

Laboratorio de Arquitectura de Redes

Punteros en lenguaje C

Punteros en lenguaje C

- Definición
- Declaración e inicialización de punteros
- Operadores de punteros: «*» y «&»
- Operaciones con punteros
 - Operaciones de asignación
 - Artimética de punteros
- Tipos de punteros
 - Genéricos
 - Nulos
 - Constantes
- Relación entre punteros y arrays
 - Puntero a un array
 - Arrays de punteros
- Puntero a un puntero
- Punteros a estructuras y uniones

Definición de puntero (I)

- Un **puntero** es una *variable* que contiene la *dirección de memoria* de otra *variable*
 - Por eso se dice que un puntero *apunta* a una variable
 - Se trata de una *indirección*: se puede acceder a la variable *indirectamente*
- Los punteros constituyen una importante herramienta en lenguaje C
 - Proporcionan acceso rápido y eficiente a los arrays
 - Facilitan el trabajo con listas enlazadas
 - Facilitan el intercambio de información entre funciones
 - Son imprescindibles para
 - Asignar memoria dinámicamente
 - El trabajo con archivos
- En el manejo de los punteros hay que extremar las precauciones para evitar errores de muy difícil localización

Definición de puntero (II)

- ❑ Variables de tipo int: `dato1` y `dato2`
- ❑ Variables puntero a entero: `puntdat1` y `pundat2`

Memoria de datos		
Etiqueta	Dirección	Contenido
<code>dato1</code>	<code>0x0F00</code>	1234
<code>dato2</code>	<code>0x0F04</code>	6789
<code>puntdat1</code>	<code>0x0F08</code>	<code>0F00</code>
<code>pundat2</code>	<code>0x0F0C</code>	<code>0F04</code>

Declaración e inicialización de punteros (I)

- La declaración de una variable puntero *asigna la memoria necesaria* para almacenar una dirección

```
tipodato *nombrepuntero;
```

- `tipodato` es el tipo de dato de la variable a la que apuntará el puntero.
 - `nombrepuntero` es la etiqueta de la zona de memoria que contendrá la dirección de una variable
 - `*nombrepuntero` hace referencia al contenido de la variable apuntada por el puntero
 - No se crea ni se reserva memoria para la variable a la que va a apuntar
- El tamaño de la memoria necesaria para almacenar una dirección es siempre el mismo, independientemente del tipo de dato al que corresponda esa dirección

Declaración e inicialización de punteros (II)

- **Inicializar** un puntero consiste en hacerle apuntar a una variable válida
 - La variable debe existir antes de la inicialización
 - Que la variable exista no significa que deba contener un dato válido

```
tipodato *punt;      /* Declaración del
                    puntero*/
tipodato var;        /* Declaración de la
                    variable */
punt = &var;         /* Inicialización
                    del puntero
                    /* La variable var no contiene todavía
                    ningún dato válido */
```

Declaración e inicialización de punteros (III)

□ Ejemplo

```
float *punt; /* Puntero */
float var; /* Variable */
/* La variable y el puntero deben
ser el mismo tipo de dato */
punt = &var; /* Inicialización
del puntero */
*punt = 7.98; /* Inicialización de
la variable. Es igual que
escribir var = 7.98: */
```

- Una declaración de la forma `int *punt=25;` hace que el puntero apunte a la dirección 25 que no sabemos qué contiene

Operadores de punteros:

«*» y «&»

- Los operadores de punteros son «*» y «&»
 - Se asocian de derecha a izquierda
 - Tienen mayor precedencia que todas las operaciones aritméticas y lógicas
- El operador «&» devuelve la dirección de memoria de su operando
 - Aplicado a una variable hace referencia a la *dirección de esa variable*
 - Sólo puede aplicarse a identificadores de variables y elementos de arrays, no es válida sobre expresiones
- El operador «*» devuelve el dato almacenado en la dirección representada por el identificador sobre el que actúa
 - Sólo tiene sentido sobre variables que contienen una dirección de memoria válida (punteros)
 - Aplicado a una variable puntero permite utilizar esa expresión como una variable cualquiera, sin limitaciones

Operaciones con punteros (I)

- Las operaciones con punteros son operaciones con direcciones de memoria
 - Sólo tienen sentido la suma, la resta, incrementos y decrementos
- Operaciones de asignación
 - Es posible asignar el contenido de un puntero a otro puntero
 - Lo que se consigue es hacer que ambos punteros apunten a la misma posición de memoria
 - Deben ser del mismo tipo
 - Ejemplo

```
int dato, *punt1, *punt2; /* Declaraciones */
punt1 = &dato;          /* Inicialización de punt1 */
punt2 = punt1;          /* asignación del contenido de
                          punt1 a punt2. Ahora punt2
                          apunta a la variable dato */
```

Operaciones con punteros (II)

