Fundamentos de la programación

9A

ANEXO

Punteros y memoria dinámica

Grado en Ingeniería Informática
Grado en Ingeniería del Software
Grado en Ingeniería de Computadores

Luis Hernández Yáñez
Facultad de Informática
Universidad Complutense

Índice

Luis Hernández Yáñez

Aritmética de punteros

Luis Hernández Yáñez

Aritmética de punteros

Operaciones aritméticas con punteros

La aritmética de punteros es una aritmética un tanto especial...

Trabaja tomando como unidad de cálculo el tamaño del tipo base

```
int dias[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
typedef int* tIntPtr;
tIntPtr punt = dias;
```

punt empieza apuntando al primer elemento del array:

```
cout << *punt << endl; // Muestra 31 (primer elemento)
```

```
punt++;
```

punt++ hace que punt pase a apuntar al siguiente elemento

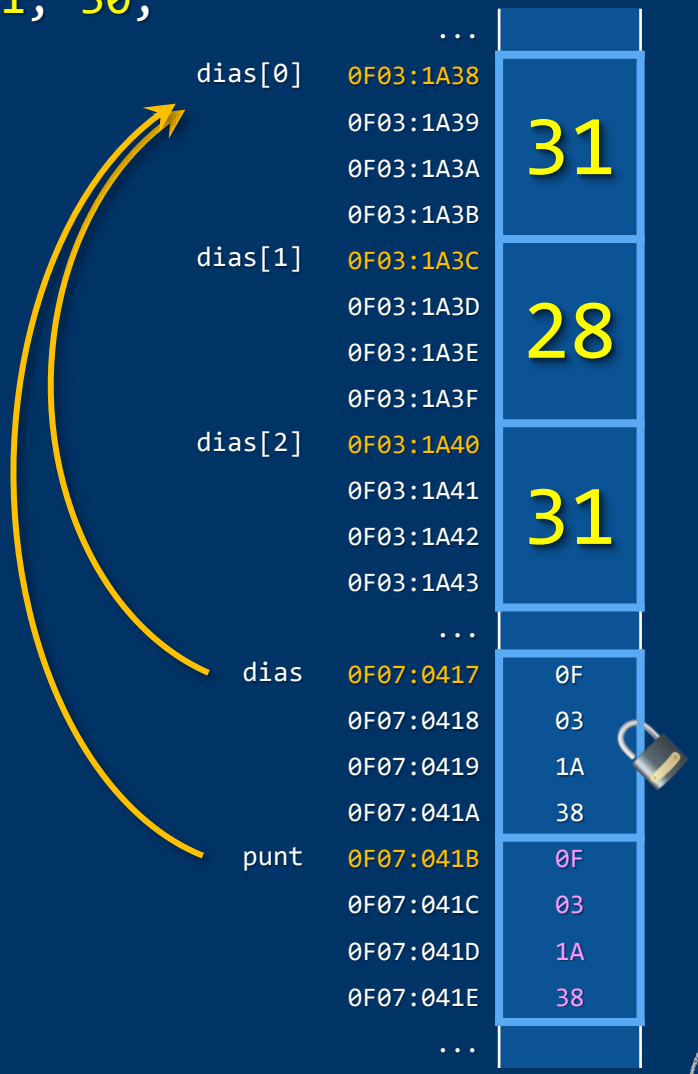
```
cout << *punt << endl; // Muestra 28 (segundo elemento)
```

A la dirección de memoria actual se le suman tantas unidades como bytes (4) ocupe en memoria un dato de ese tipo (int)

Luis Hernández Yáñez

Aritmética de punteros

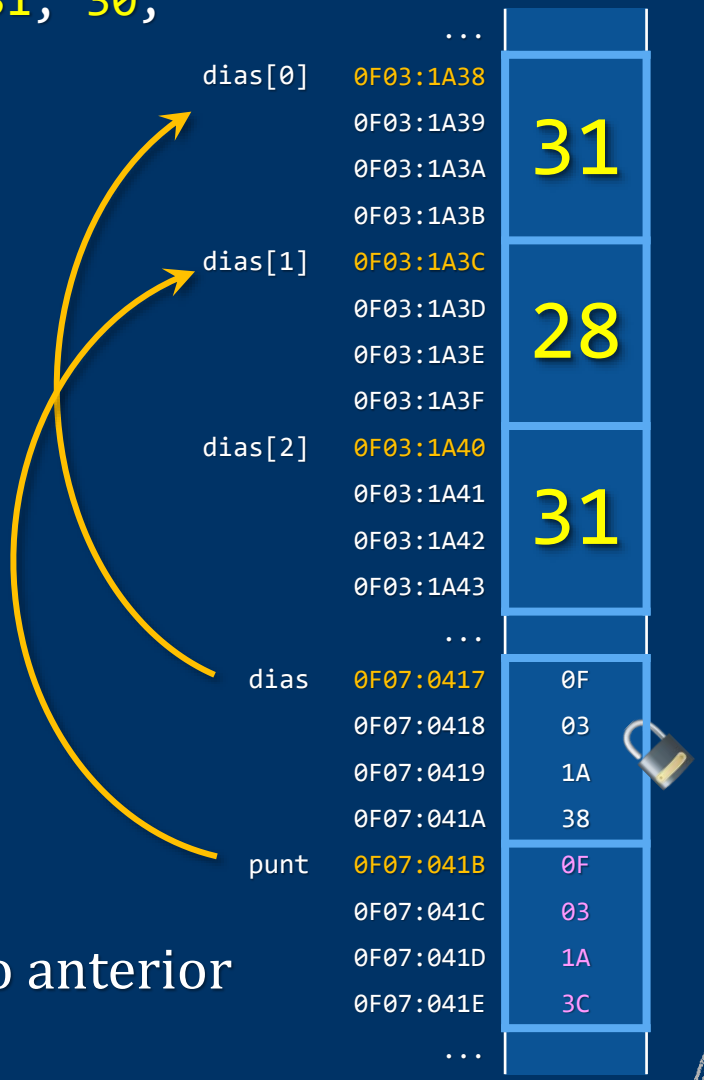
```
int dias[12] = { 31, 28, 31, 30, 31, 30,
   31, 31, 30, 31, 30, 31 };
typedef int* tIntPtr;
tIntPtr punt = dias;
```



Luis Hernández Yáñez

Aritmética de punteros

```
int dias[12] = { 31, 28, 31, 30, 31, 30,
   31, 31, 30, 31, 30, 31 };
typedef int* tIntPtr;
tIntPtr punt = dias;
punt++;
```

