Felipe Reyes González

Estructura de Datos

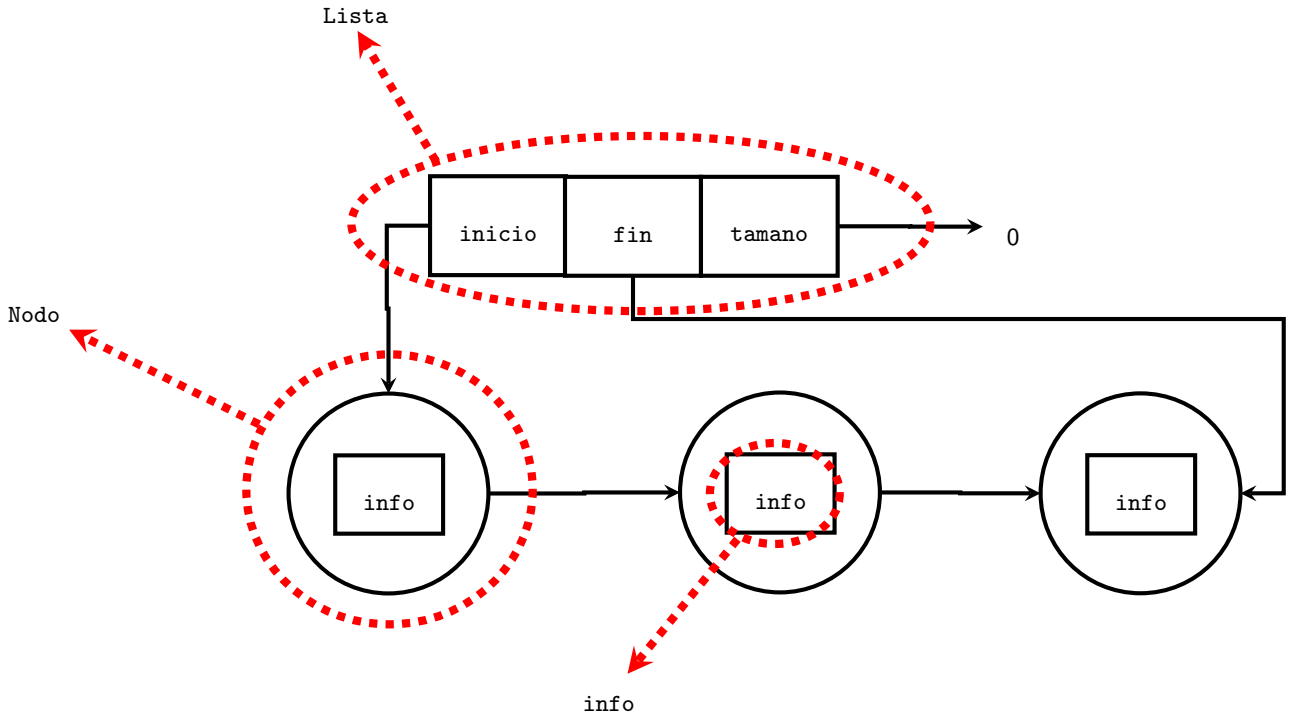
Listas Enlazadas
Ejercicios resueltos

Ingeniería en Computación e Informática

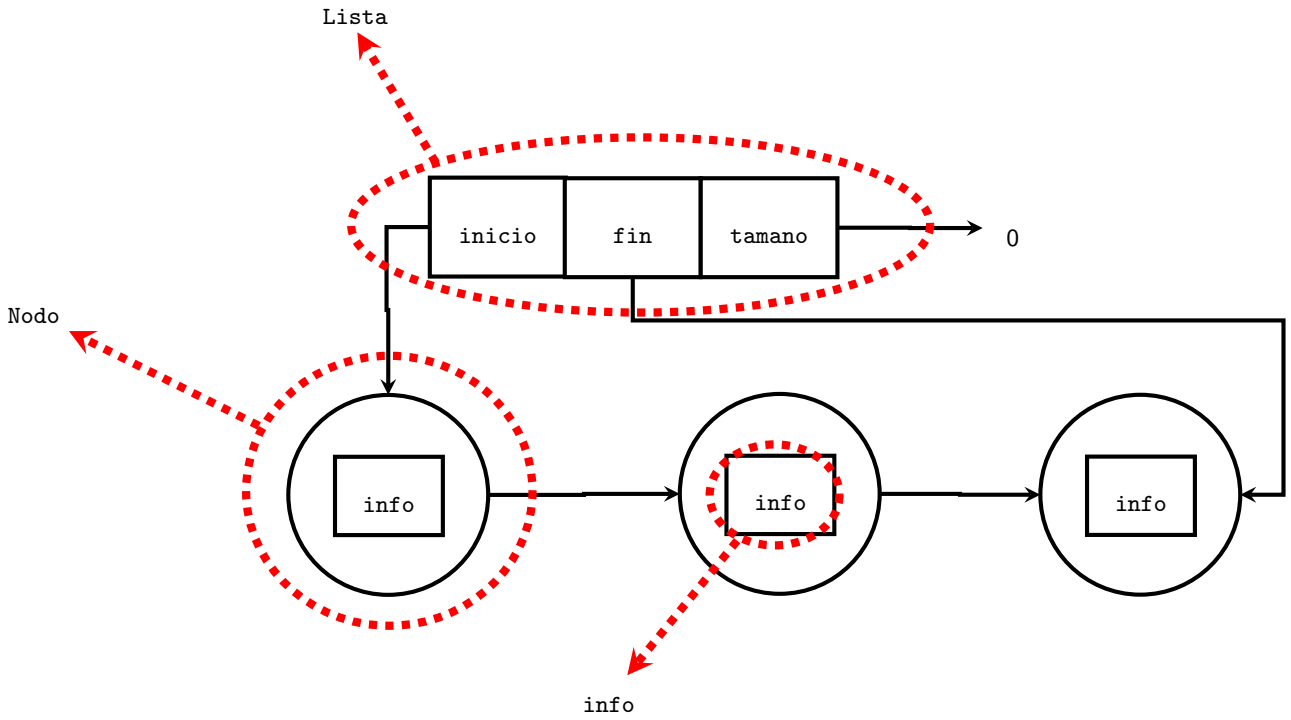
Universidad
Andrés Bello

Definición de la estructura

Las listas enlazadas son estructuras de datos dinámicas. Son una colección de nodos dispuestos uno a continuación de otro, conectados al siguiente por un **enlace**. Al ser una estructura flexible, nos permite modificar su tamaño sin mayores problemas



Definición de la estructura



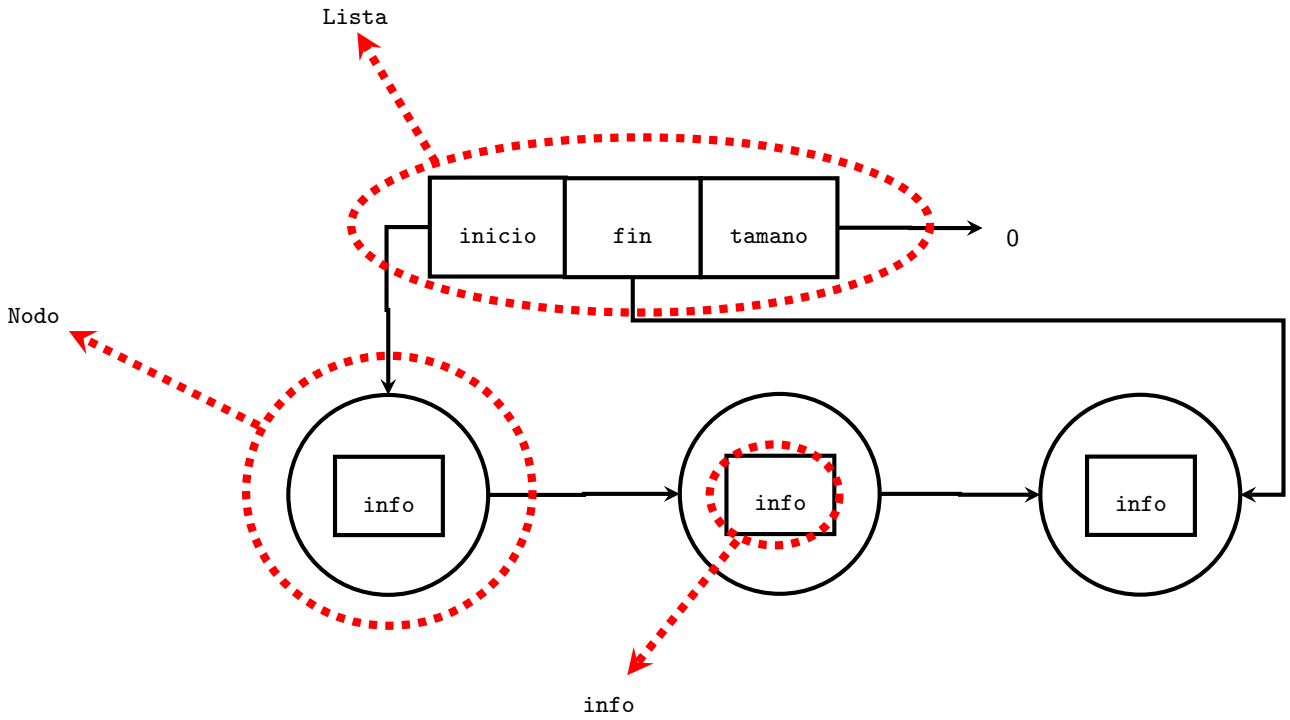
Estructura Info

```
typedef struct info {
  int dato1;
  /* int dato2;*/
  /*...*/
} Info;
```

