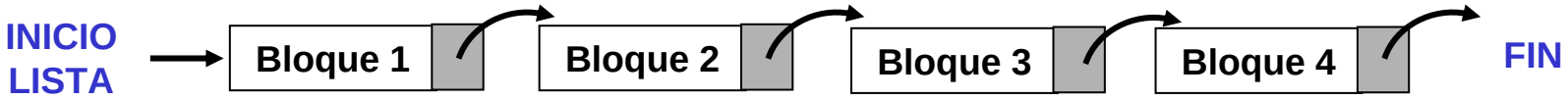
UNIDAD 12: Listas Enlazadas

Lista enlazada = Estructura de datos lineal, formada por bloques de información, con un formato común, enlazados entre sí mediante punteros.



Características de listas enlazadas

- **Los elementos se distribuyen de forma dispersa por la memoria:**
  Los bloques de información no ocupan posiciones consecutivas en la memoria. El orden de la lista la establecen los enlaces entre bloques de información.

- **Acceso a los elementos aleatorio:**
  Puede extraerse información de cualquier elemento o insertar un bloque en cualquier posición.

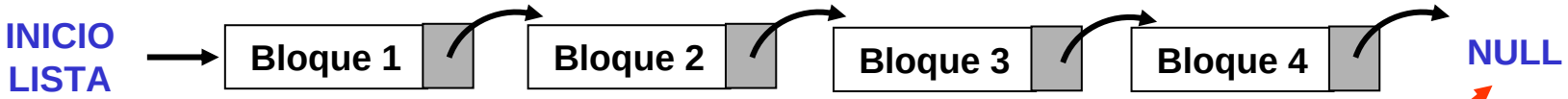
- **Acceso a los elementos no destructivo:**
  Al contrario que en colas y pilas, la extracción de un bloque no implica su eliminación.

- **El tamaño de la lista puede modificarse de forma dinámica:**
  Al contrario que en colas y pilas, no hay un número máximo de elementos de la lista (salvo limitaciones de la capacidad de almacenamiento de la máquina)

Listas simplemente enlazadas

Cada elemento contiene un enlace con el siguiente elemento de la lista.

- Conocida la posición de un elemento, podemos recorrer la lista hacia delante
- No es posible recorrer la lista hacia atrás.



Listas simplemente enlazadas en C

```
struct direc {
        char nombre[40];
        char calle[40];
        char ciudad[20];
        int codigo;
        struct direc  * sig;
        } info;
```

Información del Elemento

Enlace con el siguiente elemento

Puntero a la propia estructura

Por convenio, el fin de lista se indica como NULL

- Cada elemento requiere una reserva dinámica de memoria al ser creado (liberar memoria al ser eliminado).
- La gestión de la lista se hace con funciones apropiadas para añadir, insertar, borrar, buscar elementos, etc.

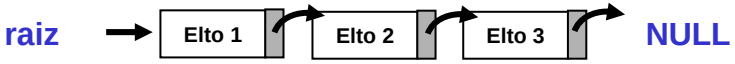
Listas simplemente enlazadas

Construcción de una Lista simplemente enlazada en C

Dos Formas

Lista Desordenada

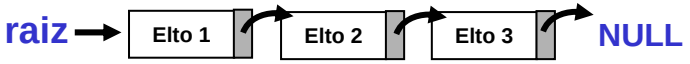
- No hay orden específico para los elementos.
- La lista se construye añadiendo cada nuevo elemento al final de la lista existente.
- Si no existe lista, el nuevo elemento pasa a ser el primero y único de la lista.



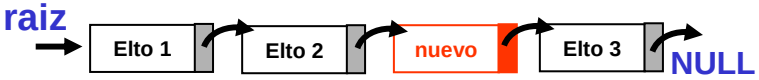
nuevo elemento
AL FINAL

Lista Ordenada

- Los elementos se añaden para que la lista final tenga un orden específico.
- Hay que buscar el punto de inserción del elemento según el orden establecido.
- La inserción puede ser al principio, en medio o al final.
- Si no existe lista, el nuevo elemento pasa a ser el primero y único de la lista.



nuevo elemento
INSERCIÓN

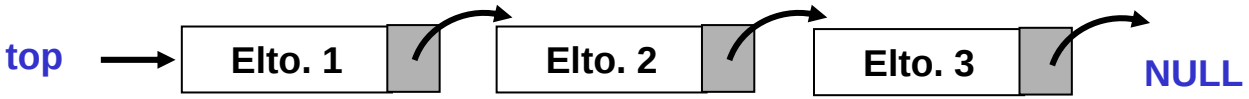


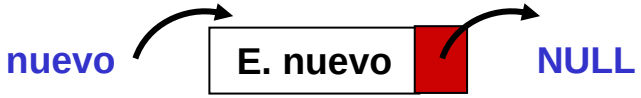
Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

1.- Lista Desordenada

```
struct direc * almac_fin( struct direc *top, struct direc * nuevo)
{
struct direc * p = top; /* puntero que recorrerá la lista hasta el final */
if (top = = NULL) /* si no existe lista la crea */
 return nuevo ;
while( p -> sig ) /* recorre la lista hasta que p apunte al último */
 { p = p -> sig ; }
p -> sig = nuevo; /* enlaza el nuevo elemento al final */
return top; /* devuelve la cabeza */
}
```





