Cap. 12 – Listas Enlazadas

Esquema

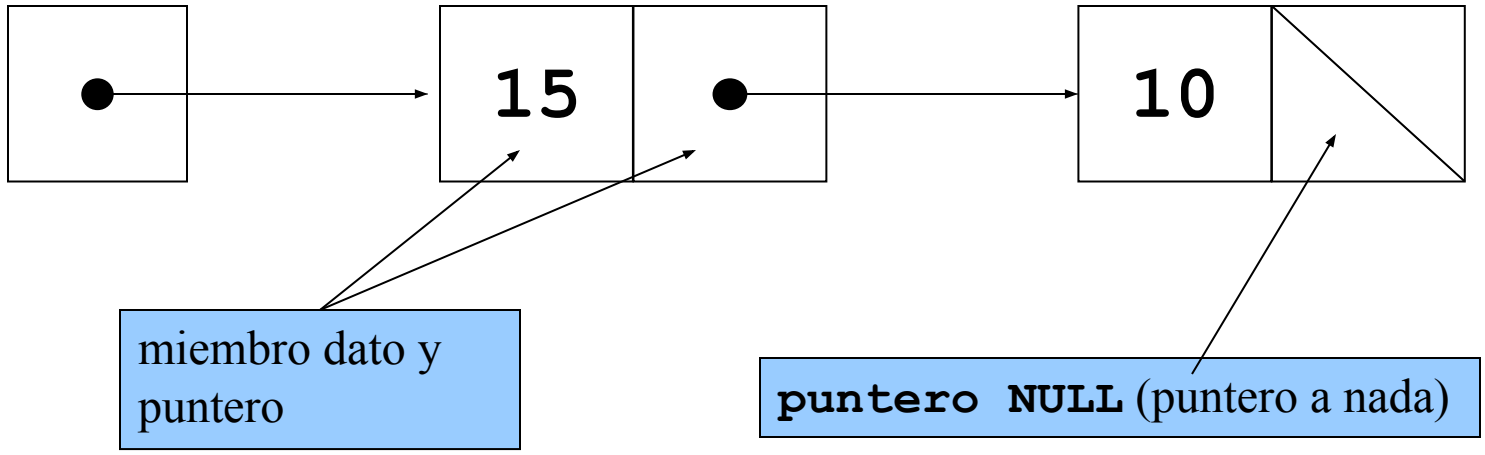
12.1 Introducción

- *Estructuras de datos dinámicas* - crecen y se encogen durante la ejecución
- *Listas enlazadas* - inserciones y retiros hechos en cualquier lugar
- *Pilas* - inserciones y retiros hechos sólo en la parte superior de la pila
- *Colas* - inserciones hechas en la parte posterior y retiros hechos desde el frente
- *Árboles binarios* - búsqueda y clasificación de datos a alta velocidad y eliminación eficiente de elementos de datos duplicados

12.2 Estructuras Autorreferenciadas

- Estructuras autorreferenciadas
  - Estructura que contiene un puntero a una estructura del mismo tipo
  - Pueden enlazarse para formar estructuras de datos útiles como listas, colas, pilas y árboles
  - Terminado con un puntero NULL (0)

- Dos objetos de estructura autorreferencial unidos entre sí



12.2 Estructuras Autorreferenciadas (II)

```
struct nodo {
    int data;
    struct nodo *nextPtr;
}
```

- **`nextPtr`** - puntero a un objeto de tipo **`node`**
  - Denominado como un **enlace** (*link*) – une un `nodo` a otro `nodo`

12.3 Ubicación de Memoria Dinámica

- Ubicación de memoria dinámica
  - Obtiene y libera memoria durante la ejecución
- **`malloc`**
  - Toma un número de bytes para ubicar
    - Use **`sizeof`** para determinar el tamaño de un objeto
  - Retorna puntero a tipo **`void *`**
    - Un puntero **`void *`** podría ser asignado a cualquier puntero
    - Si la memoria no está disponible, retorna **`NULL`**
  - `nuevoPtr = malloc( sizeof( struct nodo ) );`
- **`free`**
  - Libera memoria reservada por **`malloc`**
  - Toma un puntero como un argumento
  - `free (nuevoPtr);`

12.4 Lista Enlazada

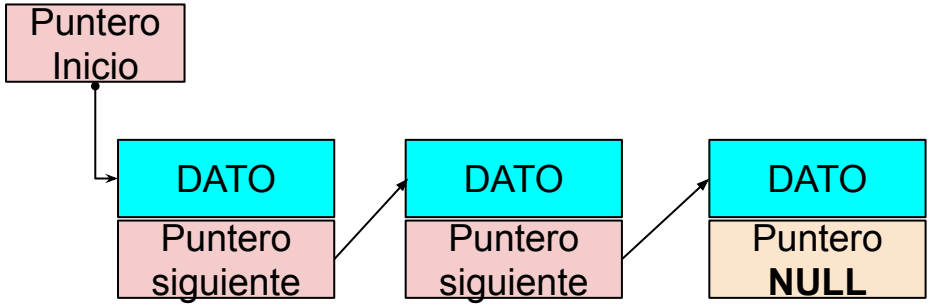
- Lista Enlazada
  - Colección lineal de objetos de clase auto-referencial, llamados nodos, conectados por enlaces punteros
  - Accediendo a través de un puntero al primer nodo de la lista
  - Los nodos subsiguientes se acceden a través del miembro de enlace
  - El puntero de enlace en el último nodo se pone en nulo para marcar el final de la lista.

- Utilice una lista de enlaces en lugar de una arreglo cuando
  - El número de elementos de datos es impredecible
  - La lista debe ser ordenada

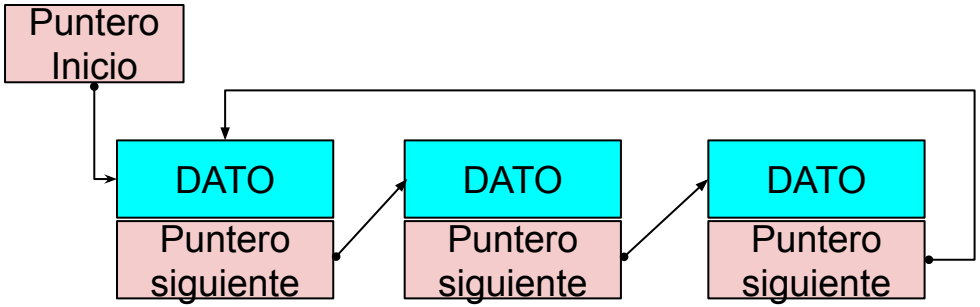
12.4 Lista Enlazada (II)

- Tipos de listas enlazadas:
  - *lista simplemente enlazada*
    - Comienza con un puntero al primer nodo
    - Termina con un puntero nulo
    - Sólo atravesó en una dirección
  - *lista circular simplemente enlazada*
    - El puntero en el último nodo apunta al primer nodo
  - *lista doblemente enlazada*
    - Dos "puntos de partida": el primer elemento y el último elemento...
    - Cada nodo tiene un puntero hacia adelante y otro hacia atrás
    - Permite atravesar tanto hacia adelante como hacia atrás
  - *lista circular doblemente enlazada*
    - El puntero hacia adelante del último nodo apunta al primer nodo y el puntero hacia atrás del primer nodo apunta al último nodo.