Uso de la estructura Info

```
Info a, *b;

a.dato1 = 1;

b = malloc(sizeof(Info));
b->dato1 = 2;

printf("%d %d\n", a.dato1, b->dato1);
```

Definición de la estructura



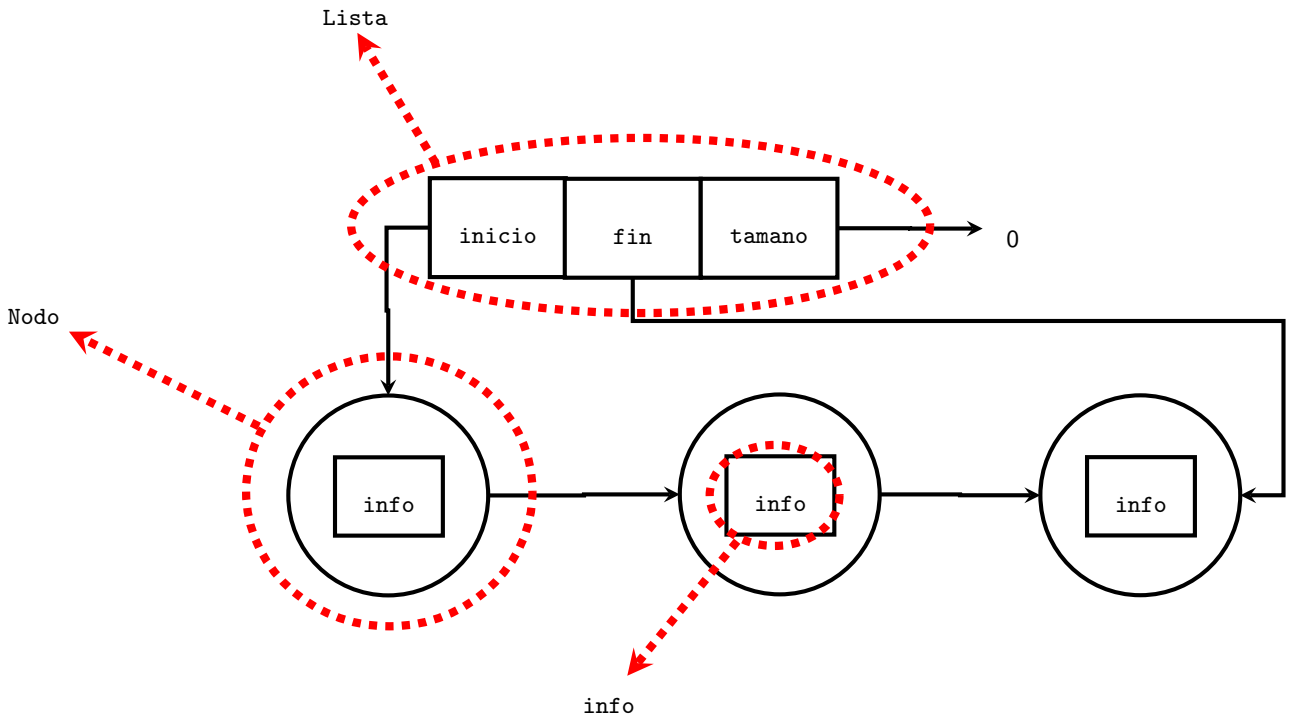
Estructura Nodo

```
typedef struct nodo {
  Info *datos;
  struct nodo *siguiente;
} Nodo;
```

Uso de la estructura Info

```
a.datos = (Info *) malloc(sizeof(Info));
a.datos->dato1 = 1;

b = malloc(sizeof(Nodo));
b->datos = (Info *) malloc(sizeof(Info));
b->datos->dato1 = 2;

printf("%d %d\n", a.datos->dato1, b->datos->dato1);
```

Definición de la estructura

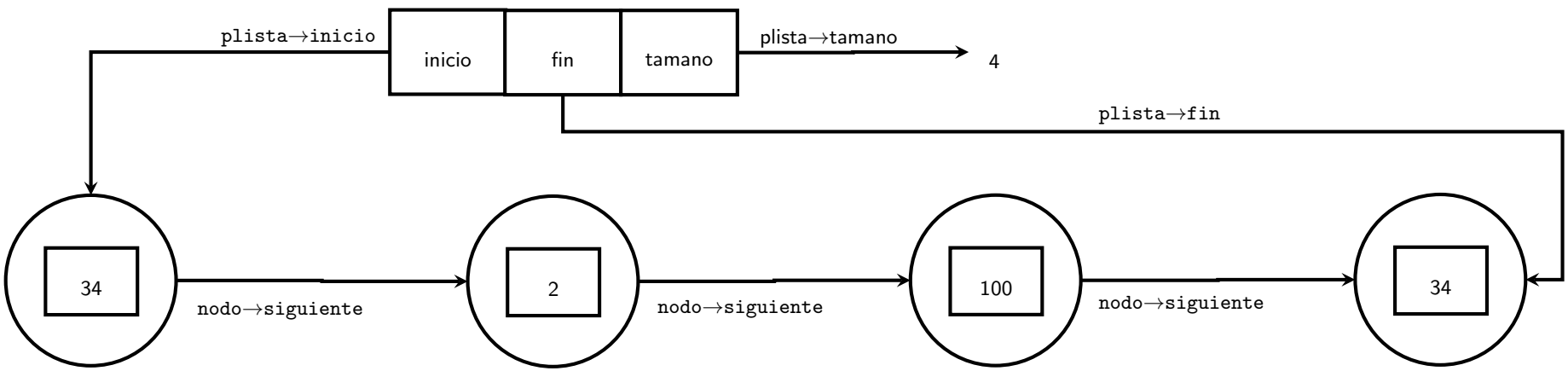


Lista

```
typedef struct lista {
  Nodo *inicio;
  Nodo *fin;
  int tamano;
} Lista;
```

