Tema 7: Punteros

Introducción a la programación I

Marcos Novalbos
Elena García Gamella

Índice

7.1. Variables puntero

- Una variable puntero (o puntero como se llama normalmente) es una variable que contiene direcciones de otras variables.
- Cuando una variable se declara, de cualquier tipo, se asocian tres atributos fundamentales:
  - Su nombre
  - Su tipo
  - Su dirección
- Al valor de una variable se accede por su nombre
- A la dirección se accede por el operador dirección &

7.1. Variables puntero. Definición

- Un puntero es una variable que contiene una posición (dirección) de memoria que puede corresponder a una variable declarada en el programa o no.
- Un puntero es una variable que “apunta” a otro lugar de memoria
- Sus ventajas:
  - Permiten definir arrays (vectores y matrices) de tamaño variable, que utilizan sólo la cantidad de memoria necesaria y cuyo tamaño puede reajustarse dinámicamente.
  - Permiten definir estructuras de datos más complejas, como listas o árboles.
  - Mediante los punteros es posible que las funciones devuelvan más de un valor

7.1. Variables puntero. Declaración

- Los punteros se declaran igual que las variables normales, pero con un asterisco (*) delante del nombre de las variables:

```
<tipo> * <identificador de puntero>;
```

Donde `tipo` hace referencia al tipo de datos al que "apuntará" nuestro puntero.

- Es muy importante respetar el tipo de datos al que apunta un puntero. Al igual que en el caso de las variables simples, un descuido en el tipo de datos generará errores.
- Se recomienda, para el nombre de la variable de tipo puntero, poner una 'p' que nos "recuerde" que se trata de un puntero.

```
double *pdoble;   /* Variable pdoble que es de tipo puntero a doble */
int *pentero; /* Variable pentero que es de tipo puntero a entero */
```

7.1. Variables puntero. Declaración

- Analogía: Dirección de una casa
  - Imaginad que alguien os pide la dirección de vuestra casa:
    - La apuntáis en un papel y dais la dirección, no dais vuestra casa
  - Igualmente, los punteros "apuntan direcciones" de "variables":

```
int  casa=0;              //tenemos la casa declarada
int* direccion;           //tenemos un lugar donde apuntar direcciones
direccion = &casa;        //apuntamos la dirección de la casa

int*  direccion2;         //un segundo lugar donde apuntar la dirección
dirección2 = direccion;   //copiamos la dirección, no la casa
casa=1;                   //cualquier modificación en la casa se verá reflejada cuando
                          //accedamos a través de su direcciones. Las próximas "visitas"
                          //se realizarán a través de su "dirección"
```

7.1. Variables puntero. Declaración

- Ejemplo:
  - double *pdoble; /* Variable "pdoble" que es de tipo puntero a doble */
  - int *pentero; /* Variable "pentero" que es de tipo puntero a entero */

- Tanto “pdoble” como “pentero” son variables de tipo puntero (o punteros para abreviar) y almacenan direcciones de memoria (por lo tanto, ambas ocupan la misma cantidad de memoria).
- La diferencia es que el dato almacenado en la dirección de memoria contenida en el puntero pentero es un entero, mientras que el dato almacenado en la dirección de memoria contenida en el puntero pdoble es un double.
- Se dice que pentero “apunta” a un entero y pdoble “apunta” a un double

7.1. Variables puntero. Declaración

- C no inicializa los punteros cuando se declaran, y después de su declaración una variable de tipo puntero contiene un valor inicial cualquiera (como ocurre con cualquier tipo de variables), que apuntará a una dirección aleatoria que puede incluso no existir.

- Es preciso inicializar los punteros antes de su uso.

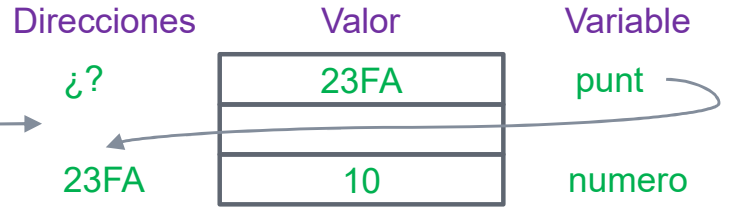
```
int i;          /* Define una variable */
int *p;         /* Define un puntero a un entero p */
p = &i;         /* Asigna la dirección de i a p */
```

7.2 Punteros. Operadores de puntero

- Operador &:
  - Operador unario de dirección.
  - Opera sobre cualquier tipo de variable.
  - Obtiene la dirección de memoria de su operando, es decir, la dirección de memoria de la variable sobre la que se aplica.
  - El compilador dará un aviso si se intenta realizar una asignación en la que no corresponden los tipos.

