**Código base con int main “easy”**

**PILAS:**

**Estática…………………2**

**Dinámica……………….12**

**COLAS:**

**Estática…………………7**

**Dinámica……………….16**

**Circular…………………22**

**Doble……………………28**

**LISTAS:**

**Simple…………………..35**

**Circular…………………44**

**Doble……………………56**

**Doble Circular…………67**

**Pila estática**

#include <stdio.h>

#define pf printf

#define sf scanf

// funciones

struct pila{

    int datos[10];

    int tope;

};

// Basicas

void crearPila(struct pila\*);

int pop(struct pila\*);

int push(struct pila\*, int);

int top(struct pila\*);

// Extra

void verPila(struct pila\*);

int empty(struct pila\*);

int full(struct pila\*);

int size(struct pila\*);

// operaciones

void crearPila(struct pila\* pila){

    pila->tope = -1;

    pf("Pila creada\n");

}

int push(struct pila\* pila, int nuevoDato){

    if(pila->tope==9) return -1;

    pila->tope++;

    pila->datos[pila->tope] = nuevoDato;

    pf("Dato ingresado: %d\n", nuevoDato);

    return 0;

}

int pop(struct pila\* pila){

    if(pila->tope == -1) return -1;

    pf("Elemento eliminado: %d\n",pila->datos[pila->tope]);

    pila->tope--;

    return 0;

}

int top(struct pila\* pila){

    if(pila->tope == -1) return -1;

    return pila->datos[pila->tope];

}

// extras

void verPila(struct pila\* pila){

    if(pila->tope == -1) return;

    pf("Datos de la pila\n");

    for(int i = pila->tope; i >= 0; i--){

        pf("%d\n", pila->datos[i]);

    }

}

int empty(struct pila\* pila){

    if(pila->tope != -1){

        pf("Pila con elementos\n");

        return -1;

    }

    pf("Pila vacia\n");

    return 0;

}

int full(struct pila\* pila){

    if(pila->tope != 9){

        pf("Pila no llena\n");

        return -1;

    }

    pf("Pila llena\n");

    return 0;

}

int size(struct pila\* pila){

    return pila->tope +1;

}

// main

int main(){

    struct pila miPila;

    crearPila(&miPila);

    push(&miPila, 1);

    push(&miPila, 2);

    push(&miPila, 3);

    push(&miPila, 4);

    verPila(&miPila);

    pop(&miPila);

    pop(&miPila);

    verPila(&miPila);

    empty(&miPila);

    full(&miPila);

    pf("Tamaño de la pila: %d\n",size(&miPila));

    return 0;

}

**Cola estática**

#include <stdio.h>

#define pf printf

#define sf scanf

// funciones

struct cola {

    int datos[10];

    int head;

    int tail;

    int size;

};

// Basicas

void crearCola(struct cola \*);

int encolar(struct cola \*, int);

int desencolar(struct cola \*);

int front(struct cola \*);

// Extra

void verCola(struct cola \*);

int emptyC(struct cola \*);

int fullC(struct cola \*);

int sizeC(struct cola \*);

// operaciones

void crearCola(struct cola \*cola) {

    cola->head = 0;

    cola->tail = -1;

    cola->size = 0;

    pf("Cola creada\n");

}

int encolar(struct cola \*cola, int nuevoDato) {

    if (cola->size == 10) return -1;

    cola->tail++;

    cola->size++;

    cola->datos[cola->tail] = nuevoDato;

    pf("Elemento agregado: %d\n", nuevoDato);

    return 0;

}

int desencolar(struct cola \*cola) {

    if (cola->size == 0) return -1;

    pf("Elemento eliminado: %d\n", cola->datos[cola->head]);

    cola->head++;

    cola->size--;

    return 0;

}

int front(struct cola \*cola) {

    if (cola->size == 0) return -1;

    return cola->datos[cola->head];

}

// extras

void verCola(struct cola \*cola){

    if(cola->size == 0) return;

    pf("Datos de la cola:\n");

    for(int i = cola->head; i <= cola->tail; i--){

        pf("%d\n", cola->datos[i]);

    }

}

int emptyC(struct cola \*cola) {

    return cola->size == 0; // Devuelve 1 si esta vacia, 0 si no

}

int fullC(struct cola \*cola) {

    return cola->size == 10; // Devuelve 1 si esta llena, 0 si no

}

int sizeC(struct cola \*cola) {

    return cola->size;

}

// main

int main() {

    struct cola miCola;

    crearCola(&miCola);

    verCola(&miCola);

    pf("¿Cola vacia? %s\n", emptyC(&miCola) ? "Si" : "No");

    pf("¿Cola llena? %s\n", fullC(&miCola) ? "Si" : "No");

    pf("Tamaño de la cola: %d\n", sizeC(&miCola));

    encolar(&miCola, 1);

    encolar(&miCola, 2);

    encolar(&miCola, 3);

    encolar(&miCola, 4);

    encolar(&miCola, 5);

    encolar(&miCola, 6);

    encolar(&miCola, 7);

    encolar(&miCola, 8);

    encolar(&miCola, 9);

    encolar(&miCola, 10);

    pf("¿Cola vacia? %s\n", emptyC(&miCola) ? "Si" : "No");

    pf("¿Cola llena? %s\n", fullC(&miCola) ? "Si" : "No");

    pf("Tamaño de la cola: %d\n", sizeC(&miCola));

    desencolar(&miCola);

    desencolar(&miCola);

    desencolar(&miCola);

    verCola(&miCola);

    return 0;

}

**Pila Dinámica**

#include <stdio.h>

#include <stdlib.h>

#define pf printf

#define sf scanf

// funciones

struct nodo {

    int dato;

    struct nodo \*siguiente;