Recorriendo una lista

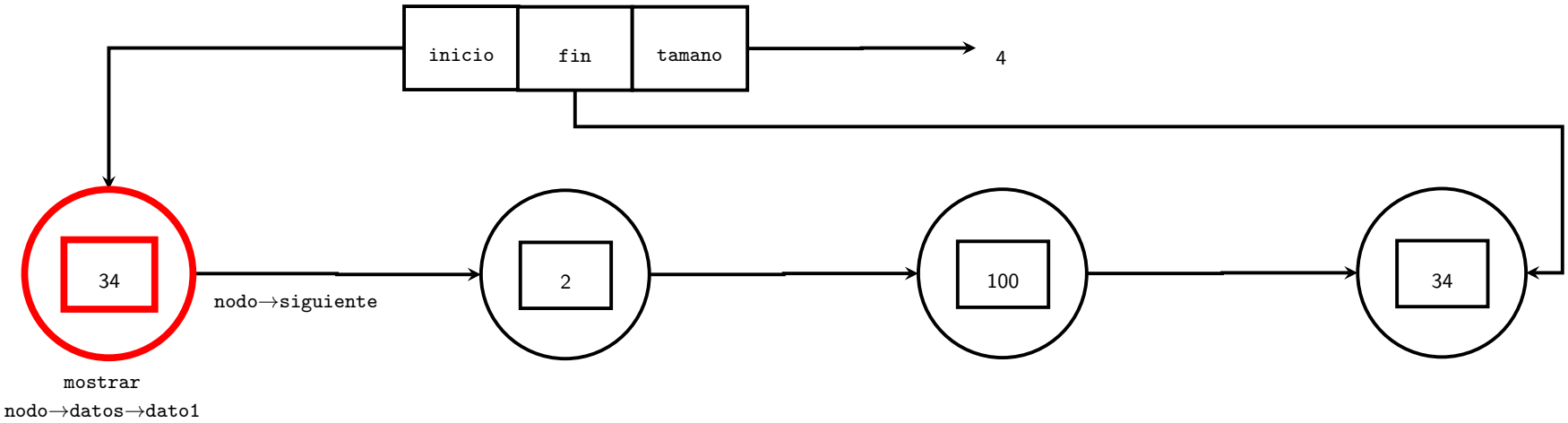
Diseñe una función que muestre la lista desde el primer elemento hasta el último



Recorriendo una lista

Acciones repetitivas

El proceso es sencillo, consiste en acceder a cada uno de los nodos y mostrar la información

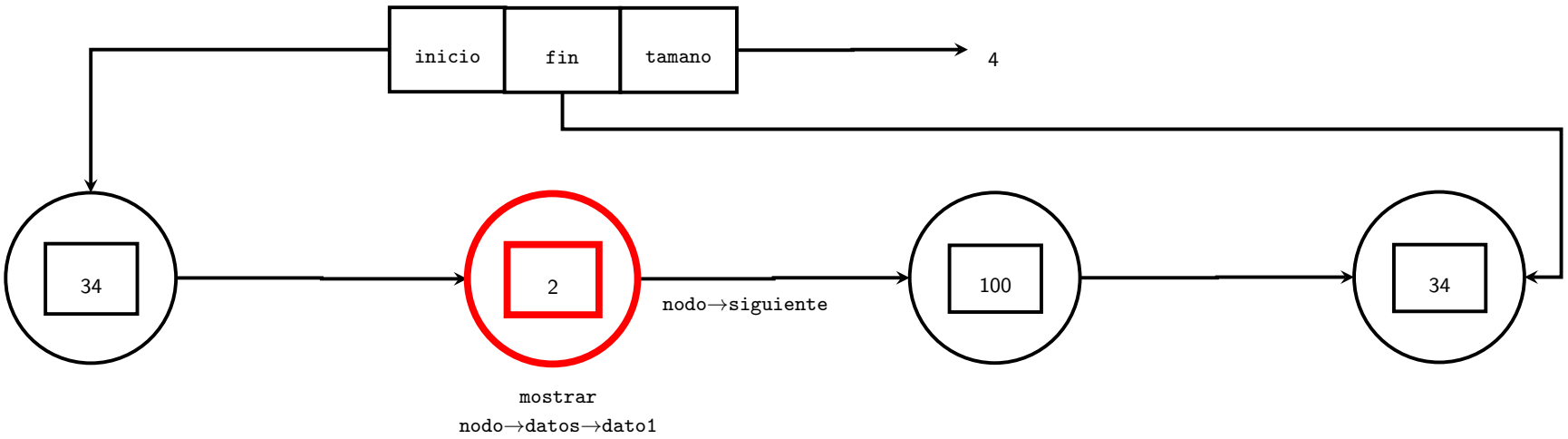



```
printf("%d ", nodo->datos->dato1);
nodo = nodo->siguiente;
```

Recorriendo una lista

Acciones repetitivas

El proceso es sencillo, consiste en acceder a cada uno de los nodos y mostrar la información

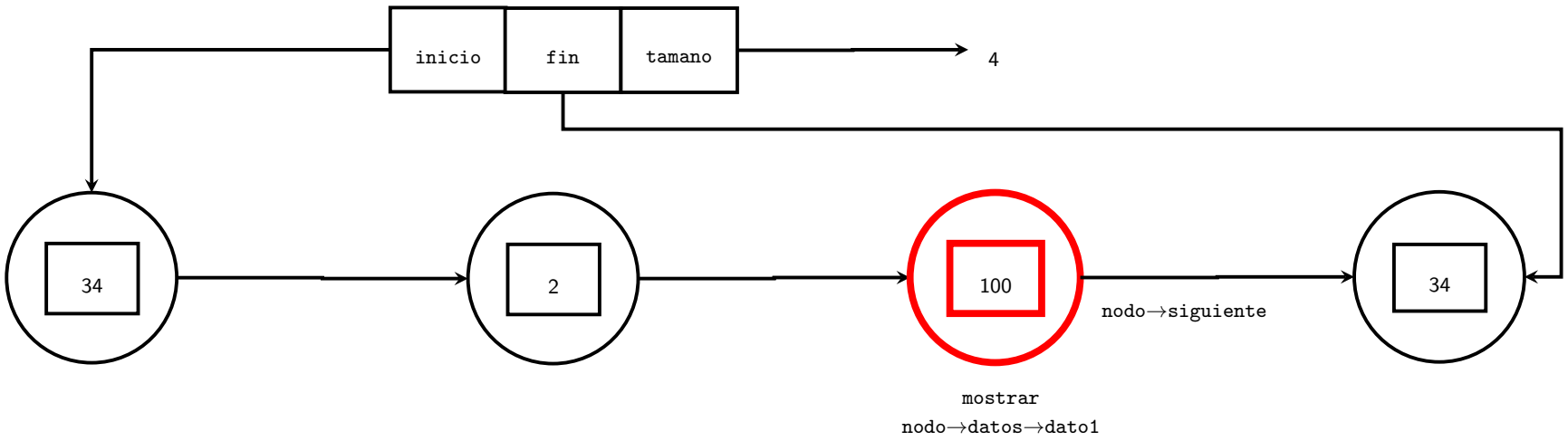



```
printf("%d ", nodo->datos->dato1);
nodo = nodo->siguiente;
```

Recorriendo una lista

Acciones repetitivas

El proceso es sencillo, consiste en acceder a cada uno de los nodos y mostrar la información

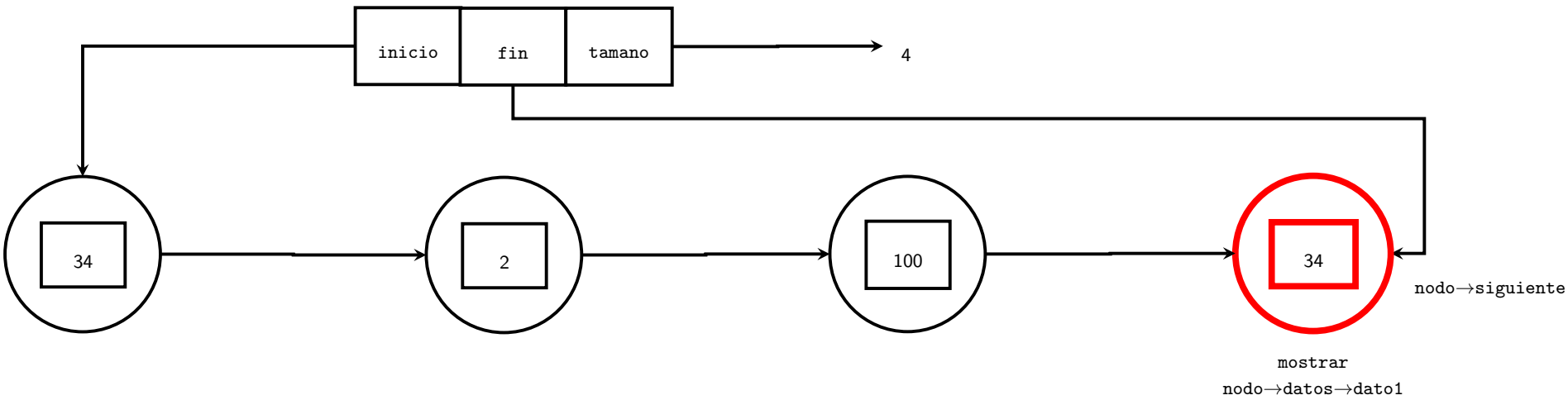



```
printf("%d ", nodo->datos->dato1);
nodo = nodo->siguiente;
```

