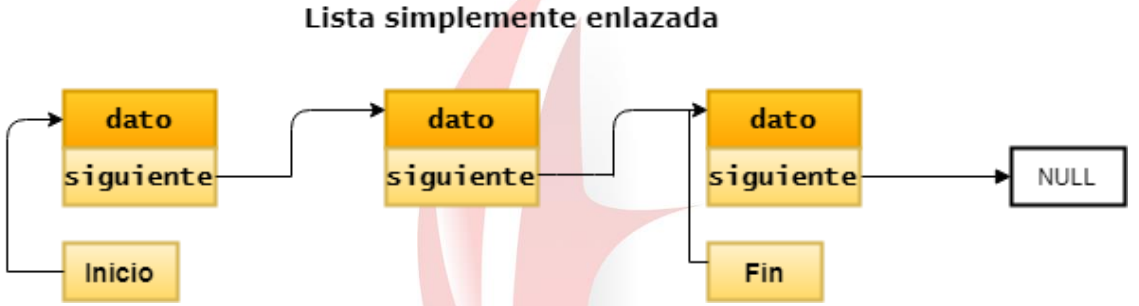
Listas simples

Las listas enlazadas son estructuras de datos semejantes a los array salvo que el acceso a un elemento no se hace mediante un índice sino mediante un puntero.
La asignación de memoria es hecha durante la ejecución.

En una lista los elementos son contiguos en lo que concierne al enlazado.



En cambio, mientras que en un *array* los elementos están contiguos en la memoria, en una lista los elementos están dispersos. El enlace entre los elementos se hace mediante un puntero. En realidad, en la memoria la representación es aleatoria en función del espacio asignado.

El puntero siguiente del último elemento tiene que apuntar hacia NULL (el fin de la lista).

Para acceder a un elemento, la lista es recorrida comenzando por el inicio, el puntero Siguiente permite el cambio hacia el próximo elemento. El desplazamiento se hace en una sola dirección, del primer al último elemento. Si deseas desplazarte en las dos direcciones (hacia delante y hacia atrás) deberás utilizar las listas doblemente enlazadas.

Construcción del modelo de un elemento de la lista

Para establecer un elemento de la lista, será utilizado el tipo struct. El elemento de la lista tendrá un campo dato y un puntero siguiente.
El puntero siguiente tiene que ser del mismo tipo que el elemento, si no, no

podrá apuntar hacia el elemento. El puntero siguiente permitirá el acceso al próximo elemento.

```
typedef struct ElementoLista {
char *dato;
struct ElementoLista *siguiente;
}Elemento;
```

El puntero inicio tendrá la dirección del primer elemento de la lista. El puntero fin albergará la dirección del último elemento de la lista. La variable tamaña contiene el número de elementos.

Cualquiera sea la posición en la lista, los punteros inicio y fin apuntan siempre al primer y último elemento. El campo tamaño contendrá al número de elementos de la lista cualquiera sea la operación efectuada sobre la lista.

Operaciones sobre las listas enlazadas

Para la inserción y la eliminación, una única función bastará si está bien concebida de acuerdo a lo que se necesite. Debo recordar que este artículo es puramente didáctico. Por lo tanto, he escrito una función para cada operación de inserción y eliminación.

Inicialización

Modelo de la función:

```
void inicializacion (Lista *lista);
```

Esta operación debe ser hecha antes de otra operación sobre la lista. Esta comienza el puntero inicio y el puntero fin con el puntero NULL, y el tamaño con el valor 0.

La función

```
void inicializacion (Lista *lista){
lista->inicio = NULL;
lista->fin = NULL;
tamaño = 0;
}
```

Inserción de un elemento en la lista

A continuación, el algoritmo de inserción y el registro de los elementos: declaración del elemento que se va a insertar, asignación de la memoria para el nuevo elemento, llena el contenido del campo de datos, actualización de los punteros hacia el primer y último elemento si es necesario. Caso particular: en una lista con un único elemento, el primero es al mismo tiempo el último. Actualizar el tamaño de la lista
Para añadir un elemento a la lista se presentan varios casos: la inserción en una lista vacía, la inserción al inicio de la lista, la inserción al final de la lista y la inserción en otra parte de la lista.