- Aritmética con punteros:
 - La *suma* de un número entero n a un puntero hace que este apunte al *enésimo* elemento del mismo tipo a partir del originalmente apuntado.
 - La *resta* de un número entero n a un puntero hace que este apunte al elemento *n-veces* anterior al que apuntaba
 - Independientemente de que el dato al que se traslada el puntero exista o no.
 - Los incrementos o decrementos de un puntero hacen que este apunte al elemento siguiente o anterior del mismo tipo.
 - En valor numérico absoluto, el incremento o decremento de un puntero cambia su valor un número de unidades que depende del tipo de dato al que apunta
 - La aritmética de punteros sólo coincide con la aritmética ordinaria cuando se opera con punteros que apunten a elementos de tipo byte
 - Son posibles las comparaciones y las operaciones lógicas con punteros siempre que se realicen entre punteros del mismo tipo de dato

Tipos de punteros

- ❑ **Puntero genérico** es aquel que no apunta a ningún tipo de dato

```
void *nombrepuntero;
```

- ❑ Se declaran así para que posteriormente se les pueda hacer a puntar a cualquier tipo de dato.

- ❑ **Puntero nulo** es aquel que no apunta a ningún dato

```
tipodato *nombrepuntero = NULL;
```

- ❑ NULL es una constante definida en `stdio.h`
- ❑ Se utiliza para indicar situaciones de error

- ❑ **Puntero constante** es aquel que se declara como tal y, por tanto, siempre apunta a la misma posición

```
tipodato *const nombrepuntero;
```

- ❑ El contenido, el dato apuntado, si puede cambiar
- ❑ Si es el dato lo que se declara como constante se escribe

```
const tipodato *nombrepuntero;
```
- ❑ Si el puntero y el dato al que apunta se declaran constantes

```
const tipodato *const nombrepuntero;
```

Relación entre punteros y arrays (I)

- En general: todo lo que puede hacerse con arrays puede hacerse también con punteros
 - Las versiones con punteros son más rápidas y más utilizadas
- El identificador de un array (sin índice) es un *puntero constante* al primer elemento del array
- En un array de N elementos, se puede acceder al elemento M (tal que $0 \leq M < N$)
 - Mediante arrays
$$\text{elementoM} = \text{nombreakarray}[M];$$
 - Mediante punteros
$$\text{elementoM} = *(\text{nombreakarray}+M);$$

Relación entre punteros y arrays (I)

- Un **puntero a un array de caracteres** o un **puntero a una cadena** es un puntero a carácter que apunta al primer elemento de la cadena
 - Se puede inicializar en la declaración:

```
char *nombrepuntero = "cadena";
```

 - `nombrepuntero` es un puntero a carácter: contiene la dirección del primer elemento de la cadena
 - Si en el transcurso del programa, se le hace apuntar a otro sitio, ya no será posible volver a apuntar a la posición original
 - `cadena` es una *cadena constante* que se almacena en una tabla de constantes. Finaliza con el carácter nulo `'\0'`
 - Cuando a una función se le pasa como argumento una constante de cadena (cadena de caracteres entre comillas dobles), en realidad se le pasa un puntero al primer elemento de esa cadena

```
char *mensaje = "Error de lectura";  
puts(mensaje);
```