Listas simplemente enlazadas

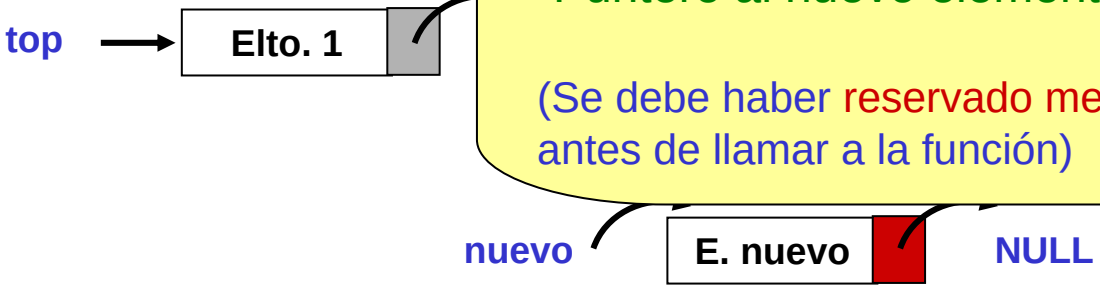
Funciones de creación de Listas simplemente enlazadas

1.- Lista Desordenada

```
struct direc * almac_fin( struct direc *top, struct direc * nuevo)
{
struct direc * p = top; /* puntero que recorrerá la lista hasta el final */
if (top = = NULL) /* si no existe lista la cre
 return nuevo ;
while( p -> sig ) /* recorre la lista hasta  punte al último */
 { p = p -> sig ; }
p -> sig = nuevo; /* enlaza el nuevo e inal */
return top; /* devuelve la cabeza */
}
```

Recibe:
-Puntero al comienzo de la lista top
-Puntero al nuevo elemento a insertar nuevo

(Se debe haber reservado memoria dinámicamente antes de llamar a la función)



Listas simplemente enlazadas

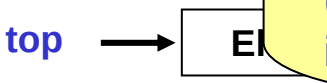
Funciones de creación de Listas simplemente enlazadas

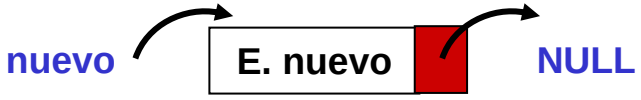
1.- Lista Desordenada

```
struct direc * almac_fin( struct direc *top, struct direc * nuevo)
{
struct direc * p = top; /* puntero que recorrerá la lista hasta el final */
if (top = = N    ) /* si no existe lista la crea */
 return nuevo ;
while( p -> sig ) /*    la lista hasta que p apunte al último */
 { p = p -> sig ; }
p -> sig = nuevo; /* enla        lemento al final */
return top; /* de
}
```

Devuelve:
-Puntero al comienzo de la lista

(En este caso la cabecera de la lista sólo varía al insertar el primer elemento)





Listas simplemente enlazadas

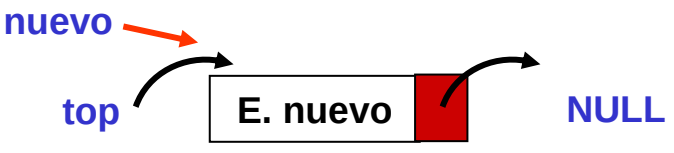
Funciones de creación de Listas simplemente enlazadas

1.- Lista Desordenada

```
struct direc * almac_fin( struct direc *top, struct direc * nuevo)
{
struct direc * p = top; /* puntero que recorrerá la lista hasta el final */
if (top = = NULL) /* si no existe lista la crea */
  return nuevo ;
while( p -> sig ) /* reco      ista hasta que p apunte al último */
 { p = p -> sig ; }
p -> sig = nuevo;
return top; /* dev
}
```

Si el principio de la lista es NULL, la lista no existe y la creamos: Su primer elemento es el que le pasamos, nuevo, y es también la nueva cabecera de la lista.

La lista no existe



El nuevo Elto es el primero y único de la lista

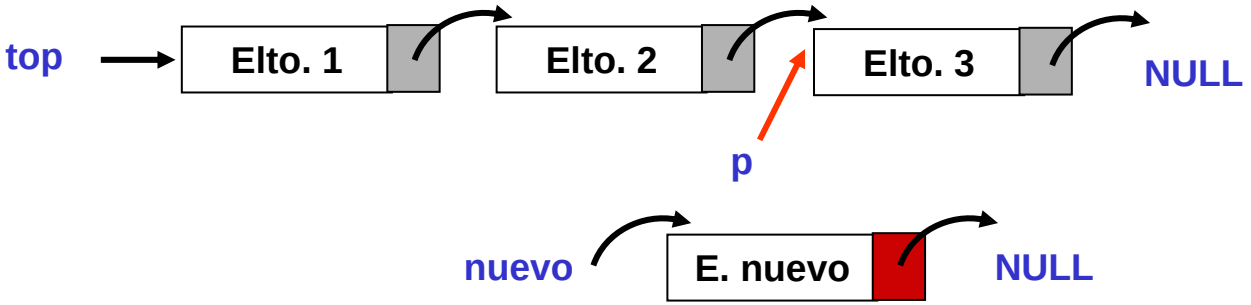
Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

1.- Lista Desordenada

```
struct direc * almac_fin( stru
{
struct direc * p = top; /* punte
if (top = = NULL) /* si no existe lista
 return nuevo ;
while( p -> sig ) /* recorre la lista hasta que p apunte al último */
 { p = p -> sig ; }
p -> sig = nuevo; /* enlaza el nuevo elemento al final */
return top; /* devuelve la cabeza */
}
```

Si el principio de la lista no es NULL, la lista ya tiene al menos un elemento.
Recorremos la lista hasta encontrar el último elemento (Al salir del while, p apunta al último elemento)