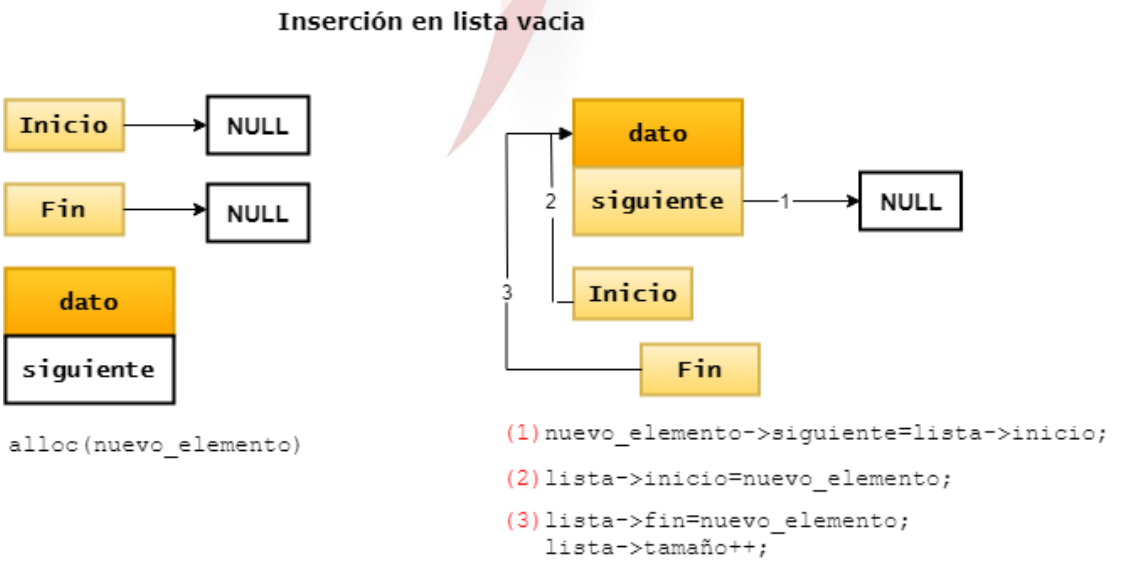
Inserción en una lista vacía

Ejemplo de la función:

```
int ins_en_lista_vacia (Lista *lista, char *dato);
```

La función retorna 1 en caso de error, si no devuelve 0.

Las etapas son asignar memoria para el nuevo elemento, completa el campo de datos de ese nuevo elemento, el puntero siguiente de este nuevo elemento apuntará hacia NULL (ya que la inserción es realizada en una lista vacía, se utiliza la dirección del puntero inicio que vale NULL), los punteros inicio y fin apuntaran hacia el nuevo elemento y el tamaño es actualizado.



La función

```
/* inserción en una lista vacía */
int ins_en_lista_vacia (Lista * lista, char *dato){
Element *nuevo_elemento;
if ((nuevo_elemento = (Element *) malloc (sizeof (Element))) == NULL)
return -1;
if ((nuevo _elemento->dato = (char *) malloc (50 * sizeof (char)))
== NULL)
return -1;
strcpy (nuevo_elemento->dato, dato);

nuevo_elemento->siguiente = NULL;
lista->inicio = nuevo_elemento;
lista->fin = nuevo_elemento;
lista->tamaño++;
return 0;
}
```

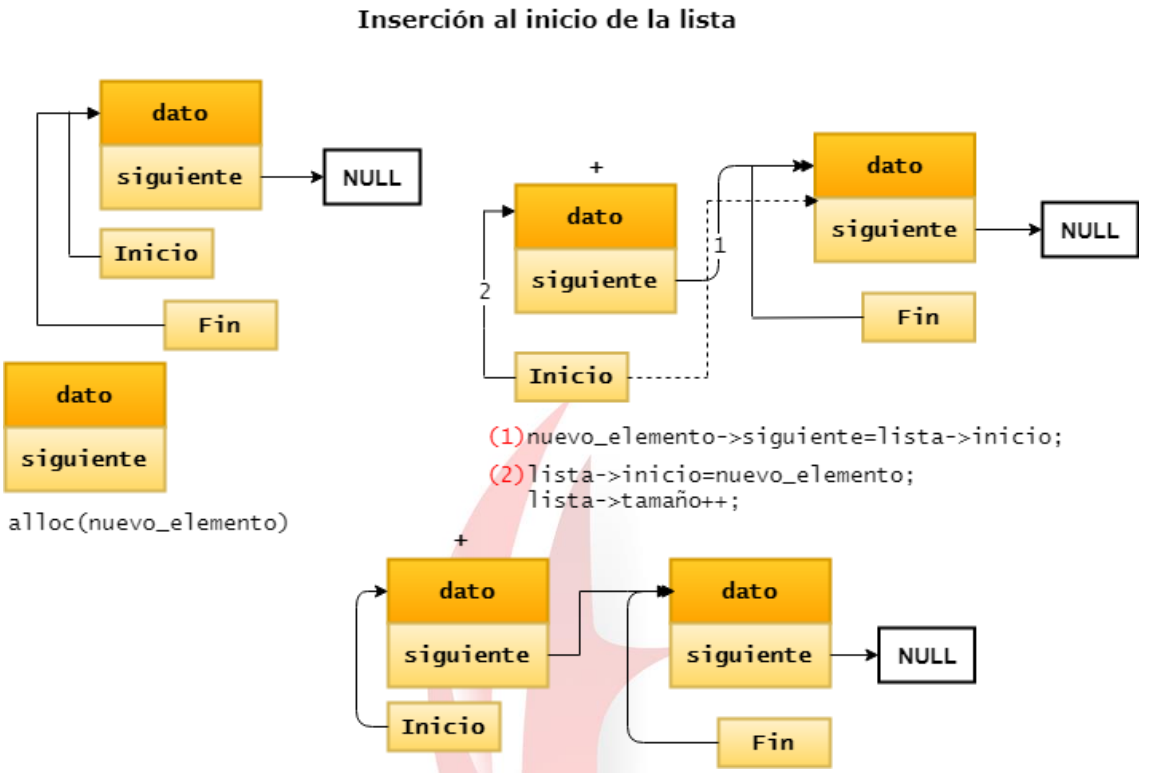
Inserción al inicio de la lista

Ejemplo de la función:

```
int ins_inicio_lista (Lista *lista,char *dato);
```

La función da -1 en caso de error, de lo contrario da 0.

