Capítulo 4 Listas circulares:

4.1 Definición

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.

El nodo típico es el mismo que para construir listas abiertas:

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

4.2 Declaraciones de tipos para manejar listas circulares en C:

Los tipos que definiremos normalmente para manejar listas cerradas son los mismos que para para manejar listas abiertas:

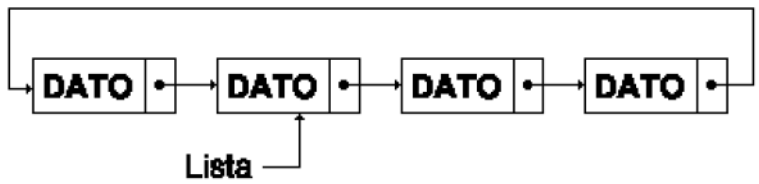
```
typedef struct _nodo {
   int dato;
   struct _nodo *siguiente;
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Lista;
```

tipoNodo es el tipo para declarar nodos, evidentemente.

pNodo es el tipo para declarar punteros a un nodo.

Lista es el tipo para declarar listas, tanto abiertas como circulares. En el caso de las circulares, apuntará a un nodo cualquiera de la lista.



A pesar de que las listas circulares simplifiquen las operaciones sobre ellas, también introducen algunas complicaciones. Por ejemplo, en un proceso de búsqueda, no es tan sencillo dar por terminada la búsqueda cuando el elemento buscado no existe.

Por ese motivo se suele resaltar un nodo en particular, que no tiene por qué ser siempre el mismo. Cualquier nodo puede cumplir ese propósito, y puede variar durante la ejecución del programa.

Otra alternativa que se usa a menudo, y que simplifica en cierto modo el uso de listas circulares es crear un nodo especial de hará la función de nodo cabecera. De este modo, la lista nunca estará vacía, y se eliminan casi todos los casos especiales.

4.3 Operaciones básicas con listas circulares:

A todos los efectos, las listas circulares son como las listas abiertas en cuanto a las operaciones que se pueden realizar sobre ellas:

- Añadir o insertar elementos.
- Buscar o localizar elementos.
- Borrar elementos.
- Moverse a través de la lista, siguiente.

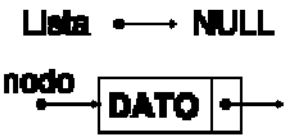
Cada una de éstas operaciones podrá tener varios casos especiales, por ejemplo, tendremos que tener en cuenta cuando se inserte un nodo en una lista vacía, o cuando se elimina el único nodo de una lista.

4.4 Añadir un elemento:

El único caso especial a la hora de insertar nodos en listas circulares es cuando la lista esté vacía.

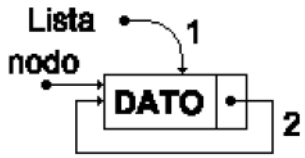
Añadir elemento en una lista circular vacía:

Partiremos de que ya tenemos el nodo a insertar y, por supuesto un puntero que apunte a él, además el puntero que define la lista, que valdrá NULL:



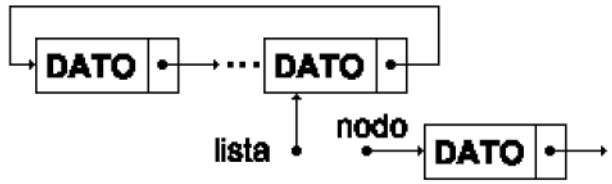
El proceso es muy simple, bastará con que:

1. lista apunta a nodo.
2. lista->siguiente apunte a nodo.



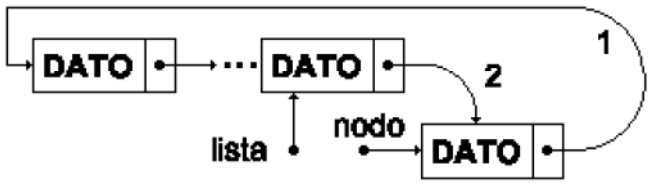
Añadir elemento en una lista circular no vacía:

De nuevo partiremos de un nodo a insertar, con un puntero que apunte a él, y de una lista, en este caso, el puntero no será nulo:



El proceso sigue siendo muy sencillo:

1. Hacemos que nodo->siguiente apunte a lista->siguiente.
2. Después que lista->siguiente apunte a nodo.



Añadir elemento en una lista circular, caso general:

Para generalizar los dos casos anteriores, sólo necesitamos añadir una operación:

1. Si lista está vacía hacemos que lista apunte a nodo.
2. Si lista no está vacía, hacemos que nodo->siguiente apunte a lista->siguiente.
3. Después que lista->siguiente apunte a nodo.

4.5 Buscar o localizar un elemento de una lista circular:

A la hora de buscar elementos en una lista circular sólo hay que tener una precaución, es necesario almacenar el puntero del nodo en que se empezó la búsqueda, para poder detectar el caso en que no exista el valor que se busca. Por lo demás, la búsqueda es igual que en el caso de las listas abiertas, salvo que podemos empezar en cualquier punto de la lista.

4.6 Eliminar un elemento de una lista circular:

Para ésta operación podemos encontrar tres casos diferentes:

1. Eliminar un nodo cualquiera, que no sea el apuntado por lista.
2. Eliminar el nodo apuntado por lista, y que no sea el único nodo.
3. Eliminar el único nodo de la lista.