12.4 Lista Enlazada (III)

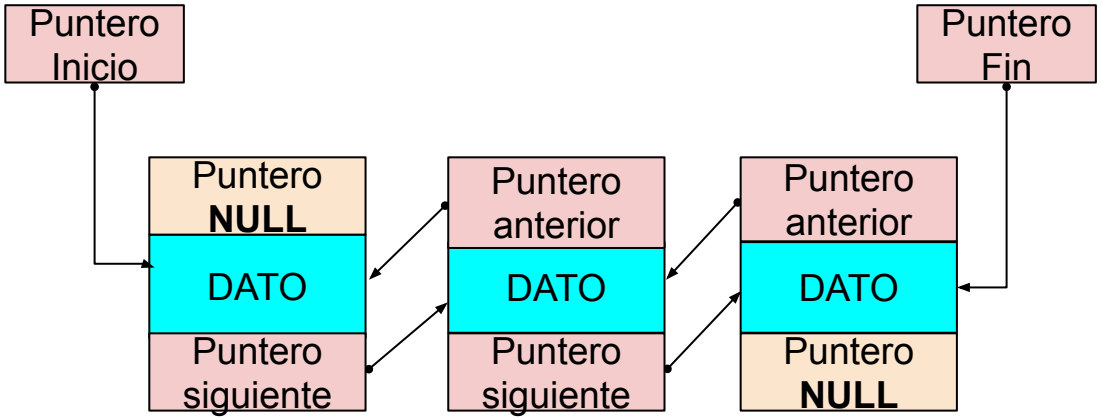


Lista simplemente enlazada

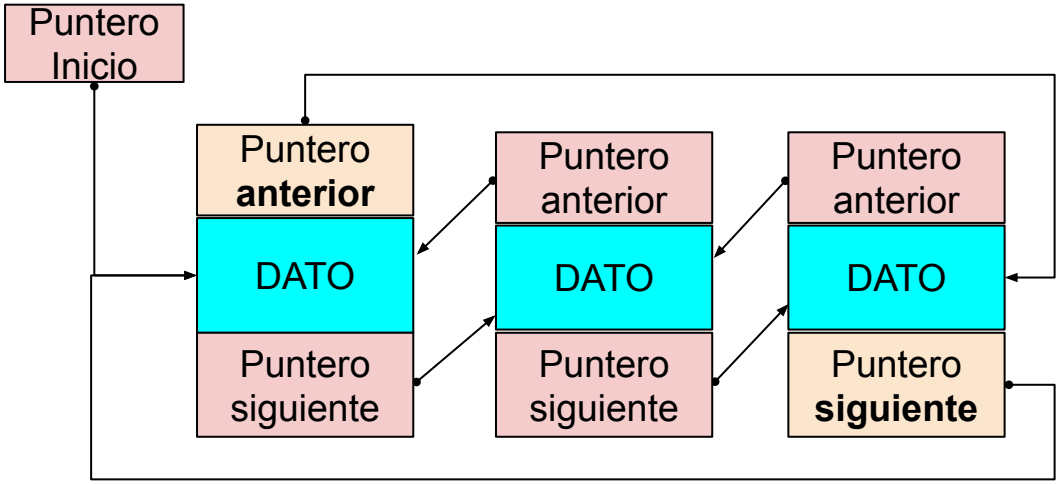


Lista circular simplemente enlazada

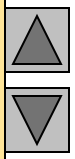
12.4 Lista Enlazada (III)



Lista doblemente enlazada



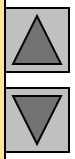
Lista circular doblemente enlazada



Outline

Cabecera
"lista_enlazada.h"

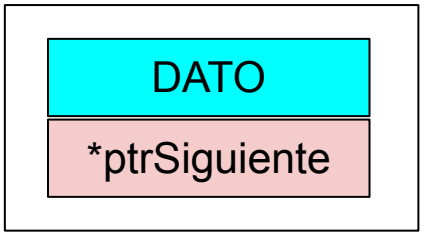
```
/* Fig. 12.3: fig12_03.c => "lista_enlazada.h"
 Operación y mantenimiento de una lista */

struct nodoLista{ /*estructura auto referenciada*/
    char dato;
    struct nodoLista * ptrSiguiente;
};

typedef struct nodoLista NodoLista;

typedef NodoLista *ptrNodoLista;

void insertar (ptrNodoLista *, char valor);
char eliminar (ptrNodoLista *, char valor);
int  estaVacia(ptrNodoLista);
void imprimeLista (ptrNodoLista);
```



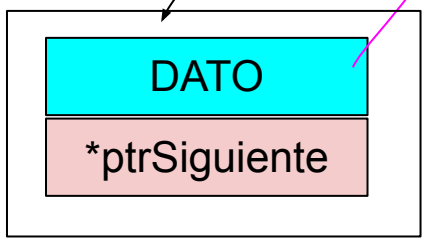
Outline

Cabecera "lista.h"

```
/* Fig. 12.3: fig12_03.c => "lista_enlazada.h"
 Operación y mantenimiento de una lista */

struct nodoLista{ /*estructura auto referenciada*/
    char dato;
    struct nodoLista * ptrSiguiente;
};

typedef struct nodoLista NodoLista;

typedef NodoLista *ptrNodoLista;

void insertar (ptrNodoLista *, char valor);
char eliminar (ptrNodoLista *, char valor);
int  estaVacia(ptrNodoLista);
void imprimeLista (ptrNodoLista);
void instrucciones (void);
```



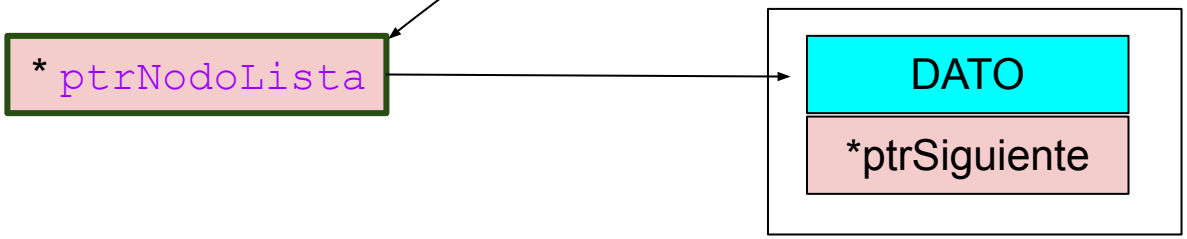
```
/* Fig. 12.3: fig12_03.c => "lista_enlazada.h"
 Operación y mantenimiento de una lista */

struct nodoLista{ /*estructura auto referenciada*/
    char dato;
    struct nodoLista * ptrSiguiente;
};

typedef struct nodoLista NodoLista;

typedef NodoLista *ptrNodoLista;

void insertar (ptrNodoLista *, char valor);
char eliminar (ptrNodoLista *, char valor);
int  estaVacia(ptrNodoLista);
void imprimeLista (ptrNodoLista);
void instrucciones (void);
```



Outline

Cabecera "lista.h"

```
/* Fig. 12.3: fig12_03.c => "lista_enlzada.h"
 Operación y mantenimiento de una lista */

struct nodoLista{ /*estructura auto referenciada*/
    char dato;
    struct nodoLista * ptrSiguiente;
};

typedef struct nodoLista NodoLista;

typedef NodoLista *ptrNodoLista;

void insertar (ptrNodoLista *, char valor);
char eliminar (ptrNodoLista *, char valor);
int  estaVacia(ptrNodoLista);
void imprimeLista (ptrNodoLista);
void instrucciones (void);
```



Outline

Función main

```
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lista_enlazada.h"
void instrucciones (void);
int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección del usuario*/
      char elemento;                  /*carácter a ser ingresado en la lista enlazada*/

      instrucciones();  /* despliega el menú */
      printf("? ");    scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: " );
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter para eliminar: "
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, elemento ) ) {
                                    printf( "%c eliminado.\n", elemento);
                                    imprimeLista( ptrInicial);
                              }else{
                                    printf( "no se encuentra el caracter %c\n", elemento );
                              }
                        }else{
                              printf( "La lista esta vacia.\n\n" );
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" ); return 0;
} /*fin main*/
```

Outline

Función main

```
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lita_enlazada.h"

int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección del usuario*/
      char elemento;                  /*carácter a ser ingresado en la lista enlazada*/

      instructiones();  /* despliega el menú */
      printf("? ");    scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: "
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter para eliminar: "
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, elemento ) ) {
                                    printf( "%c eliminado.\n", elemento);
                                    imprimeLista( ptrInicial);
                              }else{
                                    printf( "no se encuentra el caracter %c\n", elemento );
                              }
                        }else{
                              printf( "La lista esta vacia.\n\n" );
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" )
} /*fin main*/
```

```
/* despliega las instrucciones del programa para
el usuario */
void instructiones( void ){
     printf("Introduza su elección: \n"
            " 1 para insertar un elemento.\n"
            " 2 para eliminar un elemento.\n"
            " 3 para terminar\n");
}/*fin instrucciones*/
```

```
Introduzca su elección:
1 para insertar un elemento.
2 para eliminar un elemento.
3 para terminar.
?
```

