temp->siguiente;
        temp->siguiente = nuevo;
    }
}

void borrar_intermedio(struct listaLigada * lista, int pos){
    if(lista->cabecera == NULL) {
        printf("Lista Vacia!!\n");
        return;
    } else {
        // Se toma el 0 como posicion valida
        int noNodos = contar_nodos(lista);
        if(pos<-1 || pos>=noNodos){
            // Rango valido de pos: [0, pos-1]
```

```
            printf("Posicion invalida\n");
        } else {
            struct nodo * temp = NULL;
            if(pos == 0){
                temp = lista->cabecera;
                if(lista->cabecera == lista->final){
                    // Si solo queda un nodo en la lista
                    lista->cabecera = lista->final = NULL;
                } else{
                    // Si todavía hay más nodos
                    lista->cabecera = lista->cabecera->siguiente;
                }
                free(temp);
            } else{
                int i=0;
                struct nodo * temp = lista->cabecera;
                while(i<pos-1){ // <-- Linea corregida
                    // Recorremos la lista hasta el nodo previo a borrar
                    temp = temp->siguiente; // (pos-1)-th nodo
                    i++;
                }
                // Nos desplazamos al nodo a borrar
                struct nodo * borrado = temp->siguiente; // (pos)-th nodo
                // Si el nodo a borrar no es el último
                temp->siguiente = borrado->siguiente; // (pos+1)-th nodo
                if(temp->siguiente == NULL){
                    lista->final = temp;
                }
                free(borrado);
            }
        }
    }
}

int frente(struct listaLigada * lista){
    if(lista->cabecera == NULL){
        return 0;
```

```
    } else{
        return lista->cabecera->dato;
    }
}

int ultimo(struct listaLigada * lista){
    if(lista->cabecera == NULL){
        return 0;
    } else{
        return lista->final->dato;
    }
}