- Ejemplo:

```
int numero;
int *punt;
double *punt_d;
…
numero = 10;
punt = &numero; /*asignar la dirección de numero a una variable puntero*/
punt_d = &numero; /* Aviso, no se corresponden los tipos */ …
```



- Resultado: la variable número está en la posición (dirección) 23FA, después de la asignación a la variable puntero punt tendrá el valor 23FA.

7.2 Punteros. Operadores de puntero

- Operador * :
  - Operador unario de indirección
  - Complemento del operador &.
  - Permite operar sobre las variables a las que apunta la variable de tipo puntero.
  - Proporciona el valor (contenido) de la dirección de memoria a la que apunta el puntero sobre el que se aplica, es decir, permite acceder al valor por medio del puntero.

7.2 Punteros. NULL y Void

- NULL
  - Un puntero `NULL` (nulo) no direcciona ningún dato válido.
  - `NULL` es un valor

- void
  - Un puntero `void` direcciona datos de un tipo no especificado.
  - Un puntero `void` se puede igualar a nulo si no se direcciona ningún dato válido.
  - `void` es un tipo de dato

7.2 Punteros. Operadores de puntero

- Ejemplo Operador * :

```
…
int var1, var2;
…
```

7.2 Punteros. Operadores de puntero

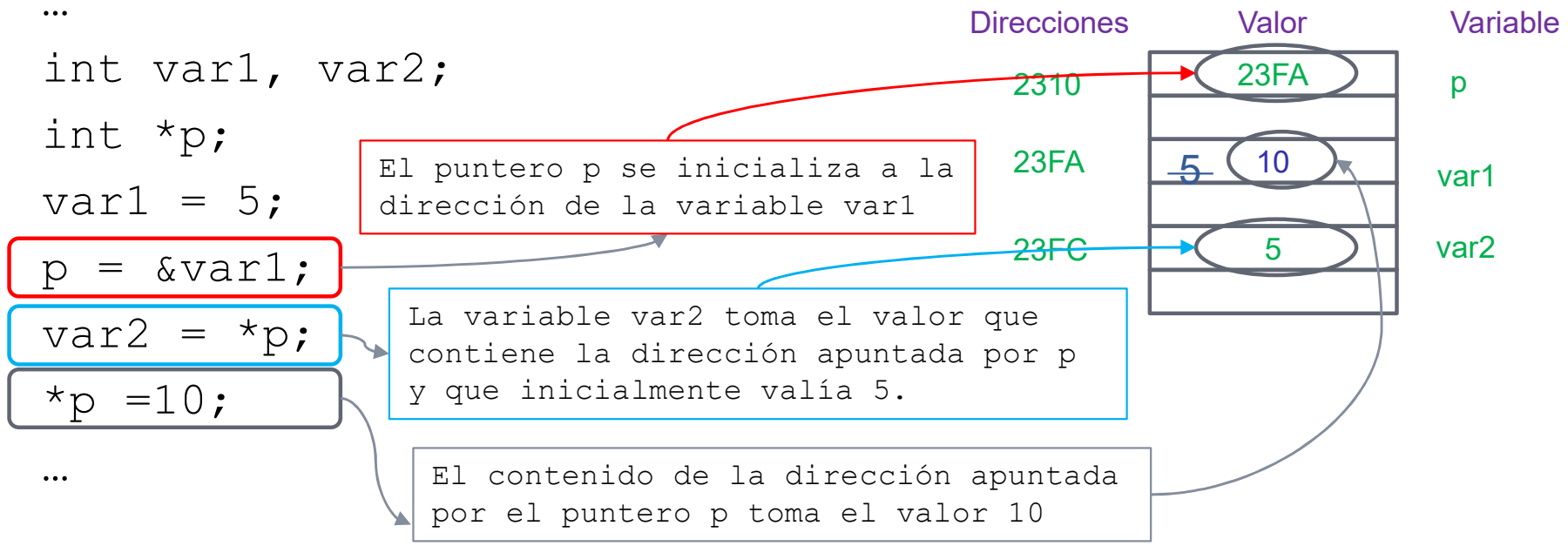
- Ejemplo Operador * :

```
…
int var1, var2;

int *p;
var1 = 5;

…
```

7.2 Punteros. Operadores de puntero

- Ejemplo Operador * :

```
…
int var1, var2;
int *p;
var1 = 5;
p = &var1;
var2 = *p;
*p =10;
…
```



ATENCIÓN: Después de la asignación p = &var1; tenemos dos maneras de manipular los valores enteros almacenados en la variable var1: directamente mediante var1 o indirectamente mediante el puntero p.