Outline

Función main

```
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lita_enlazada.h"

int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección del usuario*/
      char elemento;                  /*carácter a ser ingresado en la lista enlazada*/

      instructiones();  /* despliega el menú */
      printf("? ");     scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: "
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter para eliminar: "
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, elemento ) ) {
                                    printf( "%c eliminado.\n", elemento);
                                    imprimeLista( ptrInicial);
                              }else{
                                    printf( "no se encuentra el caracter %c\n", elemento );
                              }
                        }else{
                              printf( "La lista esta vacia.\n\n" );
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" )
} /*fin main*/
```

```
/* despliega las instrucciones del programa para
el usuario */
void instructiones( void ){
     printf("Introduza su elección: \n"
            " 1 para insertar un elemento.\n"
            " 2 para eliminar un elemento.\n"
            " 3 para terminar\n");
}/*fin instrucciones*/
```

```
Introduzca su elección:
1 para insertar un elemento.
2 para eliminar un elemento.
3 para terminar.
? 1
```

Outline

Función main

```
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lita_enlazada.h"

int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección del usuario*/
      char elemento;                  /*carácter a ser ingresado en la lista enlazada*/

      instructiones();  /* despliega el menú */
      printf("? ");    scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: " );
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter para eliminar: "
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, elemento ) ) {
                                    printf( "%c eliminado.\n", elemento);
                                    imprimeLista( ptrInicial);
                              }else{
                                    printf( "no se encuentra el caracter %c\n", elemento );
                              }
                        }else{
                              printf( "La lista esta vacia.\n\n" );
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" )
} /*fin main*/
```

```
Introduzca su elección:
1 para insertar un elemento.
2 para eliminar un elemento.
3 para terminar.
? 1
Introduzca un caracter: B
```

```c
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lita_enlazada.h"

int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección del usuario*/
      char elemento;                  /*carácter a ser ingresado en la lista enlazada*/

      instructiones();  /* despliega el menú */
      printf("? ");    scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: " );
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter para eliminar: "
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, elemento ) ) {
                                    printf( "%c eliminado.\n", elemento);
                                    imprimeLista( ptrInicial);
                              }else{
                                    printf( "no se encuentra el caracter %c\n", elemento );
                              }
                        }else{
                              printf( "La lista esta vacia.\n\n" );
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" )
} /*fin main*/
```

```
Introduzca su elección:
1 para insertar un elemento.
2 para eliminar un elemento.
3 para terminar.
? 1
Introduzca un caracter: B
```

Outline

Función main

Outline

Función main

```
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lita_enlazada.h"

int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección del usuario*/
      char elemento;                  /*carácter a ser ingresado en la lista enlazada*/

      instructiones();  /* despliega el menú */
      printf("? ");    scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: " );
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter para eliminar: "
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, elemento ) ) {
                                    printf( "%c eliminado.\n", elemento);
                                    imprimeLista( ptrInicial);
                              }else{
                                    printf( "no se encuentra el caracter %c\n", elemento );
                              }
                        }else{
                              printf( "La lista esta vacia.\n\n" );
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" )
} /*fin main*/
```

```
Introduzca su elección:
1 para insertar un elemento.
2 para eliminar un elemento.
3 para terminar.
? 1
Introduzca un caracter: B
La lista es:
B --> NULL
```

Outline

Función main

```
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lita_enlazada.h"

int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección del usuario*/
      char elemento;                  /*carácter a ser ingresado en la lista enlazada*/

      instructiones();  /* despliega el menú */
      printf("? ");    scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: " );
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, e
                                    printf( "%c eliminado.\
                                    imprimeLista( ptrInicia
                              }else{
                                    printf( "no se encuentra
                              }
                        }else{
                              printf( "La lista esta vacia.
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" )
} /*fin main*/
```

```
Introduzca su elección:
1 para insertar un elemento.
2 para eliminar un elemento.
3 para terminar.
? 1
Introduzca un caracter: B
La lista es:
B --> NULL
? 1
Introduzca un caracter: A
La lista es:
A --> B --> NULL
? 1
Introduzca un caracter: C
La lista es:
A --> B --> C --> NULL
```

Outline

Función main

```
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lita_enlazada.h"

int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección del usuario*/
      char elemento;                  /*carácter a ser ingresado en la lista enlazada*/

      instructiones();  /* despliega el menú */
      printf("? ");    scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: " );
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter para eliminar: "
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, elemento ) ) {
                                    printf( "%c eliminado.\n", elemento);
                                    imprimeLista( ptrInicial);
                              }else{
                                    printf( "no se encuentra el caracter %c\n",
                              }
                        }else{
                              printf( "La lista esta vacia.\n\n" );
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" ); return 0;
} /*fin main*/
```

```
La lista es:
A --> B --> C --> NULL
```

```
Introduzca su elección:
1 para insertar un elemento.
2 para eliminar un elemento.
3 para terminar.
? 2
```

Outline

Función main

```
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lita_enlazada.h"

int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección
      char elemento;                  /*carácter a ser ingresado en la lista

      instructiones();  /* despliega el menú */
      printf("? ");    scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: " );
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter para eliminar: ");
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, elemento ) ) {
                                    printf( "%c eliminado.\n", elemento);
                                    imprimeLista( ptrInicial);
                              }else{
                                    printf( "No se encuentra el caracter %c\n",
                              }
                        }else{
                              printf( "La lista esta vacia.\n\n" );
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" ); return 0;
} /*fin main*/
```

```
/* Devuelve 1 si la lista está
vacía, de lo contrario, 0 */
int estaVacia( ptrNodoLista ptrS )
{
      return ptrS == NULL;

} /* fin de la función function
estaVacia */
```

```
La lista es:
A --> B --> C --> NULL
```

```
Introduzca su elección:
1 para insertar un elemento.
2 para eliminar un elemento.
3 para terminar.
? 2
Introduzca un caracter para
eliminar: D
No se encuentra el caracter D.
?
```

Outline

Función main

```
/* Fig. 12.3: fig12_03.c Operación y mantenimiento de una lista */
#include <stdio.h>
#include <stdlib.h>
#include "lita_enlazada.h"

int main(){
      ptrNodoLista ptrInicial = NULL; /*puntero a lista enlazada*/
      int eleccion;                   /*variable en donde se guarda elección del usuario*/
      char elemento;                  /*carácter a ser ingresado en la lista enlazada*/

      instructiones();  /* despliega el menú */
      printf("? "); scanf( "%d", &eleccion);

      while ( eleccion != 3 ) {
            switch ( eleccion ) {
                  case 1:
                        printf( "Introduzca un caracter: " );
                        scanf( "\n%c", &elemento );
                        insertar ( &ptrInicial, elemento);
                        imprimeLista( ptrInicial);
                        break;
                  case 2:
                        if ( !estaVacia( ptrInicial ) ) {
                              printf( "Introduzca un caracter para eliminar: ");
                              scanf( "\n%c", &elemento );
                              if ( eliminar( &ptrInicial, elemento ) ) {
                                    printf( "%c eliminado.\n", elemento);
                                    imprimeLista( ptrInicial);
                              }else{
                                    printf( "No se encuentra el caracter %c\n",
                              }
                        }else{
                              printf( "La lista esta vacia.\n\n" );
                        }
                        break;
                  default:
                        printf( "Opcion invalida.\n\n" );
                        instructions();
                        break;
            }/*fin switch*/
            printf("? "); scanf( "%d", &eleccion);
      ]/*fin While*/ printf( "Fin de la ejecucion.\n" ); return 0;
} /*fin main*/
```

```
La lista es:
A --> B --> C --> NULL
```

```
Introduzca su elección:
1 para insertar un elemento.
2 para eliminar un elemento.
3 para terminar.
? 2
Introduzca un caracter para
eliminar: D
No se encuentra el caracter D.
? 2
Introduzca un caracter para
eliminar: B
caracter B eliminado.
La lista es:
A --> C --> NULL
?
```

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrActual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

```
La lista es:
NULL
```