void desplegar(struct listaLigada * lista){
        struct nodo * temp = lista->cabecera;
        if(temp == NULL){
                printf("Lista Vacia\n");
        } else{
                while(temp != NULL){
                        printf("%d->",temp->dato);
                        temp = temp->siguiente;
                }
                printf("NULL\n");
        }
}
```

ListaLigadaDoble

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

struct nodo{
        int dato;
        struct nodo * siguiente;
    struct nodo * anterior;
};
```

```
struct listaLigada{
   struct nodo * cabecera;
};

struct nodo * crearNodo(int);
struct listaLigada * crearLista();
void desplegarAdelante(struct listaLigada *);
void desplegarAtras(struct listaLigada *);
void insertar_inicio(struct listaLigada *, int);
void insertar_final(struct listaLigada *, int);
void insertar_intermedio(struct listaLigada *, int, int);
void borrar_inicio(struct listaLigada *);
void borrar_final(struct listaLigada *);
void borrar_intermedio(struct listaLigada *, int);
int contar_nodos(struct listaLigada *);
int frente(struct listaLigada *);
int final(struct listaLigada *);

int main(){
        // Desplegando nodos en la lista
        struct listaLigada * l1 = crearLista(); // l1 es una instancia de Lista
Ligada
        // Probando las funciones implementadas
        printf("\nOperaciones de insercion: \n\n");
        // Insertando nodos al principio
        insertar_inicio(l1, 23);
        insertar_inicio(l1, 12);
        insertar_inicio(l1, 76);
        insertar_inicio(l1, 93);
        insertar_inicio(l1, 8);
        desplegarAdelante(l1);
   desplegarAtras(l1);
   if(frente(l1) != INT_MIN){
      printf("Nodo al frente de la lista: %d\n", frente(l1));
   }
   if(final(l1) != INT_MIN){
```

```
    printf("Nodo al final de la lista: %d\n", final(l1));
}
     printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
     // Insertando nodos al final
     insertar_final(l1, 9);
     insertar_final(l1, 7);
     insertar_final(l1, -5);
     insertar_final(l1, 16);
     desplegarAdelante(l1);
desplegarAtras(l1);
if(frente(l1) != INT_MIN){
    printf("Nodo al frente de la lista: %d\n", frente(l1));
}
if(final(l1) != INT_MIN){
    printf("Nodo al final de la lista: %d\n", final(l1));
}
     printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
     // Insertando nodos en una posicion intermedia
     insertar_intermedio(l1, 0, 14);
     insertar_intermedio(l1, 5, 25);
     insertar_intermedio(l1, 2, 17);
insertar_intermedio(l1, 12, 39);
desplegarAdelante(l1);
desplegarAtras(l1);
if(frente(l1) != INT_MIN){
    printf("Nodo al frente de la lista: %d\n", frente(l1));
}
if(final(l1) != INT_MIN){
    printf("Nodo al final de la lista: %d\n", final(l1));
}
     printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));

     printf("\n\nOperaciones de borrado: \n\n");
     // Borrando nodos al principio
     borrar_inicio(l1);
     borrar_inicio(l1);
     borrar_inicio(l1);
```

```
    desplegarAdelante(l1);
desplegarAtras(l1);
if(frente(l1) != INT_MIN){
   printf("Nodo al frente de la lista: %d\n", frente(l1));
}
if(final(l1) != INT_MIN){
   printf("Nodo al final de la lista: %d\n", final(l1));
}
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
    // Borrando nodos al final
    borrar_final(l1);
    borrar_final(l1);
    desplegarAdelante(l1);
desplegarAtras(l1);
if(frente(l1) != INT_MIN){
   printf("Nodo al frente de la lista: %d\n", frente(l1));
}
if(final(l1) != INT_MIN){
   printf("Nodo al final de la lista: %d\n", final(l1));
}
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
    // Borrando nodos en una posicion intermedia
    borrar_intermedio(l1, 1);
    borrar_intermedio(l1, 4);
    borrar_intermedio(l1, 0);
borrar_intermedio(l1, 4);
    desplegarAdelante(l1);
desplegarAtras(l1);
if(frente(l1) != INT_MIN){
   printf("Nodo al frente de la lista: %d\n", frente(l1));
}
if(final(l1) != INT_MIN){
   printf("Nodo al final de la lista: %d\n", final(l1));
}
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));

    return 0;
```

```
}

struct nodo * crearNodo(int x){
        struct nodo * nuevo = NULL;
        nuevo = (struct nodo *) malloc(sizeof(struct nodo));
        if(nuevo == NULL) exit(0);
        nuevo->dato = x;
        nuevo->siguiente = NULL;
    nuevo->anterior = NULL;
        return nuevo;
}

struct listaLigada * crearLista(){
    struct listaLigada * nuevaLista = NULL;
    nuevaLista = (struct listaLigada *) malloc(sizeof(struct listaLigada));
    if(nuevaLista == NULL) return NULL;
    nuevaLista->cabecera = NULL;
    return nuevaLista;
}

void insertar_inicio(struct listaLigada * lista, int x){
        struct nodo * nuevo = crearNodo(x);
        if(lista->cabecera == NULL){
       // Si la lista esta vacia apenas
       // insertamos el primer nodo
                lista->cabecera = nuevo;
        } else{
       // Si ya hay mas nodos en la lista
       // entonces el nuevo nodo será el primero
                nuevo->siguiente = lista->cabecera;
       lista->cabecera->anterior = nuevo;
                lista->cabecera = nuevo;
        }
}

void insertar_final(struct listaLigada * lista, int x){
        struct nodo * nuevo = crearNodo(x);
```

```
        if(lista->cabecera == NULL){
        // Si la lista esta vacia apenas
        // insertamos el primer nodo
                lista->cabecera = nuevo;
        } else{
        // Si ya hay mas nodos en la lista
                struct nodo * temp = lista->cabecera;
                while(temp->siguiente != NULL){
           // Recorremos la lista hasta llegar
           // al ultimo nodo
                        temp = temp->siguiente;
                }
        // La parte siguiente del nodo que actualmente
        // es el último apunta al nuevo nodo
                temp->siguiente = nuevo;
        // La parte anterior del nuevo nodo apunta al nodo
        // que era el ultimo
        nuevo->anterior = temp;
        }
}

void insertar_intermedio(struct listaLigada * lista, int pos, int x){
    struct nodo * nuevo = crearNodo(x);
    // Se toma el 0 como posicion valida
    int noNodos = contar_nodos(lista);
    if(pos<-1 || pos>noNodos+1){
        // Rango valido: [0, pos+1]
        printf("Posicion invalida\n");
    } else if(pos == 0){
        if(lista->cabecera == NULL){
            lista->cabecera = nuevo;
        } else{
            nuevo->siguiente = lista->cabecera;
            lista->cabecera->anterior = nuevo;
            lista->cabecera = nuevo;
        }
    } else{
```

```
        struct nodo * temp = lista->cabecera;
        int i=0;
        while(i<pos-1){ // <-- Linea corregida
            // Recorremos la lista hasta la posicion
            // anterior a la cual vamos a insertar el nodo
            temp = temp->siguiente; // (pos-1)-th nodo
            i++;
        }
        if(temp->siguiente == NULL){
            // El nuevo nodo se inserta al final
            temp->siguiente = nuevo;
            nuevo->anterior = temp;
        } else{
            nuevo->siguiente = temp->siguiente;
            temp->siguiente = nuevo;
            nuevo->anterior = temp;
            nuevo->siguiente->anterior = nuevo;
        }
    }
}

void borrar_inicio(struct listaLigada * lista){
         if(lista->cabecera == NULL){
        // No borramos nada si la lista
        // esta vacia
                printf("\nLista Vacia");
         } else{
        // Si la lista no esta vacia ahora
        // el segundo nodo de la lista pasa a ser
        // el primero y borramos el nodo anterior
                struct nodo * aux = lista->cabecera;
                lista->cabecera = lista->cabecera->siguiente;
        lista->cabecera->anterior = NULL;
                free(aux);
         }
}
```

```
void borrar_final(struct listaLigada * lista){
        struct nodo * temp = NULL, * prev = NULL;
        if(lista->cabecera == NULL){
      // No borramos nada si la lista
      // esta vacia
            printf("\nLista Vacia");
       } else{
            temp = lista->cabecera;
            while(temp->siguiente != NULL){
        // Recorremos la lista hasta llegar al
        // ultimo nodo (temp) y tambien almacenamos
        // el penultimo nodo (prev)
                    prev = temp;
                    temp = temp->siguiente;
            }
            if(temp == lista->cabecera){
                    // Si solo queda un nodo en la lista
                    lista->cabecera = NULL;
            } else{
                    prev->siguiente = NULL;
            }
            free(temp);
       }
}

void borrar_intermedio(struct listaLigada * lista, int pos){
   if(lista->cabecera == NULL) {
      // La lista esta vacia no borramos nada
      printf("Lista Vacia!!\n");
   } else {
      // Se toma el 0 como posicion valida
      int noNodos = contar_nodos(lista);
      if(pos<-1 || pos>=noNodos){
         // Rango valido: [0, pos]
         printf("Posicion invalida\n");
      } else {
         struct nodo * temp = NULL;
```

```
            if(pos == 0){
                temp = lista->cabecera;
                if(temp->siguiente == NULL){
                    // Solo queda un nodo en la lista
                    lista->cabecera = NULL;
                } else{
                    // Hay mas nodos en la lista
                    lista->cabecera = lista->cabecera->siguiente;
                    lista->cabecera->anterior = NULL;
                }
                free(temp);
            } else{
                int i=0;
                temp = lista->cabecera;
                while(i<pos-1){ // <-- Linea corregida
                    temp = temp->siguiente; // (pos-1)-th nodo
                    i++;
                }
                struct nodo * borrado = temp->siguiente; // (pos)-th nodo
                if(borrado->siguiente == NULL){
                    borrado->anterior->siguiente = NULL;
                } else{
                    temp->siguiente = borrado->siguiente; // (pos+1)-th nodo
                    borrado->siguiente->anterior = temp;
                }
                free(borrado);
            }
        }
    }
}

int contar_nodos(struct listaLigada * lista){
        if(lista->cabecera == NULL){
                return 0;
        } else{
                int cont = 0;
                struct nodo * temp = lista->cabecera;
```

```
                while(temp != NULL){
                        cont++;
                        temp = temp->siguiente;
                }
                return cont;
        }
}

void desplegarAdelante(struct listaLigada * lista){
        struct nodo * temp = lista->cabecera;
        if(temp == NULL){
                printf("Lista Vacia");
        } else{
                while(temp != NULL){
                        printf("%d->",temp->dato);
                        temp = temp->siguiente;
                }
                printf("NULL\n");
        }
}

void desplegarAtras(struct listaLigada * lista){
        struct nodo * temp = lista->cabecera;
        if(temp == NULL){
                printf("Lista Vacia");
        } else{
       while(temp->siguiente != NULL){
          // Nos desplazamos hasta el último nodo
          temp = temp->siguiente;
       }

                while(temp != NULL){
                        printf("%d->",temp->dato);
                        temp = temp->anterior;
                }
                printf("NULL\n");
        }
```

```
}

int frente(struct listaLigada * lista){
    if(lista->cabecera == NULL){
        return INT_MIN;
    } else{
        return lista->cabecera->dato;
    }
}

int final(struct listaLigada * lista){
    if(lista->cabecera == NULL){
        return INT_MIN;
    } else{
        // Recorremos la lista hasta el ultimo
        // nodo
        struct nodo * temp = lista->cabecera;
        while (temp->siguiente != NULL){
            temp = temp->siguiente;
        }
        return temp->dato;
    }
}
```