};

struct pila {

    struct nodo \*tope;

};

// basicas

void crearPila(struct pila \*);

void verPila(struct pila \*);

int push(struct pila \*, int);

int pop(struct pila \*);

int top(struct pila \*);

// extras

int empty(struct pila \*);

// operaciones

void crearPila(struct pila \*pila) {

    pila->tope = NULL;

    pf("Pila creada\n");

}

void verPila(struct pila \*pila) {

    pf("Datos en la pila:\n");

    struct nodo \*nodoActual = pila->tope;

    while (nodoActual != NULL) {

        pf("%d\n", nodoActual->dato);

        nodoActual = nodoActual->siguiente;

    }

}

int push(struct pila \*pila, int nuevoDato) {

    struct nodo \*nuevoNodo = (struct nodo \*)malloc(sizeof(struct nodo));

    if (nuevoNodo == NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    nuevoNodo->siguiente = pila->tope;

    pila->tope = nuevoNodo;

    return 0;

}

int pop(struct pila \*pila) {

    if (pila->tope == NULL) return -1;

    struct nodo \*nodoEliminar = pila->tope;

    pila->tope = nodoEliminar->siguiente;

    free(nodoEliminar);

    return 0;

}

int top(struct pila \*pila) {

    if (pila->tope == NULL) return -1;

    return pila->tope->dato;

}

// extras

int empty(struct pila \*pilaChecar) {

    return pilaChecar->tope == NULL;

}

// main

int main() {

    struct pila miPila;

    crearPila(&miPila);

    pop(&miPila);

    push(&miPila, 10);

    push(&miPila, 8);

    push(&miPila, 666);

    pf("Elemento en la cima: %d\n", top(&miPila));

    pf("¿Pila vacia? %s\n", empty(&miPila) ? "Sí" : "No");

    verPila(&miPila);

    // Liberar memoria

    while (!empty(&miPila)) {

        pop(&miPila);

    }

    return 0;

}

**Cola Dinámica**

#include <stdio.h>

#include <stdlib.h>

#define pf printf

#define sf scanf

// funciones

struct nodo {

    int dato;

    struct nodo \*siguiente;

};

struct cola {

    struct nodo \*head;

    struct nodo \*tail;

};

// basicas

void crearCola(struct cola \*);

void verCola(struct cola \*);

int encolar(struct cola \*, int);

int desencolar(struct cola \*);

int front(struct cola \*);

// extras

int empty(struct cola \*);

// operaciones

void crearCola(struct cola \*cola) {

    cola->head = NULL;

    cola->tail = NULL;

    pf("Cola creada\n");

    return;

}

void verCola(struct cola \*cola) {

    struct nodo \*nodoActual = cola->head;

    pf("Datos en la cola:\n");

    while (nodoActual != NULL) {

        pf("%d\n", nodoActual->dato);

        nodoActual = nodoActual->siguiente;

    }

    pf("\n");

}

int encolar(struct cola \*cola, int nuevoDato) {

    struct nodo \*nuevoNodo = (struct nodo\*)malloc(sizeof(struct nodo));

    if (nuevoNodo == NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    nuevoNodo->siguiente = NULL;

    if (cola->head == NULL){

        cola->head = nuevoNodo;

    } else{

        cola->tail->siguiente = nuevoNodo;

    }

    cola->tail = nuevoNodo;

    return 0;

}

int desencolar(struct cola \*cola) {

    if (cola->head == NULL) return -1;

    struct nodo \*nodoEliminar = cola->head;

    cola->head = nodoEliminar->siguiente;

    if (cola->head == NULL) {

        cola->tail = NULL;

    }

    free(nodoEliminar);

    return 0;

}

int front(struct cola \*cola) {

    if (cola->head == NULL) return -1;

    return cola->head->dato;

}

// extras

int empty(struct cola \*cola) {

    if (cola->head == NULL) {

        pf("La cola esta vacia\n");

        return 0;

    }

    pf("Cola con elementos");

    return -1;

}

// main

int main() {

    struct cola miCola;

    crearCola(&miCola);

    empty(&miCola);

    encolar(&miCola, 1);

    encolar(&miCola, 2);

    encolar(&miCola, 3);

    verCola(&miCola);

    encolar(&miCola, 4);

    verCola(&miCola);

    desencolar(&miCola);

    desencolar(&miCola);

    verCola(&miCola);

    int datoHead = front(&miCola);

    pf("Dato del head: %d\n", datoHead);

    desencolar(&miCola);

    int datoHeadDos = front(&miCola);

    pf("Dato del head: %d\n", datoHeadDos);

    // Liberar memoria

    while (miCola.head != NULL) {

        desencolar(&miCola);

    }

    system("PAUSE");

    return 0;

}

**Cola Circular**

#include <stdio.h>

#include <stdlib.h>

#define pf printf

#define sf scanf

// funciones

struct nodo {

    int dato;

    struct nodo \*siguiente;

};

struct cola{

    struct nodo \*head;

    struct nodo \*tail;

};

void crearCola(struct cola \*);

void verCola(struct cola \*);

int encolar(struct cola \*, int);

int desencolar(struct cola \*);

int front(struct cola \*);

// extra NECESARIA

void liberarCola(struct cola \*);

// operaciones

void crearCola(struct cola \*cola){

    cola->head=NULL;

    cola->tail=NULL;

    pf("Cola creada\n");

}

void verCola(struct cola \*cola){

    struct nodo \*nodoActual = cola->head;

    pf("Datos de la cola:\n");

   do{

        pf("%d\n",nodoActual->dato);

        nodoActual = nodoActual->siguiente;

    } while(nodoActual!=cola->head);

    pf("\n");