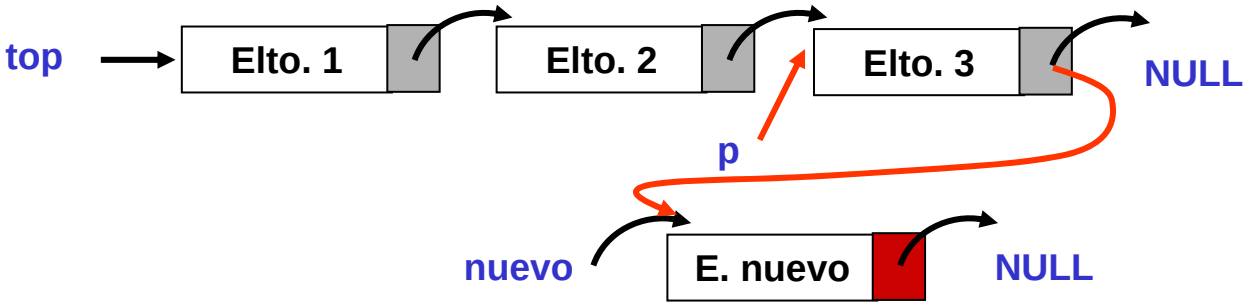
Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

1.- Lista Desordenada

```
struct direc * almac_fin( stru
{
struct direc * p = top; /* punte
if (top = = NULL) /* si no existe lista
 return nuevo ;
while( p -> sig ) /* recorre la  hasta que p apunte al último */
 { p = p -> sig ; }
p -> sig = nuevo; /* enlaza el nuevo elemento al final */
return top; /* devuelve la cabeza */
}
```

Alcanzado el final de la lista, sólo queda "enlazar" el nuevo elemento a continuación de éste.
(El siguiente elemento al último (p->sig) deberá apuntar a nuevo)



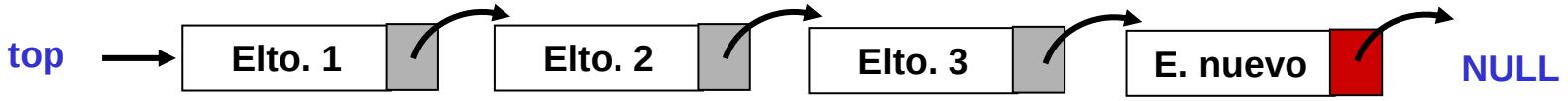
Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

1.- Lista Desordenada

```
struct direc * almac_fin( stru
{
struct direc * p = top; /* punt
if (top = = NULL) /* si no existe lista
 return nuevo ;
while( p -> sig ) /* recorre l      hasta que p apunte al último */
 { p = p -> sig ; }
p -> sig = nuevo;  /* enlaza el nuevo elemento al final */
return top; /* devuelve la cabeza */
}
```

Para finalizar devolvemos la cabecera de la lista (Haya ésta cambiado o no)



Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

Estructura base para el ejemplo de la función insertar

```
struct ciclista{
        int num; /* número del ciclista */
        char nombre[80]; /* nombre del ciclista */
        int tiempo; /* tiempo en la clasificación */

        struct ciclista * sig; /* puntero a siguiente */
        };
```

El objetivo es diseñar la función insertar para crear listas ordenadas de elementos struct ciclista.
El criterio de ordenación será de menor a mayor tiempo de clasificación.

```
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabecera de la lista */
}
```

Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

```
struct ciclista *insertar ( struct ciclista * top, struct ciclista * nuevo)
{
struct ciclista *actual = top; /* puntero para moverse por la lista */
struct ciclista *anterior = NULL; /*puntero elemento anterior al "actual" */
/* Búsqueda del punto de inserción */
while ( (actual != NULL) && (actual -> <= nuevo -> tiempo) )
 {
 anterior = actual;
 actual = actual -> sig;
 }
/* Inserción */
if (anterior != NULL) /*
 anterior -> sig = nuev
else /* inserción al pri
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabecera de la lista */
}
```

Recibe:
-Puntero al comienzo de la lista top
-Puntero al nuevo elemento a insertar nuevo

(Se debe haber reservado memoria dinámicamente antes de llamar a la función)

Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

```
struct ciclista *insertar ( struct ciclista * top, struct ciclista * nuevo)
{
struct cicl ta *actual = top; /* puntero para moverse por la lista */
struct ciclista nterior = NULL; /*puntero al elemento anterior al "actual" */
/* Búsqueda del  o de inserción */
while ( (actual != N & (actual -> tiempo <= nuevo -> tiempo) )
 {
 anterior = actual;
 actual = actual -> si
 }
/* Inserción */
if (anterior != NU
 anterior -> sig =
else /* inserción
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabecera de la lista */
}
```

Devuelve:
-Puntero al comienzo de la lista

(En este caso la cabecera de la lista varía si el nuevo elemento tiene un tiempo de clasificación inferior a todos los demás)

Listas simplemente enlazadas

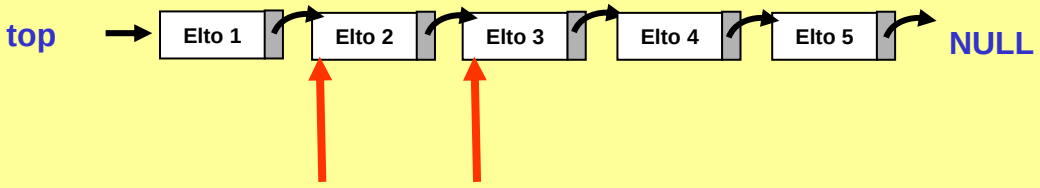
Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

```
struct ciclista *insertar ( struct ciclista * top, struct ciclista * nuevo)
{
struct ciclista *actual = top; /* puntero para moverse por la lista */
struct ciclista *anterior = NULL; /*puntero al elemento anterior al "actual" */
/* Búsqueda del punto de inserción */
while ( (actual != NULL) && (a                tiempo <= nuevo -> tiempo) )
 {
 anterior = actual;
 actual = actual -> sig
 }
/* Inserción */
if (anterior != NULL) /
 anterior -> sig = nue
else /* inserción al pr
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelv
}
```

Recorreremos la lista con dos punteros:

- actual que apunta al elemento que queremos examinar en ese momento
- anterior, que apunta al elemento precedente a actual



Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

```
struct ciclista *insertar ( struct ciclista * top, struct ciclista * nuevo)
{
struct ciclista *actual = top; /* puntero para moverse por la lista */
struct ciclista *anterior = NULL; /*puntero al elemento anterior al "actual" */
/* Búsqueda del punto de inserción */
while ( (actual != NULL) && (actual -> tiempo <= nuevo -> tiempo) )
 {
 anterior = actual;
 actual = actual -> sig;
 }
/* Inserción */
if (anterior != NULL) /* inserc
 anterior -> sig = nuevo;
else /* inserción al principio
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabe
}
```

Recorremos la lista desde el principio mientras:

1.- No lleguemos al final

Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

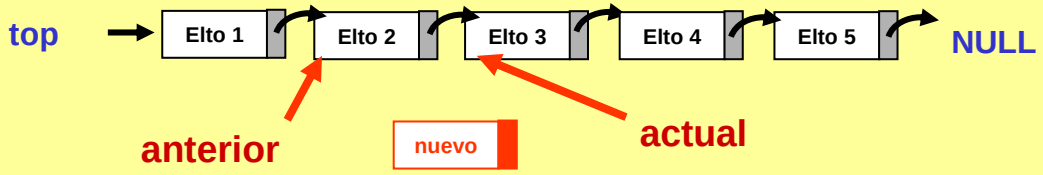
2.- Lista Ordenada

```
struct ciclista *insertar ( struct ciclista * top, struct ciclista * nuevo)
{
struct ciclista *actual = top; /* puntero para moverse por la lista */
struct ciclista *anterior = NULL; /*puntero al elemento anterior al "actual" */
/* Búsqueda del punto de inserción */
while ( (actual != NULL) && (actual -> tiempo <= nuevo -> tiempo) )
 {
 anterior = actual;
 actual = actua
 }
/* Inserción
if (anterior !=
 anterior ->
else /* inser
 top = nuevo
nuevo -> sig
return top; /
}
```

Y

2.- El elemento actual tenga un tiempo de clasificación inferior al del nuevo elemento.

Si no es así, el nuevo elemento debe insertarse entre los elementos de la lista apuntados por anterior y actual.



Listas simplemente enlazadas

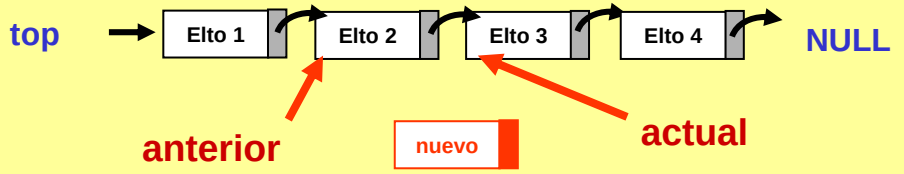
Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

```
struct ciclista *insertar
{
struct ciclista *actual =
struct ciclista *anterior
/* Búsqueda del punto
while ( (actual != NULL)
 {
 anterior = actual;
 actual = actual -> sig;
 }
/* Inserción */
if (anterior != NULL) /* inserción en medio o final */
 anterior -> sig = nuevo;
else /* inserción al principio de la lista */
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabecera de la lista */
}
```

Una vez localizado el punto de inserción, hay que distinguir dos casos:

1.- Que el punto de inserción sea en medio o al final



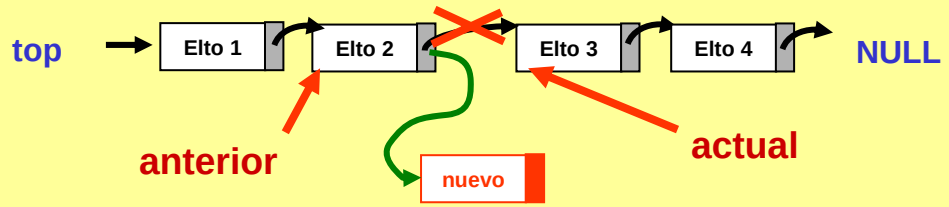
Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

```
struct ciclista *insertar
{
struct ciclista *actual =
struct ciclista *anterior
/* Búsqueda del punto
while ( (actual != NULL)
 {
 anterior = actual;
 actual = actual -> sig;
 }
/* Inserción */
if (anterior != NULL) /* inserción en medio o final */
 anterior -> sig = nuevo;
else /* inserción al principio de la lista */
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabecera de la lista */
}
```

En cuyo caso “Reconectamos” como



“anterior -> sig = nuevo”

Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

```
struct ciclista *insertar ( struct ciclista * top, struct ciclista * nuevo)
{
struct ciclista *actual = top; /* puntero para moverse por la lista */
struct ciclista *anterior = NULL; /* puntero al elemento anterior a "actual" */
/* Búsqueda del pu
while ( (actual != N
 {
 anterior = actual;
 actual = actual -> sig;
 }
/* Inserción */
if (anterior != NULL) /* inserción en medio o final */
 anterior -> sig = nuevo;
else /* inserción al principio de la lista */
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabecera de la lista */
}
```

Si no

2.- La inserción es al principio de la lista.