ListaLigadaDoble2Apuntadores

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

struct nodo{
         int dato;
         struct nodo * siguiente;
    struct nodo * anterior;
};

struct listaLigada{
```

```
    struct nodo * cabecera;
    struct nodo * final;
};

struct nodo * crearNodo(int);
struct listaLigada * crearLista();
void desplegarAdelante(struct listaLigada *);
void desplegarAtras(struct listaLigada *);
void insertar_inicio(struct listaLigada *, int);
void insertar_final(struct listaLigada *, int);
void insertar_intermedio(struct listaLigada *, int, int);
void borrar_inicio(struct listaLigada *);
void borrar_final(struct listaLigada *);
void borrar_intermedio(struct listaLigada *, int);
int contar_nodos(struct listaLigada * cabecera);
int frente(struct listaLigada *);
int ultimo(struct listaLigada *);

int main(){
        // Desplegando nodos en la lista
        struct listaLigada * l1 = crearLista(); // l1 es una instancia de Lista
Ligada
        // Probando las funciones implementadas
        printf("\nOperaciones de insercion: \n\n");
        // Insertando nodos al principio
        insertar_inicio(l1, 23);
        insertar_inicio(l1, 12);
        insertar_inicio(l1, 76);
        insertar_inicio(l1, 93);
        insertar_inicio(l1, 8);
        desplegarAdelante(l1);
    desplegarAtras(l1);
    if(frente(l1) != INT_MIN){
        printf("Primer nodo de la lista: %d\n", frente(l1));
    }
    if(ultimo(l1) != INT_MIN){
        printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
```