}

int encolar(struct cola \*cola,int nuevoDato){

    struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

    if(nuevoNodo==NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    nuevoNodo->siguiente = NULL;

    if(cola->head == NULL){

        cola->head = nuevoNodo;

    } else{

        cola->tail->siguiente = nuevoNodo;

    }

    cola->tail = nuevoNodo;

    nuevoNodo->siguiente = cola->head;

    return 0;

}

int desencolar(struct cola \*cola){

    if(cola->tail == NULL) return -1;

    struct nodo \*nodoEliminar = cola->head;

    if(cola->head == cola->tail){

        cola->tail = NULL;

        cola->head = NULL;

    } else{

        cola->head = nodoEliminar->siguiente;

        cola->tail->siguiente = cola->head;

    }

    free(nodoEliminar);

    return 0;

}

int front(struct cola \*cola){

    if(cola->head==NULL) return -1;

    return cola->head->dato;

}

// extra NECESARIA

void liberarCola(struct cola \*cola) {

    if (cola->head == NULL) return;

    struct nodo \*nodoActual = cola->head;

    struct nodo \*nodoLiberar;

    do {

        nodoLiberar = nodoActual;

        nodoActual = nodoActual->siguiente;

        free(nodoLiberar);

    } while(nodoActual != cola->head);

    cola->head = NULL;

    cola->tail = NULL;

    pf("Memoria liberada");

}

// main

int main(){

    struct cola miCola;

    crearCola(&miCola);

    desencolar(&miCola);

    encolar(&miCola,10);

    encolar(&miCola,9);

    encolar(&miCola,11);

    verCola(&miCola);

    int dato1=front(&miCola);

    pf("El valor del head es: %d\n",dato1);

    desencolar(&miCola);

    verCola(&miCola);

    dato1=front(&miCola);

    pf("El valor del head es: %d\n",dato1);

    // Liberar memoria

    liberarCola(&miCola);

    return 0;

}

**Cola Doble**

#include <stdio.h>

#include <stdlib.h>

#define pf printf

#define sf scanf

// funciones

struct nodo {

    int dato;

    struct nodo \*siguiente;

    struct nodo \*anterior;

};

struct cola{

    struct nodo \*head;

    struct nodo \*tail;

};

// basicas

void crearCola(struct cola \*);

void verCola(struct cola \*);

int encolarHead(struct cola \*, int);

int encolarTail(struct cola \*, int);

int desencolarHead(struct cola \*);

int desencolarTail(struct cola \*);

int front(struct cola \*);

int back(struct cola \*);

// operaciones

void crearCola(struct cola \*cola){

    cola->head=NULL;

    cola->tail=NULL;

    pf("Cola creada\n");

}

void verCola(struct cola \*cola){

    struct nodo \*nodoActual = cola->head;

    pf("Datos en la cola\n");

    while(nodoActual != NULL){

        pf("%d\n",nodoActual->dato);

        nodoActual = nodoActual->siguiente;

    }

    pf("\n");

}

int encolarHead(struct cola \*cola, int nuevoDato){

    struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

    if(nuevoNodo==NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    nuevoNodo->anterior = NULL;

    nuevoNodo->siguiente = cola->head;

    if(cola->head == NULL){

        cola->head = nuevoNodo;

        cola->tail = nuevoNodo;

    } else{

        cola->head->anterior = nuevoNodo;

        cola->head = nuevoNodo;

    }

    return 0;

}

int encolarTail(struct cola \*cola, int nuevoDato){

    struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

    if(nuevoNodo==NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    nuevoNodo->siguiente = NULL;

    nuevoNodo->anterior = cola->tail;

    if(cola->head == NULL){

        cola->head = nuevoNodo;

        cola->tail = nuevoNodo;

    } else{

        cola->tail->siguiente = nuevoNodo;

        cola->tail = nuevoNodo;

    }

    return 0;

}

int desencolarHead(struct cola \*cola){

    if(cola->head==NULL) return -1;

    struct nodo \*nodoEliminar=cola->head;

    if(cola->head == cola->tail){

        cola->head = NULL;

        cola->tail = NULL;

    } else{

        cola->head = nodoEliminar->siguiente;

        cola->head->anterior = NULL;

    }

    free(nodoEliminar);

    return 0;

}

int desencolarTail(struct cola \*cola){

    if(cola->head==NULL) return -1;

    struct nodo \*nodoEliminar = cola->tail;

    if(cola->head == cola->tail){

        cola->head = NULL;

        cola->tail = NULL;

    } else{

        cola->tail = nodoEliminar->anterior;

        cola->tail->siguiente = NULL;

    }

    free(nodoEliminar);

    return 0;

}

int front(struct cola \*cola){

    if(cola->head==NULL) return -1;

    return cola->head->dato;

}

int back(struct cola \*cola){

    if(cola->tail==NULL) return -1;

    return cola->tail->dato;

}

// main

int main(){

    struct cola miCola;

    crearCola(&miCola);

    desencolarHead(&miCola);

    desencolarTail(&miCola);

    encolarHead(&miCola,1);

    encolarHead(&miCola,2);

    encolarTail(&miCola,3);

    verCola(&miCola);

    desencolarHead(&miCola);

    verCola(&miCola);

    desencolarTail(&miCola);

    verCola(&miCola);

    // Liberar memoria

    while(miCola.head != NULL){

        desencolarHead(&miCola);

    }

    return 0;

}

**Lista Simple**

#include <stdio.h>

#include <stdlib.h>

#define pf printf

#define sf scanf

// funciones

struct nodo{

   int dato;

   struct nodo \*siguiente;

};

struct lista{

   struct nodo \*head;

};