Etapas: asignar memoria al nuevo elemento, rellenar el campo de datos de este nuevo elemento, el puntero siguiente del nuevo elemento apunta hacia el primer elemento, el puntero inicio apunta al nuevo elemento, el puntero fin no cambia, el tamaño es incrementado.



La función

```
/* inserción al inicio de la lista */
int ins_inicio_lista (Lista * lista, char *dato){
Element *nuevo_elemento;
if ((nuevo_elemento = (Element *) malloc (sizeof (Element))) == NULL)
return -1;
if ((nuevo_elemento->dato = (char *) malloc (50 * sizeof (char)))
== NULL)
return -1;
strcpy (nuevo_elemento->dato, dato);

nuevo_elemento->siguiente = lista->inicio
lista->inicio = nuevo_elemento;
lista->tamaño++;
return 0;
}
```

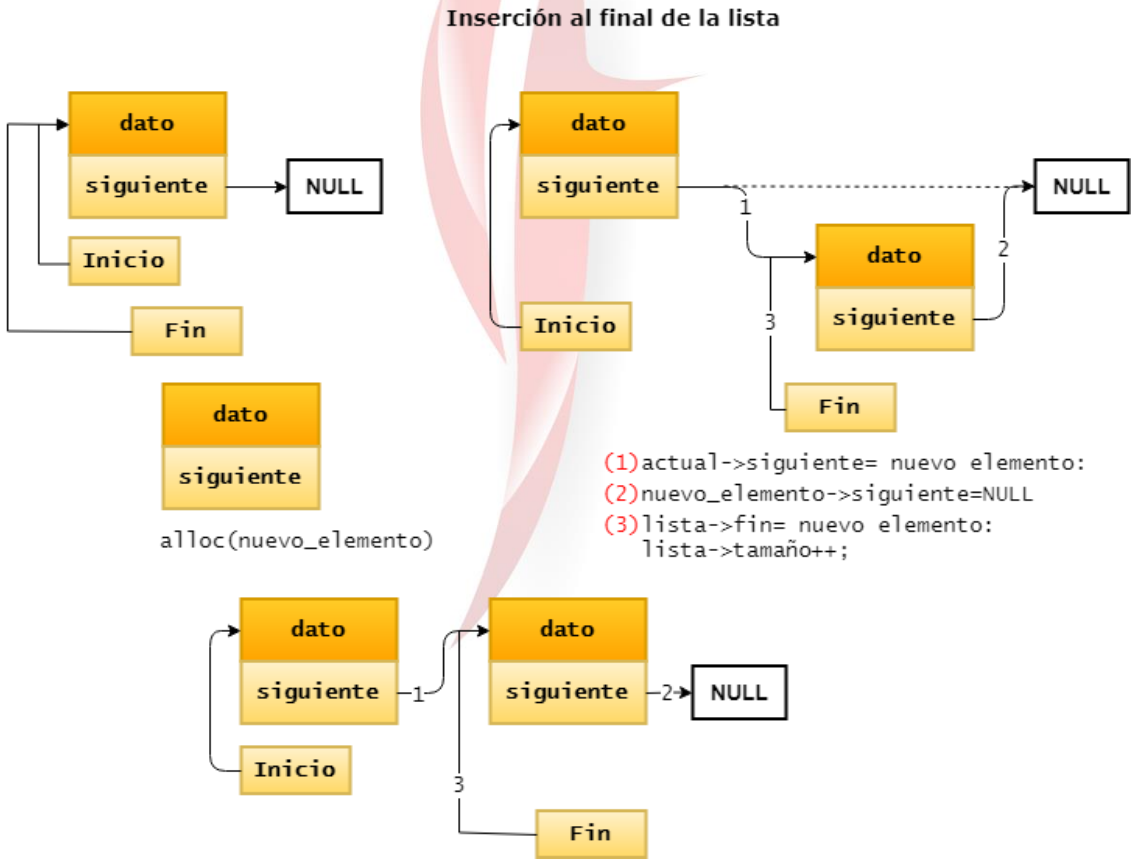
Inserción al final de la lista

Ejemplo de la función:

```
int ins_fin_lista (Lista *lista, Element *actual, char *dato);
```

La función da -1 en caso de error, si no arroja 0.