```
}
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
    // Insertando nodos al final
    insertar_final(l1, 9);
    insertar_final(l1, 7);
    insertar_final(l1, -5);
    insertar_final(l1, 16);
    desplegarAdelante(l1);
desplegarAtras(l1);
if(frente(l1) != INT_MIN){
   printf("Primer nodo de la lista: %d\n", frente(l1));
}
if(ultimo(l1) != INT_MIN){
   printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
}
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
    // Insertando nodos en una posicion intermedia
    insertar_intermedio(l1, 0, 14);
    insertar_intermedio(l1, 5, 25);
    insertar_intermedio(l1, 2, 17);
insertar_intermedio(l1, 12, 89);
    desplegarAdelante(l1);
desplegarAtras(l1);
if(frente(l1) != INT_MIN){
   printf("Primer nodo de la lista: %d\n", frente(l1));
}
if(ultimo(l1) != INT_MIN){
   printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
}
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));

    printf("\n\nOperaciones de borrado: \n\n");
    // Borrando nodos al principio
borrar_inicio(l1);
    borrar_inicio(l1);
    borrar_inicio(l1);
    desplegarAdelante(l1);
```

```
desplegarAtras(l1);
if(frente(l1) != INT_MIN){
    printf("Primer nodo de la lista: %d\n", frente(l1));
}
if(ultimo(l1) != INT_MIN){
    printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
}
     printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
     // Borrando nodos al final
     borrar_final(l1);
     borrar_final(l1);
borrar_final(l1);
     borrar_final(l1);
borrar_final(l1);
     borrar_final(l1);
insertar_final(l1, -4);
     desplegarAdelante(l1);
desplegarAtras(l1);
if(frente(l1) != INT_MIN){
    printf("Primer nodo de la lista: %d\n", frente(l1));
}
if(ultimo(l1) != INT_MIN){
    printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
}
     printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
     // Borrando nodos en una posicion intermedia
borrar_intermedio(l1, 0);
borrar_intermedio(l1, 4);
     borrar_intermedio(l1, 1);
borrar_intermedio(l1, 2);
borrar_intermedio(l1, 0);
borrar_intermedio(l1, 1);
borrar_intermedio(l1, 0);
insertar_inicio(l1, 1);
borrar_intermedio(l1, 0);
     desplegarAdelante(l1);
desplegarAtras(l1);
```

```
    if(frente(l1) != INT_MIN){
        printf("Primer nodo de la lista: %d\n", frente(l1));
    }
    if(ultimo(l1) != INT_MIN){
        printf("Ultimo nodo de la lista: %d\n", ultimo(l1));
    }
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));

        return 0;
}

struct nodo * crearNodo(int x){
        struct nodo * nuevo = NULL;
        nuevo = (struct nodo *) malloc(sizeof(struct nodo));
        if(nuevo == NULL) exit(0);
        nuevo->dato = x;
        nuevo->siguiente = NULL;
    nuevo->anterior = NULL;
        return nuevo;
}

struct listaLigada * crearLista(){
    struct listaLigada * nuevaLista = NULL;
    nuevaLista = (struct listaLigada *) malloc(sizeof(struct listaLigada));
    if(nuevaLista == NULL) return NULL;
    nuevaLista->cabecera = NULL;
    nuevaLista->final = NULL;
    return nuevaLista;
}

void insertar_inicio(struct listaLigada * lista, int x){
        struct nodo * nuevo = crearNodo(x);
        if(lista->cabecera == NULL){
       // Insertamos el primer nodo
                lista->cabecera = lista->final = nuevo;
        } else{
                nuevo->siguiente = lista->cabecera;
```

```
        lista->cabecera->anterior = nuevo;
                lista->cabecera = nuevo;
         }
}

void insertar_final(struct listaLigada * lista, int x){
         struct nodo * nuevo = crearNodo(x);
         if(lista->cabecera == NULL){
        // Insertamos el primer nodo
                lista->cabecera = lista->final = nuevo;
         } else{
        lista->final->siguiente = nuevo;
        nuevo->anterior = lista->final;
        lista->final = nuevo;
         }
}

void borrar_inicio(struct listaLigada * lista){
         if(lista->cabecera == NULL){
        // Si la lista esta vacia no borrramos nada
                printf("\nLista Vacia");
         } else{
                struct nodo * temp = lista->cabecera;
        if(lista->cabecera == lista->final){
           // Solo queda un nodo en la lista
           lista->cabecera = lista->final = NULL;
        } else{
           lista->cabecera = lista->cabecera->siguiente;
           lista->cabecera->anterior = NULL;
        }
                free(temp);
         }
}

void borrar_final(struct listaLigada * lista){
         if(lista->cabecera == NULL){
        // Si la lista esta vacia no borrramos nada
```