// basicas

void crearLista(struct lista \*);

void verLista(struct lista \*);

int insertar(struct lista \*,int);

int buscar(struct lista \*,int);

int borrar(struct lista \*,int);

// extras

void liberarLista(struct lista \*);

int size(struct lista \*);

int insertarFinal(struct lista \*, int);

int insertarCualquierPosicion(struct lista \*, int, int);

int borrarIndice(struct lista \*, int);

// operaciones

void crearLista(struct lista \*lista){

   lista->head=NULL;

   pf("Lista creada\n");

}

void verLista(struct lista \*lista){

   if(lista->head==NULL){

       pf("Lista Vacia\n");

   }else{

       struct nodo \*nodoActual=lista->head;

       pf("Datos de la lista\n");

       while(nodoActual != NULL){

           pf("%d\n",nodoActual->dato);

           nodoActual = nodoActual->siguiente;

       }

   }

   pf("\n");

}

int insertar(struct lista \*lista,int nuevoDato){

   struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

   if(nuevoNodo==NULL) return -1;

   nuevoNodo->dato = nuevoDato;

   nuevoNodo->siguiente = lista->head;

   lista->head = nuevoNodo;

   return 0;

}

int buscar(struct lista \*lista,int datoBuscar){

   if(lista->head==NULL) return -1;

   struct nodo \*nodoActual = lista->head;

   while(nodoActual != NULL){

       if(nodoActual->dato == datoBuscar){

           pf("Dato encontrado\n");

           return nodoActual->dato;

       }

       nodoActual=nodoActual->siguiente;

   }

   pf("Dato no encontrado\n");

   return -1;

}

int borrar(struct lista \*lista,int datoBorrar){

   if(lista->head==NULL) return -1;

   struct nodo \*nodoEliminar= lista->head;

   struct nodo \*nodoAnterior= NULL;

   while(nodoEliminar != NULL && nodoEliminar->dato != datoBorrar){

       nodoAnterior = nodoEliminar;

       nodoEliminar = nodoEliminar->siguiente;

   }

   if(nodoEliminar==NULL) return -1; //no esta el elemento

   if(nodoEliminar == lista->head){

       lista->head = nodoEliminar->siguiente;

   } else{

       nodoAnterior->siguiente = nodoEliminar->siguiente;

   }

   free(nodoEliminar);

   return 0;

}

// extras

void liberarLista(struct lista \*lista) {

    struct nodo \*nodoActual = lista->head;

    struct nodo \*nodoLiberar;

    while(nodoActual != NULL) {

        nodoLiberar = nodoActual;

        nodoActual = nodoActual->siguiente;

        free(nodoLiberar);

    }

    lista->head = NULL;

    pf("Memoria liberada\n");

 }

int size(struct lista \*lista) {

   int tamano = 0;

   struct nodo \*nodoActual = lista->head;

   while(nodoActual != NULL) {

       tamano++;

       nodoActual = nodoActual->siguiente;

   }

   return tamano;

}

int insertarFinal(struct lista \*lista, int nuevoDato) {

    struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

    if(nuevoNodo==NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    nuevoNodo->siguiente = NULL;

    if(lista->head == NULL) {

        lista->head = nuevoNodo;

    } else{

        struct nodo \*nodoActual = lista->head;

        while(nodoActual->siguiente != NULL) {

            nodoActual = nodoActual->siguiente;

        }

        nodoActual->siguiente = nuevoNodo;

    }

    return 0;

 }

 int insertarCualquierPosicion(struct lista \*lista, int nuevoDato, int posicion) {

     int tamano = size(lista);

     if(posicion < 1 || posicion > tamano +1) return 1; // posicion invalida

     if(posicion == 1) return insertar(lista, nuevoDato);

     if(posicion == tamano +1) return insertarFinal(lista, nuevoDato);

     struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

     if(nuevoNodo == NULL) return -1;

     nuevoNodo->dato = nuevoDato;

     struct nodo \*nodoActual = lista->head;

     for(int i = 1; i < posicion-1; i++) nodoActual = nodoActual->siguiente;

     nuevoNodo->siguiente = nodoActual->siguiente;

     nodoActual->siguiente = nuevoNodo;

     return 0;

 }

 int borrarIndice(struct lista \*lista, int posicion) {

     if(lista->head == NULL) return -1;

     int tamano = size(lista);

     if(posicion < 1 || posicion > tamano) return -1; // posicion invalida

     struct nodo \*nodoEliminar;

     if(posicion == 1) {

         nodoEliminar = lista->head;

         lista->head = lista->head->siguiente;

     } else{

         struct nodo \*nodoActual = lista->head;

         for(int i = 1; i < posicion - 1; i++) nodoActual = nodoActual->siguiente;

         nodoEliminar = nodoActual->siguiente;

         nodoActual->siguiente = nodoEliminar->siguiente;

     }

     free(nodoEliminar);

     return 0;

 }

// main

int main() {

    struct lista miLista;

    crearLista(&miLista);

    insertar(&miLista, 3);

    insertar(&miLista, 2);

    insertar(&miLista, 1);

    verLista(&miLista);

    insertarFinal(&miLista, 10);

    insertarFinal(&miLista, 20);

    verLista(&miLista);

    insertarCualquierPosicion(&miLista, 30, 1);

    insertarCualquierPosicion(&miLista, 40, 2);

    verLista(&miLista);

    buscar(&miLista, 2);

    buscar(&miLista, 9);

    pf("Tamaño de la lista: %d\n", size(&miLista));

    borrar(&miLista, 40);

    verLista(&miLista);

    borrarIndice(&miLista, 1);

    verLista(&miLista);

    borrarIndice(&miLista, 1);

    verLista(&miLista);

    liberarLista(&miLista);

    return 0;