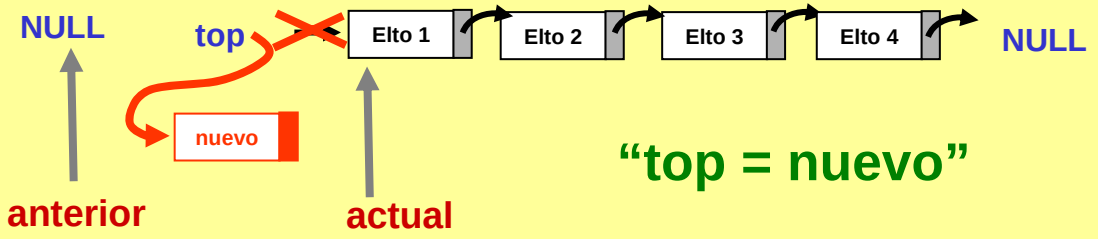
Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

```
struct ciclista *insertar
{
struct ciclista *actual =
struct ciclista *anterior
/* Búsqueda del punto
while ( (actual != NULL)
 {
 anterior = actual;
 actual = actual -> sig;
 }
/* Inserción */
if (anterior != NULL) /*
 anterior -> sig = nu
else /* inserción
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabecera de la lista */
}
```

En cuyo caso “reconectamos” como



“top = nuevo”

Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

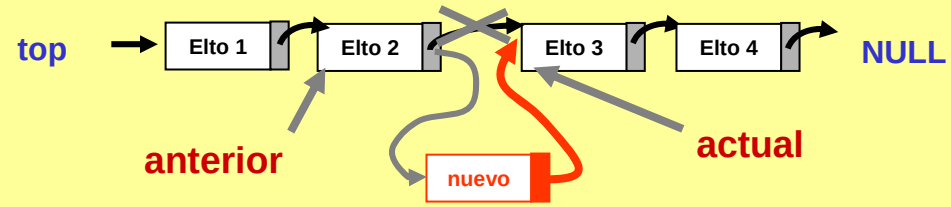
2.- Lista Ordenada

```
struct ciclista *insertar
{
struct ciclista *actual =
struct ciclista *anterior
/* Búsqueda del punto
while ( (actual != NULL)
 {
 anterior = actual;
 actual = actual -> sig
 }
/* Inserción */
if (anterior != NU
 anterior -> sig
else /* inserci al princ
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabecera de la lista */
}
```

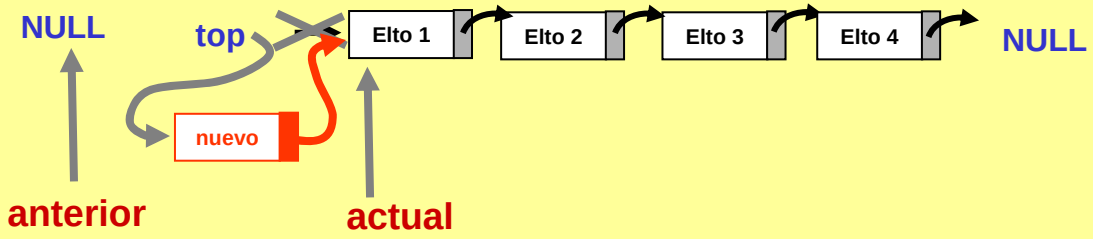
En cualquiera de los dos casos:

"nuevo -> sig = actual"

Caso 1:



Caso 2:



Listas simplemente enlazadas

Funciones de creación de Listas simplemente enlazadas

2.- Lista Ordenada

```
struct ciclista *insertar ( struct ciclista * top, struct ciclista * nuevo)
{
struct ciclista *actual = top; /* puntero para moverse por la lista */
struct ciclista *anterior = NULL; /*puntero al elemento anterior al "actual" */
/* Búsqueda del punto de inserción */
while ( (actual != NULL) && (actual -> tiempo <= nuevo -> tiempo) )
 {
 anterior = actual;
 actual = actual -> sig;
 }
/* Inserción */
if (anterior != NULL) /* inserción en medio
 anterior -> sig = nuevo;
else /* inserción al principio de la li
 top = nuevo;
nuevo -> sig = actual;
return top; /* devuelve la cabecera de la lista */
}
```

Para finalizar devolvemos la cabecera de la lista (Haya ésta cambiado o no)

Listas simplemente enlazadas

Funciones de Recuperación de elementos de Listas simplemente enlazadas

1.- Recorrer la lista completa

```
void listado (struct ciclista * top)
{
struct ciclista *p = top; /* Puntero para recorrer la lista */
while (p) {    /* Bucle que recorre la lista */
            printf("%s %d\n", p -> nombre, p -> tiempo);
            p = p -> s
            }
}
```

Función que recorre la lista al completo desde el primer al último elemento, mostrando los miembros nombre y tiempo.

Listas simplemente enlazadas

Funciones de Recuperación de elementos de Listas simplemente enlazadas

1.- Recorrer la lista completa

```
void listado (struct ciclista * top)
{
struct ciclista *p = top; /* Puntero para recorrer la lista */
while (p) {    /* Bucle que recorre la lista */
            printf("%s %d\n", p -> nombre, p -> tiempo);
            p = p -> sig;
            }
}
```

Esta sentencia pude sustituirse en el caso general por cualquier otra secuencia en función de los se desee hacer con los datos de la lista.

Listas simplemente enlazadas

Funciones de Recuperación de elementos de Listas simplemente enlazadas

2.- Búsqueda de un elemento concreto

```
struct ciclista * busca (struct ciclista *top, char *c)
{
struct ciclista *p = top; /* Puntero para recorrer la lista */
while (p)
        {
        if ( !strcmp(c, p -> nombre) ) return p; /* devuelve el puntero*/
        p = p -> sig;
        }
return NULL; /* no encontrado */
}
```

Función que busca el elemento cuyo miembro nombre coincida con el que se le pasa como argumento (char *c).
Devuelve un puntero al elemento encontrado, o NULL en caso de no encantarlo.

Listas simplemente enlazadas

Funciones Adicionales para Listas simplemente enlazadas

Borrado de elementos

```
struct ciclista * borrar (struct ciclista * top, char * c)
{
struct ciclista * p =  p , * ant = NULL; /* Punteros para recorrer lista */
struct ciclista * aux  servirá para poder liberar memoria */
while(p)  {
          if ( !strcm          > nombre))  /* lo hemos encontrado */

                              NULL)   /* es el primero */
                              ux = p -> sig;
                                                                  al */
          } /* fin del while */
return top; /* no encontrado o lista vacía */
}
```

Función que borra un elemento de una lista enlazada.
Para localizar el elemento, se le pasa el miembro nombre del elemento a eliminar (char *c), y la cabecera de la lista (top)

Devuelve un puntero a la nueva cabecera de la lista.