```
                printf("\nLista Vacia");
        } else{
                struct nodo * temp = lista->cabecera;
        if(lista->cabecera == lista->final){
           // Solo queda un nodo en la lista
           lista->cabecera = lista->final = NULL;
           free(temp);
        } else{
           while(temp->siguiente != lista->final){
              // Recorremos la lista hasta el penultimo nodo
                        temp = temp->siguiente;
                   }
           lista->final->anterior = NULL;
           free(lista->final);
           lista->final = temp;
           lista->final->siguiente = NULL;
        }
        }
}

int contar_nodos(struct listaLigada * lista){
        if(lista->cabecera == NULL){
                return 0;
        } else{
                int cont = 0;
                struct nodo * temp = lista->cabecera;
                while(temp != NULL){
                        cont++;
                        temp = temp->siguiente;
                }
                return cont;
        }
}

void insertar_intermedio(struct listaLigada * lista, int pos, int x){
   struct nodo * nuevo = crearNodo(x);
   // Se toma el 0 como posicion valida
```

```
    int noNodos = contar_nodos(lista);
    if(pos<-1 || pos>noNodos+1){
        // Rango valido de pos: [0, pos+1]
        printf("Posicion invalida\n");
    } else if(pos == 0){
        if(lista->cabecera == NULL) {
            // Apenas vamos a insertar el primer nodo
            lista->cabecera = lista->final = nuevo;
        }  else{
            // El nuevo nodo será el primero
            nuevo->siguiente = lista->cabecera;
            lista->cabecera->anterior = nuevo;
            lista->cabecera = nuevo;
        }
    } else if(pos == noNodos){
        // Si el valor de pos es igual al numero de nodos
        // el nuevo nodo será el último de la lista
        lista->final->siguiente = nuevo;
        nuevo->anterior = lista->final;
        lista->final = nuevo;
    }else{
        // Insertamos un nodo en cualquier posición
        // que no sea la primera y la última
        struct nodo * temp = lista->cabecera;
        int i=0;
        while(i<pos-1){ // <-- Linea corregida
            temp = temp->siguiente; // (pos-1)-th nodo
            i++;
        }
        nuevo->siguiente = temp->siguiente;
        nuevo->siguiente->anterior = nuevo;
        temp->siguiente = nuevo;
        nuevo->anterior = temp;
    }
}

void borrar_intermedio(struct listaLigada * lista, int pos){
```

```
if(lista->cabecera == NULL) {
    printf("Lista Vacia!!\n");
    return;
} else {
    // Se toma el 0 como posicion valida
    int noNodos = contar_nodos(lista);
    if(pos<-1 || pos>=noNodos){
        // Rango valido de pos: [0, pos]
        printf("Posicion invalida\n");
    } else {
        struct nodo * temp = NULL;
        if(pos == 0){
            temp = lista->cabecera;
            if(lista->cabecera == lista->final){
                // Si solo queda un nodo en la lista
                lista->cabecera = lista->final = NULL;
            } else{
                // Si todavía hay más nodos
                lista->cabecera = lista->cabecera->siguiente;
                lista->cabecera->anterior = NULL;
            }
            free(temp);
        } else{
            int i=0;
            struct nodo * temp = lista->cabecera;
            while(i<pos-1){ // <-- Linea corregida
                // Recorremos la lista hasta el nodo previo a borrar
                temp = temp->siguiente; // (pos-1)-th nodo
                i++;
            }
            // Nos desplazamos al nodo a borrar
            struct nodo * borrado = temp->siguiente; // (pos)-th nodo
            if(borrado == lista->final){
                borrado->anterior = NULL;
                lista->final = temp;
                temp->siguiente = NULL;
```

```
            } else{
                // Si el nodo a borrar no es el último
                temp->siguiente = borrado->siguiente; // (pos+1)-th nodo
                if(temp->siguiente == NULL){
                    lista->final = temp;
                }
            }
            free(borrado);
        }
    }
  }
}

int frente(struct listaLigada * lista){
   if(lista->cabecera == NULL){
      return INT_MIN;
   } else{
      return lista->cabecera->dato;
   }
}

int ultimo(struct listaLigada * lista){
   if(lista->cabecera == NULL){
      return INT_MIN;
   } else{
      return lista->final->dato;
   }
}

void desplegarAdelante(struct listaLigada * lista){
        struct nodo * temp = lista->cabecera;
        if(temp == NULL){
                printf("Lista Vacia\n");
        } else{
                while(temp != NULL){
                        printf("%d->",temp->dato);
                        temp = temp->siguiente;
```

