LISTAS

ESTRUCTURAS DE DATOS

Nombre: Daniela Beatriz Martínez Montes
Profesor: Oscar Flores
Grupo: 2.-B

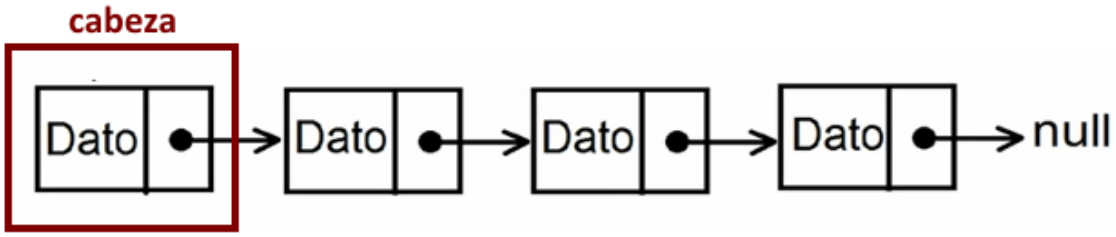
Listas

Una lista es una estructura de datos que nos permite agrupar elementos de una manera organizada. Las listas al igual que los algoritmos son muy importantes en la computación y críticas en muchos programas informáticos.

Tipos de listas

Lista enlazada

Una lista enlazada tiene un conjunto de nodos, los cuales almacenan 2 tipos de información: El dato que contienen y un puntero al siguiente nodo en la lista. El último nodo de la lista tiene como siguiente nodo el valor NULL. Entonces las listas enlazadas simples solo pueden ser recorridas en una dirección, apuntando al nodo siguiente, mas no a un nodo anterior.



Características

•Los elementos se distribuyen de forma dispersa por la memoria:

•Acceso a los elementos no destructivo:

•El tamaño de la lista puede modificarse de forma dinámica:

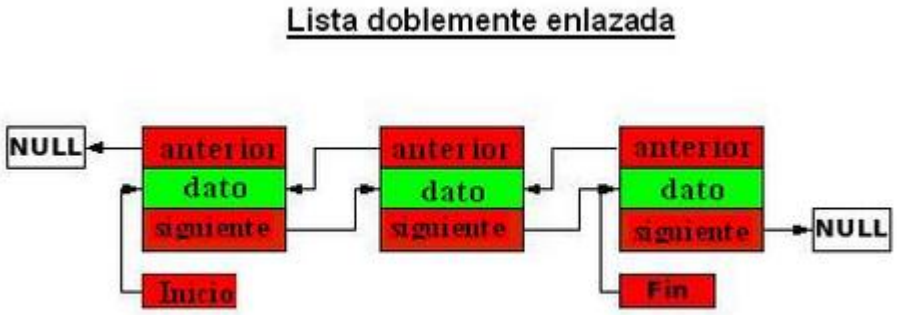
Ejemplo de lista enlazada en C++

```
/* ---------- lista.h ---------- */
typedef struct ElementoLista
{
  char *dato;
  struct ElementoLista *siguiente;
} Elemento;

typedef struct ListaIdentificar
{
  Elemento *inicio;
  Elemento *fin;
  int tamaño;
} Lista;

/* inicialización de la lista */
void inicialización (Lista * lista);


/* INSERCION */

/* inserción en une lista vacía */
int ins_en_lista_vacia (Lista * lista, char *dato);

/* inserción al inicio de la lista */
int ins_inicio_lista (Lista * lista, char *dato);

/* inserción al final de la lista */
int ins_fin_lista (Lista * lista, Elemento * actual, char *dato);

/* inserción en otra parte */
int ins_lista (Lista * lista, char *dato, int pos);
```

```
/* SUPRESION */

int sup_inicio (Lista * lista);
int sup_en_lista (Lista * lista, int pos);


int menu (Lista *lista,int *k);
void muestra (Lista * lista);
void destruir ( Lista * lista);
/* -------- FIN lista.h --------- */


===lista _function.h===
```

Lisa doblemente enlazada

Las listas doblemente enlazadas son un tipo de lista lineal en la que cada nodo tiene dos enlaces, uno que apunta al nodo siguiente, y el otro que apunta al nodo anterior. Las listas doblemente enlazadas no requieren de un nodo explicito para acceder a ellas, ya que presentan una gran ventaja comparada con las listas enlazadas y es que pueden recorrerse en ambos sentidos a partir de cualquier nodo de la lista, ya que siempre es posible desde cualquier nodo alcanzar cualquier otro nodo de la lista, hasta que se llega a uno de los extremos.