}

**Lista Circular**

#include <stdio.h>

#include <stdlib.h>

#define pf printf

#define sf scanf

// funciones

struct nodo {

    int dato;

    struct nodo \*siguiente;

};

struct lista {

    struct nodo \*head;

};

//basicas

void crearLista(struct lista \*);

void lista(struct lista \*);

int insertar(struct lista \*, int);

int buscar(struct lista \*, int);

int borrar(struct lista \*, int);

// extras

void liberarLista(struct lista \*);

int size(struct lista \*);

int insertarFinal(struct lista \*, int);

int insertarCualquierPosicion(struct lista \*, int, int);

int borrarIndice(struct lista \*, int);

// operaciones

void crearLista(struct lista \*lista) {

    lista->head = NULL;

    pf("Lista creada\n");

}

void verLista(struct lista \*lista) {

    if (lista->head == NULL) return;

    struct nodo \*nodoActual = lista->head;

    pf("Datos de la lista:\n");

    do {

        pf("%d\n", nodoActual->dato);

        nodoActual = nodoActual->siguiente;

    } while (nodoActual != lista->head);

    pf("\n");

}

int insertar(struct lista \*lista, int nuevoDato) {

    struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

    if (nuevoNodo == NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    if (lista->head == NULL) {

        nuevoNodo->siguiente = nuevoNodo;

        lista->head = nuevoNodo;

    } else{

        struct nodo \*nodoActual = lista->head;

        while (nodoActual->siguiente != lista->head) {

            nodoActual = nodoActual->siguiente;

        }

        nuevoNodo->siguiente = lista->head;

        nodoActual->siguiente = nuevoNodo;

        lista->head = nuevoNodo;

    }

    return 0;

}

int buscar(struct lista \*lista, int datoBuscar) {

    if (lista->head == NULL) return -1;

    struct nodo \*nodoActual = lista->head;

    do {

        if (nodoActual->dato == datoBuscar) {

            pf("Dato encontrado\n");

            return nodoActual->dato;

        }

        nodoActual = nodoActual->siguiente;

    } while (nodoActual != lista->head);

    pf("Dato no encontrado\n");

    return -1;

}

int borrar(struct lista \*lista, int datoBorrar) {

    if (lista->head == NULL) return -1;

    struct nodo \*nodoActual= lista->head;

    struct nodo \*nodoAnterior= NULL;

    do {

        if (nodoActual->dato == datoBorrar) {

            //Si se encuentra el dato

            break;

        }

        nodoAnterior= nodoActual;

        nodoActual= nodoActual->siguiente;

    } while (nodoActual != lista->head);

    if (nodoActual->dato != datoBorrar) return -1; //no se encontro el dato

    if (nodoActual->siguiente == nodoActual) {

        lista->head= NULL;

    } else{

        if (nodoAnterior == NULL) {

                nodoAnterior = lista->head;

                while(nodoAnterior->siguiente != lista->head) {

                    nodoAnterior = nodoAnterior->siguiente;

                }

                nodoAnterior->siguiente= nodoActual->siguiente;

                lista->head= nodoAnterior;

            } else{

                nodoAnterior->siguiente= nodoActual->siguiente;

            }

    }

    free(nodoActual);

    return 0;

}

// extras

void liberarLista(struct lista \*lista) {

    struct nodo \*nodoActual = lista->head;

    struct nodo \*nodoLiberar;

    do {

        nodoLiberar = nodoActual;

        nodoActual= nodoActual->siguiente;

        free(nodoLiberar);

    } while (nodoActual != lista->head);

    lista->head = NULL;

    pf("Memoria liberada\n");

}

int size(struct lista \*lista) {

    if (lista->head == NULL) return -1;

    int tamano = 0;

    struct nodo \*nodoActual= lista->head;

    do {

        tamano++;

        nodoActual = nodoActual->siguiente;

    } while (nodoActual != lista->head);

    return tamano;

}

int insertarFinal(struct lista \*lista, int nuevoDato) {

    struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

    if (nuevoNodo == NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    if (lista->head == NULL) {

        nuevoNodo->siguiente = nuevoNodo;

        lista->head = nuevoNodo;

    } else{

        struct nodo \*nodoActual = lista->head;

        while (nodoActual->siguiente != lista->head){

            nodoActual = nodoActual->siguiente;

        }

        nodoActual->siguiente = nuevoNodo;

        nuevoNodo->siguiente = lista->head;

    }

    return 0;

}

int insertarCualquierPosicion(struct lista \*lista, int nuevoDato, int posicion) {

    if (posicion < 1) return -1; // posicion invalida

    struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

    if (nuevoNodo == NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    if (posicion == 1){

        if (lista->head == NULL){

            nuevoNodo->siguiente = nuevoNodo;

            lista->head = nuevoNodo;

        } else{

            struct nodo \*nodoActual = lista->head;

            while (nodoActual->siguiente != lista->head) {

                nodoActual = nodoActual->siguiente;

            }

            nuevoNodo->siguiente = lista->head;

            nodoActual->siguiente = nuevoNodo;

            lista->head = nuevoNodo;

        }

        return 0;

    }

    struct nodo \*nodoActual = lista->head;

    struct nodo \*nodoAnterior = NULL;

    int indice = 1;

    while (indice < posicion && nodoActual->siguiente != lista->head){

        nodoAnterior = nodoActual;

        nodoActual = nodoActual->siguiente;

        indice++;

    }

    if (indice != posicion - 1) {

        free(nuevoNodo);

        return -1;

    }

    nuevoNodo->siguiente = nodoActual->siguiente;

    nodoAnterior->siguiente = nuevoNodo;

    return 0;