```
                }
                printf("NULL\n");
        }
}

void desplegarAtras(struct listaLigada * lista){
        struct nodo * temp = lista->final;
        if(temp == NULL){
                printf("Lista Vacia\n");
        } else{
                while(temp != NULL){
                        printf("%d->",temp->dato);
                        temp = temp->anterior;
                }
                printf("NULL\n");
        }
}
```

ListaLigadaCircular

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

struct nodo{
   int dato;
   struct nodo * siguiente;
};

struct listaCircular{
   struct nodo * cabecera;
};

struct nodo * crearNodo(int);
struct listaCircular * listaCircular();
void insertar_inicio(struct listaCircular *, int);
void insertar_final(struct listaCircular *, int);
int contar_nodos(struct listaCircular *);
```

```
void insertar_intermedio(struct listaCircular *, int, int);
void borrar_inicio(struct listaCircular *);
void borrar_final(struct listaCircular *);
void borrar_intermedio(struct listaCircular *, int);
void desplegar(struct listaCircular *);

int main() {
    // Declaramos múltiples instancias de lista
    struct listaCircular * l1 = listaCircular();
    // Probando las funciones implementadas
        printf("\nOperaciones de insercion: \n\n");
        // Insertando nodos al principio
        insertar_inicio(l1, 23);
        insertar_inicio(l1, 12);
        insertar_inicio(l1, 76);
        insertar_inicio(l1, 93);
        insertar_inicio(l1, 8);
        desplegar(l1);
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
        // Insertando nodos al final
        insertar_final(l1, 9);
        insertar_final(l1, 7);
        insertar_final(l1, -5);
        insertar_final(l1, 16);
        desplegar(l1);
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
        // Insertando nodos en una posicion intermedia
        insertar_intermedio(l1, 0, 14);
        insertar_intermedio(l1, 5, 25);
        insertar_intermedio(l1, 2, 17);
        desplegar(l1);
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));

        printf("\n\nOperaciones de borrado: \n\n");
        // Borrando nodos al principio
        borrar_inicio(l1);
        borrar_inicio(l1);
```

```
        borrar_inicio(l1);
        desplegar(l1);
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
        // Borrando nodos al final
        borrar_final(l1);
        borrar_final(l1);
        desplegar(l1);
        printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));
        // Borrando nodos en una posicion intermedia
    borrar_intermedio(l1, 1);
        borrar_intermedio(l1, 4);
        borrar_intermedio(l1, 0);
        desplegar(l1);
    printf("Numero de Nodos en la Lista: %d\n", contar_nodos(l1));

    return 0;
}

struct nodo * crearNodo(int x){
    struct nodo * nuevo = NULL;
    nuevo = (struct nodo *) malloc(sizeof(struct nodo));
    if(nuevo == NULL) return NULL;
    nuevo->dato = x;
    nuevo->siguiente = NULL;

    return nuevo;
}

struct listaCircular * listaCircular(){
    // Inicializamos la lista en NULL
    struct listaCircular * lista = NULL;
    // Asignamos memoria para la lista
    lista = (struct listaCircular *) malloc(sizeof(struct listaCircular));
    if(lista == NULL) return NULL; // En caso de que no se le pueda
asignar memoria
    lista->cabecera = NULL; // Asignamos valores
```

```
    return lista;
}

void insertar_inicio(struct listaCircular * lista, int dato){
    struct nodo * nuevo = crearNodo(dato);
    if(lista->cabecera == NULL){
        nuevo->siguiente = nuevo;
        lista->cabecera = nuevo;
    } else{
        // Recorriendo la lista hasta el final
        struct nodo * temp = lista->cabecera;
        while (temp->siguiente != lista->cabecera) {
            temp = temp->siguiente;
        };
        nuevo->siguiente = lista->cabecera;
        temp->siguiente = nuevo;
        lista->cabecera = nuevo;
    }
}

void insertar_final(struct listaCircular * lista, int dato){
    struct nodo * nuevo = crearNodo(dato);
    if(lista->cabecera == NULL){
        nuevo->siguiente = nuevo;
        lista->cabecera = nuevo;
    } else{
        // Recorriendo la lista hasta el final
        struct nodo * temp = lista->cabecera;
        while (temp->siguiente != lista->cabecera) {
            temp = temp->siguiente;
        };
        temp->siguiente = nuevo;
        nuevo->siguiente = lista->cabecera;
    }
}

int contar_nodos(struct listaCircular * lista){
```

```
    if(lista->cabecera == NULL){
        return 0;
    } else{
        struct nodo * temp = lista->cabecera;
        int cont = 0;
        do {
            temp = temp->siguiente;
            cont++;
        } while(temp != lista->cabecera);
        return cont;
    }
}

void insertar_intermedio(struct listaCircular * lista, int pos, int dato){
    struct nodo * nuevo = crearNodo(dato);
    // Se toma el 0 como posición válida
    int noNodos = contar_nodos(lista);
    if(pos<-1 || pos>noNodos){
        printf("Posición inválida\n");
        return;
    } else if(lista->cabecera == NULL) {
        nuevo->siguiente = nuevo;
        lista->cabecera = nuevo;
    } else if(pos == 0){
        // Recorriendo la lista hasta el final
        struct nodo * temp = lista->cabecera;
        while (temp->siguiente != lista->cabecera) {
            temp = temp->siguiente;
        };
        nuevo->siguiente = lista->cabecera;
        temp->siguiente = nuevo;
        lista->cabecera = nuevo;
    } else{
        struct nodo * temp = lista->cabecera;
        int i=0;
        while(i<pos-1){
            temp = temp->siguiente; // (pos-1)-th nodo
```

```
            i++;
        }
        if(temp->siguiente == lista->cabecera){
            temp->siguiente = nuevo;
            nuevo->siguiente = lista->cabecera;
        } else{
            nuevo->siguiente = temp->siguiente;
            temp->siguiente = nuevo;
        }

    }
}

void borrar_inicio(struct listaCircular * lista){
    if(lista->cabecera == NULL){
        printf("Lista Vacía!!\n");
    } else{
        // Recorriendo la lista hasta el final
        struct nodo * ultimo = lista->cabecera;
        struct nodo * temp = lista->cabecera;
        while(temp->siguiente != lista->cabecera){
            temp = temp->siguiente;
        }
        if (temp == lista->cabecera) {
            lista->cabecera = NULL;
        } else{
            temp->siguiente = lista->cabecera->siguiente;
            lista->cabecera = lista->cabecera->siguiente;
        }

        free(ultimo);
    }
}

void borrar_final(struct listaCircular * lista) {
    struct nodo *temp; // Nodo a borrar
    struct nodo *prev; // Nodo previo al ultimo
```

```
    if (lista->cabecera == NULL) {
        printf("Lista Vacía!!\n");
    } else {
        temp = lista->cabecera;
        // Recorremos hasta el último nodo de la Lista
        while (temp->siguiente != lista->cabecera) {
            prev = temp;
            temp = temp->siguiente;
        }
        if (temp == lista->cabecera) {
            lista->cabecera = NULL;
        } else {
            /* Desconectar el enlace */
            prev->siguiente = lista->cabecera;
        }
        free(temp); // Borramos el último
    }
}

void borrar_intermedio(struct listaCircular * lista, int pos){
    if(lista->cabecera == NULL) {
        printf("Lista Vacía!!\n");
    } else {
        // Se toma el 0 como posición válida
        int noNodos = contar_nodos(lista);
        if(pos<-1 || pos>noNodos){
            printf("Posición inválida\n");
            return;
        } else {
            struct nodo * temp = NULL;
            if(pos == 0){
                // Recorriendo la lista hasta el final
                struct nodo * ultimo = lista->cabecera;
                struct nodo * temp = lista->cabecera;
                while(temp->siguiente != lista->cabecera){
                    temp = temp->siguiente;
                }
```

```
            if (temp == lista->cabecera) {
                lista->cabecera = NULL;
            } else{
                temp->siguiente = lista->cabecera->siguiente;
                lista->cabecera = lista->cabecera->siguiente;
            }
            free(ultimo);
        } else{
            int i=0;
            temp = lista->cabecera;
            while(i<pos-1){
                temp = temp->siguiente; // (pos-1)-th nodo
                i++;
            }
            if(temp->siguiente == lista->cabecera){
                free(temp);
                lista->cabecera = NULL;
            } else{
                struct nodo * borrado = temp->siguiente; // (pos)-th nodo
                temp->siguiente = borrado->siguiente; // (pos+1)-th nodo
                free(borrado);
            }
        }
    }
}

void desplegar(struct listaCircular * lista){
    struct nodo * temp = lista->cabecera;
    if(temp == NULL){
        printf("Lista Vacía!!\n");
        return;
    }
    int i=0;
    do{
        printf("%d ", temp->dato);
        temp = temp->siguiente;
```

```
        i++;
    } while (temp != lista->cabecera);
    printf("\n");
}
```