Listas simplemente enlazadas

Funciones Adicionales para Listas simplemente enlazadas

Borrado de elementos

```
struct ciclista * borrar (struct ciclista * top, char * c)
{
struct ciclista * p = top , * ant = NULL; /* Punteros para recorrer lista */
struct ciclista * aux; /* servirá para poder liberar memoria */
while(p)  {
          if ( !strcmp(c , p -> nombre))  /* lo hemos encontrado */
                    {
                    if (ant == NULL)   /* es el primero */
                              aux = p -> sig;
                              free(p);
                              return aux;}
                    else /* el elemento a eliminar va en medio o al final */
                              {ant -> sig = p -> sig;
                              free(p);
                              return top;}
                    }
          ant = p; /* avance */
          p = p -> sig;
          } /* fin del while */
return top; /* no encontrado o lista vacía */
}
```

Se definen punteros para recorrer la lista (similar a la función insertar)

Listas simplemente enlazadas

Funciones Adicionales para Listas simplemente enlazadas

Borrado de elementos

```
struct ciclista * borrar (struct ciclista * top, char * c)
{
struct ciclista * p = top , * ant = NULL; /* Punteros para recorrer lista */
struct ciclista * aux; /* servirá para poder liberar memoria */
while(p)  {
          if ( !strcmp(c , p -> nombre))  /* lo hemos encontrado */
                    {
                    if (ant = = NULL)   /* es el primero */
                              { aux = p -> sig;
                              free(p);
                              return aux;}
                    else /* el elemento a eliminar va en medio o al final */
                              {ant -> sig = p -> sig;
                              free(p);
                              return top;}
                    }
          ant = p; /* avance */
          p = p -> sig;
          } /* fin del while */
return top; /* no encontrado o lista vacía */
}
```

Recorremos la lista hasta el final

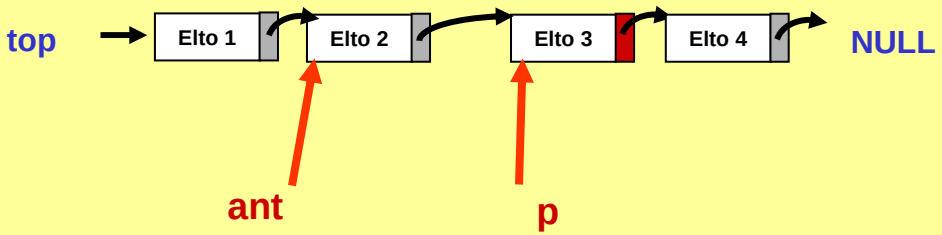
Listas simplemente enlazadas

Funciones Adicionales para Listas simplemente enlazadas

Borrado de elementos

```
struct ciclista * borrar (struct ciclista * top, char * c)
{
struct ciclista * p = top , * ant = NULL; /* Punteros para recorrer lista */
struct ciclista * aux; /* servirá para poder liberar memoria */
while(p)  {
          if ( !strcmp(c , p -> nombre)) /* lo hemos encontrado */
                    {
                    = = NULL)   /* es el primero */
                                  aux = p -> sig;
return top; /* no encontrado o lista vacía */
}
```

Si el miembro nombre coincide con el que se le pasa como argumento, entonces el elemento a borrar está apuntado por **actual**



Listas simplemente enlazadas

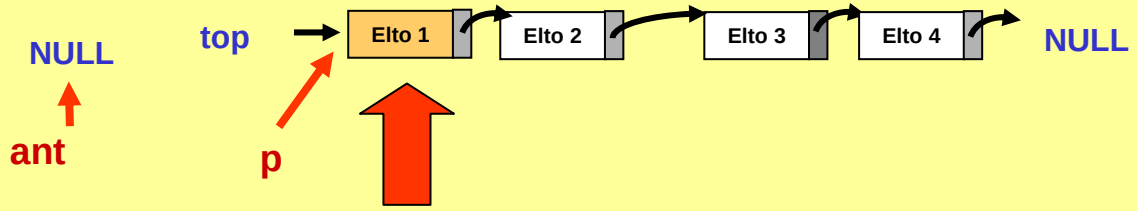
Funciones Adicionales para Listas simplemente enlazadas

Borrado de elementos

```
struct ciclista * borrar (struct ciclista * top, char * c)
{
struct ciclista * p = top , * ant = NULL; /* Punteros para recorrer lista */
struct ciclista * aux; /* servirá para poder liberar memoria */
while(p)  {
          if ( !strcmp(c , p -> nombre))  /* lo hemos encontrado */
                    {
                    if (ant = = NULL)  /* es el primero */
                              { aux = p -> sig;
                              free(p);
                                                                        */
return top; /* no encontrado o lista vacía */
}
```

Dos casos:

1.- El elemento a borrar es el primero



Listas simplemente enlazadas

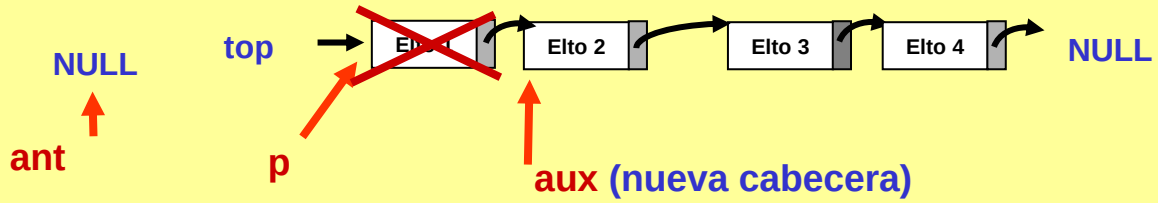
Funciones Adicionales para Listas simplemente enlazadas