Recorriendo una lista

Acciones repetitivas

El proceso es sencillo, consiste en acceder a cada uno de los nodos y mostrar la información

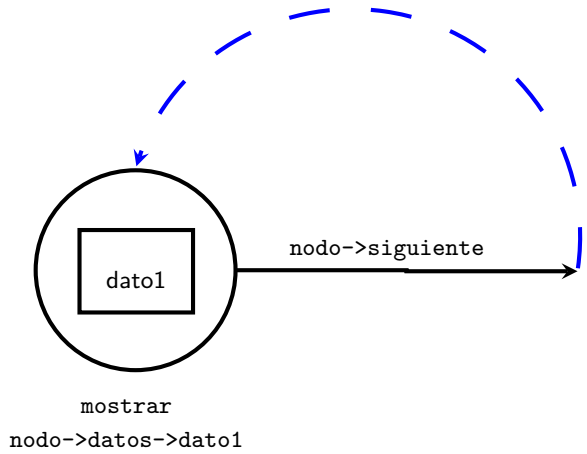



```
printf("%d ", nodo->datos->dato1);
nodo = nodo->siguiente;
```

Recorriendo una lista

Acciones repetitivas

Del proceso anterior pudimos percatarnos que se repite siempre la misma acción. De ahí deriva la idea de utilizar ciclos, donde la **condición de término** es cuando el nodo sea igual a `NULL`, además, sabemos que debemos iterar utilizando una variable auxiliar para no perder la dirección del primer nodo.

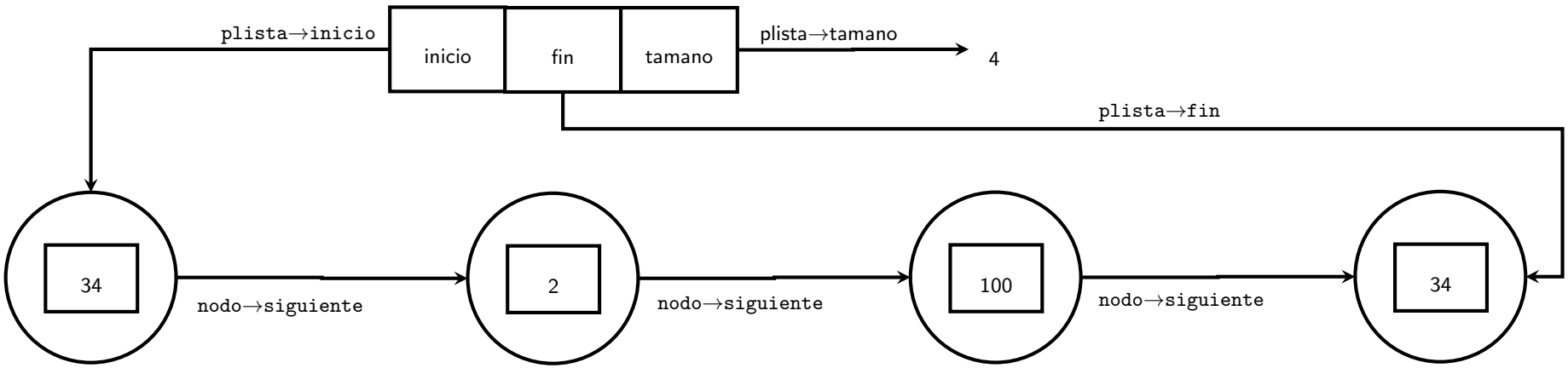



```
while(nodo != NULL) {
  printf("%d ", nodo->datos->dato1);
  nodo = nodo->siguiente;
}
```

Recorriendo una lista

Programando las acciones

Como no podemos perder el puntero al inicio de la lista, lo que hacemos es copiar `plista->inicio` en la variable auxiliar `nodo`




```
Nodo *nodo;
nodo = plista->inicio;

while(nodo != NULL) {
  printf("%d ", nodo->datos->dato1);
  nodo = nodo->siguiente;
}
```

Recorriendo una lista

La función mostrar

Por último, como sabemos que la función necesita de la información de la Lista, enviaremos por parámetro un puntero a una lista.

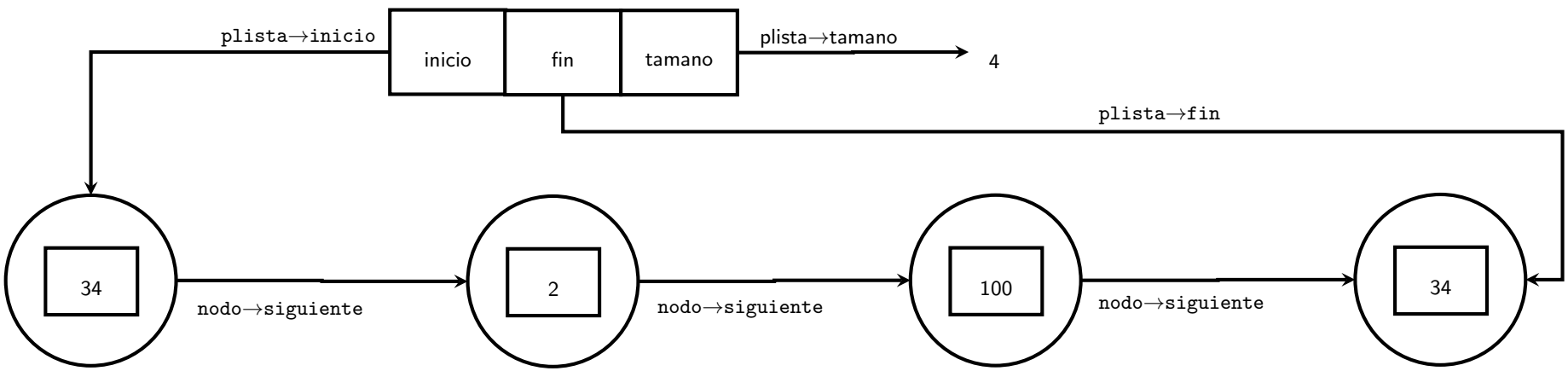
```
void mostrar(Lista *plista) {
  Nodo *nodo;
  nodo = plista->inicio;

  while(nodo != NULL) {
    printf("%d ", nodo->datos->dato1);
    nodo = nodo->siguiente;
  }
}
```

Observación

Se envía por parámetro un puntero a una lista, por que al enviar un elemento por referencia no creamos una copia de la variable, si no la dirección donde está almacenado y así no utilizamos espacio en memoria adicional.

Atsil anu odneirrocer

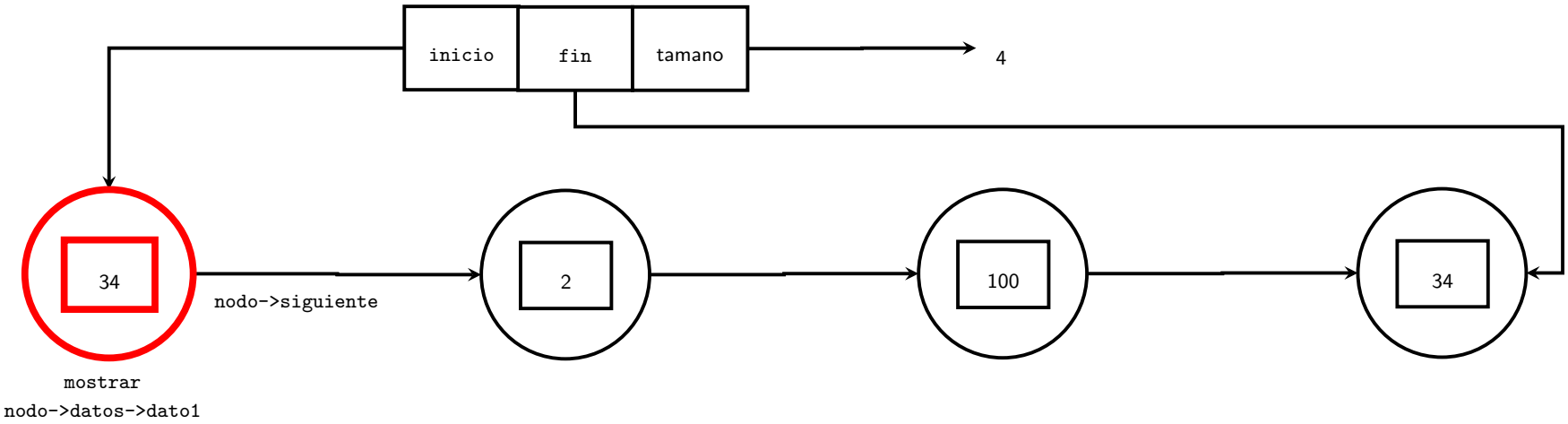
Diseñe una función que muestre la lista desde el último elemento hasta el primero. **Pista :** utilice recursividad.