PilaListaLigada

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

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


typedef struct pila{
    // La cabecera representaría el tope
    Nodo * tope;
} Pila;

Pila * init();
// Una pila sigue la metodología LIFO
void push(Pila *, int); // push() representaría la funcion insertar_inicio en
una lista ligada
void pop(Pila *); // pop() representaría la funcion borrar_inicio en una
lista ligada
int peak(Pila *); // peak() imprime valor en el tope
void desplegar(Pila *);
int contar_nodos(Pila *);

int main(){
    Pila * p1 = init();
    // Insertando elementos en la Pila
    printf("\nInsertando elementos en la pila: ");
    push(p1, 5);
```

```
    push(p1, 8);
    push(p1, 9);
    push(p1, -2);
    push(p1, 7);
    desplegar(p1);
    if(peak(p1) != INT_MIN){
        printf("Nodo en el tope de la pila: %d\n", peak(p1));
    }
    printf("Numero de Nodos en la pila: %d\n", contar_nodos(p1));
    printf("\nBorrando elementos en la pila: ");
    pop(p1);
    pop(p1);
    pop(p1);
    desplegar(p1);
    if(peak(p1) != INT_MIN){
        printf("Nodo en el tope de la pila: %d\n", peak(p1));
    }
    printf("Numero de Nodos en la pila: %d\n\n", contar_nodos(p1));

    return 0;
}

Nodo * crearNodo(int x){
        Nodo * nuevo = NULL;
        nuevo = (Nodo *) malloc(sizeof(Nodo));
        if(nuevo == NULL) exit(0);
        nuevo->dato = x;
        nuevo->siguiente = NULL;
        return nuevo;
}