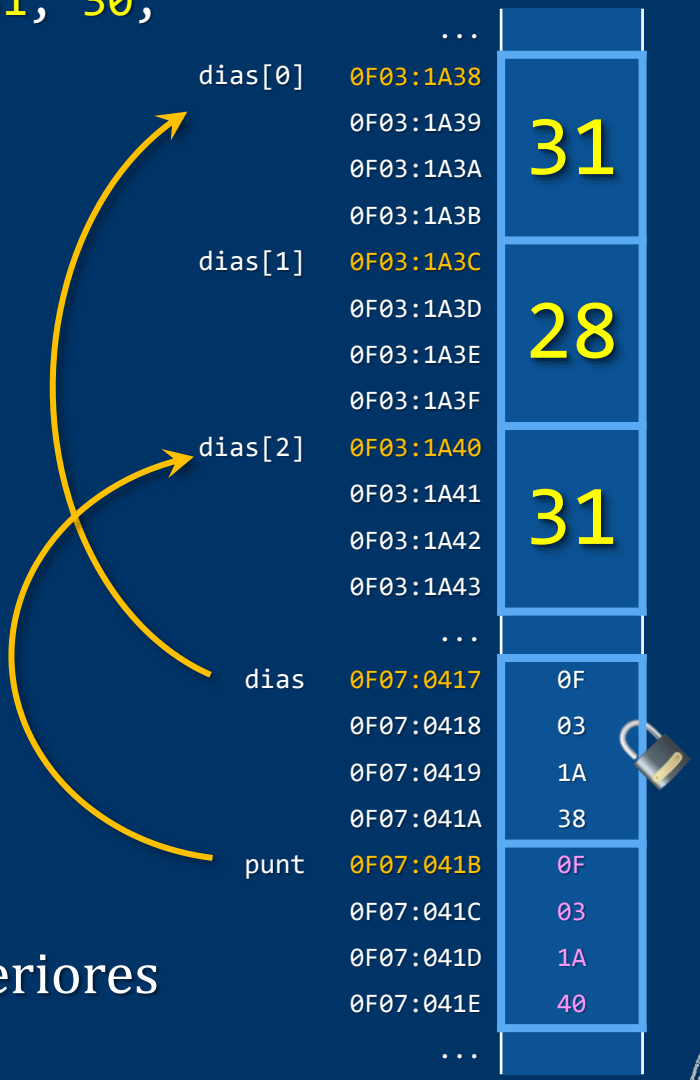


`punt--` hace que apunte al elemento anterior

Luis Hernández Yáñez

Aritmética de punteros

```
int dias[12] = { 31, 28, 31, 30, 31, 30,
   31, 31, 30, 31, 30, 31 };
typedef int* tIntPtr;
tIntPtr punt = dias;
punt = punt + 2;
```

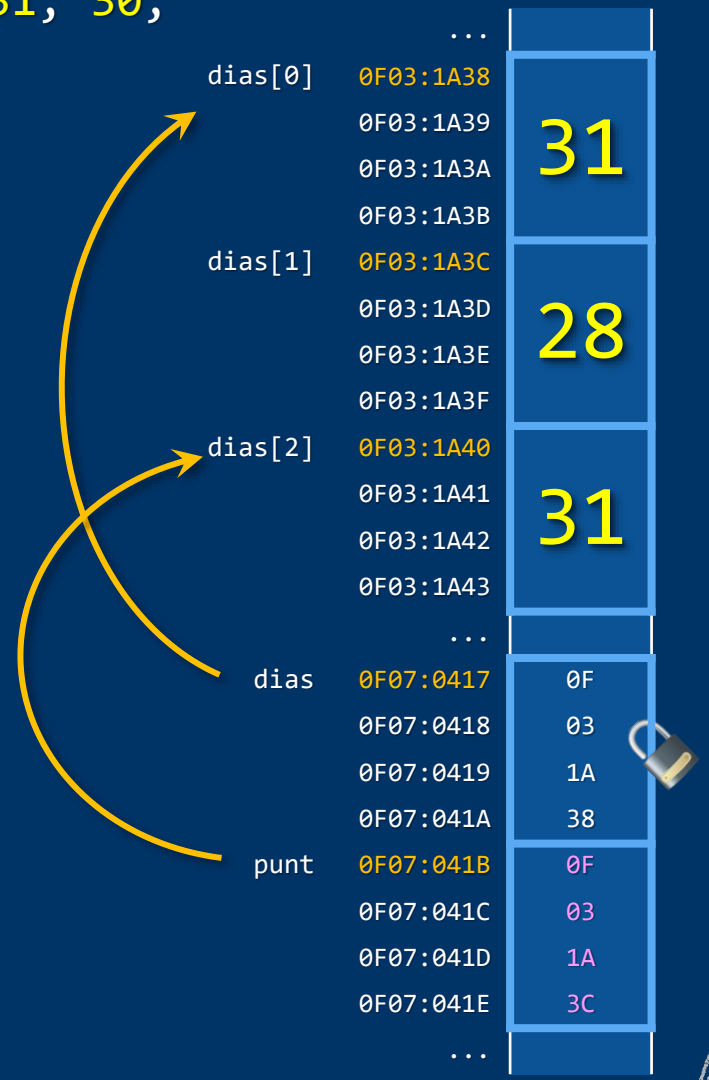


Restando pasamos a elementos anteriores

Aritmética de punteros

```
int dias[12] = { 31, 28, 31, 30, 31, 30,
   31, 31, 30, 31, 30, 31 };
typedef int* tIntPtr;
tIntPtr punt = dias;
punt = punt + 2;

int num = punt - dias;
```

Nº de elementos entre los punteros



Luis Hernández Yáñez

Aritmética de punteros

Otro tipo base

```
short int (2 bytes)
short int dias[12] = {31, 28, 31, 30,
   31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
tSIPtr punt = dias;
```

Luis Hernández Yáñez

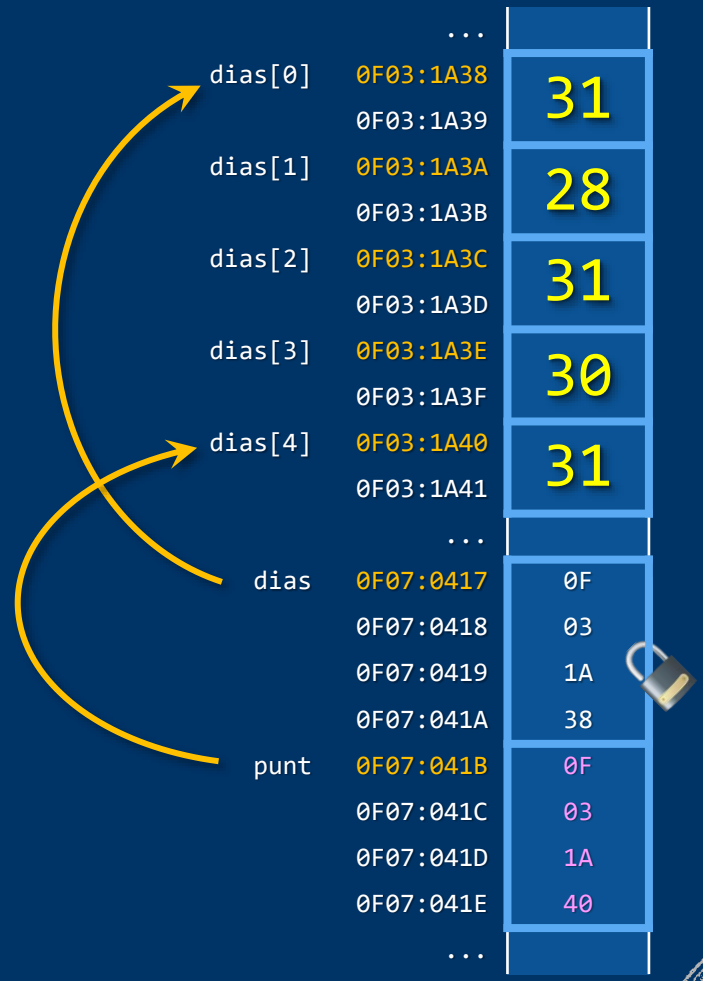
Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,
   31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
tSIPtr punt = dias;
punt++;
```

Luis Hernández Yáñez

Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,
   31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
tSIPtr punt = dias;
punt++;
punt = punt + 3;
```