Función insertar

valor

A

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

```
La lista es:
NULL
```

Función insertar

valor

A

*
ptrNuevo

*
ptrAnterior

*
ptrActual

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

```
La lista es:
NULL
```

Función insertar

valor

A

*
ptrNuevo

*
ptrAnterior

*
ptrActual

**
ptrS

*
puntero
Inicial

A

*ptrSiguiente

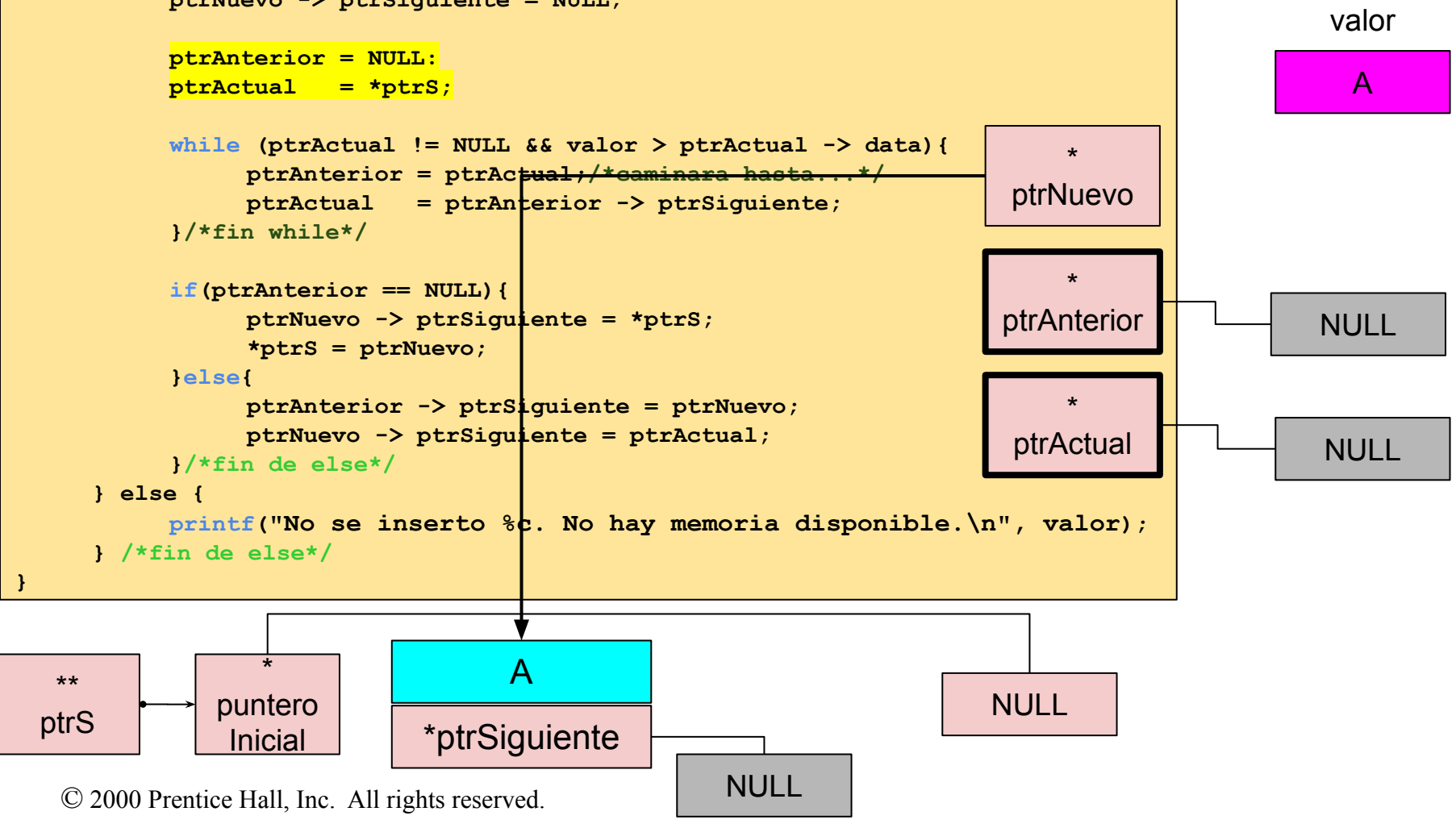
NULL

NULL

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrActual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

```
La lista es:
NULL
```

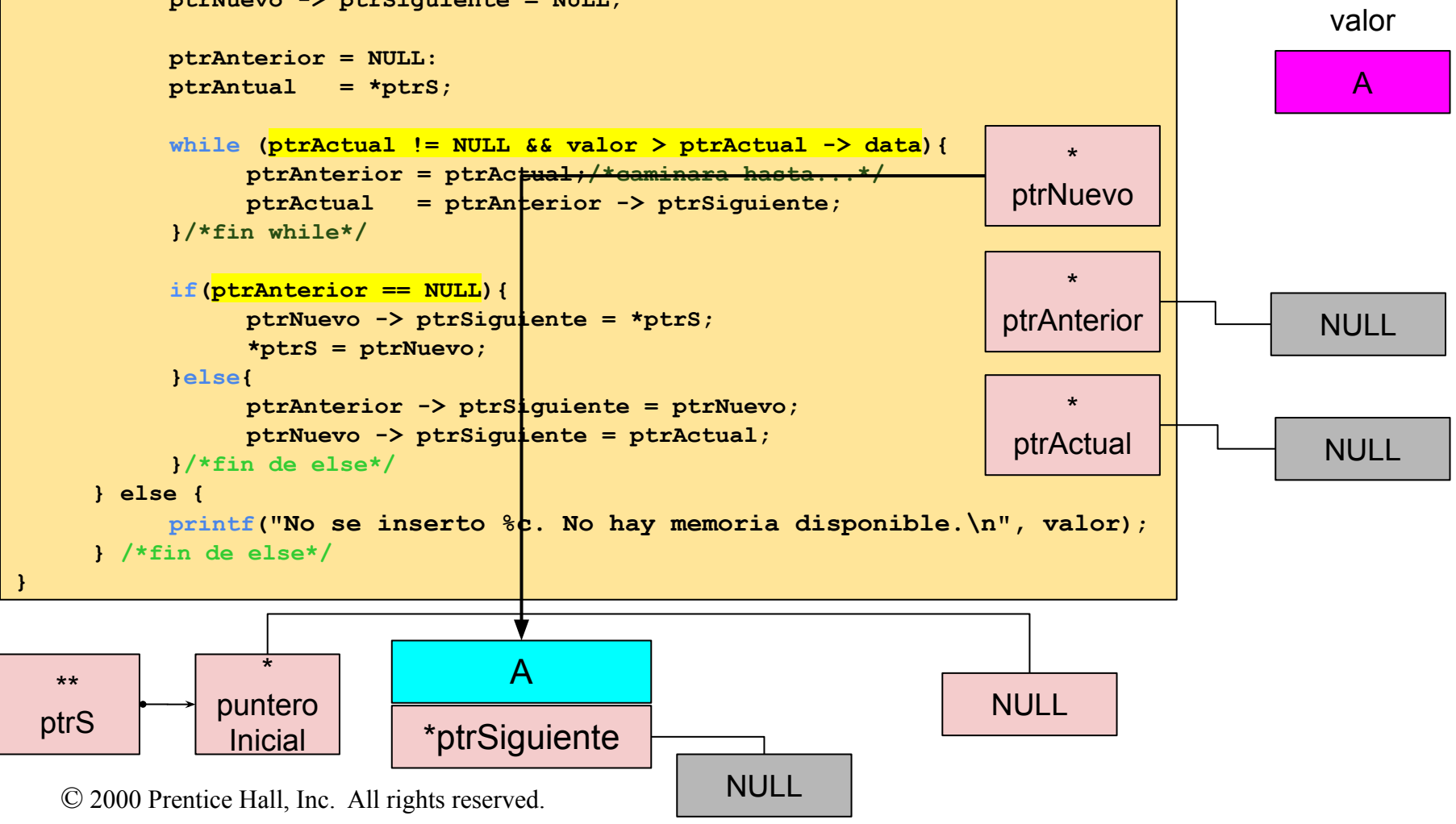
Función insertar



```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