Borrado de elementos

```
struct ciclista * borrar (struct ciclista * top, char * c)
{
struct ciclista * p = top , * ant = NULL; /* Punteros para recorrer lista */
struct ciclista * aux; /* servirá para poder liberar memoria */
while(p)  {
          if ( !strcmp(c , p -> nombre))  /* lo hemos encontrado */
                    {
                    if (ant = = NULL)   /* es el primero */
                              { aux = p -> sig;
                              free(p);
                              return aux;}
                              el elemento a eliminar va en medio o al final */
                              {ant -> sig = p -> sig;
```

Dos casos:

Liberamos memoria y devolvemos la nueva cabecera



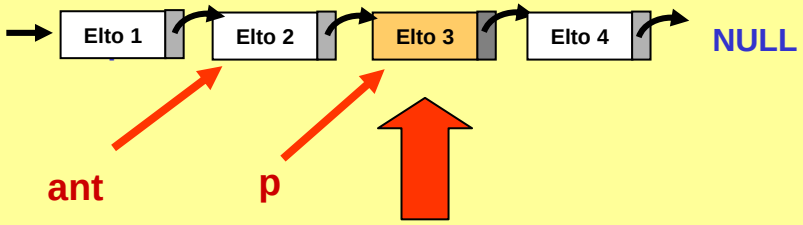
Listas simplemente enlazadas

Funciones Adicionales para Listas simplemente enlazadas

Borrado de elementos

Dos casos:

2.- El elemento a borrar está en medio o al final



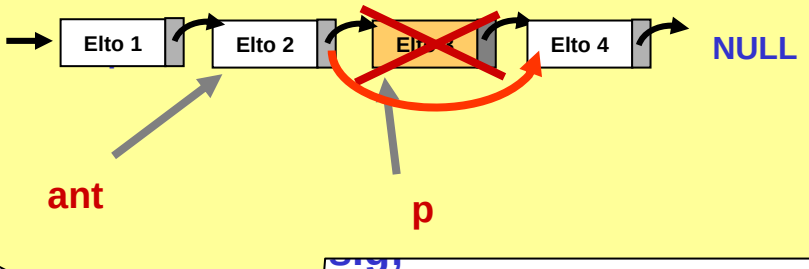

```
struct ciclista
{
struct ciclist
struct ciclist
while(p)  {
          if (
                                    sig;
                    e(p);
                    return aux;}
          else /* el elemento a eliminar va en medio o al final */
                    {ant -> sig = p -> sig;
                    free(p);
                    return top;}
               }
          ant = p; /* avance */
          p = p -> sig;
          } /* fin del while */
return top; /* no encontrado o lista vacía */
}
```

Listas simplemente enlazadas

Funciones Adicionales para Listas simplemente enlazadas

Borrado de elementos

Enlazamos saltándonos el elemento a borrar y liberamos memoria. Devolvemos la cabecera de lista.




```
struct ciclista
{
struct ciclist
struct ciclist
while(p)  {
          if (
                                  sig;
                    retu
          else /* el elemento a eliminar va en medio o al final */
                    {ant -> sig = p -> sig;
                    free(p);
                    return top;}
                    }
          ant = p; /* avance */
          p = p -> sig;
          } /* fin del while */
return top; /* no encontrado o lista vacía */
}
```

Listas simplemente enlazadas

Funciones Adicionales para Listas simplemente enlazadas

Borrado de elementos

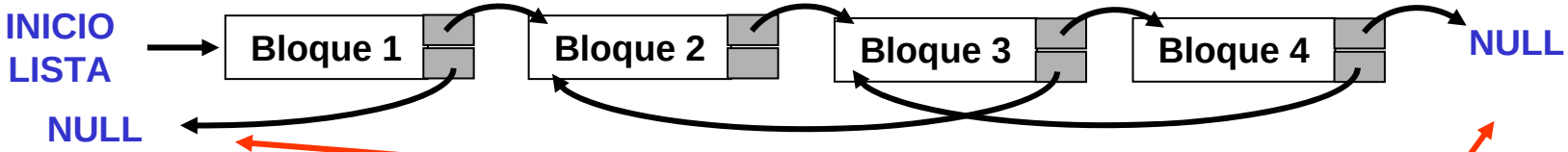
```
struct ciclista * borrar (struct ciclista * top, char * c)
{
struct ciclista * p = top , * ant = NULL; /* Punteros para recorrer lista */
struct ciclista * aux; /* servirá para poder liberar memoria */
while(p)  {
          if ( !strcmp(c , p -> nombre))  /* lo hemos encontrado */
                    {
                    if (ant = = NULL)   /* es el primero */
                              { aux = p -> sig;
                              free(p);
                              return aux;}
                    else /* el elemento a eliminar va en medio o al final */
                              {ant -> sig = p -> sig;
                              free(p);
                              return aux;}
                    }
          ant = p; /* ava
          p = p -> sig;
          } /* fin del whil
return top; /* no encontrado o lista vacía */
}
```

Si llegamos al final de la lista, significa que no se ha encontrado el elemento a eliminar.
Devolvemos la cabecera de lista.

Listas doblemente enlazadas

Cada elemento contiene dos enlaces con el siguiente y anterior elemento de la lista.