Etapas: proporcionar memoria al nuevo elemento, rellenar el campo de datos del nuevo elemento, el puntero siguiente del último elemento apunta hacia el nuevo elemento, el puntero fin apunta al nuevo elemento, el puntero inicio no varía, el tamaño es incrementado:



La función

```
/*inserción al final de la lista */
int ins_fin_lista (Lista * lista, Element * actual, char *dato){
Element *nuevo_elemento;
if ((nuevo_elemento = (Element *) malloc (sizeof (Element))) == NULL)
return -1;
if ((nuevo_elemento->dato = (char *) malloc (50 * sizeof (char)))
== NULL)
return -1;
strcpy (nuevo_elemento->dato, dato);

actual->siguiente = nuevo_elemento;
nuevo_elemento->siguiente = NULL;

lista->fin = nuevo_elemento;

lista->tamaño++;
return 0;
}
```

Inserción en la otra parte de la lista

Ejemplo de la función

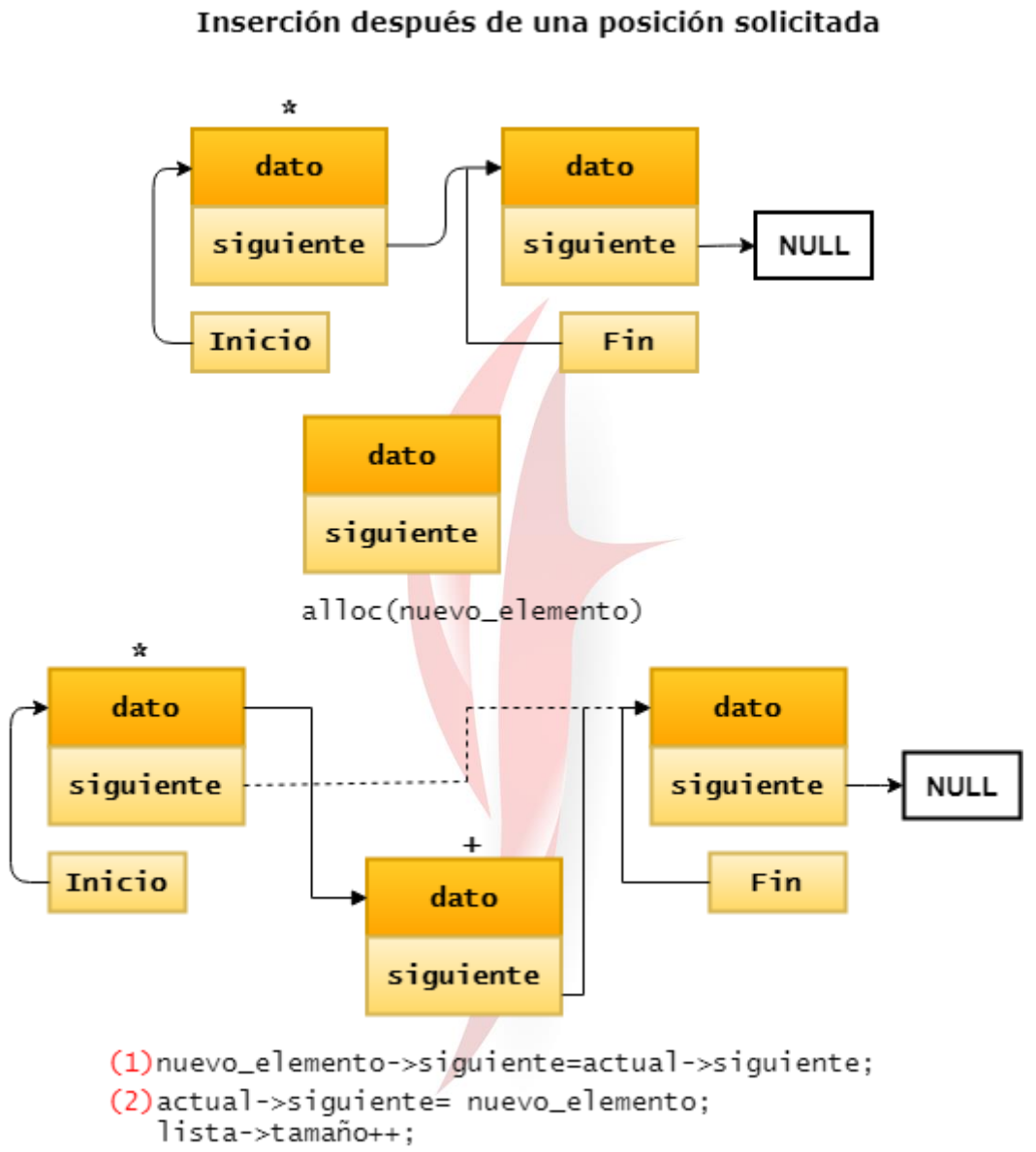
```
int ins_lista (Lista *lista, char *dato,int pos);
```

La función arroja -1 en caso de error, si no da 0.

La inserción se efectuará después de haber pasado a la función una posición como argumento.
Si la posición indicada no tiene que ser el último elemento. En ese caso se debe utilizar la función de inserción al final de la lista.