Luis Hernández Yáñez

Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,
   31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
tSIPtr punt = dias;
punt++;
punt = punt + 3;
punt--;
```

Luis Hernández Yáñez

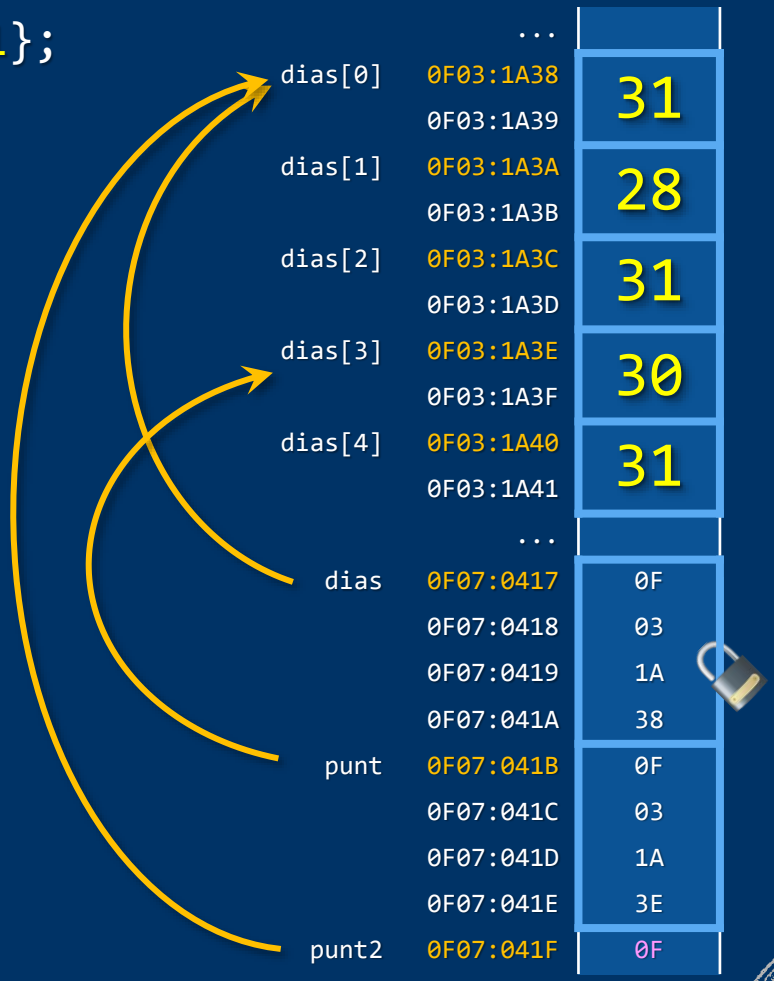
Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,
   31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
tSIPtr punt = dias;
punt++;
punt = punt + 3;
punt--;
tSIPtr punt2;
```

Luis Hernández Yáñez

Aritmética de punteros

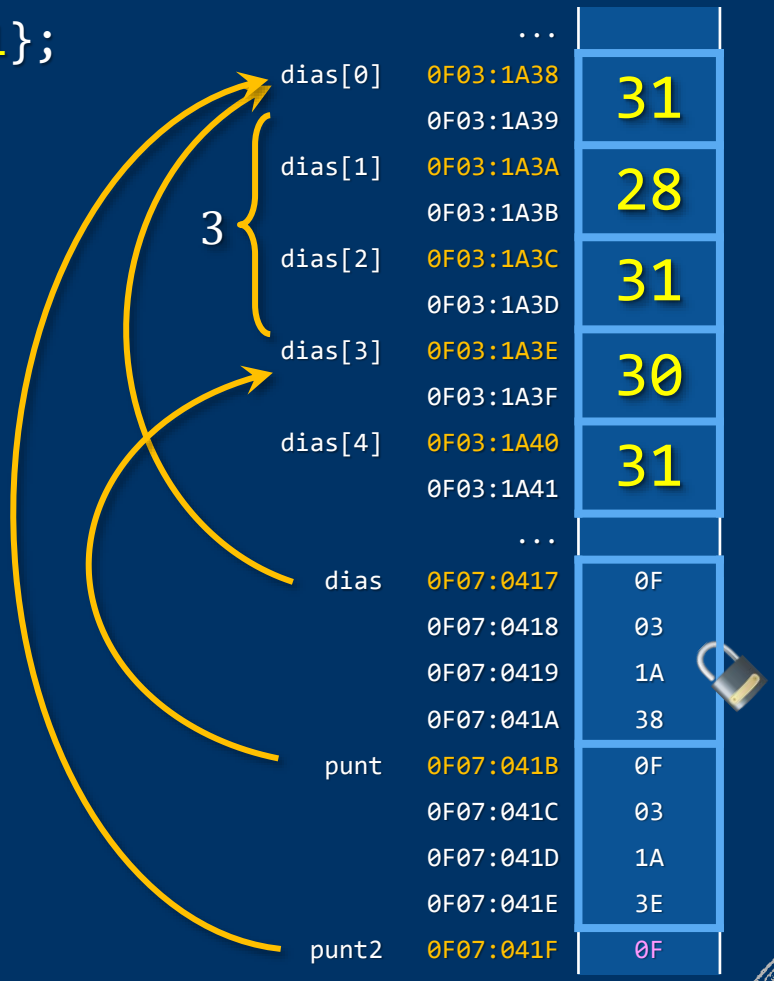
```
short int dias[12] = {31, 28, 31, 30,
   31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
siPtr punt = dias;
punt++;
punt = punt + 3;
punt--;
tSIPtr punt2;
punt2 = dias;
```



Luis Hernández Yáñez

Aritmética de punteros

```
short int dias[12] = {31, 28, 31, 30,
   31, 30, 31, 31, 30, 31, 30, 31};
typedef short int* tSIPtr;
siPtr punt = dias;
punt++;
punt = punt + 3;
punt--;
tSIPtr punt2;
punt2 = dias;
cout << punt - punt2; // 3
```