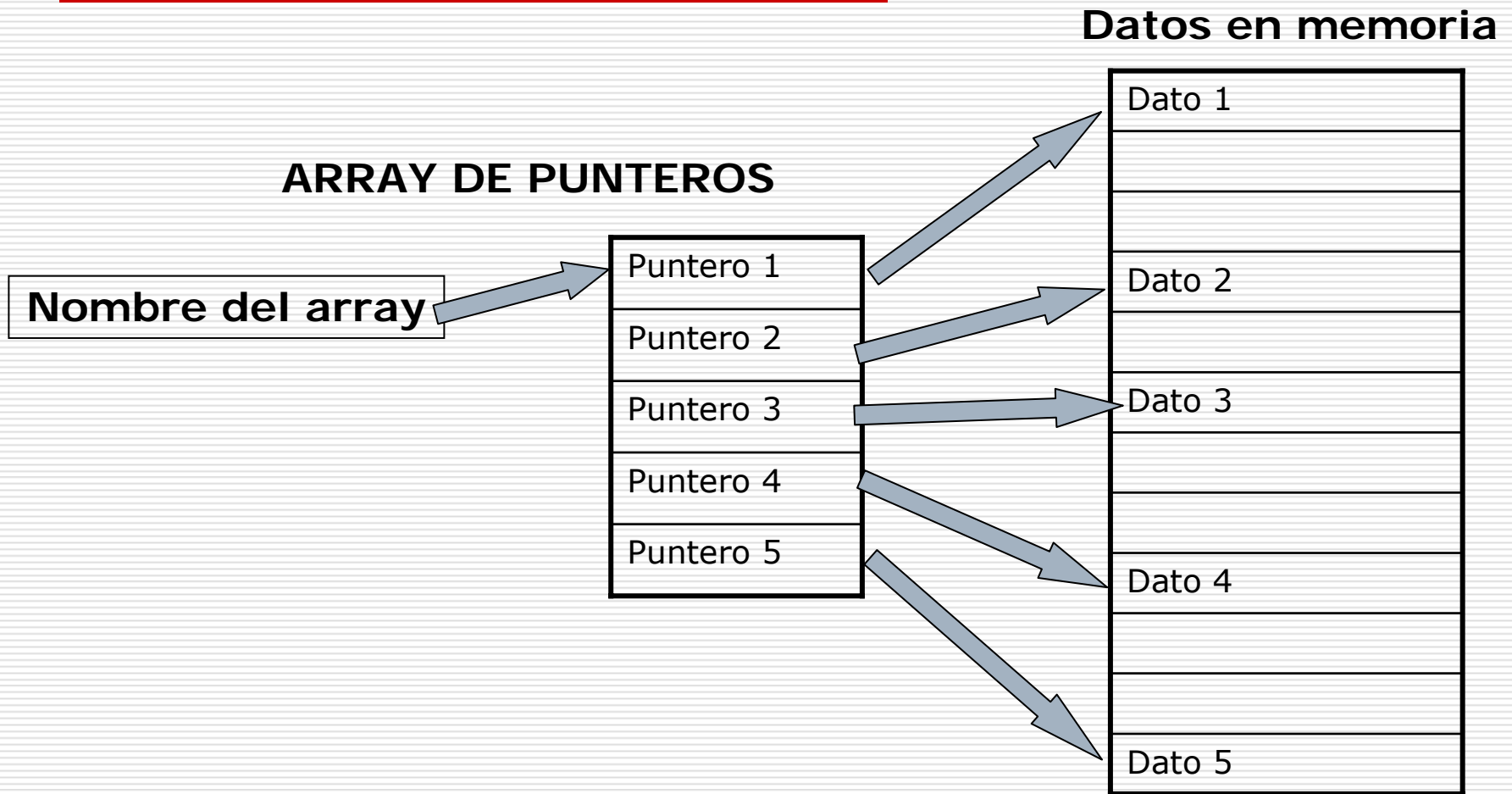
Relación entre punteros y arrays (II)

- Un **array de punteros** se declara del siguiente modo

```
tipodato *nombrevariable = [tamaño];
```

 - `tipodato` es el tipo de datos de los elementos que serán apuntados.
 - `nombrevariable` es el nombre del array de punteros
 - `tamaño` indica el número de punteros que contendrá el array
- El array de punteros contiene elementos que son direcciones
 - Cada dirección apuntará a un dato del tipo declarado.
 - Es preciso inicializar todos los punteros del array (hacerles apuntar a un tipo de dato válido)
- Un array de punteros a carácter es similar a un array bidimensional o a un array de cadenas

Relación entre punteros y arrays (III)



Relación entre punteros y arrays (IV)

□ Ejemplo

■ Array bidimensional de caracteres

```
char mens[3][80] = {"Inicial", "Central", "Último"};
    /* Array de 3 cadenas. Se reserva más memoria
    de la necesaria para que quepan todos los
    mensajes */
puts(mens[1]); /* Saca el mensaje "Central" */
```

■ Array de punteros a carácter

```
char *mens[3];          /* Array de 3 punteros a
                        carácter */
mens[0]= "Inicial";    /* Inicialización de punteros */
mens[1]= "Central";
mens[2]= "Último";
puts(mens[1]);        /* Casa el mensaje "Central" */
```


Puntero a un puntero

- Un **puntero a puntero** representa una indirección múltiple: el primer puntero contiene la dirección del segundo puntero, el cual apunta al dato

```
tipodato **nombrepuntero;
```

- `nombrepuntero` contiene la dirección de `*nombrepuntero` que, a su vez, contiene la dirección de `**nombrepuntero`

- `**nombrepuntero` es el identificador del dato

- Un puntero a puntero es *equivalente* al nombre de un array bidimensional

- Si `datos[M][N]` es un array bidimensional, el elemento `datos[i][j]` puede accederse también mediante la aritmética de punteros:

```
*(* (datos+i)+j)
```

Punteros a estructuras y uniones

- ❑ Para declarar un puntero a una estructura o a un unión, el tipo de estas debe estar previamente definido
- ❑ Declaración de un puntero a una estrucutra

```
struct tipoestructura *nombrepuntero;
union tipounion *nombrepuntero;
```

 - ❑ `tipoestructura` o `tipounion` representan el tipo de estructura o de unión ya definido
- ❑ Para acceder a los elementos de una estructura o de una unión a través de su puntero se utiliza el operador flecha «->» formado por los signos *menos* «-» y *mayor que* «>» en ese orden

```
nombrepuntero->nombreelemento
```

 - ❑ El operador flecha (con punteros) equivale al operador punto con identificadores ordinarios
- ❑ La referencia a un miembro de una estructura o una unión a través de un puntero puede utilizarse sin restricciones como un identificador ordinario