- Conocida la posición de un elemento, podemos recorrer la lista en ambas direcciones



Por convenio, el fin de lista, y el elemento precente al primero se indican como NULL

Listas doblemente enlazadas en C

```
struct direc {
        char nombre[40];
        char calle[40];
        char ciudad[20];
        int codigo;
        struct direc  * sig;
        struct direc  * ante;
        } info;
```

Información del Elemento

Enlace con el siguiente elemento y el anterior

Punteros a la propia estructura

DE NUEVO:

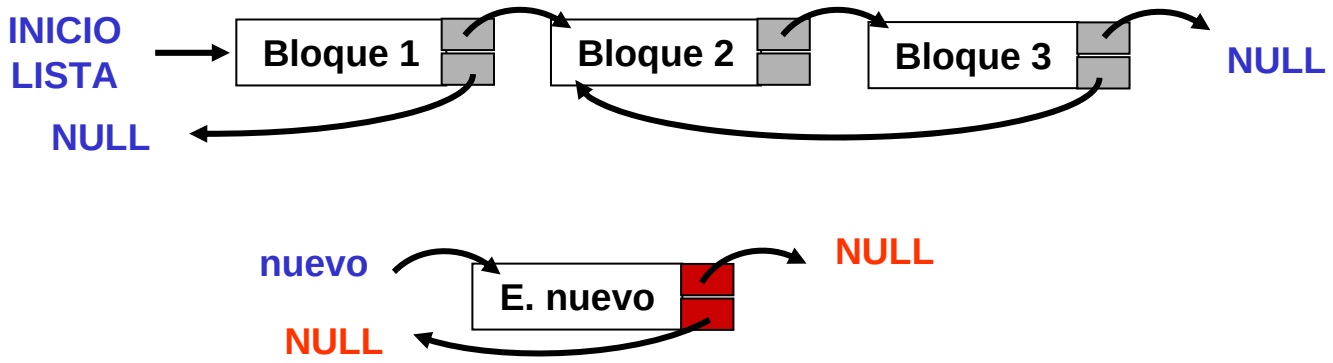
- Cada elemento requiere una reserva dinámica de memoria al ser creado (liberar memoria al ser eliminado).
- La gestión de la lista se hace con funciones apropiadas para añadir, insertar, borrar, buscar elementos, etc.

Listas doblemente enlazadas

Funciones de creación de Listas doblemente enlazadas

1.- Lista Desordenada

```
struct direc * d_almac_fin(struct direc *top, struct direc * nuevo)
{
struct direc * p = top; /* puntero que recorrerá la lista hasta el final */
if (top = = NULL)
                return nuevo ; /* si no existe lista la crea */
while( p -> sig )
               { p = p -> sig ; } /* recorre la lista hasta que p apunte al último */
p -> sig = nuevo; /* enlaza el nuevo elemento al final */
nuevo -> ante = p; /* enlace con el último */
return top; /* devuelve la cabeza */
}
```



Listas doblemente enlazadas

Funciones de creación de Listas doblemente enlazadas

2.- Lista Ordenada

Estructura base para el ejemplo de la función dinsertar

```
struct ciclista{
        int num; /* número del ciclista */
        char nombre[80]; /*nombre del ciclista*/
        int tiempo; /* tiempo en la clasificación */
        struct ciclista * sig; /* puntero a siguiente */
        struct ciclista * ant; /* puntero a anterior */
        };
```

El objetivo es diseñar la función dinsertar para crear listas ordenadas de elementos struct ciclista.
El criterio de ordenación será de menor a mayor tiempo de clasificación.

```
struct ciclista * dinsertar (struct ciclista * ppio, struct ciclista * nuevo)
                                                              lista */
                                                              or al "actual" */
                                                              mpo) )
                                                              nal */
nuevo -> sig = actual;
if ( actual != NULL )
actual -> ant = nuevo;
return ppio; /* devuelve la cabecera de la lista */
}
```

Listas doblemente enlazadas

Funciones de creación de Listas doblemente enlazadas

2.- Lista Ordenada

```
struct ciclista * dinsertar (struct ciclista * ppio, struct ciclista * nuevo)
{
struct ciclista *actual = ppio; /* puntero para moverse por la lista */
struct ciclista *anterior = NULL; /*puntero al elemento anterior al "actual" */
/* Búsqueda del punto de inserción */
while ( (actual != NULL) && ( actual -> tiempo <= nuevo -> tiempo) )
        {anterior = actual;
        actual = actual -> sig;}
/* inserción */
if (anterior != NULL)
        { anterior -> sig = nuevo; /* inserción en medio o final */
        nuevo -> ant = anterior;}
else
        { ppio = nuevo; /* inserción al principio de la lista */
        nuevo -> ant = NULL;}
nuevo -> sig = actual;
if ( actual != NULL )
actual -> ant = nuevo;
return ppio; /* devuelve la cabecera de la lista */
}
```

Listas doblemente enlazadas

Borrado de elementos

```
struct ciclista * dborrar (struct ciclista * top, char * c)
{
struct ciclista * p = top;
while(p) {
        if ( !strcmp(c, p->nombre) ) /* lo hemos encontrado */
                {
                if ( p -> ant = = NULL ) /* es el primero */
                                {top = p -> sig;
                                if (top) top -> ant = NULL; /* si el borrado no era único */
                                free(p);return top;}
                if ( p -> sig = = NULL ) /* si no, es el último */
                                {p -> ant -> sig = NULL;
                                free (p);return top;}
                else
                                {p -> ant -> sig = p -> sig; /* va en medio */
                                p -> sig -> ant = p -> ant;
                                free(p);
                                return top;}
                } /*fin de if */
        p = p -> sig; /* avance */
        } /* fin del while */
return top; /* no encontrado o lista vacía */
}
```

Información tomada de: Fundamentos de Informática, Escuela Superior de Ingenieros, Universidad de Sevilla.
Autores: Ismael Alcalá Torrego, José Ángel Acosta Rodríguez, Fernando Dorado Navas, Fabio Gómez Estern-Aguilar, Manuel López Martínez, Amparo Núñez Reyes y Carlos Vivas Venegas

Compilado por Sullin Santaella, con fines académicos. 2014