Atsil anu odneirrocer

Acciones repetitivas

El proceso es sencillo nuevamente, consiste en acceder a cada uno de los nodos hasta llegar al final, para luego devolverse y mostrar los elementos de cada nodo.

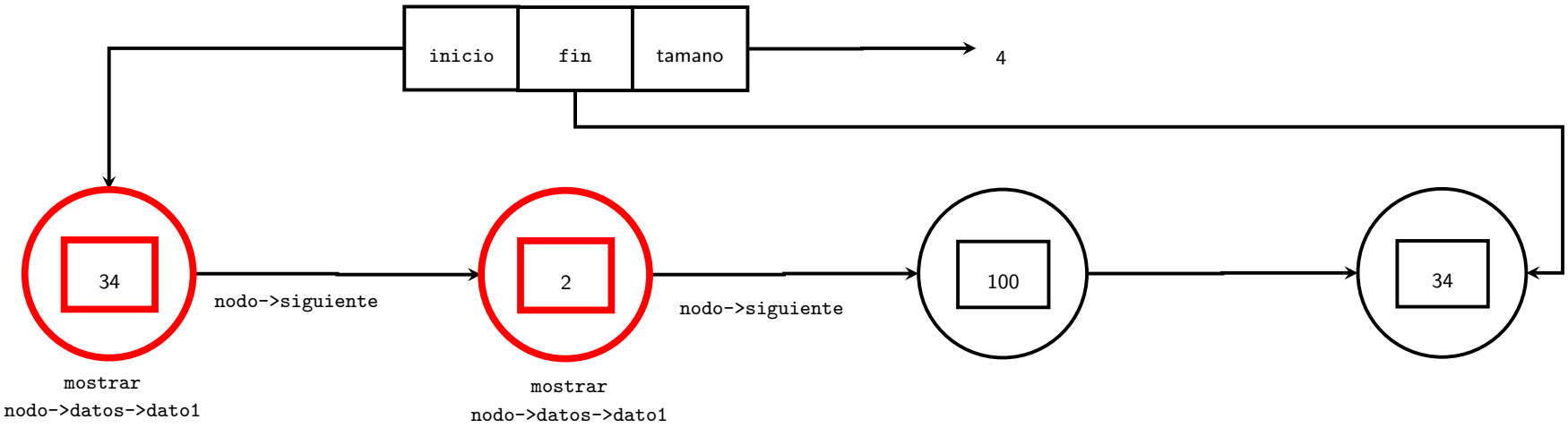



```
nodo = nodo->siguiente;
printf("%d ", nodo->datos->dato1);
```

Atsil anu odneirrocer

Acciones repetitivas

El proceso es sencillo nuevamente, consiste en acceder a cada uno de los nodos hasta llegar al final, para luego devolverse y mostrar los elementos de cada nodo.

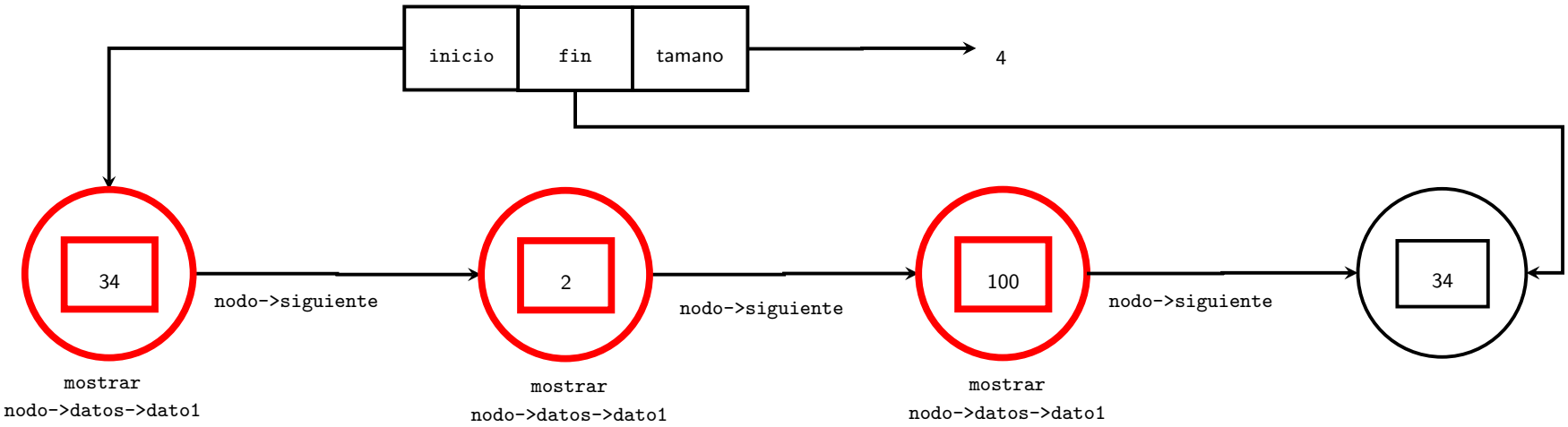



```
nodo = nodo->siguiente;
printf("%d ", nodo->datos->dato1);
```

Atsil anu odneirrocer

Acciones repetitivas

El proceso es sencillo nuevamente, consiste en acceder a cada uno de los nodos hasta llegar al final, para luego devolverse y mostrar los elementos de cada nodo.

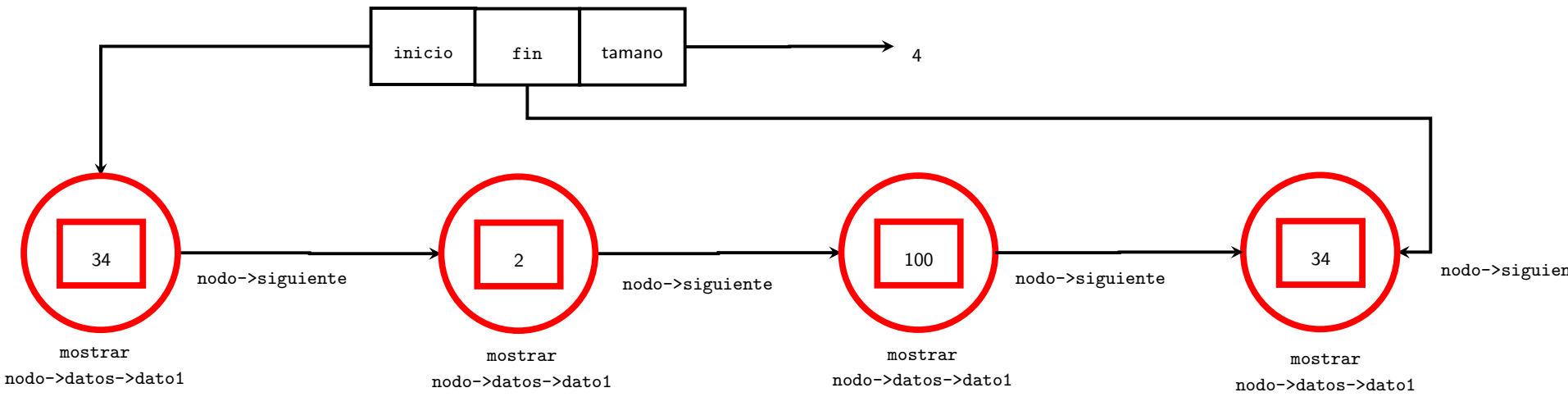



```
nodo = nodo->siguiente;
printf("%d ", nodo->datos->dato1);
```

Atsil anu odneirrocer

Acciones repetitivas

El proceso es sencillo nuevamente, consiste en acceder a cada uno de los nodos hasta llegar al final, para luego devolverse y mostrar los elementos de cada nodo.