Luis Hernández Yáñez

Recorrido de arrays con punteros

Luis Hernández Yáñez

Recorrido de arrays con punteros

arraypunt.cpp

Punteros como iteradores para arrays

```
const int MAX = 100;
typedef int tArray[MAX];
typedef struct {
   tArray elementos;
   int cont;
} tLista;
typedef int* tIntPtr;
tLista lista;
```

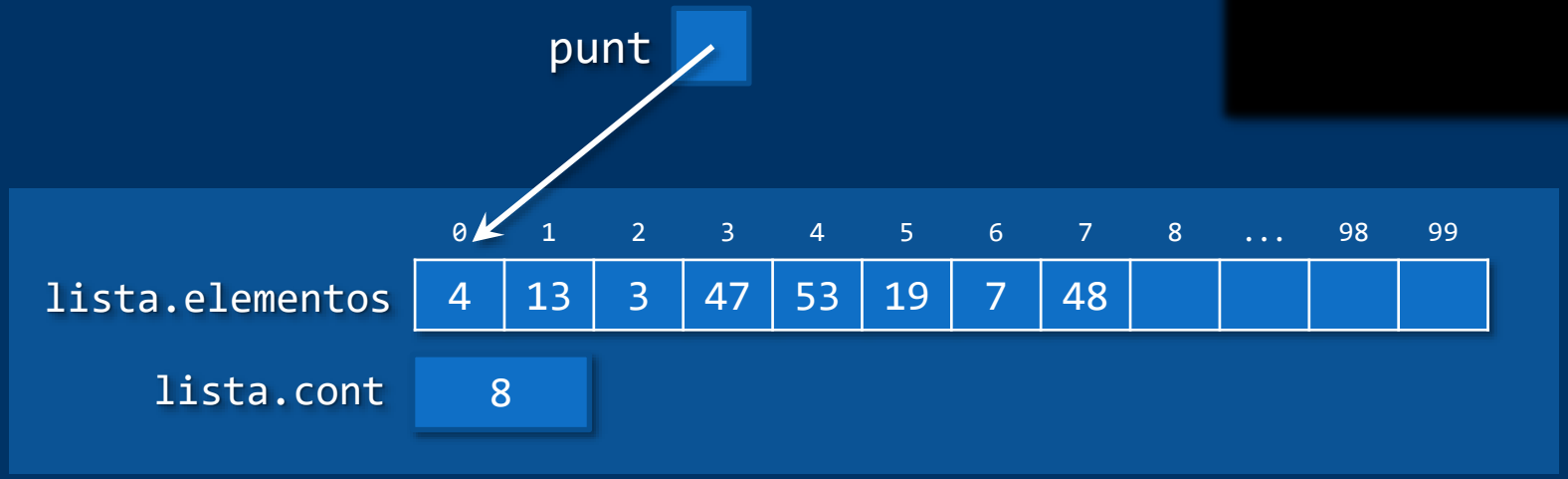
Usamos un puntero como *iterador* para recorrer el array:

```
tIntPtr punt = lista.elementos;
for (int i = 0; i < lista.cont; i++) {
   cout << *punt << endl;
   punt++;
}
```

Luis Hernández Yáñez

Punteros como iteradores para arrays

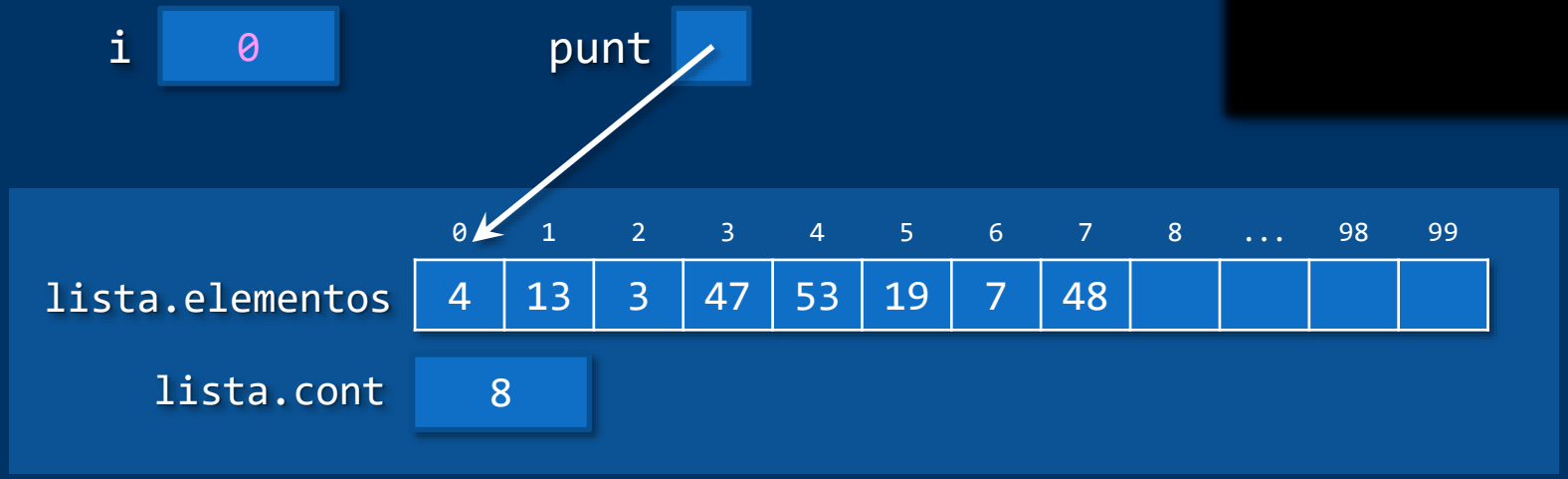
```
...
intPtr punt = lista.elementos;
```



Luis Hernández Yáñez

Punteros como iteradores para arrays

```
...
for (int i = 0; i < lista.cont; i++) {
    cout << *punt << endl;
    punt++;
}
```