Etapas: asignación de una cantidad de memoria al nuevo elemento, rellenar el campo de datos del nuevo elemento, escoger una posición en la lista (la inserción se hará luego de haber elegido la posición), el puntero siguiente del nuevo elemento apunta hacia la dirección a la que apunta el puntero siguiente del elemento actual, el puntero siguiente del elemento actual apunta al nuevo

elemento, los punteros inicio y fin no cambian, el tamaño se incrementa en una unidad:



La función

```
/* inserción en la posición solicitada */
int ins_lista (Lista * lista, char *dato, int pos){
if (lista->tamaño < 2)
return -1;
if (pos < 1 || pos >= lista->tamaño)
return -1;
```

```
Element *actual;
Element *nuevo_elemento;
int i;

if ((nuevo_elemento = (Element *) malloc (sizeof (Element))) == NULL)
return -1;
if ((nuevo_elemento->dato = (char *) malloc (50 * sizeof (char)))
== NULL)
return -1;

actual = lista->inicio;
for (i = 1; i < pos; ++i)
actual = actual->siguiente;
if (actual->siguiente == NULL)
return -1;
strcpy (nuevo_elemento->dato, dato);

nuevo_elemento->siguiente = actual->siguiente;
actual->siguiente = nuevo_elemento;
lista->tamaño++;
return 0;
```

Eliminación de un elemento de la lista

A continuación, un algoritmo para eliminar un elemento de la lista: uso de un puntero temporal para almacenar la dirección de los elementos a borrar, el elemento a eliminar se encuentra después del elemento actual, apuntar el puntero siguiente del elemento actual en dirección del puntero siguiente del elemento a eliminar, liberar la memoria ocupada por el elemento borrado, actualizar el tamaño de la lista. Para eliminar un elemento de la lista hay varios casos: eliminación al inicio de la lista y eliminación en otra parte de la lista.

Eliminación al inicio de la lista

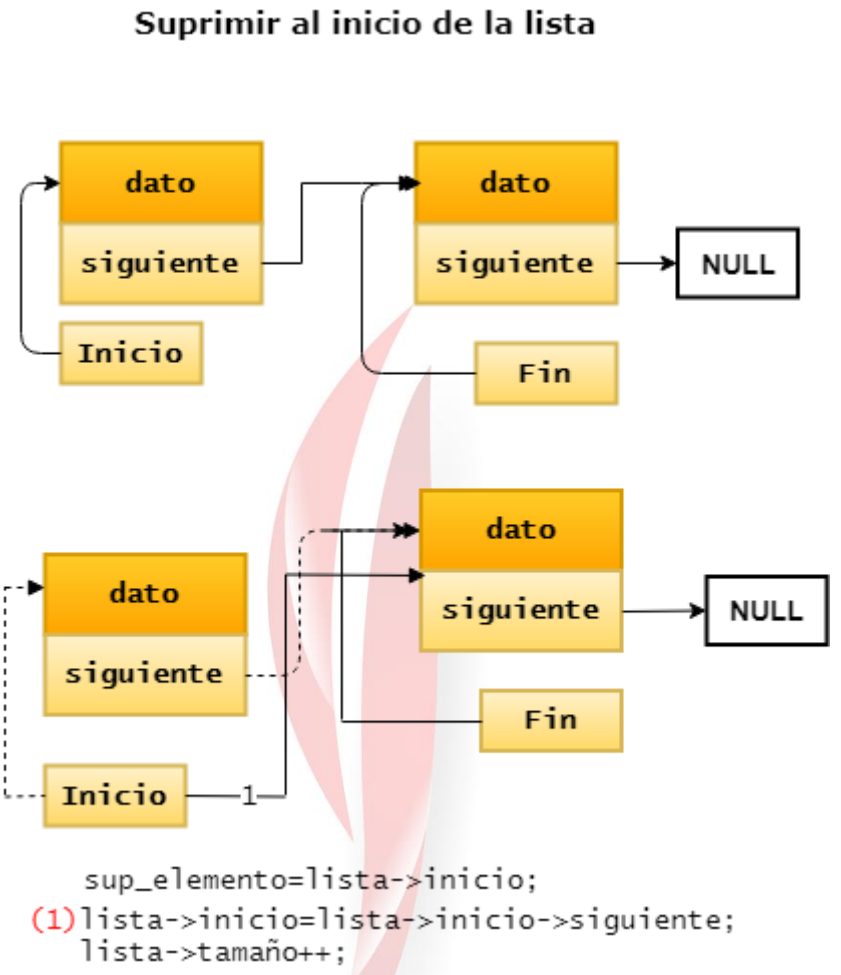
Ejemplo de la función:

```
int sup_inicio (Lista *lista);
```

La función devolverá -1 en caso de equivocación, de lo contrario da 0.