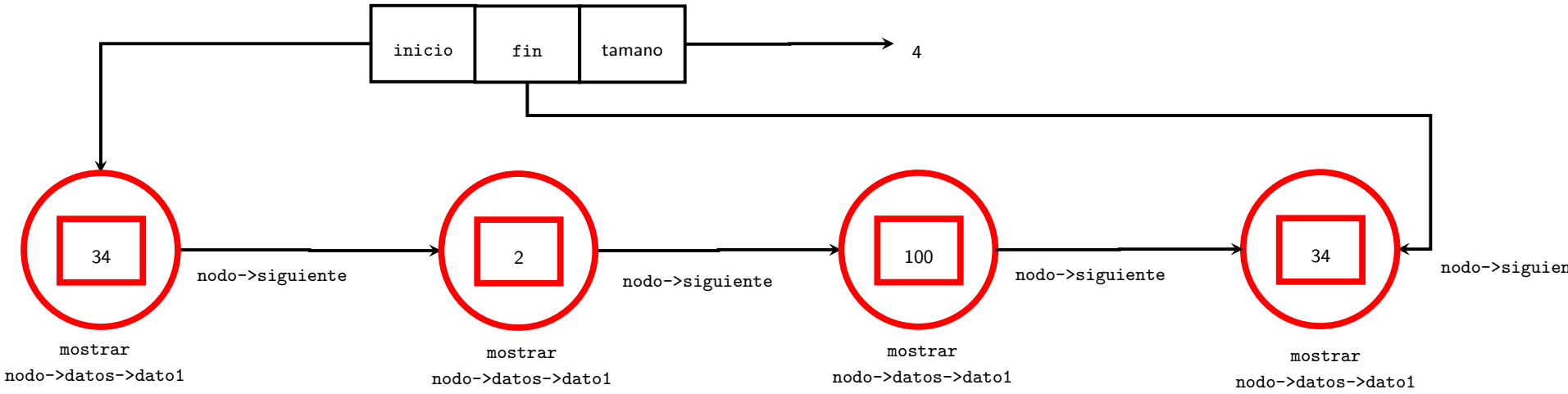



```
nodo = nodo->siguiente;
printf("%d ", nodo->datos->dato1);
```

Atsil anu odneirrocer

Programando las acciones

Del proceso anterior pudimos percatarnos que se repite siempre la misma acción. De ahí deriva la idea de utilizar ciclos, donde la **condición de término** es cuando el nodo sea igual a `NULL` ¿Pero ésto no hará que perdamos los punteros anteriores? Efectivamente, el proceso hará que se pierdan los punteros anteriores, por eso es que los dibujo quedan marcados a medida que transcurre el proceso, a la espera de que comiencen a devolverse gracias a la `recursividad` que aplicaremos.




```
if(nodo != NULL) {
  mostrarInverso(nodo->siguiente);
  printf("%d ", nodo->datos->dato1);
}
```

Atsil anu odneirrocer

La función mostrar

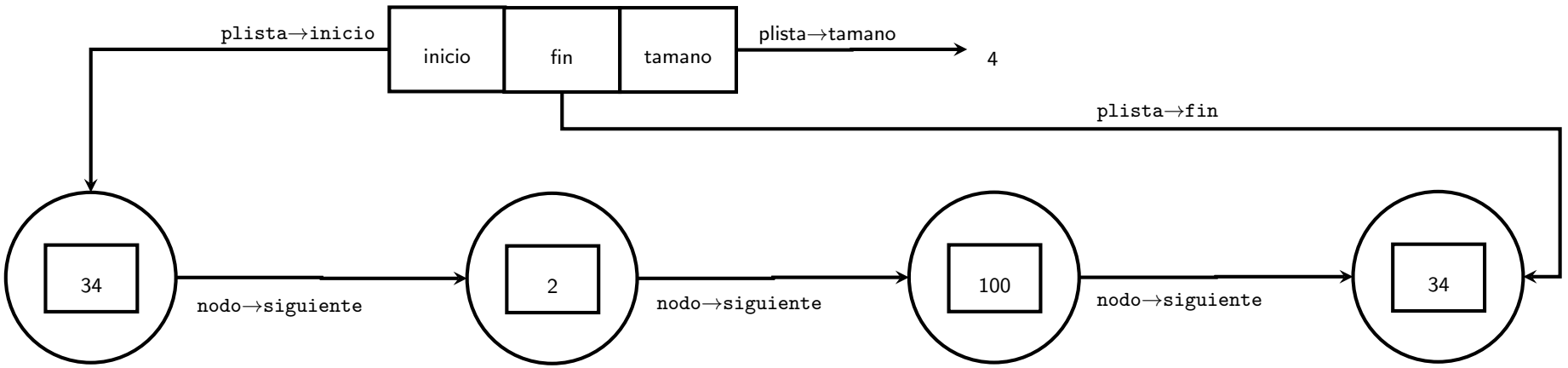
```
void mostrarInverso(Nodo *nodo) {
  if(nodo != NULL) {
    mostrarInverso(nodo->siguiente);
    printf("%d ", nodo->datos->dato1);
  }
}
```

Observación

La primera llamada a la función `mostrarInverso` debe recibir como parámetro el primer nodo de la lista, es decir, `plista->inicio`

Eliminar un elemento x

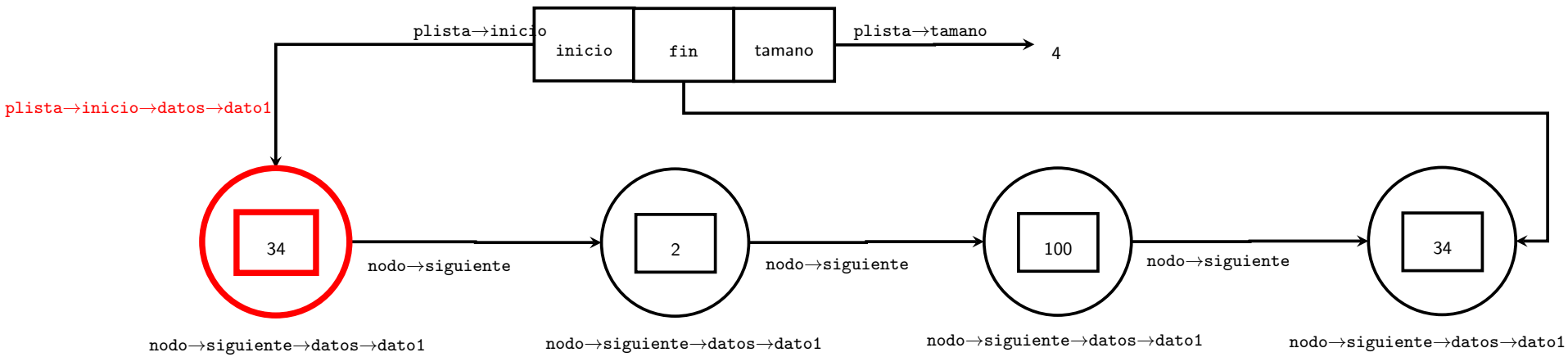
Diseñe una función que elimine un elemento x de la lista.