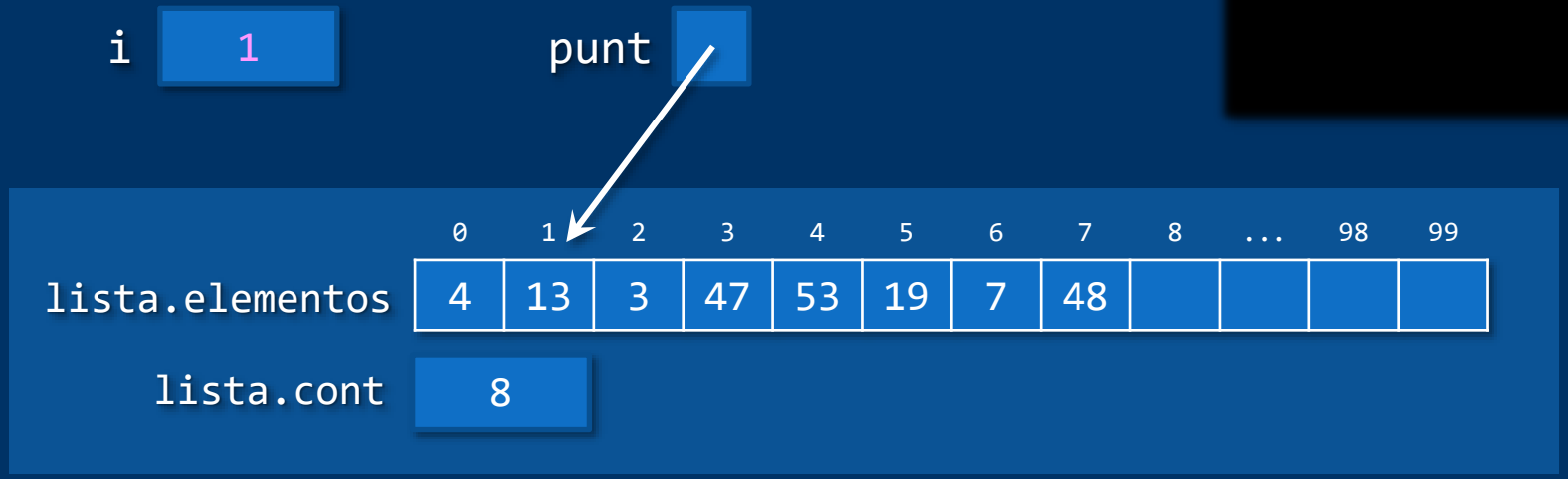


Luis Hernández Yáñez

Punteros como iteradores para arrays

```
...
for (int i = 0; i < lista.cont; i++) {
    cout << *punt << endl;
    punt++;
}
```

```
4
```

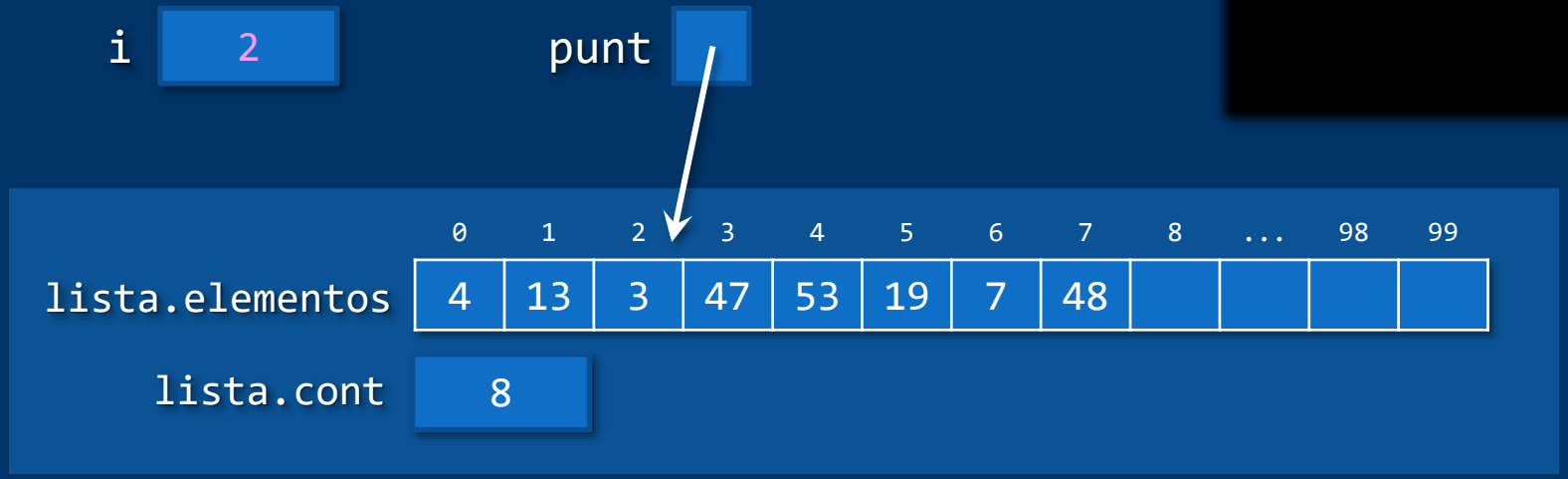


Luis Hernández Yáñez

Punteros como iteradores para arrays

```
...
for (int i = 0; i < lista.cont; i++) {
    cout << *punt << endl;
    punt++;
}
```

```
4
13
```

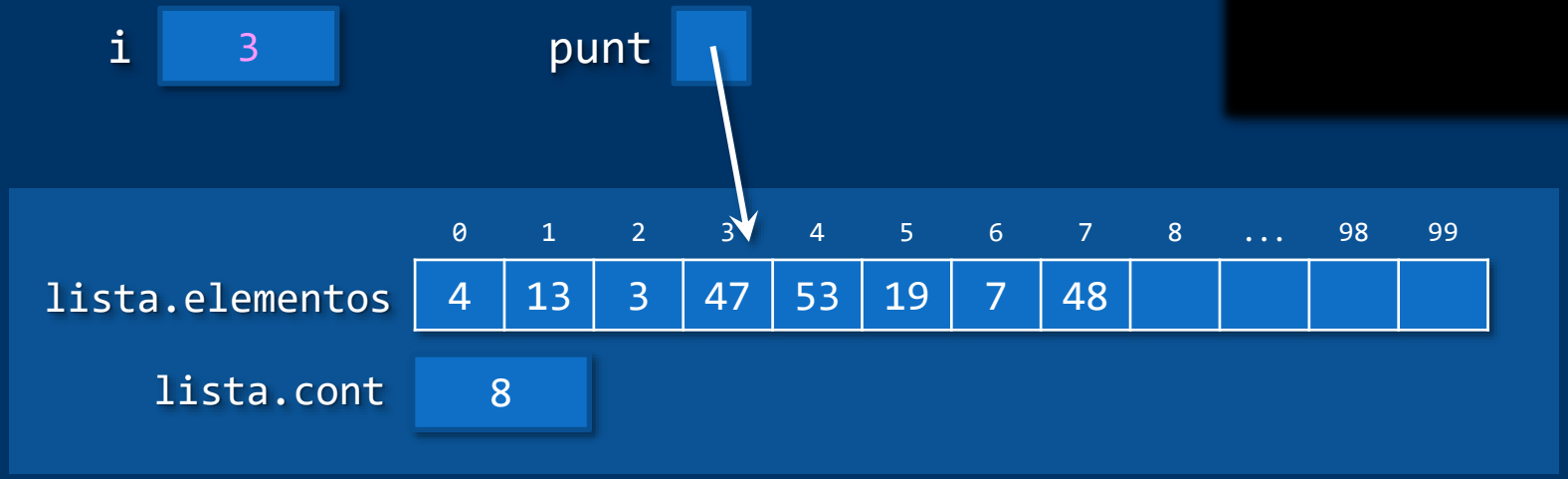


Luis Hernández Yáñez

Punteros como iteradores para arrays

```
...
for (int i = 0; i < lista.cont; i++) {
    cout << *punt << endl;
    punt++;
}
```

```
4
13
3
```