}

int borrarIndice(struct lista \*lista, int posicion) {

    if (lista->head == NULL || posicion < 1) return -1; // posicion invalida

    struct nodo \*nodoActual = lista->head;

    struct nodo \*nodoAnterior = NULL;

    int indice = 1;

    if (posicion == 1) {

        if (lista->head->siguiente == lista->head){

            free(lista->head);

            lista->head = NULL;

            return 0;

        } else{

            nodoAnterior= lista->head;

            while(nodoAnterior->siguiente != lista->head) {

                nodoAnterior= nodoAnterior->siguiente;

            }

            nodoAnterior->siguiente= lista->head->siguiente;

            free(lista->head);

            lista->head = nodoAnterior->siguiente;

            return 0;

            }

        }

    while (indice < posicion && nodoActual->siguiente != lista->head) {

        nodoAnterior= nodoActual;

        nodoActual= nodoActual->siguiente;

        indice++;

    }

    if (indice < posicion) return -1; // no se encuentra la posicion

    nodoAnterior->siguiente = nodoActual->siguiente;

    free(nodoActual);

    return 0;

}

// main

int main() {

    struct lista miLista;

    crearLista(&miLista);

    insertar(&miLista, 1);

    insertar(&miLista, 2);

    insertar(&miLista, 3);

    insertar(&miLista, 4);

    insertar(&miLista, 5);

    lista(&miLista);

    pf("Tamaño de la lista: %d\n", size(&miLista));

    insertarFinal(&miLista, 10);

    insertarCualquierPosicion(&miLista, 11, 3);

    lista(&miLista);

    borrar(&miLista, 2);

    borrarIndice(&miLista, 1);

    lista(&miLista);

    buscar(&miLista, 4);

    buscar(&miLista, 2);

    liberarLista(&miLista);

    return 0;

}

**Lista doblemente ligada**

#include <stdio.h>

#include <stdlib.h>

#define pf printf

#define sf scanf

// funciones

struct nodo{

    int dato;

    struct nodo \*siguiente;

    struct nodo \*anterior;

};

struct lista{

    struct nodo \*head;

    struct nodo \*tail;

    int size;

};

// basicas

void crearLista(struct lista \*);

void verLista(struct lista \*);

void verListaPosiciones(struct lista \*);

void liberarLista(struct lista \*);

int insertar(struct lista \*,int);

int buscar(struct lista \*,int);

int borrar(struct lista \*, int);

int borrarIndice(struct lista \*,int);

int size(struct lista \*);

struct nodo \* buscarNodo(struct lista \*,int);

// extras

int insertarCualquierPosicion(struct lista \*, int, int);

int insertarFinal(struct lista \*, int);

// operaciones

void crearLista(struct lista \*lista){

    lista->head = NULL;

    lista->tail = NULL;

    lista->size = 0;

    pf("Lista creada\n");

}

void verLista(struct lista \*lista){

    if(lista->head == NULL) return;

    struct nodo \*nodoActual = lista->head;

    pf("Datos de la lista:\n");

        while(nodoActual != NULL){

            pf("%d\n",nodoActual->dato);

            nodoActual = nodoActual->siguiente;

        }

    pf("\n");

}

void verListaPosiciones(struct lista \*verPosicionLista){

    if(verPosicionLista->head == NULL) return;

    struct nodo \*nodoActual = verPosicionLista->head;

    int indice = 0;

    pf("Datos de la lista \n");

        while(nodoActual != NULL){

            pf("Posicion:%d Dato:%d\n", indice, nodoActual->dato);

            nodoActual = nodoActual->siguiente;

            indice++;

        }

    pf("\n");

}

void liberarLista(struct lista \*lista){

    struct nodo \*nodoActual = lista->head;

    struct nodo \*nodoLiberar;

    while (nodoActual != NULL){

        nodoLiberar= nodoActual;

        nodoActual= nodoActual->siguiente;

    }

    lista->head = NULL;

    lista->size = 0;

    pf("Memoria liberada\n");

}

int insertar(struct lista \*lista, int nuevoDato){

    struct nodo \*nuevoNodo=(struct nodo \*)malloc(sizeof(struct nodo));

    if(nuevoNodo == NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    nuevoNodo->siguiente = lista->head;

    if(lista->head != NULL){

        lista->head->anterior = nuevoNodo;

    } else{

        lista->tail = nuevoNodo;

    }

    lista->head = nuevoNodo;

    nuevoNodo->anterior = NULL;

    lista->size++;

    return 0;

}

int buscar(struct lista \*lista, int datoBuscar){

    if(lista->head == NULL) return -1;

    struct nodo \*nodoActual = lista->head;

    while(nodoActual != NULL){

        if(nodoActual->dato == datoBuscar){

            pf("Dato Encontrado\n");

            return nodoActual->dato;

        }

        nodoActual=nodoActual->siguiente;

    }

    pf("Dato no encontrado\n");

    return -1;

}

int borrar(struct lista \*lista, int datoBorrar){

    if(lista->head == NULL) return -1;

    struct nodo \*nodoEliminar = lista->head;

    while (nodoEliminar != NULL && nodoEliminar->dato != datoBorrar){

        nodoEliminar = nodoEliminar->siguiente;

    }

    if(nodoEliminar == NULL) return -1; //no encontro el nodo

    // Unico nodo

    if(nodoEliminar->siguiente == NULL && nodoEliminar->anterior == NULL){

        lista->head = NULL;

    }

    // Ultimo nodo

    else if (nodoEliminar->siguiente == NULL){

        nodoEliminar->anterior->siguiente = NULL;

    }

    // Primer nodo

    else if (nodoEliminar == lista->head) {

        lista->head = nodoEliminar->siguiente;

        lista->head->anterior = NULL;