Eliminar un elemento x

Acciones repetitivas

El proceso consiste en buscar el elemento y sacarlo de la lista. En éste caso no resulta tan sencillo como suena, pues debemos almacenar el nodo anterior al nodo que deseamos eliminar

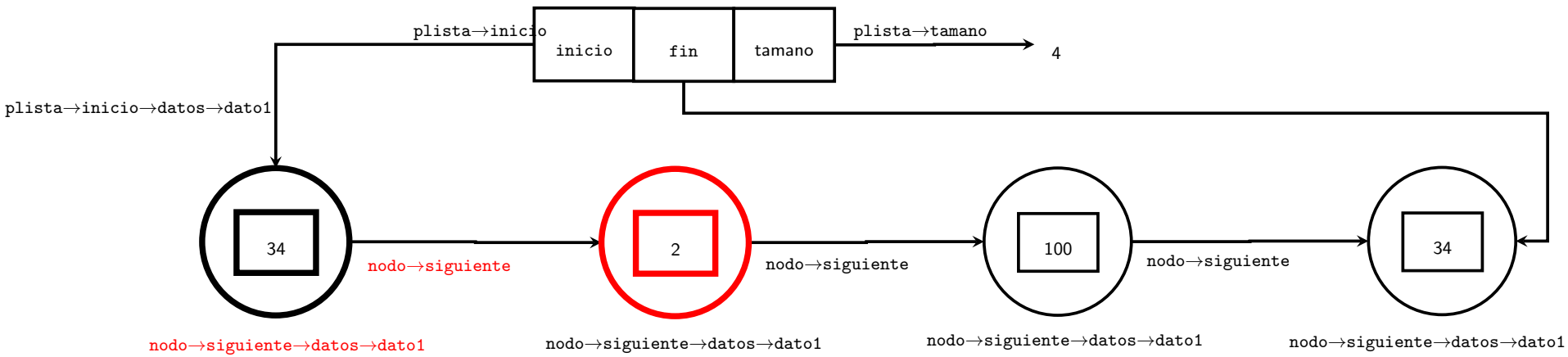



```
if(plista->inicio->datos->dato1 == x) {
  // Eliminar
}
```

Eliminar un elemento x

Acciones repetitivas

El proceso consiste en buscar el elemento y sacarlo de la lista. En éste caso no resulta tan sencillo como suena, pues debemos almacenar el nodo anterior al nodo que deseamos eliminar

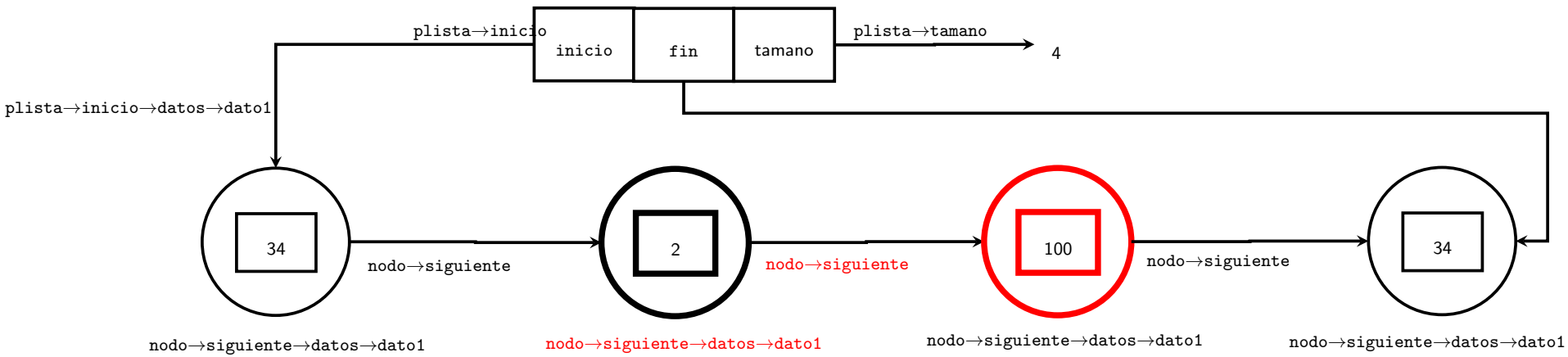



```
if(nodo->siguiente->datos->dato1 == x) {
  // Eliminar
}
```

Eliminar un elemento x

Acciones repetitivas

El proceso consiste en buscar el elemento y sacarlo de la lista. En éste caso no resulta tan sencillo como suena, pues debemos almacenar el nodo anterior al nodo que deseamos eliminar

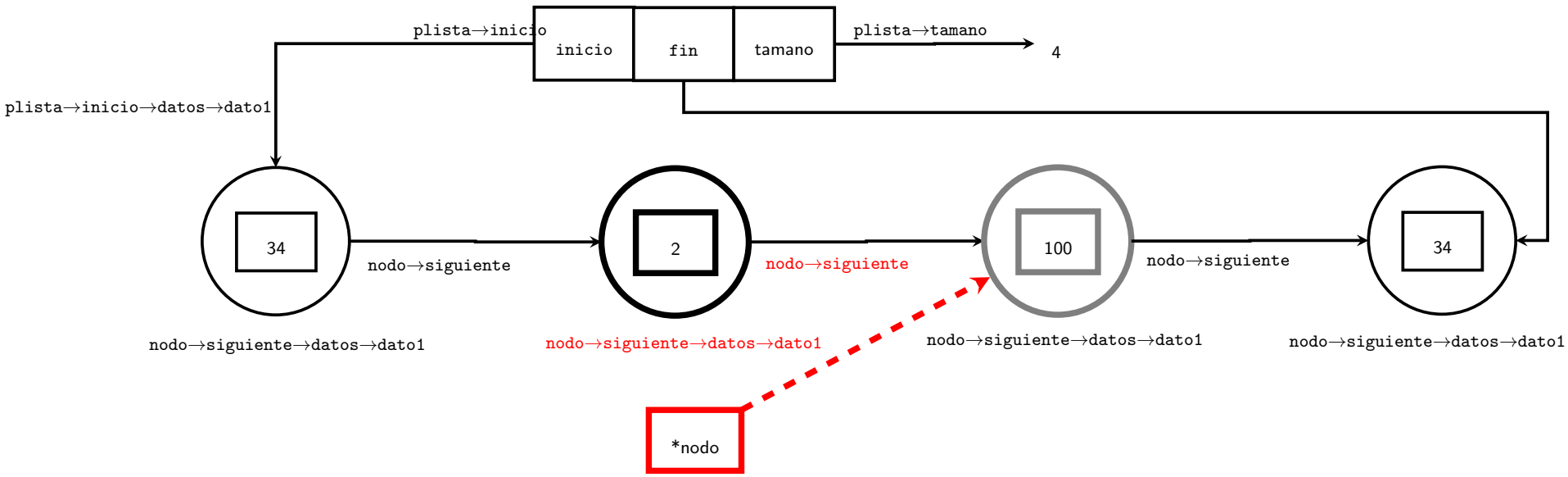



```
if(nodo->siguiente->datos->dato1 == x) {
  // Eliminar nodo
}
```

Eliminar un elemento x

Acciones repetitivas

El proceso consiste en buscar el elemento y sacarlo de la lista. En éste caso no resulta tan sencillo como suena, pues debemos almacenar el nodo anterior al nodo que deseamos eliminar

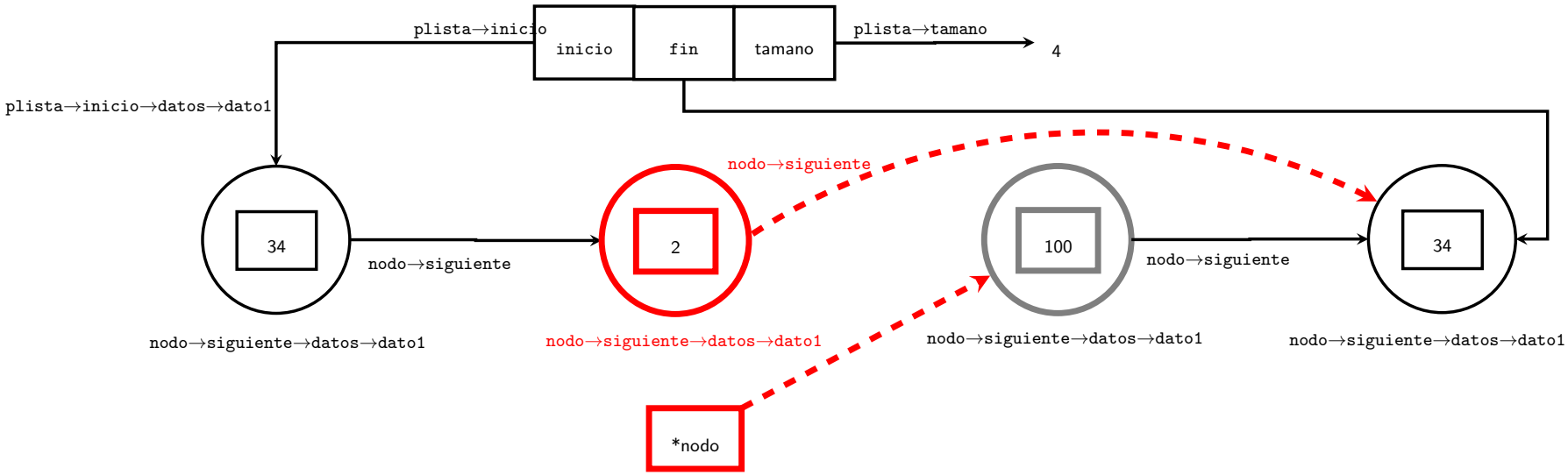



```
if(nodo->siguiente->datos->dato1 == x) {
  Nodo *aux;
  aux = nodo;
}
```