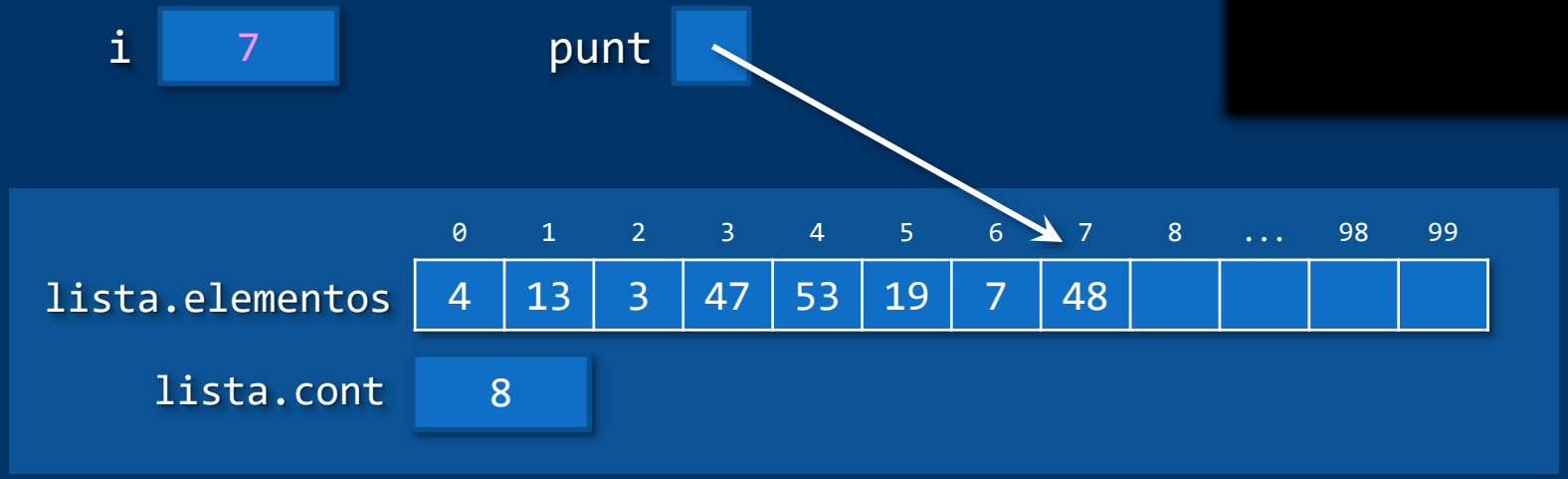
...

Luis Hernández Yáñez

Punteros como iteradores para arrays

```
...
for (int i = 0; i < lista.cont; i++) {
    cout << *punt << endl;
    punt++;
}
```

```
4
13
3
47
53
19
7
```

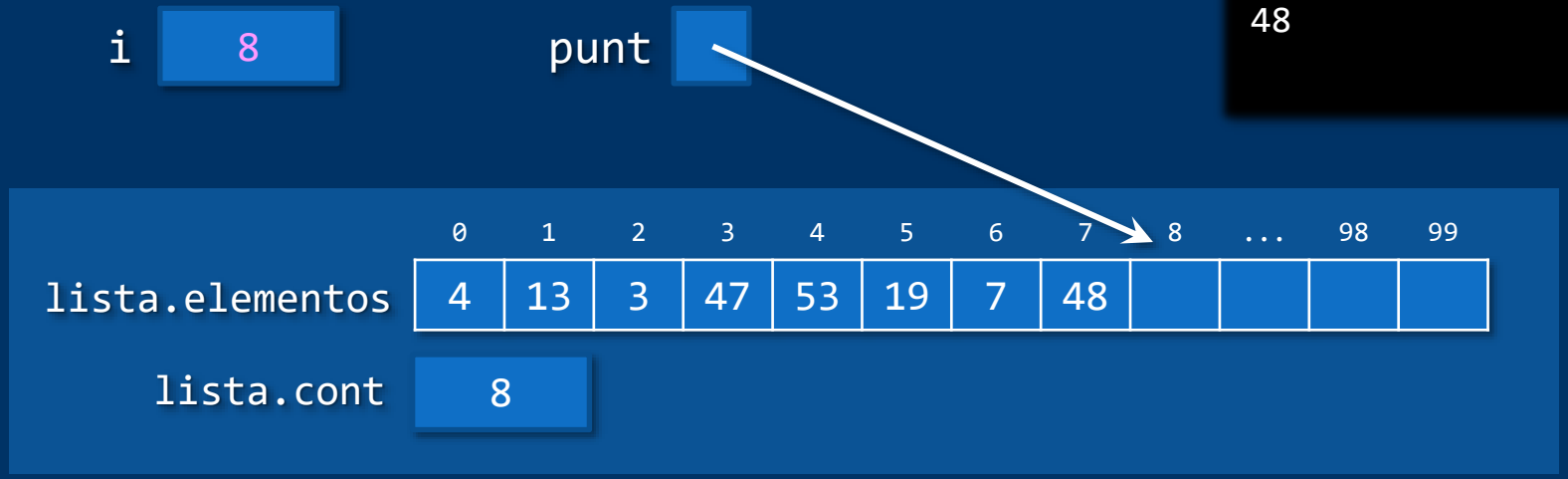


Luis Hernández Yáñez

Punteros como iteradores para arrays

```
...
for (int i = 0; i < lista.cont; i++) {
    cout << *punt << endl;
    punt++;
}
```

```
4
13
3
47
53
19
7
48
```



Luis Hernández Yáñez

Referencias

Luis Hernández Yáñez

Referencias

Nombres alternativos para los datos

Una referencia es una nueva forma de llamar a una variable

Nos permiten referirnos a una variable con otro identificador:

```
int x = 10;
int &z = x;
```

x y z son ahora la misma variable (comparten memoria)

Cualquier cambio en x afecta a z y cualquier cambio en z afecta a x

```
z = 30;
cout << x;
```

Las referencias se usan en el paso de parámetros por referencia

Luis Hernández Yáñez

Listas enlazadas

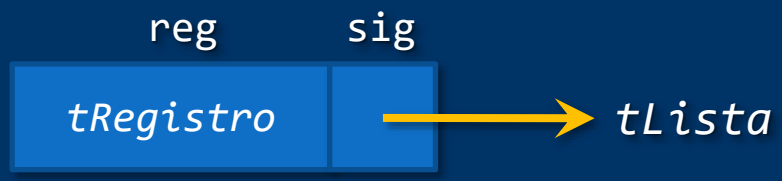
Luis Hernández Yáñez

Listas enlazadas

Una implementación dinámica de listas enlazadas

Cada elemento de la lista apunta al siguiente elemento:

```
struct tNodo; // Declaración anticipada
typedef tNodo *tLista;
struct tNodo {
   tRegistro reg;
   tLista sig;
};
```



Una lista (`tLista`) es un puntero a un nodo

Si el puntero vale `NULL`, no apunta a ningún nodo: lista vacía

Un nodo (`tNodo`) es un elemento seguido de una lista

Lista { *Vacía* / *Elemento seguido de una lista* }

¡Definición recursiva!