```
La lista es:
NULL
```

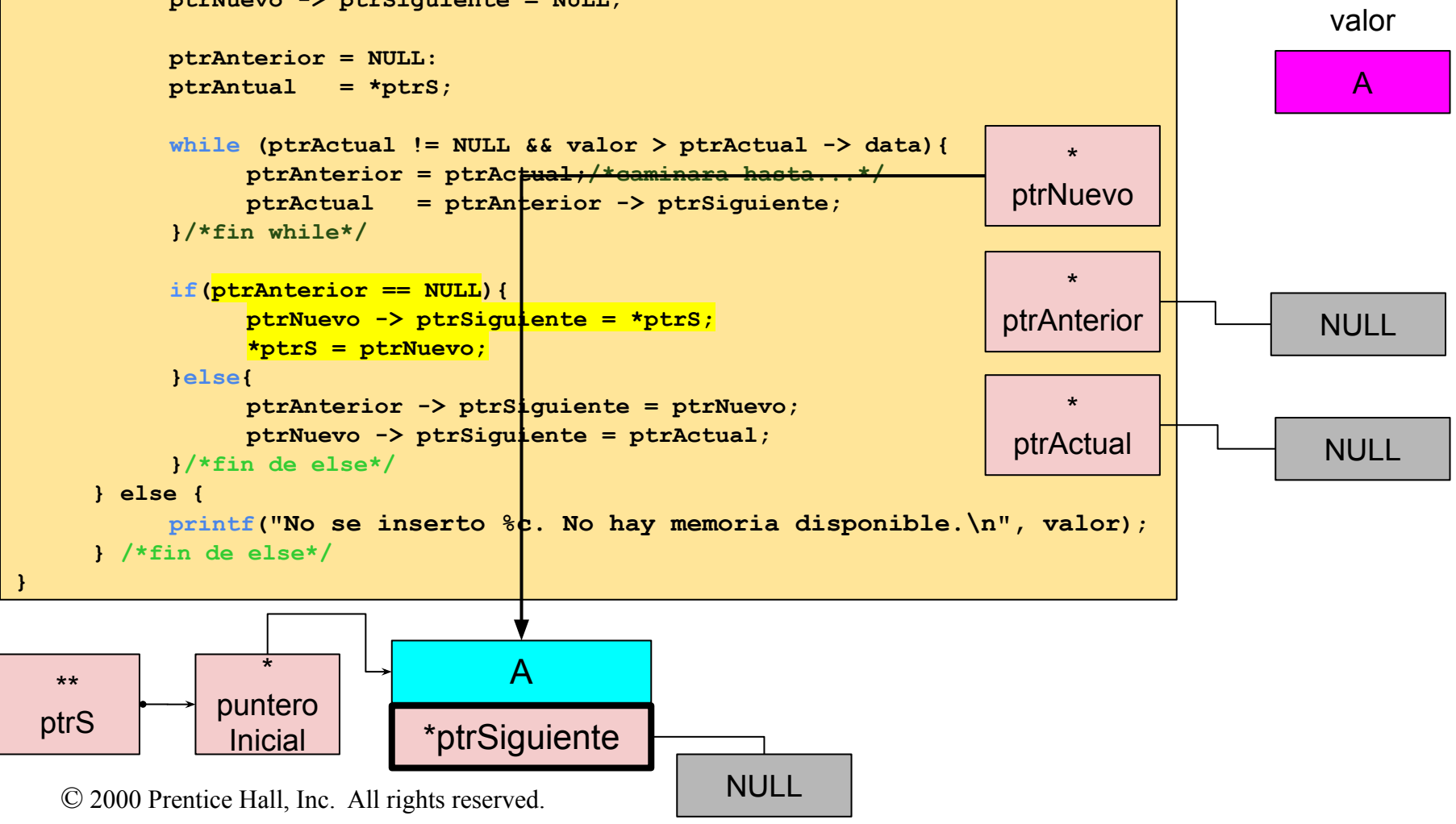
Función insertar



```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

```
La lista es:
A --> NULL
```

Función insertar



```
La lista es:
A --> NULL
```

Función insertar

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

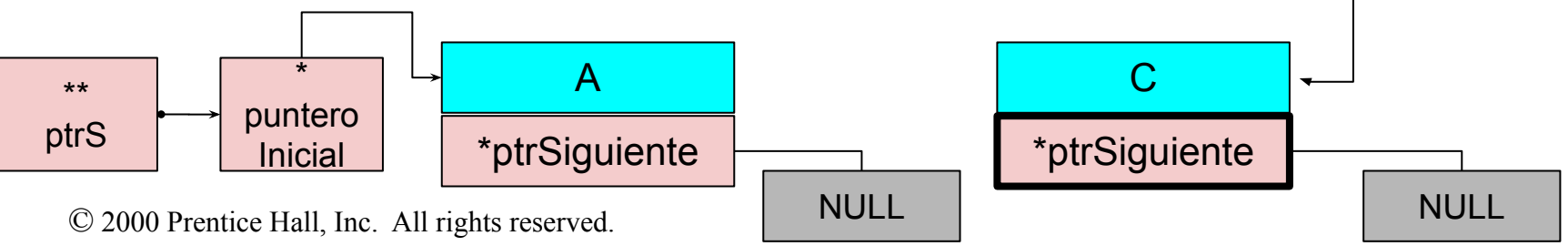
valor

C

*
ptrNuevo

*
ptrAnterior

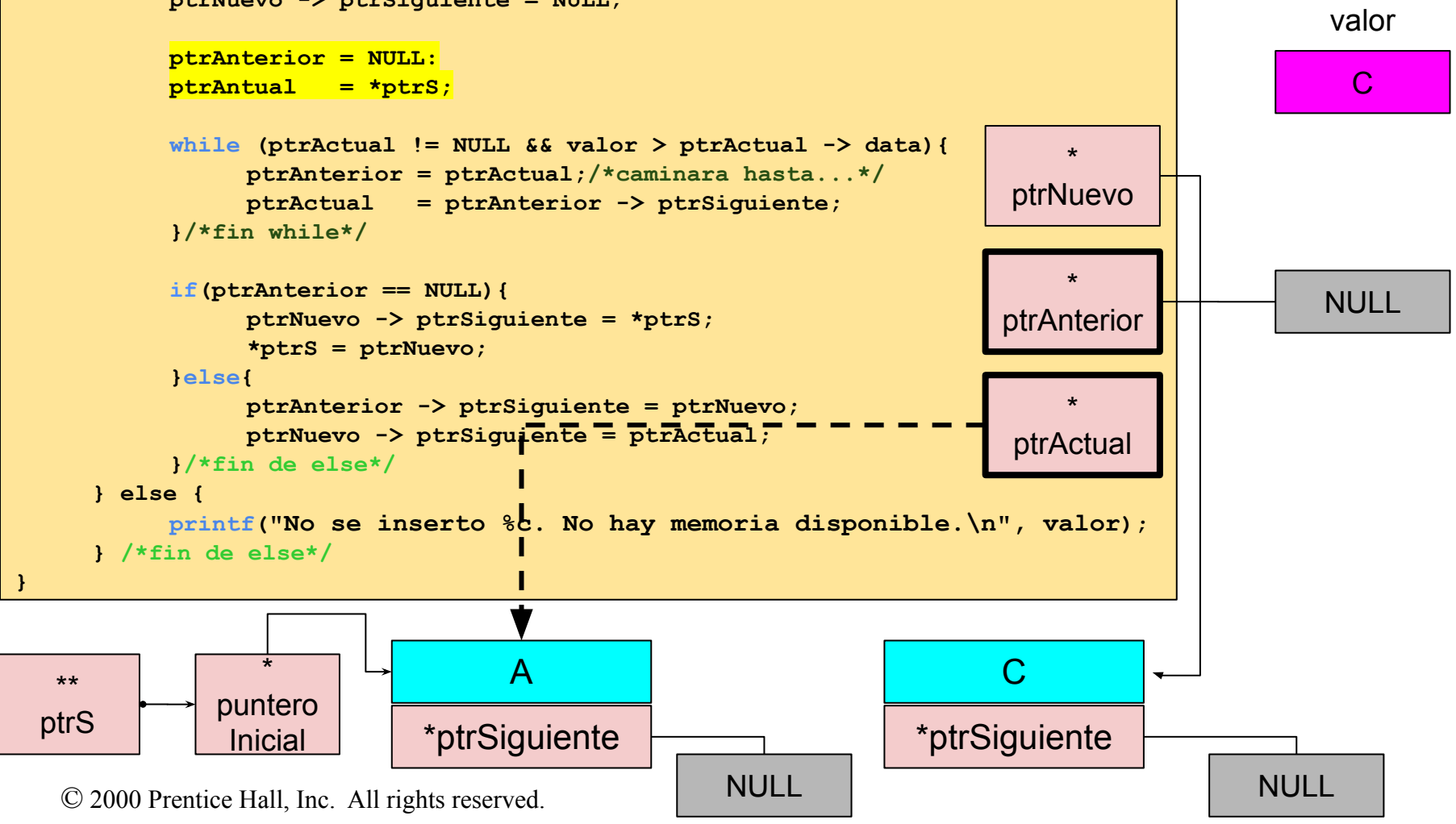
*
ptrActual



```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