Características

- Recorrido secuencial en ambas direcciones
- Mayor ocupación: 2 referencias en cada nodo
- Inserción y borrado: Modificar más referencias
- Borrado más simple: Localizado el elemento a borrar se accede al anterior / siguiente

Ejemplo de lista doblemente enlazada en C++

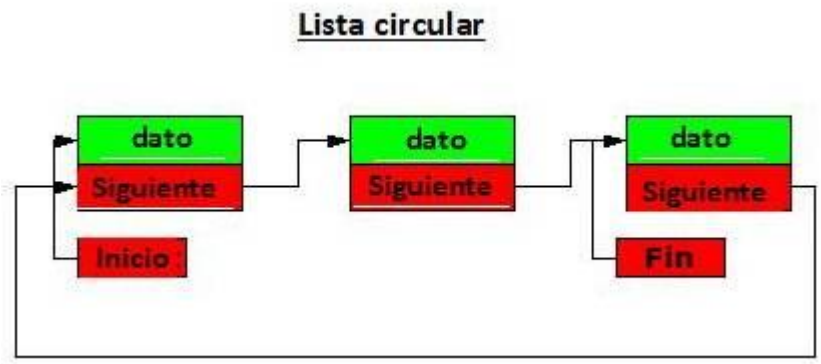
```
#include <stdio.h>
#define ASCENDENTE 1
#define DESCENDENTE 0
typedef struct _nodo {
int valor;
struct _nodo *siguiente;
struct _nodo *anterior;
} tipoNodo;
typedef tipoNodo *pNodo;
typedef tipoNodo *Lista;
/* Funciones con listas: */
void Insertar(Lista *l, int v);
void Borrar(Lista *l, int v);
void BorrarLista(Lista *);
void MostrarLista(Lista l, int orden);
int main()
{
Lista lista = NULL;
pNodo p;
Insertar(&lista, 20);
Insertar(&lista, 10);
Insertar(&lista, 40);
Insertar(&lista, 30);
MostrarLista(lista, ASCENDENTE);
MostrarLista(lista, DESCENDENTE);
Borrar(&lista, 10);
Borrar(&lista, 15);
Borrar(&lista, 45);
Borrar(&lista, 30);
MostrarLista(lista, ASCENDENTE);
MostrarLista(lista, DESCENDENTE);
BorrarLista(&lista);
system("PAUSE");
return 0;
}
void Insertar(Lista *lista, int v)
{
pNodo nuevo, actual;
/* Crear un nodo nuevo */
nuevo = (pNodo)malloc(sizeof(tipoNodo));

nuevo->valor = v;
/* Colocamos actual en la primera posición de la lista */
actual = *lista;
if(actual) while(actual->anterior) actual = actual->anterior;
```

```
/* Si la lista está vacía o el primer miembro es mayor que el nuevo */
if(!actual || actual->valor > v) {
/* Añadimos la lista a continuación del nuevo nodo */
nuevo->siguiente = actual;
nuevo->anterior = NULL;
if(actual) actual->anterior = nuevo;
if(!*lista) *lista = nuevo;
}

 else {
 /* Avanzamos hasta el último elemento o hasta que el siguiente tenga
 un valor mayor que v */
 while(actual->siguiente &&actual->siguiente->valor <= v)
 actual = actual->siguiente;
 /* Insertamos el nuevo nodo después del nodo anterior */
 nuevo->siguiente = actual->siguiente;
 actual->siguiente = nuevo;
 nuevo->anterior = actual;
 if(nuevo->siguiente) nuevo->siguiente->anterior = nuevo;
 }
 }
 void Borrar(Lista *lista, int v)
 {
 pNodo nodo;
 /* Buscar el nodo de valor v */
 nodo = *lista;
 while(nodo && nodo->valor < v) nodo = nodo->siguiente;
 /* Buscar el nodo de valor v */
nodo = *lista;
while(nodo && nodo->valor < v) nodo = nodo->siguiente;
while(nodo && nodo->valor > v) nodo = nodo->anterior;
/* El valor v no está en la lista */
if(!nodo || nodo->valor != v) return;
/* Borrar el nodo */
/* Si lista apunta al nodo que queremos borrar, apuntar a otro */
if(nodo == *lista)
if(nodo->anterior) *lista = nodo->anterior;
else *lista = nodo->siguiente;
if(nodo->anterior) /* no es el primer elemento */
nodo->anterior->siguiente = nodo->siguiente;
if(nodo->siguiente) /* no es el último nodo */
nodo->siguiente->anterior = nodo->anterior;
free(nodo);
}
void BorrarLista(Lista *lista)
{
pNodo nodo, actual;
actual = *lista;
while(actual->anterior) actual = actual->anterior;

while(actual) {
nodo = actual;
actual = actual->siguiente;
free(nodo);
}
*lista = NULL;
}

void MostrarLista(Lista lista, int orden)
{
pNodo nodo = lista;
if(!lista) printf("Lista vacía");
nodo = lista;
if(orden == ASCENDENTE) {
while(nodo->anterior) nodo = nodo->anterior;
printf("Orden ascendente: ");
while(nodo) {
printf("%d -> ", nodo->valor);
```

```
nodo = nodo->siguiente;
}
}
Else
{
while(nodo->siguiente)
   nodo = nodo->siguiente;
printf("Orden descendente: ");
```

