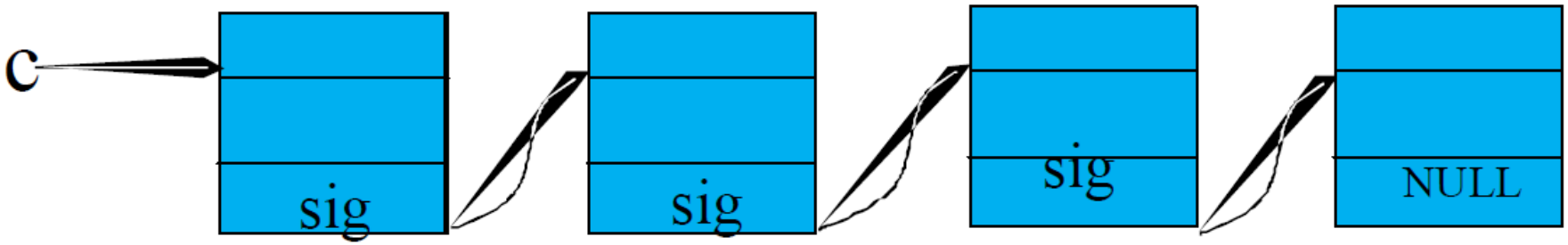
El lenguaje de programación C
- Listas Enlazadas-

Lista enlazada simple

- La forma más simple de estructura dinámica es la lista simplemente enlazada o lista abierta. En esta forma los nodos se organizan de modo que cada uno apunta al siguiente, y el último no apunta a nada, es decir, el puntero del nodo siguiente vale NULL:



Definir una lista

- Para crear un alista debemos definir la clase de elementos que van a formar parte de la misma. Un tipo de dato genérico podría ser la siguiente estructura:

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

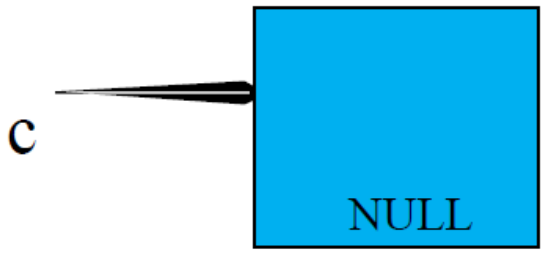
Acceder a la lista

- Para acceder a un nodo de la estructura sólo necesitaremos un puntero a un nodo. En el ejemplo anterior declaramos una variable de estructura, que va a ser un puntero a dicha estructura mediante typedef declaramos un nuevo tipo de dato:

```
typedef struct s
{
  int dato;
  struct s *siguiente;
} elemento;
elemento *c; //puntero a nodo
```

Lista vacía

- Cuando el puntero que usamos para acceder a la lista vale NULL, diremos que la lista está vacía: **c = NULL; //lista vacía**



Operaciones Básicas con Listas

- Con las listas se pueden realizar las siguientes operaciones básicas:

  a) Crear lista.

  b) Añadir o insertar elementos.

  c) Buscar o localizar elementos.

  d) Borrar elementos.

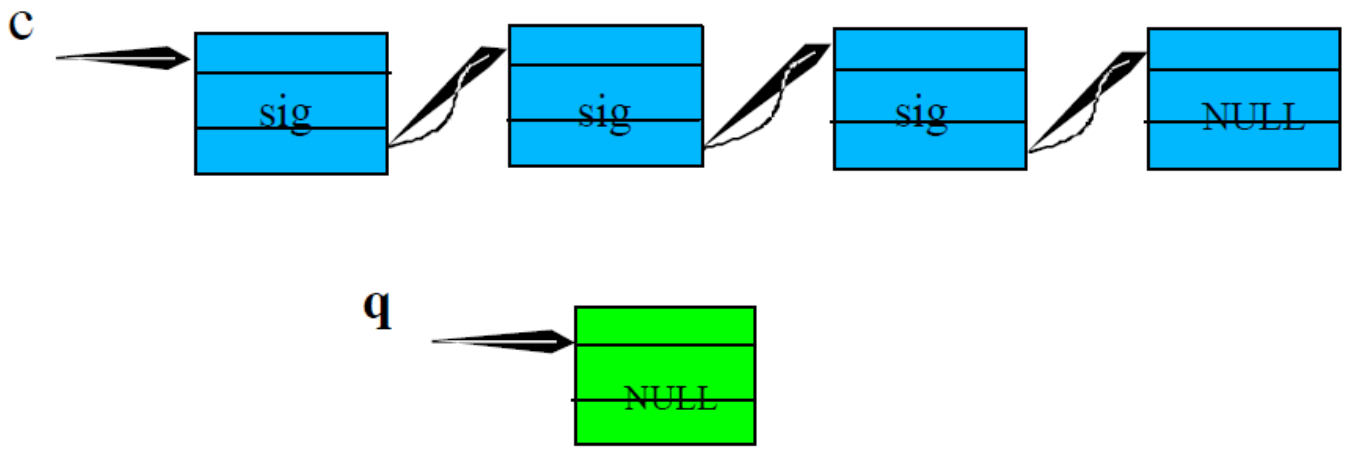
  e) Moverse a través de una lista.