Pila * init(){
    Pila * nuevaPila = NULL;
    nuevaPila = (Pila *) malloc(sizeof(Pila));
    if(nuevaPila == NULL) return NULL;
    nuevaPila->tope = NULL;
    return nuevaPila;
```

```
}

void push(Pila * p, int x){
	Nodo * n = crearNodo(x);
	if(p->tope == NULL){
		p->tope = n;
	} else{
		n->siguiente = p->tope;
		p->tope = n;
	}
}

void pop(Pila * p){
	if(p->tope == NULL){
		printf("\nLista Vacia");
	} else{
		Nodo * aux = p->tope;
		p->tope = p->tope->siguiente;
		free(aux);
	}
}

int peak(Pila * p){
   if(p->tope == NULL){
      return INT_MIN;
   } else{
      return p->tope->dato;
   }
}

int contar_nodos(Pila * p){
	if(p->tope == NULL){
		return 0;
	} else{
		int cont = 0;
		struct nodo * temp = p->tope;
		while(temp != NULL){
```

```
                cont++;
                temp = temp->siguiente;
            }
            return cont;
        }
}

void desplegar(Pila * p){
        Nodo * temp = p->tope;
        if(temp == NULL){
            printf("Lista Vacia");
        } else{
            while(temp != NULL){
                printf("%d->",temp->dato);
                temp = temp->siguiente;
            }
            printf("NULL\n");
        }
}
```

ColaListaLigada

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

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

typedef struct cola{
    Nodo * frente;
    Nodo * final;
} Cola;

Nodo * crearNodo(int);
Cola * init();
```

```
void encolar(Cola *, int);
void desencolar(Cola *);
int frente(Cola *);
int final(Cola *);
void desplegar(Cola *);
int contar_nodos(Cola *);

int main(){
    Cola * c1 = init();
    // Insertando elementos en la Cola
    printf("\nInsertando elementos en la cola: ");
    encolar(c1, 5);
    encolar(c1, 8);
    encolar(c1, 9);
    encolar(c1, -2);
    encolar(c1, 7);
    desplegar(c1);
    if(frente(c1) != 0){
        printf("Nodo al frente de la cola: %d\n", frente(c1));
    }
    if(final(c1) != 0){
        printf("Nodo al final de la cola: %d\n", final(c1));
    }
    printf("Numero de Nodos en la cola: %d\n", contar_nodos(c1));

    printf("\nBorrando elementos en la pila: ");
    desencolar(c1);
    desencolar(c1);
    desencolar(c1);
    desplegar(c1);
    if(frente(c1) != 0){
        printf("Nodo al frente de la cola: %d\n", frente(c1));
    }
    if(final(c1) != 0){
        printf("Nodo al final de la cola: %d\n", final(c1));
    }
    printf("Numero de Nodos en la cola: %d\n\n", contar_nodos(c1));
```

```
    return 0;
}

Nodo * crearNodo(int x){
        Nodo * nuevo = NULL;
        nuevo = (Nodo *) malloc(sizeof(Nodo));
        if(nuevo == NULL) exit(0);
        nuevo->dato = x;
        nuevo->siguiente = NULL;
        return nuevo;
}

Cola * init(){
    Cola * nuevaCola = NULL;
    nuevaCola = (Cola  *) malloc(sizeof(Cola));
    if(nuevaCola == NULL) return NULL;
    nuevaCola->frente = NULL;
    nuevaCola->final = NULL;
    return nuevaCola;
}

void encolar(Cola * c, int x){
        Nodo * n = crearNodo(x);
        if(c->frente == NULL){
                c->frente = c->final = n;
        } else{
       c->final->siguiente = n;
       c->final = n;
        }
}

void desencolar(Cola * c){
        if(c->frente == NULL){
                printf("\nLista Vacia");
        } else{
                Nodo * temp = c->frente;
```

```
        if(c->frente == c->final){
            // Solo queda un nodo en la lista
            c->frente = c->final = NULL;
        } else{
            c->frente = c->frente->siguiente;
        }
                free(temp);
         }
}

int frente(Cola * c){
    if(c->frente == NULL){
        return 0;
    } else{
        return c->frente->dato;
    }
}

int final(Cola * c){
    if(c->frente == NULL){
        return 0;
    } else{
        return c->final->dato;
    }
}

int contar_nodos(Cola * c){
         if(c->frente == NULL){
                return 0;
         } else{
                int cont = 0;
                Nodo * temp = c->frente;
                while(temp != NULL){
                        cont++;
                        temp = temp->siguiente;
                }
                return cont;
```

```
        }
}

void desplegar(Cola * c){
        Nodo * temp = c->frente;
        if(temp == NULL){
                printf("Lista Vacia\n");
        } else{
                while(temp != NULL){
                        printf("%d->",temp->dato);
                        temp = temp->siguiente;
                }
                printf("NULL\n");
        }
}
```