    }

    // Nodo en medio

    else {

        nodoEliminar->anterior->siguiente = nodoEliminar->siguiente;

        nodoEliminar->siguiente->anterior = nodoEliminar->anterior;

    }

    lista->size--;

    free(nodoEliminar);

    return 0;

}

int borrarIndice(struct lista \*lista, int posicion) {

    if (lista->head == NULL) return -1;

    if (posicion > lista->size || posicion < 0) return -1; // posicion no valida

    struct nodo \*nodoEliminar = lista->head;

    int indice = 0;

    while (indice != posicion) {

        nodoEliminar = nodoEliminar->siguiente;

        indice++;

    }

    // Unico nodo

    if (nodoEliminar->siguiente == NULL && nodoEliminar->anterior == NULL) {

        lista->head = NULL;

    }

    // Ultimo nodo

    else if (nodoEliminar->siguiente == NULL) {

        nodoEliminar->anterior->siguiente = NULL;

    }

    // Primer nodo

    else if (nodoEliminar == lista->head) {

        lista->head = nodoEliminar->siguiente;

        lista->head->anterior = NULL;

    }

    // Nodo en medio

    else {

        nodoEliminar->anterior->siguiente = nodoEliminar->siguiente;

        nodoEliminar->siguiente->anterior = nodoEliminar->anterior;

    }

    lista->size--;

    free(nodoEliminar);

    return 0;

}

int size(struct lista \*lista){

    return lista->size;

}

// NO SE USA EN ESTE CODIGO

struct nodo \*buscarNodo(struct lista \*lista, int datoBuscar){

    if(lista->head==NULL){

        pf("La lista esta vacia\n");

        return NULL;

    }

    struct nodo \*nodoActual = lista->head;

    while(nodoActual != NULL){

        if(nodoActual->dato == datoBuscar){

            pf("Dato Encontrado\n");

            return nodoActual;

        }

        nodoActual = nodoActual->siguiente;

    }

    pf("Dato no encontrado\n");

    return NULL;

}

// main

int main() {

    struct lista miLista;

    crearLista(&miLista);

    insertar(&miLista, 1);

    insertar(&miLista, 2);

    insertarFinal(&miLista, 3);

    insertarCualquierPosicion(&miLista, 4, 1); // Inserta el 4 en la posicion 1

    // Ver la lista

    verLista(&miLista);

    verListaPosiciones(&miLista);

    // Buscar elementos

    buscar(&miLista, 2);

    buscar(&miLista, 5);

    // Borrar elementos

    borrar(&miLista, 1); // Borra el nodo que contiene el dato 1

    verLista(&miLista);

    // Borrando por indice

    borrarIndice(&miLista, 1); // Borra el nodo en la posicion 1

    verLista(&miLista);

    // Mostrar el tamaño de la lista

    int tamanio = size(&miLista);

    pf("El tamaño de la lista es %d\n", tamanio);

    // Liberar la memoria de la lista

    liberarLista(&miLista);

    return 0;

}

**Lista doblemente ligada Circular**

#include <stdio.h>

#include <stdlib.h>

#define pf printf

#define sf scanf

// funciones

struct nodo{

    int dato;

    struct nodo \*siguiente;

    struct nodo \*anterior;

};

struct lista{

    struct nodo \*head;

    struct nodo \*tail;

    int size;

};

// basicas

void crearLista(struct lista \*);

void lista(struct lista \*);

void listaPosiciones(struct lista \*);

void lista(struct lista \*);

int insertar(struct lista \*,int);

int buscar(struct lista \*,int);

struct nodo \* buscarNodo(struct lista \*,int);

int borrar(struct lista \*, int);

int borrarIndice(struct lista \*,int);

int size(struct lista \*);

// extras

int lista(struct lista \*, int, int);

int insertarFinal(struct lista \*, int);

// operaciones

void crearLista(struct lista \*lista){

    lista->head = NULL;

    lista->tail = NULL;

    lista->size = 0;

    pf("Lista creada\n");

}

void verLista(struct lista \*lista){

    if(lista->head == NULL) return;

    struct nodo \*nodoActual = lista->head;

    pf("Datos de la lista:\n");

        do {

            pf("%d\n", nodoActual->dato);

            nodoActual = nodoActual->siguiente;

        } while(nodoActual != lista->head);

}

void listaPosiciones(struct lista \*lista){

    if(lista->head == NULL) return;

    struct nodo \*nodoActual = lista->head;

    int indice = 0;

    pf("Datos de la lista:\n");

        do {

            pf("Posicion:%d Dato:%d\n", indice, nodoActual->dato);

            nodoActual = nodoActual->siguiente;

            indice++;

        } while(nodoActual != lista->head);

    pf("\n");

}

void liberarLista(struct lista \*lista) {

    if(lista->head == NULL) return;

    struct nodo \*nodoActual = lista->head;

    struct nodo \*nodoLiberar;

    do {

        nodoLiberar = nodoActual;

        nodoActual = nodoActual->siguiente;

        free(nodoLiberar);

    } while(nodoActual != lista->head);

    lista->head = NULL;

    lista->tail = NULL;

    lista->size = 0;

    pf("Memoria liberada\n");

}

int insertar(struct lista \*lista, int nuevoDato){

    struct nodo \*nuevoNodo=(struct nodo\*)malloc(sizeof(struct nodo));

    if(nuevoNodo == NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    if(lista->head == NULL){

        nuevoNodo->siguiente = nuevoNodo;

        nuevoNodo->anterior = nuevoNodo;

        lista->head = nuevoNodo;

        lista->tail = nuevoNodo;