Luis Hernández Yáñez

Implementación dinámica de listas enlazadas

Cada elemento de la lista en su nodo

Apuntará al siguiente elemento o a ninguno (NULL)

```
struct tNodo; // Declaración anticipada
typedef tNodo *tLista;
struct tNodo {
   tRegistro reg;
   tLista sig;
};
```

Además, un puntero al primer elemento (nodo) de la lista

```
tLista lista = NULL; // Lista vacía
```

Luis Hernández Yáñez

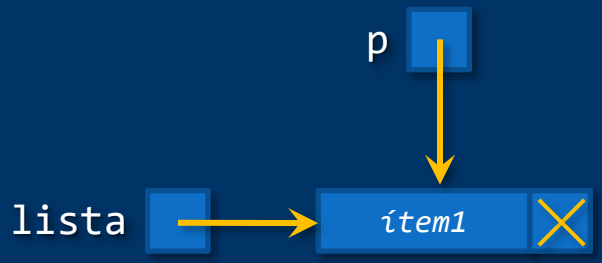
Implementación dinámica de listas enlazadas

```
struct tNodo;
typedef tNodo *tLista;
struct tNodo {
   tRegistro reg;
   tLista sig;
};
tLista lista = NULL; // Lista vacía
lista = new tNodo;
lista->reg = nuevo();
lista->sig = NULL;
```

Luis Hernández Yáñez

Implementación dinámica de listas enlazadas

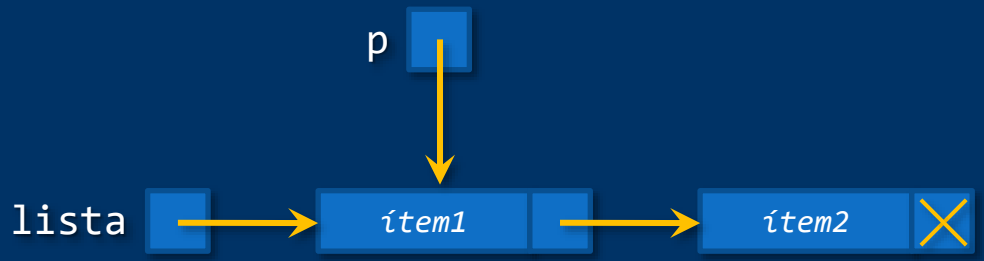
```
tLista lista = NULL; // Lista vacía
lista = new tNodo;
lista->reg = nuevo();
lista->sig = NULL;
tLista p;
p = lista;
```



Luis Hernández Yáñez

Implementación dinámica de listas enlazadas

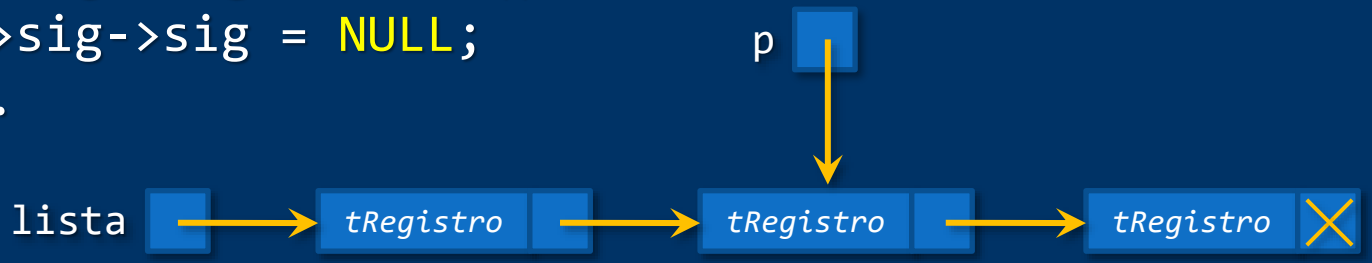
```
tLista lista = NULL; // Lista vacía
lista = new tNodo;
lista->reg = nuevo();
lista->sig = NULL;
tLista p;
p = lista;
p->sig = new tNodo;
p->sig->reg = nuevo();
p->sig->sig = NULL;
```



Luis Hernández Yáñez

Implementación dinámica de listas enlazadas

```
tLista lista = NULL; // Lista vacía
lista = new tNodo;
lista->reg = nuevo();
lista->sig = NULL;
tLista p;
p = lista;
p->sig = new tNodo;
p->sig->reg = nuevo();
p->sig->sig = NULL;
p = p->sig;
p->sig = new tNodo;
p->sig->reg = nuevo();
p->sig->sig = NULL;
...
```



Luis Hernández Yáñez

Implementación dinámica de listas enlazadas

Usamos la memoria que necesitamos, ni más ni menos

Tantos elementos, tantos nodos hay en la lista

¡Pero perdemos el acceso directo!

Algunas operaciones de la lista se complican y otras no

A continuación tienes el módulo de lista implementado como lista enlazada...

Luis Hernández Yáñez

Ejemplo de lista enlazada

listaenlazada.h

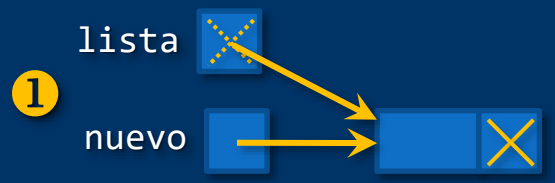
```
struct tNodo;
typedef tNodo *tLista;
struct tNodo {
   tRegistro reg;
   tLista sig;
};

const string BD = "bd.txt";

void mostrar(tLista lista);
void insertar(tLista &lista, tRegistro registro, bool &ok);
void eliminar(tLista &lista, int code, bool &ok);
tLista buscar(tLista lista, int code); // Devuelve puntero
void cargar(tLista &lista, bool &ok);
void guardar(tLista lista);
void destruir(tLista &lista); // Liberar la memoria dinámica
```