7.3 Punteros. Operaciones con punteros

- Asignación de punteros:
  - Es posible asignar una dirección de una variable a un puntero (ej1)
  - Es posible asignar el contenido de un puntero a otro puntero (ej2)

```
#include <stdio.h>
int main(void)
{
   int a;
   int *punt1;
   int *punt2;
   a = 10;
   punt1 = &a;              /* ej1 */
   punt2 = punt1;           /* ej2 */
   printf(" El valor %d esta en la posicion %x ", *punt2, punt2);
   return 0;
}
```

  - Resultado: El valor 10 se encuentra en la posición 203c .

7.3 Punteros. Operaciones con punteros

Aritmética de punteros:

- Se trata de sumar y restar, incrementar o decrementar variables de tipo puntero
- Debe entenderse como cambios en la dirección a la que apunta el puntero (se produce un cambio en la dirección de memoria contenida en el puntero).
- El incremento o decremento de un puntero depende exclusivamente del tipo de dato base dado en la declaración del puntero.
- Al declarar un puntero es necesario indicar a qué tipo de dato apunta para que cuando se utilicen los incrementos o decrementos se conozca cuánto hay que sumar o restar.
- **La operación de sumar 1 a un puntero hace que su dirección se incremente la cantidad necesaria para pasar a apuntar al siguiente dato del mismo tipo** (cantidad que coincide con el número de bytes que ocupa dicho tipo de dato).

  Por lo tanto, sólo en el caso de variables que ocupan 1 byte en memoria (variables de tipo “char”) la operación de incremento aumenta en 1 la dirección de memoria; en los demás casos aumenta más.

7.3 Punteros. Operaciones con punteros

Aritmética de punteros:

- La sustracción o resta de punteros tiene una semántica especial. La resta de punteros debe hacerse siempre entre punteros del mismo tipo (es decir, que apunten al mismo tipo de datos) y **el resultado no es el número de bytes que los separan, sino el número de elementos**.
  - Sólo tiene sentido si apuntan a direcciones diferentes de un mismo VECTOR.
  - El resultado de la sustracción es la diferencia de posiciones del VECTOR que existe entre ambos

- Es posible comparar punteros mediante cualquier operador de comparación y relacional, pero:
  - Es necesario que ambos punteros sean del mismo tipo.
  - Resultan de gran utilidad cuando se trabaja con vectores utilizando punteros.

- OPERACIONES PROHIBIDAS CON PUNTEROS
  - SUMAR PUNTEROS
  - MULTIPLICAR PUNTEROS
  - DIVIDIR PUNTEROS

7.3 Punteros. Operaciones con punteros

- **Ejemplo de aritmética de punteros**: incremento de un puntero

```
#include <stdio.h>
int main( void)
{
   int var=7;
   int *punt1;
   punt1 = &var;
   printf("\nLa direccion del dato =%x",&var);
   printf("\nEl valor apuntado por el puntero = %d", *punt1);
   printf("\nEl contenido de punt1 =%x",punt1);
   punt1++;
   printf("\nel contenido de punt1 =%x",punt1);
   printf("\nEl valor apuntado ahora por  el puntero = %d",*punt1);
   return 0;
}
```

Dirección de var: 1000

Al incrementar punt1 en 1, si un entero ocupa 4 bytes el resultado en punt1 es: 1004

7.3 Punteros. Operaciones con punteros. Ejercicio1

- Cread un programa que sume dos números utilizando punteros

7.3 Punteros. Operaciones con punteros. Ejercicio1

- Cread un programa que sume dos números utilizando punteros

```
#include <stdio.h>

void main()
{
    int num1, num2, sum;
    int *pnum1, *pnum2;

    pnum1 = &num1; // pnum1 stores the address of num1
    pnum2 = &num2; // pnum2 stores the address of num2

    printf("Introduce dos numeros: \n");
    scanf("%d%d", pnum1, pnum2);

    sum = *pnum1 + *pnum2;

    printf("Sum = %d", sum);
}
```

7.4 Punteros. Arrays y punteros

- **Existe una relación estrecha entre punteros y arrays**
- Los punteros son más eficientes en el manejo de arrays
- Se puede generar un puntero al primer elemento de un array simplemente especificando el nombre del array, sin índice
- Por ejemplo, dado