Eliminar un elemento x

Acciones repetitivas

El proceso consiste en buscar el elemento y sacarlo de la lista. En éste caso no resulta tan sencillo como suena, pues debemos almacenar el nodo anterior al nodo que deseamos eliminar

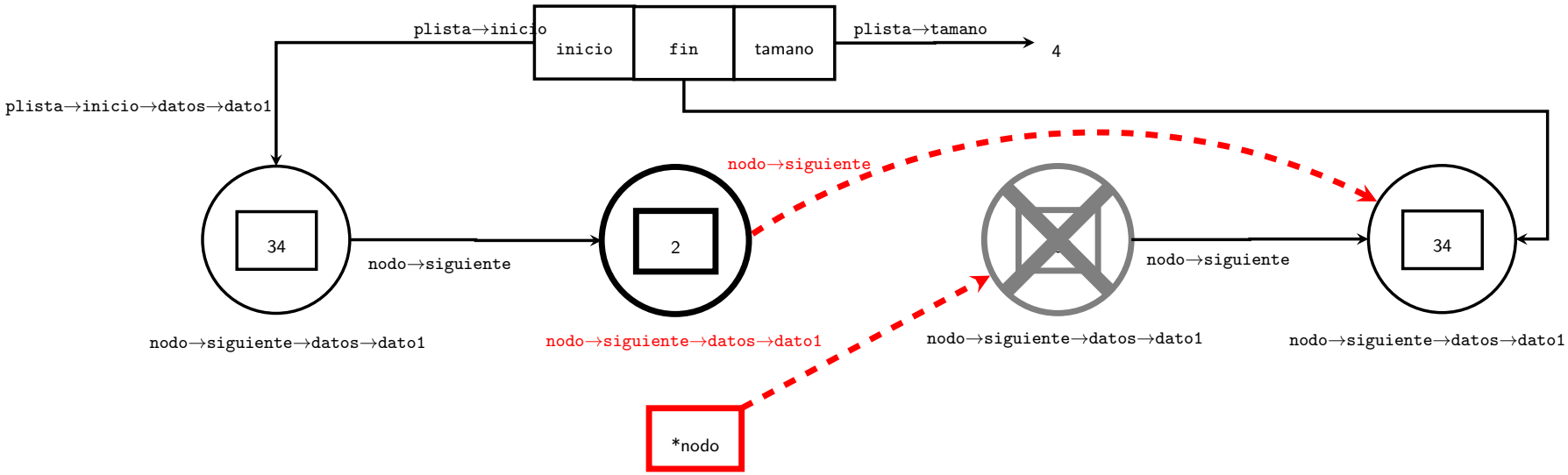



```
if(nodo->siguiente->datos->dato1 == x) {
  Nodo *aux;
  aux = nodo;

  nodo->siguiente = aux->siguiente;
}
```

Eliminar un elemento x

Acciones repetitivas

El proceso consiste en buscar el elemento y sacarlo de la lista. En éste caso no resulta tan sencillo como suena, pues debemos almacenar el nodo anterior al nodo que deseamos eliminar

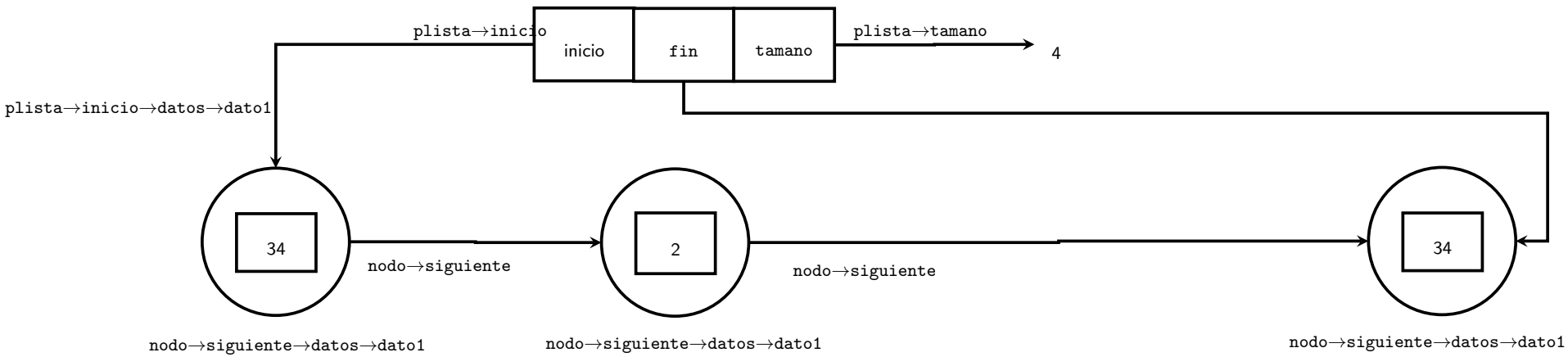



```
if(nodo->siguiente->datos->dato1 == x) {
  Nodo *aux;
  aux = nodo;

  nodo->siguiente = aux->siguiente;
  free(aux);
}
```

Eliminar un elemento x

Acciones repetitivas

El proceso consiste en buscar el elemento y sacarlo de la lista. En éste caso no resulta tan sencillo como suena, pues debemos almacenar el nodo anterior al nodo que deseamos eliminar



Eliminar un elemento x

Programando las acciones

Del proceso anterior pudimos percatarnos que hay acciones que se repiten. De ahí deriva la idea de utilizar ciclos, donde la **condición de término** es cuando el nodo sea igual a `NULL`

```
Nodo *nodo;
nodo = plista->inicio;

while(nodo != NULL ) {
  if(nodo->siguiente->datos->dato1 == x) {
    Nodo *aux;
    aux = nodo->siguiente;

    nodo->siguiente = aux->siguiente;
    free(aux->datos);
    free(aux);
    break;
  }
  nodo = nodo->siguiente;
}
```

Eliminar un elemento x

Casos a considerar

Pero hay dos situaciones que nuestro programa no considera ¿Que pasa si el elemento que se desea eliminar es el primero o el último de la lista? Debemos contemplar dicha situación.

```
void eliminarElemento(Lista *plista, int x) {
  Nodo *nodo;
  nodo = plista->inicio;

  if(plista->inicio->datos->dato1 == x) {
    eliminar(plista);
  } else {
    while(nodo != NULL ) {
      if(nodo->siguiente->datos->dato1 == x) {
        Nodo *aux;
        aux = nodo->siguiente;

        nodo->siguiente = aux->siguiente;
        free(aux->datos);
        free(aux);

        if(nodo->siguiente == NULL) {
         plista->fin = nodo;
        }
        plista->tamano--;
        break;
      }
      nodo = nodo->siguiente;
    }
  }
}
```

Eliminar un elemento x

La función eliminarElemento

```
void eliminarElemento(Lista *plista, int x) {
  Nodo *nodo;
  nodo = plista->inicio;

  if(plista->inicio->datos->dato1 == x) {
    eliminar(plista);
  } else {
    while(nodo != NULL ) {
      if(nodo->siguiente->datos->dato1 == x) {
        Nodo *aux;
        aux = nodo->siguiente;

        nodo->siguiente = aux->siguiente;
        free(aux->datos);
        free(aux);

        if(nodo->siguiente == NULL) {
         plista->fin = nodo;
        }
        plista->tamano--;
        break;
      }
      nodo = nodo->siguiente;
    }
  }
}
```

Felipe Reyes González

Estructura de Datos

Listas Enlazadas
Ejercicios resueltos

Ingeniería en Computación e Informática

Universidad
Andrés Bello