En el primer caso necesitamos localizar el nodo anterior al que queremos borrar. Como el principio de la lista puede ser cualquier nodo, haremos que sea precisamente lista quien apunte al nodo anterior al que queremos eliminar.

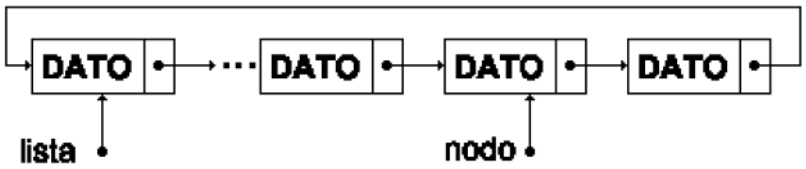
Esto elimina la excepción del segundo caso, ya que lista nunca será el nodo a eliminar, salvo que sea el único nodo de la lista.

Una vez localizado el nodo anterior y apuntado por lista, hacemos que lista->siguiente apunte a nodo->siguiente. Y a continuación borramos nodo.

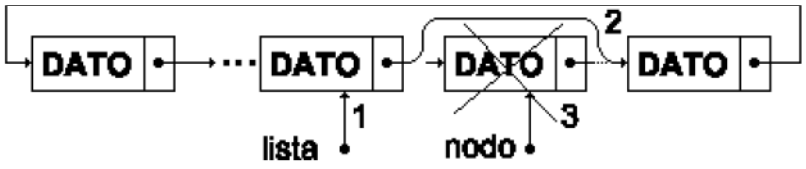
En el caso de que sólo exista un nodo, será imposible localizar el nodo anterior, así que simplemente eliminaremos el nodo, y haremos que lista valga NULL.

Eliminar un nodo en una lista circular con más de un elemento:

Consideraremos los dos primeros casos como uno sólo.



1. El primer paso es conseguir que lista apunte al nodo anterior al que queremos eliminar. Esto se consigue haciendo que lista valga lista->siguiente mientras lista->siguiente sea distinto de nodo.
2. Hacemos que lista->siguiente apunte a nodo->siguiente.
3. Eliminamos el nodo.



Eliminar el único nodo en una lista circular:

Este caso es mucho más sencillo. Si lista es el único nodo de una lista circular:

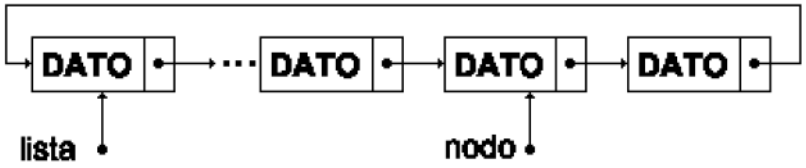
1. Borramos el nodo apuntado por lista.

2. Hacemos que lista valga NULL.

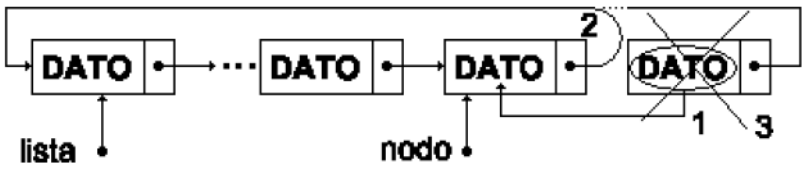
Otro algoritmo para borrar nodos:

Existe un modo alternativo de eliminar un nodo en una lista circular con más de un nodo.

Supongamos que queremos eliminar un nodo apuntado por nodo:



1. Copiamos el contenido del nodo->siguiente sobre el contenido de nodo.
2. Hacemos que nodo->siguiente apunte a nodo->siguiente->siguiente.
3. Eliminamos nodo->siguiente.
4. Si lista es el nodo->siguiente, hacemos lista = nodo.



Este método también funciona con listas circulares de un sólo elemento, salvo que el nodo a borrar es el único nodo que existe, y hay que hacer que lista apunte a NULL.

4.7 Ejemplo de lista circular en C:

Construiremos una lista cerrada para almacenar números enteros. Haremos pruebas insertando varios valores, buscándolos y eliminándolos alternativamente para comprobar el resultado.

Algoritmo de la función "Insertar":

1. Si lista está vacía hacemos que lista apunte a nodo.
2. Si lista no está vacía, hacemos que nodo->siguiente apunte a lista->siguiente.
3. Después que lista->siguiente apunte a nodo.

```
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;
}
```

Algoritmo de la función "Borrar":

1. ¿Tiene la lista un único nodo?
2. **SI:**
    1. Borrar el nodo lista.
    2. Hacer lista = NULL.
3. **NO:**
    1. Hacemos lista->siguiente = nodo->siguiente.
    2. Borramos 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);
      }
   }
}
```

Código del ejemplo completo:

Tan sólo nos queda escribir una pequeña prueba para verificar el funcionamiento:

```
#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)
{
   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
   free(*lista);
   *lista = NULL;
}

void MostrarLista(Lista lista)
{
   pNodo nodo = lista;

   do {
      printf("%d -> ", nodo->valor);
      nodo = nodo->siguiente;
   } while(nodo != lista);
   printf("\n");
}
```