Etapas: el puntero sup_elem contendrá la dirección del 1er elemento, el

puntero inicio apuntará hacia el segundo elemento, el tamaño de la lista disminuirá un elemento:



La función

```
/* eliminación al inicio de la lista */
int sup_inicio (Lista * lista){
if (lista->tamaño == 0)
return -1;
Element *sup_elemento;
sup_element = lista->inicio;
lista->inicio = lista->inicio->siguiente;
if (lista->tamaño == 1)
lista->fin = NULL;
free (sup_elemento->dato);
free (sup_elemento);
lista->tamaño--;
return 0;
}
```

Eliminación en otra parte de la lista

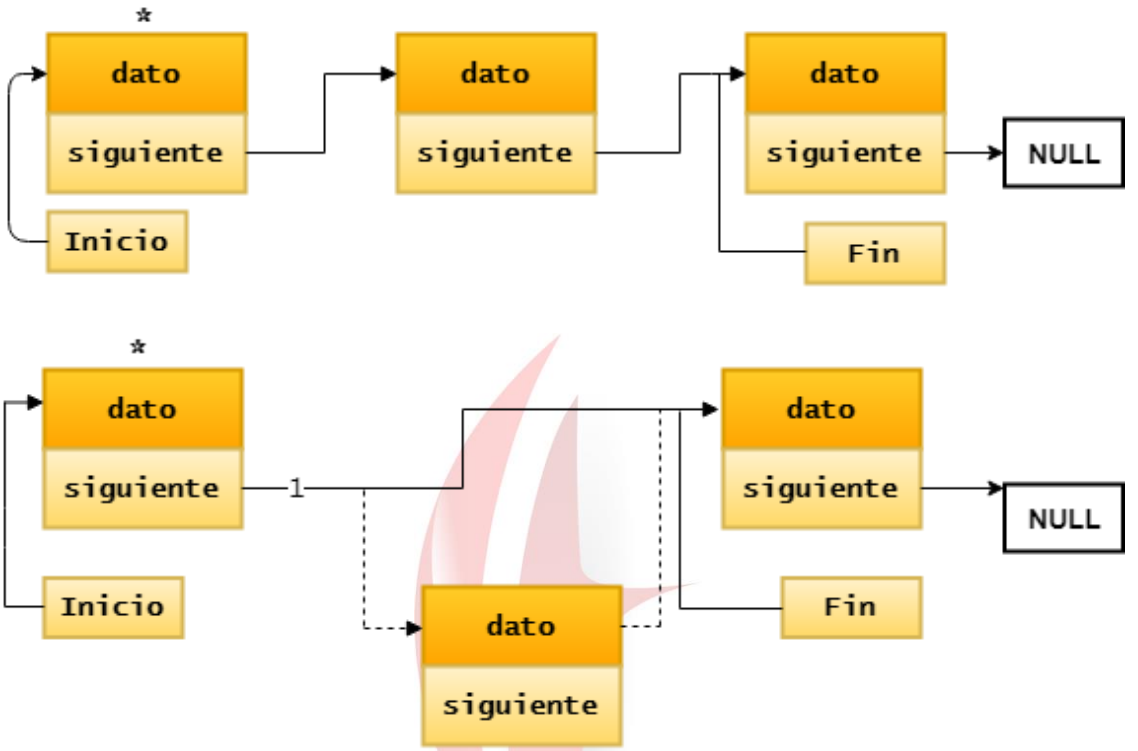
Ejemplo de la función:

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

La función da -1 en caso de error, si no devuelve 0.

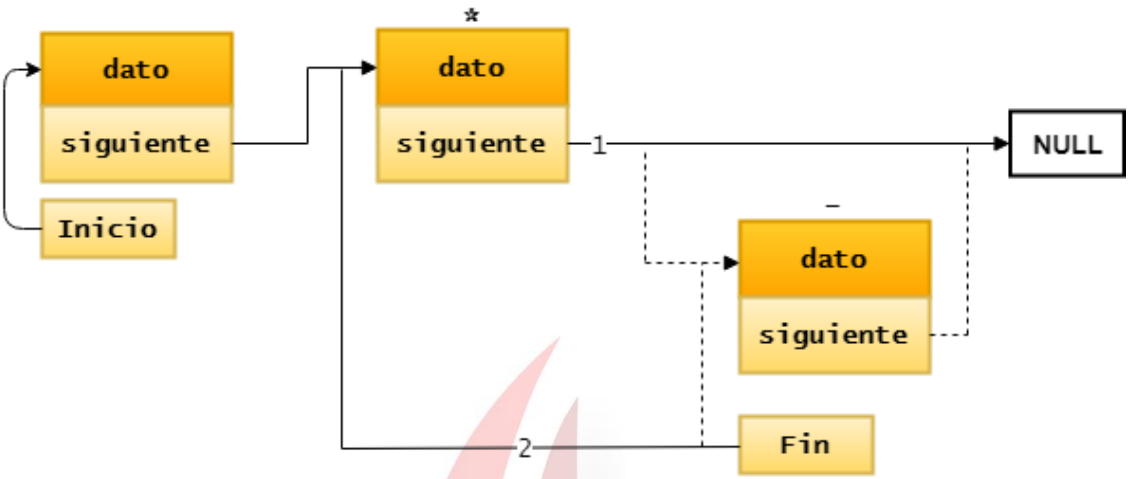
Etapas: el puntero **sup_elem** contendrá la dirección hacia la que apunta el puntero siguiente del elemento actual, el puntero siguiente del elemento actual apuntará hacia el elemento al que apunta el puntero siguiente del elemento que sigue al elemento actual en la lista. Si el elemento actual es el penúltimo elemento, el puntero fin debe ser actualizado. El tamaño de la lista será disminuido en un elemento:

Eliminación después de una posición solicitada




```
sup_elemento=actual->siguiente;
(1)actual->siguiente=actual->siguiente->siguiente;
   lista->tamaño++;
```

Eliminación después del penúltimo elemento




```
   sup_elemento=actual->siguiente;
(1)actual->siguiente=actual->siguiente->siguiente;
(2)lista->fin=actual;
   lista->tamaño++;
```

La función

```
/* eliminar un elemento después de la posición solicitada */

int sup_en_lista (Lista * lista, int pos){
if (lista->tamaño <= 1 || pos < 1 || pos >= lista->tamaño)
        return -1;
        int i;
        Element *actual;
        Element *sup_elemento;
        actual = lista->inicio;

        for (i = 1; i < pos; ++i)
        actual = actual->siguiente;
        sup_elemento = actual->siguiente;
        actual->siguiente = actual->siguiente->siguiente;
        if(actual->siguiente == NULL)
        lista->fin = actual;
        free (sup_elemento->dato);
        free (sup_elemento);
        lista->tamaño--;
        return 0;
}
```

Visualización de la lista

Para mostrar la lista entera hay que posicionarse al inicio de la lista (el puntero inicio lo permitirá). Luego usando el puntero siguiente de cada elemento la lista es recorrida del primero al último elemento.
La condición para detener es proporcionada por el puntero siguiente del último elemento que vale NULL.

La función

```
/* visualización de la lista */
void visualización (Lista * lista){
Element *actual;
actual = lista->inicio;
while (actual != NULL){
printf ("%p - %s\n", actual, actual->dato);
actual = actual->siguiente;
}
}
```

Destrucción de la lista

Para destruir la lista entera, he utilizado la eliminación al inicio de la lista porque el tamaño es mayor a cero.

La función

```
/* destruir la lista */
void destruir (Lista *lista){
while (lista->tamaño > 0)
sup_inicio (lista);
}
```