```
La lista es:
A --> NULL
```

Función insertar



```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

```
La lista es:
A --> NULL
```

Función insertar

valor

C

*
ptrNuevo

*
ptrAnterior

*
ptrActual

**
ptrS

*
puntero
Inicial

A

*ptrSiguiente

NULL

C

*ptrSiguiente

NULL

```
La lista es:
A --> NULL
```

Función insertar

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

valor

C

* ptrNuevo

* ptrAnterior

* ptrActual

** ptrS

* puntero Inicial

A

*ptrSiguiente

NULL

C

*ptrSiguiente

NULL

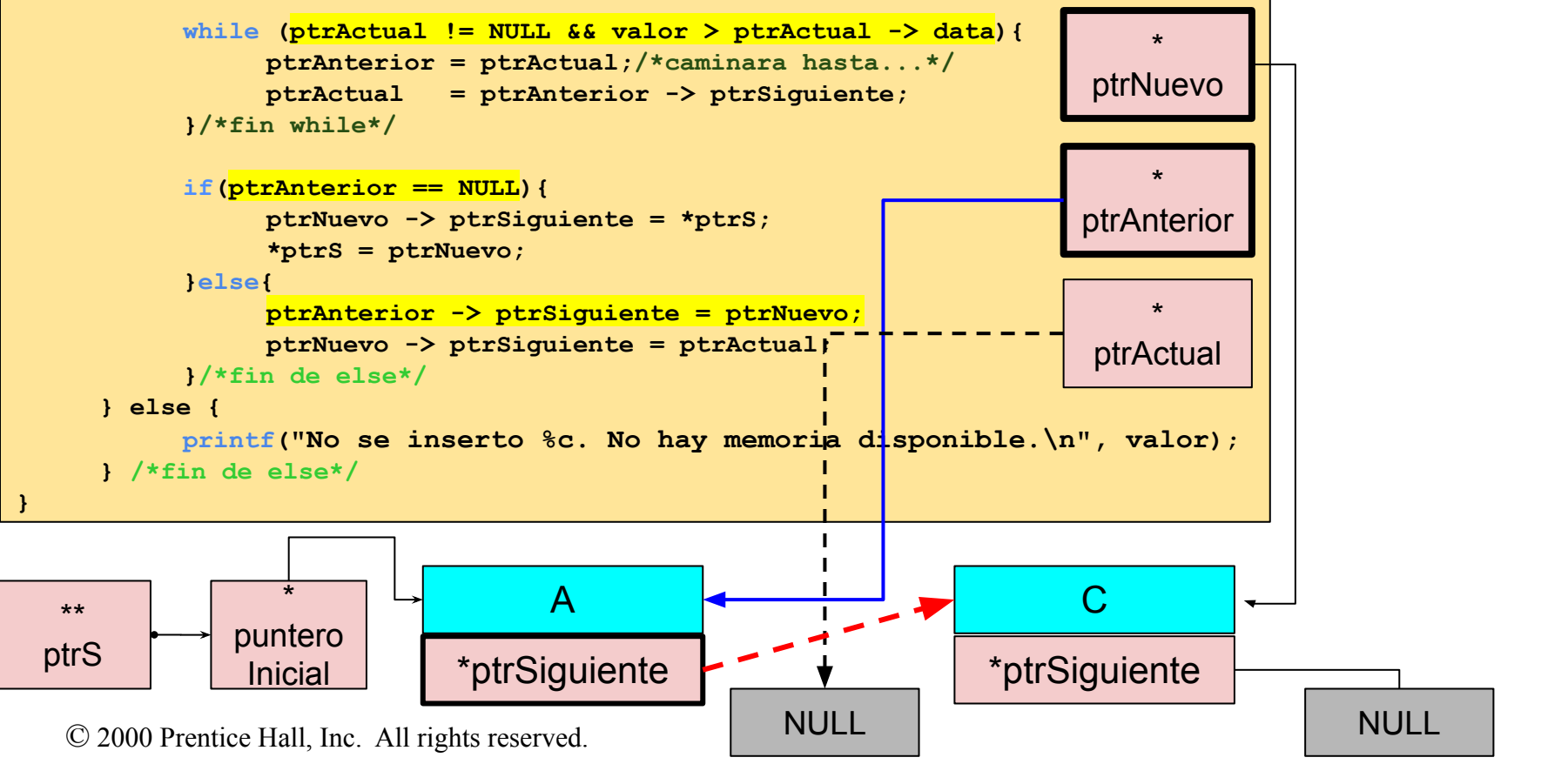
```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

```
La lista es:
A --> C --> NULL
```

Función insertar

valor

C



```
La lista es:
A --> C --> NULL
```

Función insertar

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
void insertar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrNuevo    ;/*puntero a lista enlazada nuevo nodo*/
      ptrNodoLista ptrAnterior ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrAtual    ;/*puntero a lista enlazada nodo actual*/

      ptrNodoLista ptrNuevo = malloc( sizeof( NodoLista ) );

      if(ptrNuevo != NULL){ /* es espacio disponible*/
            ptrNuevo -> dato = valor;
            ptrNuevo -> ptrSiguiente = NULL;

            ptrAnterior = NULL:
            ptrAntual   = *ptrS;

            while (ptrActual != NULL && valor > ptrActual -> data){
                  ptrAnterior = ptrActual;/*caminara hasta...*/
                  ptrActual   = ptrAnterior -> ptrSiguiente;
            }/*fin while*/

            if(ptrAnterior == NULL){
                  ptrNuevo -> ptrSiguiente = *ptrS;
                  *ptrS = ptrNuevo;
            }else{
                  ptrAnterior -> ptrSiguiente = ptrNuevo;
                  ptrNuevo -> ptrSiguiente = ptrActual;
            }/*fin de else*/
      } else {
            printf("No se inserto %c. No hay memoria disponible.\n", valor);
      } /*fin de else*/
}
```

valor

C

*
ptrNuevo

*
ptrAnterior

*
ptrActual

**
ptrS

*
puntero
Inicial

A

*ptrSiguiente

C

*ptrSiguiente

NULL

```
La lista es:
A --> B --> C --> NULL
```

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
char eliminar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrAnterior  ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrActual    ;/*puntero a lista enlazada nodo actual*/
      ptrNodoLista tempPtr;

      if(valor == (*ptrS) -> dato){
            tempPtr = *ptrS;
            *ptrS = (* ptrS) -> ptrSiguiente; /*desata el nodo*/
            free(tempPtr);    /*libera el nodo*/
            return valor;
      } else{
            ptrAnterior = *ptrS;
            ptrActual = (*ptrS) -> ptrSiguiente;

            while(ptrAcutal != NULL && ptrActual -> dato != valor ){
                  ptrAnterior = ptrActual;
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
            }/*fin de while*/

            if(ptrActual != NULL){
                  trmpPtr = ptrAcutal:
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
                  free(tempPtr);
                  return valor;
            }/*fin de if*/
      }/*fin de else*/
      return '\0';
}/*fin de función eliminar*/
```

Función eliminar

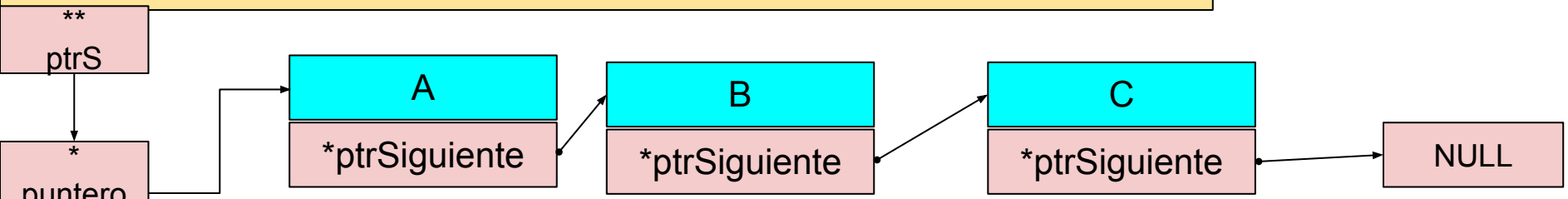
valor

B

*
ptrAnterior

*
ptrActual

*
tempPtr