  f) Ordenar una lista.

Crear Lista

```
elemento *q;
 q = (elemento *)malloc(sizeof(elemento));
 if(!q)
    perror("No se ha reservado memoria para el nuevo nodo");
```

Añadir o insertar elementos




```
elemento *q;
 q = (elemento *)malloc(sizeof(elemento));
 q->siguiente = NULL;
```

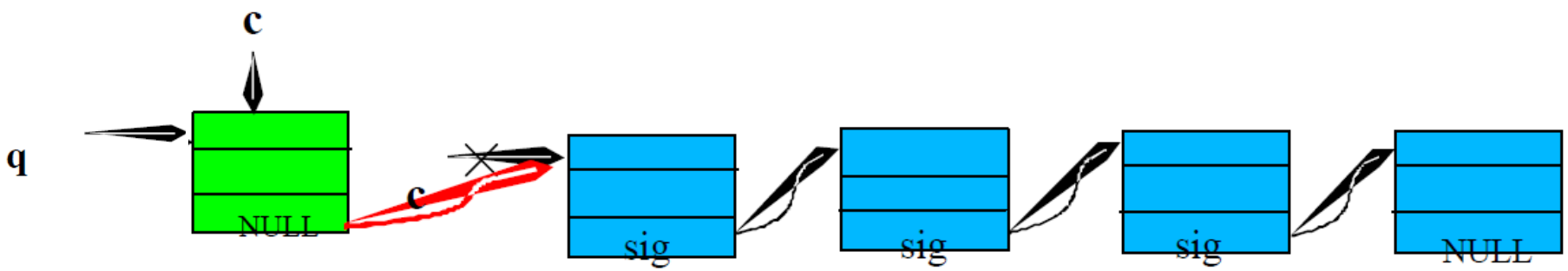
Recorrer una lista

```
. . . .
q = c;
while(q != NULL)
{
        printf(“%d\t”, q->num);
        q = q->sig;
}
. . . .
```

Encontrar un elemento

```
. . . .
q = c;
printf("¿Valor a localizar?");
scanf("%d", &x);
while(q != NULL && q->num != x)
        q = q->sig;
 . . . .
```

Agregar al inicio



Apuntamos q->sig a c:

q->sig = c;