Ejemplo completo

lista.h

```
/* ---------- 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===

/**************************\
lista_function.h *\************************/voidinicialisacion (Lista *
lista){ lista ->inicio = NULL; lista ->fin = NULL; lista ->tamaño = 0;}/*
insercion en une lista vacia */int ins_en_lista _vacia (Lista * lista, char
*dato){ Elemento *nuevo_elemento; if ((nuevo_elemento = (Elemento *) malloc
(sizeof (Elemento))) == NULL) return -1; if ((nuevo_elemento->dato = (char
*) malloc (50 * sizeof (char))) == NULL) return -1; strcpy (nuevo_elemento-
>dato, dato); nuevo_elemento->siguiente = NULL; lista ->inicio =
nuevo_elemento; lista ->fin = nuevo_elemento; lista ->tamaño++; return
0;}/* inserción al inicio de la lista */int ins_inicio_lista (Lista *
```

```
lista, char *dato){ Elemento *nuevo_elemento; if ((nuevo_elemento =
(Elemento *) malloc (sizeof (Elemento))) == NULL) return -1; if
((nuevo_elemento->dato = (char *) malloc (50 * sizeof (char))) == NULL)
return -1; strcpy (nuevo_elemento->dato, dato); nuevo_elemento->siguiente =
lista->inicio; lista ->inicio = nuevo_elemento; lista ->tamaño++; return
0;}/*insercion al final de la lista */int ins_fin_lista (Lista * lista,
elemento * actual, char *dato){ Elemento *nuevo_elemento; if
((nuevo_elemento = (Elemento *) malloc (sizeof (Elemento))) == NULL) return
-1; if ((nuevo_elemento->dato = (char *) malloc (50 * sizeof (char))) ==
NULL) return -1; strcpy (nuevo_elemento->dato, dato); actual->siguiente =
nuevo_elemento; nuevo_elemento->siguiente = NULL; lista ->fin =
nuevo_elemento; lista ->tamaño++; return 0;}/* insercion en la posicion
solicitada */int ins_ lista (Lista * lista, char *dato, int pos){ if (lista
->tamaño < 2) return -1; if (pos < 1 || pos >= lista ->tamaño) return -1;
Elemento *actual; Elemento *nuevo_elemento; int i; if ((nuevo_elemento =
(Elemento *) malloc (sizeof (Elemento))) == NULL) return -1; if
((nuevo_elemento->dato = (char *) malloc (50 * sizeof (char))) == NULL)
return -1; actual = lista ->inicio; for (i = 1; i < pos; ++i) actual =
actual->siguiente; if (actual->siguiente == NULL) return -1; strcpy
(nuevo_elemento->dato, dato); nuevo_elemento->siguiente = actual-
>siguiente; actual->siguiente = nuevo_elemento; lista ->tamaño++; return
0;}/* supresión al inicio de la lista */int sup_inicio (Lista * lista){ if
(lista ->tamaño == 0) return -1; Elemento *sup_elemento; sup_elemento =
lista ->inicio; lista ->inicio = lista ->inicio->siguiente; if (lista -
>tamaño == 1) lista ->fin = NULL; free (sup_elemento->dato); free
(sup_elemento); lista ->tamaño--; return 0;}/* suprimir un elemento después
de la posición solicitada */int sup_en_lista (Lista * lista, int pos){ if
(lista ->tamaño <= 1 || pos < 1 || pos >= lista ->tamaño) return -1; int i;
Elemento *actual; Elemento *sup_elemento; actual = lista ->inicio; for (i =
1; i < pos; ++i) actual = actual->siguiente; sup_elemento = actual-
>siguiente; actual->siguiente = actual->siguiente->siguiente; if(actual-
>siguiente == NULL) lista ->fin = actual; free (sup_elemento->dato); free
(sup_elemento); lista ->tamaño--; return 0;}/* visualización de la Lista
*/void muestra (Lista * lista){ Elemento *actual; actual = lista ->inicio;
while (actual != NULL){ printf ("%p - %s\n", actual, actual->dato); actual
= actual->siguiente; }}/* destruir la Lista */void destruir (La lista*La
lista){ while (lista ->tamaño > 0) sup_inicio (lista);}int menu (Lista
*lista,int *k){ int elección; printf("********** MENU **********\n"); if
(lista ->tamaño == 0){ printf ("1. Adición del1er elemento\n"); printf ("2.
Quitar\n"); }else if(lista ->tamaño == 1 || *k == 1){ printf ("1. Adición
al inicio de la lista\n"); printf ("2. Adición al final de la lista\n");
printf ("4. Supresión al inicio de la lista\n"); printf ("6. Destruir la
lista\n"); printf ("7. Quitar\n"); }else { printf ("1. Adición al inicio de
la lista\n"); printf ("2. Adición al final de la lista\n"); printf ("3.
Adición después de la posición indicada\n"); printf ("4. Supresión al
inicio de la lista\n"); printf ("5. Supresión después de la posición
indicada\n"); printf ("6. Destruir la lista\n"); printf ("7. Quitar\n"); }
printf ("\n\nElegir: "); scanf ("%d", &elección); getchar(); if(lista-
>tamaño == 0 && elección == 2) elección = 7; return elección;}/* --------
FIN lista_function.h --------- */===lista.c===/*********************\ *
lista.c *\*********************/#include <stdio.h>#include
<stdlib.h>#include <string.h>#include "lista.h"#include "lista
```

```
_function.h"int main (void){ char elección; char *nom; Lista *lista;
Elemento *actual; if ((lista = (Lista *) malloc (sizeof (Lista))) == NULL)
return -1; if ((nom = (char *) malloc (50)) == NULL) return -1; actual =
NULL; elección = 'o'; inicialisacion (lista); int pos, k; while (elección!=
7){ elección = menu (lista, &k); switch (elección){ case 1: printf
("Ingresa un elemento: "); scanf ("%s", nom); getchar (); if (lista->tamaño
== 0) ins_en_lista_vacia (lista, nom); else ins_inicio_lista (lista, nom);
printf ("%d elementos:ini=%s,fin=%s\n", lista->tamaño, lista->inicio->dato,
lista->fin->dato); muestra (lista); break; case 2: printf ("Ingrese un
elemento: "); scanf ("%s", nom); getchar (); ins_fin_lista (lista, lista-
>fin, nom); printf ("%d elementos:ini=%s,fin=%s\n", lista->tamaño, lista-
>inicio->dato, lista->fin->dato); muestra (lista); break; case 3: printf
("Ingrese un elemento: "); scanf ("%s", nom); getchar (); do{ printf
("Ingrese la posicion: "); scanf ("%d", &pos); } while (pos < 1 || pos >
lista->tamaño); getchar (); if (lista->tamaño == 1 || pos == lista-
>tamaño){ k = 1; printf("-----------------------------------------------
\n"); printf("/!\\Fracaso la insercion.Utilice el menu {1|2} /!\\\n");
printf("-----------------------------------------------\n"); break; }
ins_lista (lista, nom, pos); printf ("%d elementos:ini=%s,fin=%s\n", lista-
>tamaño, lista->inicio->dato, lista->fin->dato); muestra (lista); break;
case 4: sup_inicio (lista); if (lista->tamaño != 0) printf ("%d
elementos:ini=%s,fin=%s\n", lista->tamaño, lista->inicio->dato, lista->fin-
>dato); else printf ("lista vacia\n"); muestra (lista); break; case 5: do{
printf ("Ingrese la posicion : "); scanf ("%d", &pos); } while (pos < 1 ||
pos > lista->tamaño); getchar (); sup_en_lista (lista, pos); if (lista-
>tamaño != 0) printf ("%d elementos:ini=%s,fin=%s\n", lista->tamaño, lista-
>inicio->dato, lista->fin->dato); else printf ("lista vacia\n"); muestra
(lista); break; case 6: destruir (lista); printf ("la lista ha sido
destruida: %d elementos\n", lista->tamaño); break; } } return 0;}
```