    } else{

        nuevoNodo->siguiente = lista->head;

        nuevoNodo->anterior = lista->tail;

        lista->head->anterior = nuevoNodo;

        lista->tail->siguiente = nuevoNodo;

        lista->head = nuevoNodo;

    }

    lista->size++;

    return 0;

}

int insertarFinal(struct lista \*lista, int nuevoDato){

    struct nodo \*nuevoNodo=(struct nodo\*)malloc(sizeof(struct nodo));

    if(nuevoNodo == NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    if(lista->head == NULL){

        nuevoNodo->siguiente = nuevoNodo;

        nuevoNodo->anterior = nuevoNodo;

        lista->head = nuevoNodo;

        lista->tail = nuevoNodo;

    } else{

        nuevoNodo->siguiente = lista->head;

        nuevoNodo->anterior = lista->tail;

        lista->tail->siguiente = nuevoNodo;

        lista->head->anterior = nuevoNodo;

        lista->tail = nuevoNodo;

    }

    lista->size++;

    return 0;

}

int insertarCualquierPosicion(struct lista \*lista, int nuevoDato, int posicion){

    if(posicion < 0 || posicion > lista->size) return -1; // posicion no valida

    if(posicion == 0) return insertar(lista, nuevoDato);

    if(posicion == lista->size) return insertarFinal(lista, nuevoDato);

    struct nodo \*nuevoNodo=(struct nodo\*)malloc(sizeof(struct nodo));

    if(nuevoNodo == NULL) return -1;

    nuevoNodo->dato = nuevoDato;

    struct nodo \*nodoActual = lista->head;

    for(int i = 0; i < posicion; i++) nodoActual = nodoActual->siguiente;

    nuevoNodo->siguiente = nodoActual;

    nuevoNodo->anterior = nodoActual->anterior;

    nodoActual->anterior->siguiente = nuevoNodo;

    nodoActual->anterior = nuevoNodo;

    lista->size++;

    return 0;

}

struct nodo \* buscarNodo(struct lista \*lista, int datoBuscar){

    if(lista->head == NULL) return NULL;

    struct nodo \*nodoActual = lista->head;

    do {

        if(nodoActual->dato == datoBuscar) return nodoActual;

        nodoActual = nodoActual->siguiente;

    } while(nodoActual != lista->head);

    return NULL;

}

int buscar(struct lista \*lista, int datoBuscar){

    struct nodo \*nodoAux = buscarNodo(lista, datoBuscar);

    return (nodoAux != NULL) ? nodoAux->dato : -1; //Regresa 0 si se encontro el dato, si no lo encontro regresa -1

}

int borrar(struct lista \*lista, int datoBorrar){

    if(lista->head == NULL) return -1;

    struct nodo \*nodoEliminar = buscarNodo(lista, datoBorrar);

    if(nodoEliminar == NULL) return -1; // no encontro el dato

    if(lista->size == 1){

        lista->head = NULL;

        lista->tail = NULL;

    } else{

        nodoEliminar->anterior->siguiente = nodoEliminar->siguiente;

        nodoEliminar->siguiente->anterior = nodoEliminar->anterior;

        if(nodoEliminar == lista->head) lista->head = nodoEliminar->siguiente;

        if(nodoEliminar == lista->tail) lista->tail = nodoEliminar->anterior;

    }

    free(nodoEliminar);

    lista->size--;

    return 0;

}

int borrarIndice(struct lista \*lista, int posicion){

    if(posicion < 0 || posicion >= lista->size) return -1; // posicion no valida

    struct nodo \*nodoEliminar = lista->head;

    for(int i = 0; i < posicion; i++) nodoEliminar = nodoEliminar->siguiente;

    if(lista->size == 1){

        lista->head = NULL;

        lista->tail = NULL;

    } else{

        nodoEliminar->anterior->siguiente = nodoEliminar->siguiente;

        nodoEliminar->siguiente->anterior = nodoEliminar->anterior;

        if(nodoEliminar == lista->head) lista->head = nodoEliminar->siguiente;

        if(nodoEliminar == lista->tail) lista->tail = nodoEliminar->anterior;

    }

    free(nodoEliminar);

    lista->size--;

    return 0;

}

int size(struct lista \*lista){

    return lista->size;

}

// main

int main(){

    struct lista miLista;

    crearLista(&miLista);

    pf("\n-----Insertar elementos-----\n\n");

    insertar(&miLista, 1);

    insertar(&miLista, 2);

    insertarFinal(&miLista, 3);

    lista(&miLista, 4, 1);

    lista(&miLista);

    listaPosiciones(&miLista);

    pf("\n-----Buscar elementos----\n\n");

    pf("Buscando el 2... %s\n", buscar(&miLista, 2) != -1 ? "Dato encontrado" : "Dato No encontrado");

    pf("Buscando el 5... %s\n", buscar(&miLista, 5) != -1 ? "Dato encontrado" : "Dato No encontrado");

    pf("\n-----Buscar el primer nodo----\n\n");

    struct nodo \*nodoBuscado = buscarNodo(&miLista, 1);

    if(nodoBuscado != NULL){

        pf("Nodo encontrado: %d\n", nodoBuscado->dato);

        pf("Nodo: Anterior(%d) - Actual(%d) - Siguiente(%d)\n",

           nodoBuscado->anterior->dato,

           nodoBuscado->dato,

           nodoBuscado->siguiente->dato);

    }

    pf("\n-----Borrar por elementos----\n\n");

    borrar(&miLista, 2);

    lista(&miLista);

    pf("\n-----Borrar por indice----\n\n");

    borrarIndice(&miLista, 1);

    lista(&miLista);

    pf("\nTamaño de la lista: %d\n", size(&miLista));

    // Liberar memoria

    lista(&miLista);

    return 0;

}