```
La lista es:
A --> B --> C --> NULL
```

Función eliminar

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
char eliminar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrAnterior  ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrActual    ;/*puntero a lista enlazada nodo actual*/
      ptrNodoLista tempPtr;

      if(valor == (*ptrS) -> dato){
            tempPtr = *ptrS;
            *ptrS = (* ptrS) -> ptrSiguiente; /*desata el nodo*/
            free(tempPtr);    /*libera el nodo*/
            return valor;
      } else{
            ptrAnterior = *ptrS;
            ptrActual = (*ptrS) -> ptrSiguiente;

            while(ptrAcutal != NULL && ptrActual -> dato != valor ){
                  ptrAnterior = ptrActual;
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
            }/*fin de while*/

            if(ptrActual != NULL){
                  trmpPtr = ptrAcutal:
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
                  free(tempPtr);
                  return valor;
            }/*fin de if*/
      }/*fin de else*/
      return '\0';
}/*fin de función eliminar*/
```

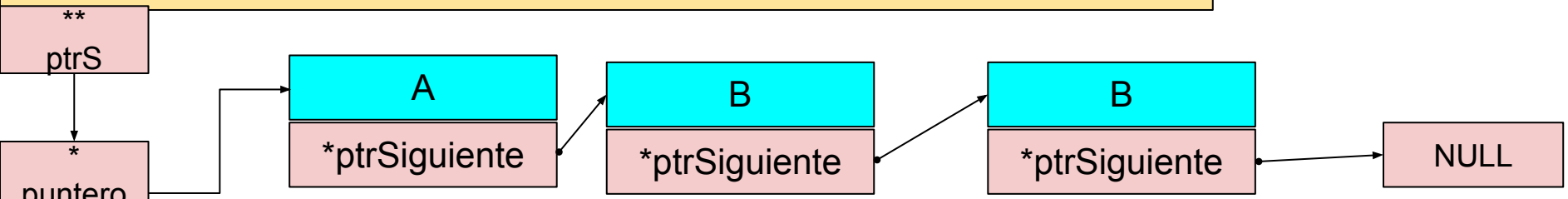
valor

B

* ptrAnterior

* ptrActual

* tempPtr



```
La lista es:
A --> B --> C --> NULL
```

Función eliminar

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
char eliminar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrAnterior  ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrActual    ;/*puntero a lista enlazada nodo actual*/
      ptrNodoLista tempPtr;

      if(valor == (*ptrS) -> dato){
            tempPtr = *ptrS;
            *ptrS = (* ptrS) -> ptrSiguiente; /*desata el nodo*/
            free(tempPtr);    /*libera el nodo*/
            return valor;
      } else{
            ptrAnterior = *ptrS;
            ptrActual = (*ptrS) -> ptrSiguiente;

            while(ptrAcutal != NULL && ptrActual -> dato != valor ){
                  ptrAnterior = ptrActual;
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
            }/*fin de while*/

            if(ptrActual != NULL){
                  trmpPtr = ptrAcutal:
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
                  free(tempPtr);
                  return valor;
            }/*fin de if*/
      }/*fin de else*/
      return '\0';
}/*fin de función eliminar*/
```

valor

B

* ptrAnterior

* ptrActual

* tempPtr

** ptrS

* puntero Inicial

A *ptrSiguiente

B *ptrSiguiente

C *ptrSiguiente

NULL

```
La lista es:
A --> B --> C --> NULL
```

Función eliminar

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
char eliminar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrAnterior  ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrActual    ;/*puntero a lista enlazada nodo actual*/
      ptrNodoLista tempPtr;

      if(valor == (*ptrS) -> dato){
            tempPtr = *ptrS;
            *ptrS = (* ptrS) -> ptrSiguiente; /*desata el nodo*/
            free(tempPtr);    /*libera el nodo*/
            return valor;
      } else{
            ptrAnterior = *ptrS;
            ptrActual = (*ptrS) -> ptrSiguiente;

            while(ptrAcutal != NULL && ptrActual -> dato != valor ){
                  ptrAnterior = ptrActual;
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
            }/*fin de while*/

            if(ptrActual != NULL){
                  trmpPtr = ptrAcutal:
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
                  free(tempPtr);
                  return valor;
            }/*fin de if*/
      }/*fin de else*/
      return '\0';
}/*fin de función eliminar*/
```

valor

B

*
ptrAnterior

*
ptrActual

*
tempPtr

**
ptrS

A
*ptrSiguiente

B
*ptrSiguiente

C
*ptrSiguiente

NULL

```
La lista es:
A --> B --> C --> NULL
```

Función eliminar

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
char eliminar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrAnterior  ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrActual    ;/*puntero a lista enlazada nodo actual*/
      ptrNodoLista tempPtr;

      if(valor == (*ptrS) -> dato){
            tempPtr = *ptrS;
            *ptrS = (* ptrS) -> ptrSiguiente; /*desata el nodo*/
            free(tempPtr);    /*libera el nodo*/
            return valor;
      } else{
            ptrAnterior = *ptrS;
            ptrActual = (*ptrS) -> ptrSiguiente;

            while(ptrAcutal != NULL && ptrActual -> dato != valor ){
                  ptrAnterior = ptrActual;
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
            }/*fin de while*/

            if(ptrActual != NULL){
                  tempPtr = ptrAcutal:
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
                  free(tempPtr);
                  return valor;
            }/*fin de if*/
      }/*fin de else*/
      return '\0';
}/*fin de función eliminar*/
```

valor

B

*
ptrAnterior

*
ptrActual

*
tempPtr

**
ptrS

*
puntero
Inicial

A
*ptrSiguiente

B
*ptrSiguiente

C
*ptrSiguiente

NULL

```
La lista es:
A --> B --> C --> NULL
```

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
char eliminar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrAnterior  ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrActual    ;/*puntero a lista enlazada nodo actual*/
      ptrNodoLista tempPtr;

      if(valor == (*ptrS) -> dato){
            tempPtr = *ptrS;
            *ptrS = (* ptrS) -> ptrSiguiente; /*desata el nodo*/
            free(tempPtr);    /*libera el nodo*/
            return valor;
      } else{
            ptrAnterior = *ptrS;
            ptrActual = (*ptrS) -> ptrSiguiente;

            while(ptrAcutal != NULL && ptrActual -> dato != valor ){
                  ptrAnterior = ptrActual;
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
            }/*fin de while*/

            if(ptrActual != NULL){
                  tempPtr = ptrAcutal:
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
                  free(tempPtr);
                  return valor;
            }/*fin de if*/
      }/*fin de else*/
      return '\0';
}/*fin de función eliminar*/
```