Listas Circulares

Una lista circular es una lista lineal en la que el último nodo a punta al primero. Las listas circulares evitan excepciones en la operaciones que se realicen sobre ellas. No existen casos especiales, cada nodo siempre tiene uno anterior y uno siguiente. En algunas listas circulares se añade un nodo especial de cabecera, de ese modo se evita la única excepción posible, la de que la lista esté vacía



Características

- No existe algún elemento que apunte a NULL
- Se integra una estructura tipo anillo
- Solo hay una cabeza
- La cabeza siempre será el siguiente enlace para algún nodo
- Se pueden llegar a crear recorridos en bucles infinitos

Ejemplo de lista circular en c++

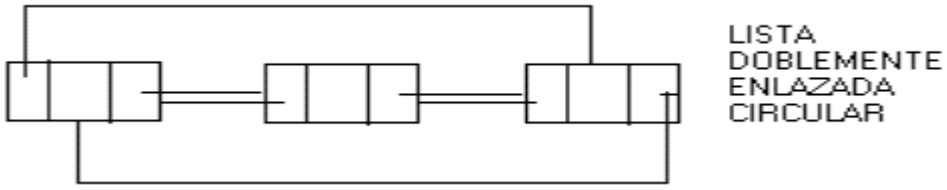
```
#include <stdlib.h>
#include <stdio.h>
typedef struct _nodo {
int valor;
struct _nodo *siguiente;
} tipoNodo;
typedef tipoNodo *pNodo;
typedef tipoNodo *Lista;
// Funciones con listas:
void Insertar(Lista *l, int v);
void Borrar(Lista *l, int v);
void BorrarLista(Lista *);
void MostrarLista(Lista l);
int main()
{
Lista lista = NULL;
pNodo p;
Insertar(&lista, 10);
Insertar(&lista, 40);
Insertar(&lista, 30);
Insertar(&lista, 20);
Insertar(&lista, 50);
MostrarLista(lista);
Borrar(&lista, 30);
Borrar(&lista, 50);
MostrarLista(lista);
BorrarLista(&lista);
system("PAUSE");
return 0;
}
```

```
void Insertar(Lista *lista, int v)
{
   pNodo nodo;
  // Creamos un nodo para el nuvo valor a insertar
     nodo = (pNodo)malloc(sizeof(tipoNodo));
     nodo->valor = v;
      // Si la lista está vacía, la lista será el nuevo nodo
      // Si no lo está, insertamos el nuevo nodo a continuación del
      //  apuntado por lista
   if(*lista == NULL) *lista = nodo;
   else nodo->siguiente = (*lista)->siguiente;
      // En cualquier caso, cerramos la lista circular
     (*lista)->siguiente = nodo;
}

void Borrar(Lista *lista, int v)
{
    pNodo nodo;
    nodo = *lista;
     // Hacer que lista apunte al nodo anterior al de valor v
    do {
      if((*lista)->siguiente->valor != v) *lista = (*lista)->siguiente;
      } while((*lista)->siguiente->valor != v && *lista != nodo);
 // Si existe un nodo con el valor v:
     if((*lista)->siguiente->valor == v)
     {
        // Y si la lista sólo tiene un nodo
        if(*lista == (*lista)->siguiente) {
       // Borrar toda la lista
        free(*lista);
    *lista = NULL;
    }
    else
    {
     // Si la lista tiene más de un nodo, borrar el nodo de valor v
      nodo = (*lista)->siguiente;
      (*lista)->siguiente = nodo->siguiente;
       free(nodo);
    }
 }
}

void BorrarLista(Lista *lista)
```

```
void BorrarLista(Lista *lista)
{
pNodo nodo;
// Mientras la lista tenga más de un nodo
while((*lista)->siguiente != *lista) {
// Borrar el nodo siguiente al apuntado por lista
nodo = (*lista)->siguiente;
(*lista)->siguiente = nodo->siguiente;
free(nodo);
}
// Y borrar el último nodo
```

Listas circulares dobles

En las listas circulares doblemente enlazadas cada nodo tiene un par de campos de enlace, uno al nodo siguiente, y otro al anterior. Un campo de enlace permite atravesar la lista hacia adelante, mientras que el otro permite atravesar la lista hacia atrás.



En las listas circulares dobles, nunca se llega a una posición en la que ya no sea posible desplazarse.

Cuando se llegue al último elemento, el desplazamiento volverá a comenzar desde el primer elemento