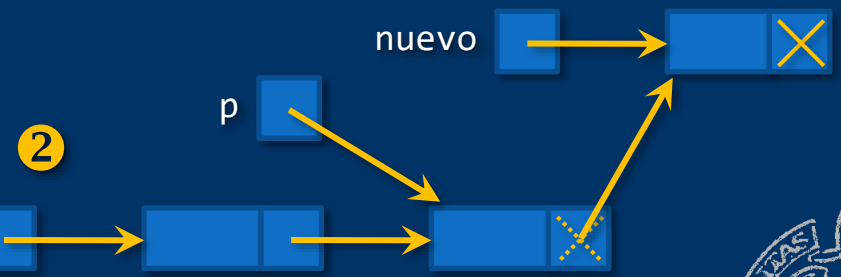
Luis Hernández Yáñez

Ejemplo de lista enlazada

listaenlazada.cpp

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {
   ok = true;
   tLista nuevo = new tNodo;
   if (nuevo == NULL) {
      ok = false; // No hay más memoria dinámica
   }
   else {
      nuevo->reg = registro;
      nuevo->sig = NULL;
      if (lista == NULL) { // Lista vacía
1        lista = nuevo;
      }
      else {
         tLista p = lista;
         // Localizamos el último nodo...
         while (p->sig != NULL) {
2           p = p->sig;
         }
         p->sig = nuevo;
      }
   }
} ...
```

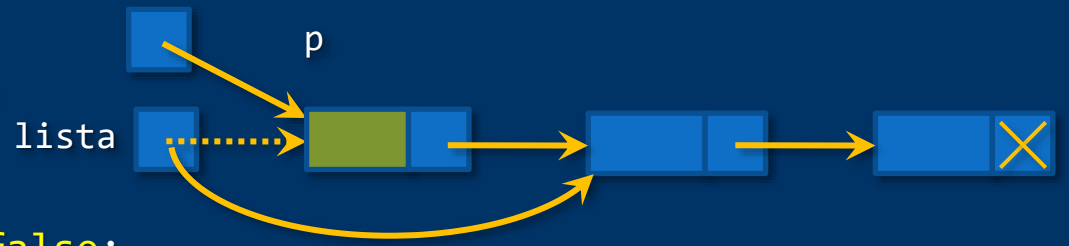




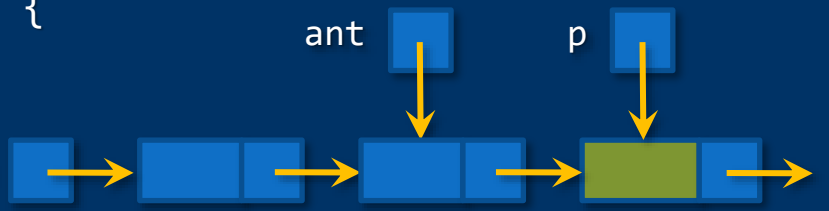
Ejemplo de lista enlazada

```
void eliminar(tLista &lista, int code, bool &ok) {
   ok = true;
   tLista p = lista;
   if (p == NULL) {
      ok = false; // Lista vacía
   }
   else if (p->reg.codigo == code) { // El primero
      lista = p->sig;
❶     delete p;
   }
   else {
      tLista ant = p;
      p = p->sig;
      bool encontrado = false;
      while ((p != NULL) && !encontrado) {
         if (p->reg.codigo == code) {
            encontrado = true;
         }
❷        else {
            ant = p;
            p = p->sig;
         }
      } ...
```

❶



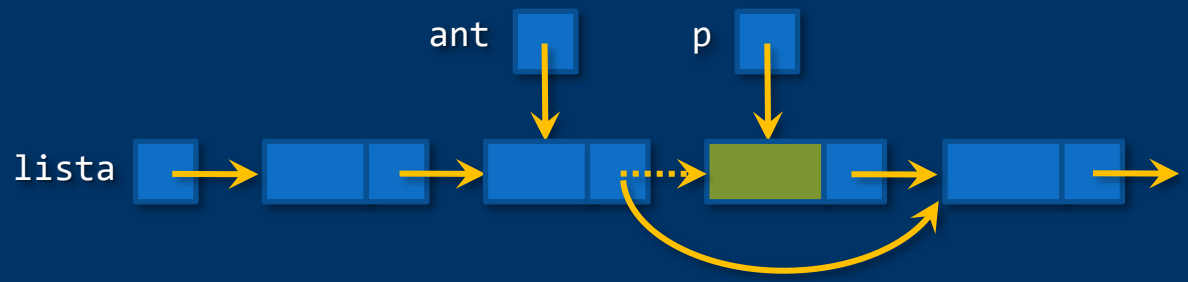
❷



Luis Hernández Yáñez

Ejemplo de lista enlazada

```
        if (!encontrado) {
            ok = false; // No existe ese código
        }
        else {
            ant->sig = p->sig;
            delete p;
        }
    }
}
...
```



Luis Hernández Yáñez

Ejemplo de lista enlazada

```
tLista buscar(tLista lista, int code) {
// Devuelve un puntero al nodo, o NULL si no se encuentra
   tLista p = lista;
   bool encontrado = false;
   while ((p != NULL) && !encontrado) {
      if (p->reg.codigo == code) {
         encontrado = true;
      }
      else {
         p = p->sig;
      }
   }
   return p;
}

void mostrar(tLista lista) {
   cout << endl << "Elementos de la lista:" << endl
        << "----------------------" << endl;
   tLista p = lista;
   while (p != NULL) {
      mostrar(p->reg);
      p = p->sig;
   }
} ...
```

Luis Hernández Yáñez

Ejemplo de lista enlazada

```
void cargar(tLista &lista, bool &ok) {
   ifstream archivo;
   char aux;
   ok = true;
   lista = NULL;
   archivo.open(BD.c_str());
   if (!archivo.is_open()) {
      ok = false;
   }
   else {
      tRegistro registro;
      tLista ult = NULL;
      archivo >> registro.codigo;
      while (registro.codigo != -1) {
         archivo >> registro.valor;
         archivo.get(aux); // Saltamos el espacio
         getline(archivo, registro.nombre);
         ...
```

Luis Hernández Yáñez

Ejemplo de lista enlazada

```
            if (lista == NULL) {
               lista = new tNodo;
               ult = lista;
            }
            else {
               ult->sig = new tNodo;
               ult = ult->sig;
            }
            ult->reg = registro;
            ult->sig = NULL;
            archivo >> registro.codigo;
         }
         archivo.close();
      }
      return ok;
   } ...
```

Luis Hernández Yáñez

Ejemplo de lista enlazada

```
void guardar(tLista lista) {
   ofstream archivo;
   archivo.open(BD);
   tLista p = lista;
   while (p != NULL) {
      archivo << p->registro.codigo << " ";
      archivo << p->registro.valor << " ";
      archivo << p->registro.nombre << endl;
      p = p->sig;
   }
   archivo.close();
}

void destruir(tLista &lista) {
   tLista p;
   while (lista != NULL) {
      p = lista;
      lista = lista->sig;
      delete p;
   }
}
```

Luis Hernández Yáñez

Acerca de *Creative Commons*

Licencia CC (Creative Commons)

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:

- Reconocimiento (*Attribution*):
  En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.
- No comercial (*Non commercial*):
  La explotación de la obra queda limitada a usos no comerciales.
- Compartir igual (*Share alike*):
  La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.

Luis Hernández Yáñez