Función eliminar

valor

B

*
ptrAnterior

*
ptrActual

*
tempPtr

**
ptrS

*
puntero
Inicial

A
*ptrSiguiente

B
*ptrSiguiente

C
*ptrSiguiente

NULL

```
La lista es:
A --> B --> C --> NULL
```

Función eliminar

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
char eliminar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrAnterior  ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrActual    ;/*puntero a lista enlazada nodo actual*/
      ptrNodoLista tempPtr;

      if(valor == (*ptrS) -> dato){
            tempPtr = *ptrS;
            *ptrS = (* ptrS) -> ptrSiguiente; /*desata el nodo*/
            free(tempPtr);    /*libera el nodo*/
            return valor;
      } else{
            ptrAnterior = *ptrS;
            ptrActual = (*ptrS) -> ptrSiguiente;

            while(ptrAcutal != NULL && ptrActual -> dato != valor ){
                  ptrAnterior = ptrActual;
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
            }/*fin de while*/

            if(ptrActual != NULL){
                  tempPtr = ptrAcutal:
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
                  free(tempPtr);
                  return valor;
            }/*fin de if*/
      }/*fin de else*/
      return '\0';
}/*fin de función eliminar*/
```

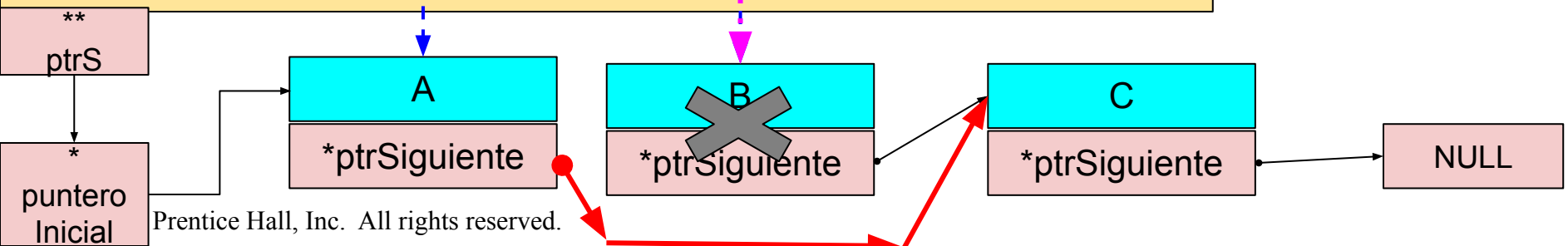
valor

B

* ptrAnterior

* ptrActual

* tempPtr



```
La lista es:
A --> C --> NULL
```

```
/* Fig. 12.3: fig12_03.c => "lista_anlazada.c"
Operación y mantenimiento de una lista */
char eliminar(ptrNodoLista *ptrS, char valor){
      ptrNodoLista ptrAnterior  ;/*puntero a lista enlazada nodo anterio*/
      ptrNodoLista ptrActual    ;/*puntero a lista enlazada nodo actual*/
      ptrNodoLista tempPtr;

      if(valor == (*ptrS) -> dato){
            tempPtr = *ptrS;
            *ptrS = (* ptrS) -> ptrSiguiente; /*desata el nodo*/
            free(tempPtr);    /*libera el nodo*/
            return valor;
      } else{
            ptrAnterior = *ptrS;
            ptrActual = (*ptrS) -> ptrSiguiente;

            while(ptrAcutal != NULL && ptrActual -> dato != valor ){
                  ptrAnterior = ptrActual;
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
            }/*fin de while*/

            if(ptrActual != NULL){
                  tempPtr = ptrAcutal:
                  ptrAnterior -> ptrSiguiente = ptrActual -> ptrSiguiente;
                  free(tempPtr);
                  return valor;
            }/*fin de if*/
      }/*fin de else*/
      return '\0';
}/*fin de función eliminar*/
```

Función eliminar

valor

B

*
ptrAnterior

*
ptrActual

*
tempPtr

NULL

**
ptrS

*
puntero
Inicial

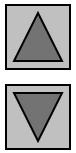
A

*ptrSiguiente

C

*ptrSiguiente

NULL

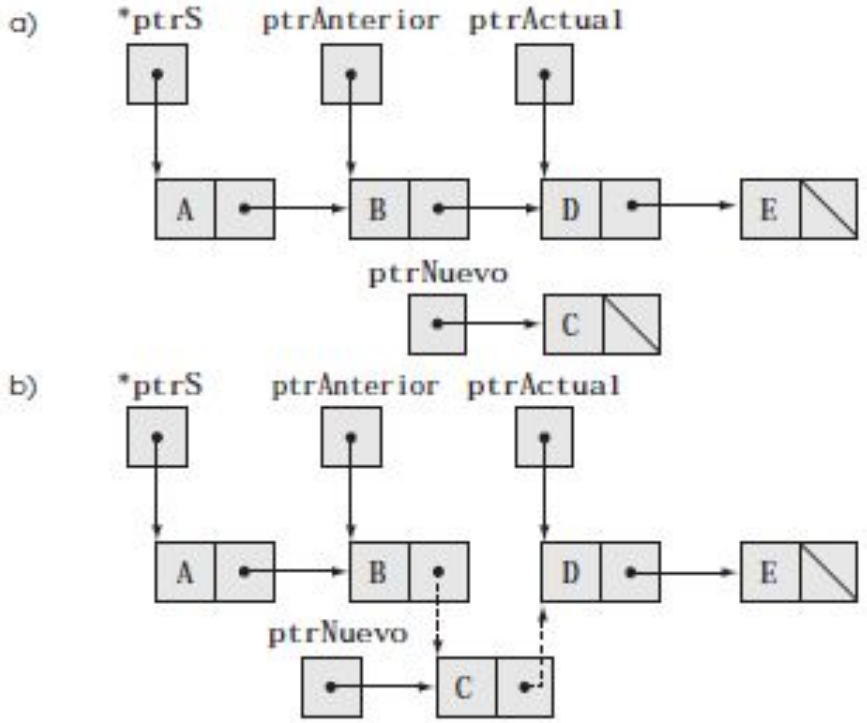


Outline

Salida del Programa

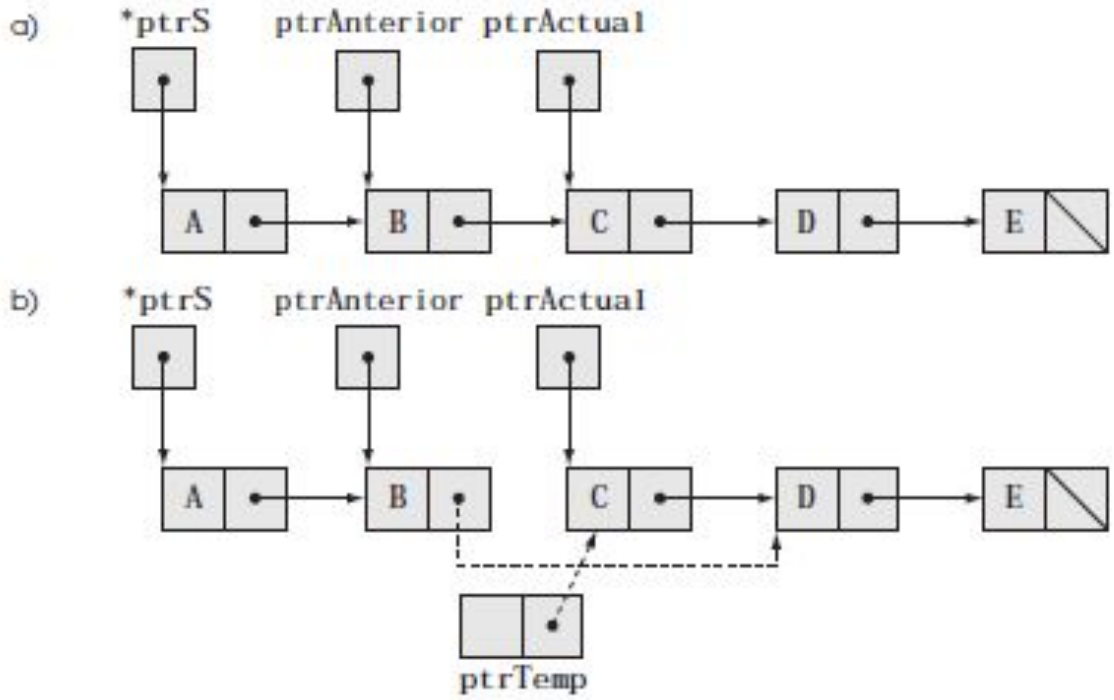
```
Introduzca su eleccion:
1 para insertar un elemento en la lista.
2 para eliminar un elemento de la lista.
3 para terminar.
? 1
Introduzca un caracter: B
La lista es:
B --> NULL
? 1
Introduzca un caracter: A
La lista es:
A --> B --> NULL
? 1
Introduzca un caracter: C
La lista es:
A --> B --> C --> NULL
? 2
Introduzca un caracter para eliminar: D
no se encuentra el caracter D.
? 2
Introduzca un caracter para eliminar: B
caracter B eliminado.
La lista es:
A --> C --> NULL
? 2
Introduzca un caracter para eliminar: C
caracter C eliminado.
La lista es:
A --> NULL
```

12.4 Lista Enlazada (IV)



Inserción ordenada de un nodo en una lista

12.4 Lista Enlazada (IV)



Eliminación de un nodo de una lista

Cap. 12 – Listas Enlazadas

Aguije!!