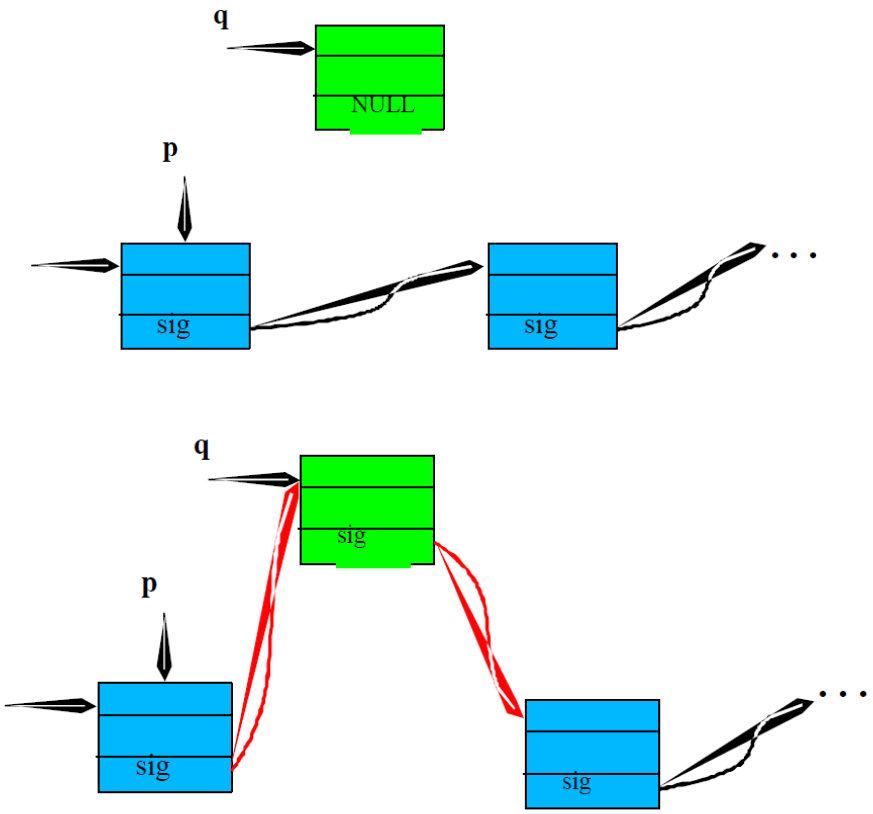
Ahora reapuntamos c a q:

c = q;

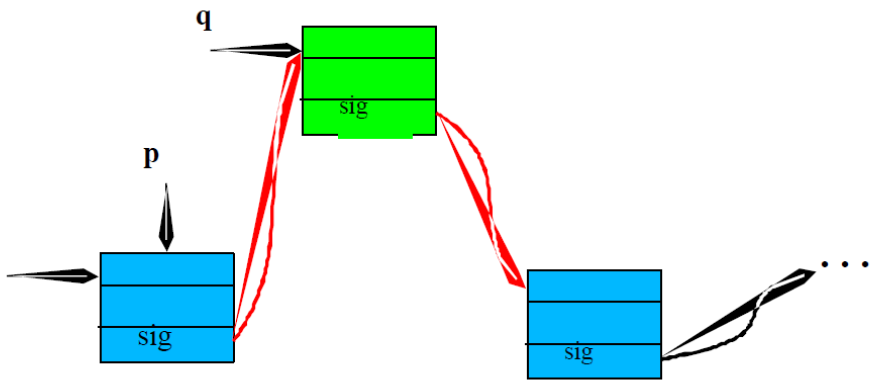
Es fundamental no alterar el orden de las operaciones.

c->dato = 10;

Insertar en el medio



Insertar en el medio(2)



q->num = n; //n es el entero que insertamos

q->sig = p->sig;

p->sig = q;

Insertar al final de la lista

- El proceso en este caso tampoco es excesivamente complicado: Necesitamos un puntero que señale al último elemento de la lista. La manera de conseguirlo es empezar por el primero y avanzar hasta que el nodo que tenga como siguiente el valor NULL (es decir, el último). Hacer que nodo->siguiente sea NULL. Hacer que ultimo->siguiente sea nodo.

Borrar toda la lista

- Borrar todos los elementos de una lista equivale a liberar la memoria reservada para cada uno de los elementos de la misma. Si el primer nodo está apuntado por c, empleamos el puntero auxiliar q = c; //salvamos el puntero a la lista

```
while(q != NULL)
{
  q = c;
  c = c->sig;
  free(q);
        }
```

Hay que observar que antes de borrar el elemento apuntado por q, hacemos que c apunte al siguiente elemento ya que si no perdemos el resto de la lista

Borrar un elemento de la lista

- En todos los demás casos, eliminar un nodo se puede hacer siempre del mismo modo. Supongamos que tenemos una lista con al menos dos elementos, y un puntero **p** al nodo anterior al que queremos eliminar. Y un puntero auxiliar **q**.
- Hacemos que **q** apunte al nodo que queremos borrar (el siguiente a p): **q = p->sig**;
- Ahora, asignamos como nodo siguiente del nodo anterior, el siguiente al que queremos eliminar**: p->sig = q->sig;**
- Eliminamos la memoria asociada al nodo que queremos eliminar. free(q);
- Si el nodo a eliminar es el último, es procedimiento es igualmente válido, ya que **p** pasará a ser el último, y **p->sig** valdrá **NULL**.