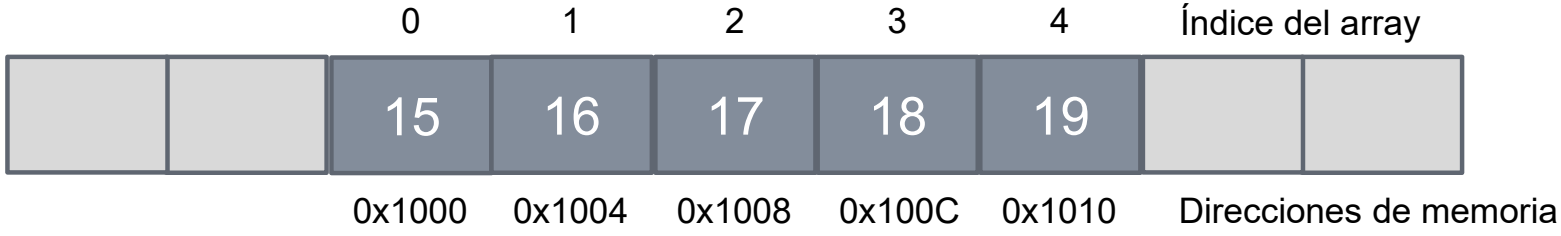
```
int alturas [10];
int *paltura;
…
paltura = alturas;
```

- También se puede especificar la dirección del primer elemento de un array utilizando el operador `&`. Así `alturas` y `&alturas [0]` producen el mismo resultado. Esto último casi nunca se utiliza.

7.4 Punteros. Arrays y punteros

- Un array se almacena en memoria de manera secuencial

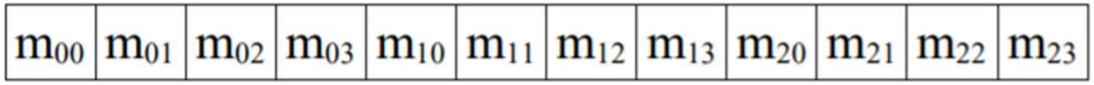
```
int edades [5] = {15,16,17,18,19};   int *pedades;
pedades=edades; // pedades toma el valor 0x1000
```



- Si tenemos un puntero `pedades` que apunte al array `edades[]` se puede aplicar fácilmente la aritmética de punteros para hacer referencia al siguiente elemento del array.
- `&edades[1]` sería equivalente a: `pedades+1` o `pedeades++`
- `edades[1]` es equivalente a `*(pedades+1)` o `*(pedades++)`

7.4 Indexación de punteros

- En cuanto a las matrices sabemos que se almacenan en memoria de forma contigua, empezando por filas y dentro de cada fila los elementos de las columnas. Ejemplo de matriz 3x4:



- Por tanto, una forma bastante común de acceder a un elemento de una matriz bidimensional de FxC a partir de la posición de inicio de la matriz es mediante la fórmula siguiente:

elemento(fila, columna)= *(puntero+(fila*C)+columna)

elemeto (i,j) = *(matriz+(i*NumColumnas)+j)

Donde "puntero" o “matriz” es la dirección del primer elemento.

7.4 Indexación de punteros

- Cualquier variable puntero puede indexarse como si estuviera declarada como un array. Por ejemplo

```
int *paltura, alturas [10];
paltura = alturas;
alturas [3] = 100; /* asignación usando índice */
*(paltura + 3) = 100; /*asignación usando aritmética de punteros */
```

- Ambas asignaciones ponen el valor 100 en el cuarto elemento del array alturas. La primera indexa y la segunda la aritmética de punteros
- Este mismo concepto se puede aplicar también a los arrays de dos o más dimensiones.

7.4 Punteros. Arrays y punteros. Ejercicio 2

- Crear un programa que copie un array de enteros con un tamaño máximo MAX_SIZE = 100, en otro array usando punteros

7.4 Punteros. Arrays y punteros. Ejercicio 2

- Crear un programa que copie un array en otro array usando punteros

1. Solicitar al usuario e Introducir el tamaño del array origen, así como los elementos de dicho array
2. Declarar otro array destino para dejar la copia.
3. Declarar punteros que apunten a ambos arrays
4. Copiar elementos del array origen al array destino utilizando las operaciones con punteros
5. Incrementar los punteros
6. Repetir 4 y 5 hasta que el puntero que apunta al array origen deje de apuntar a un elemento del array

7.4 Punteros. Arrays y punteros. Ejercicio 2

- Crear un programa que copie un array en otro array usando punteros

7.4 Punteros. Array de punteros

- Algunos de los problemas de los arrays dobles:
  - El número de columnas es fijo para cada fila.
  - Si necesitamos un tamaño variable de columnas para cada fila estaríamos desperdiciando memoria con el uso de arrays dobles.

- La solución pasa por el uso de un array de punteros
- Se puede declarar un array de punteros, como un array que contiene como elementos punteros, cada uno de los cuales apuntará a otro dato específico. Es decir:

```
int *ptr [10];
```

  - Reserva un array de **10 variables puntero**.

7.4 Punteros. Array de punteros

```
int *ptr [10];
```

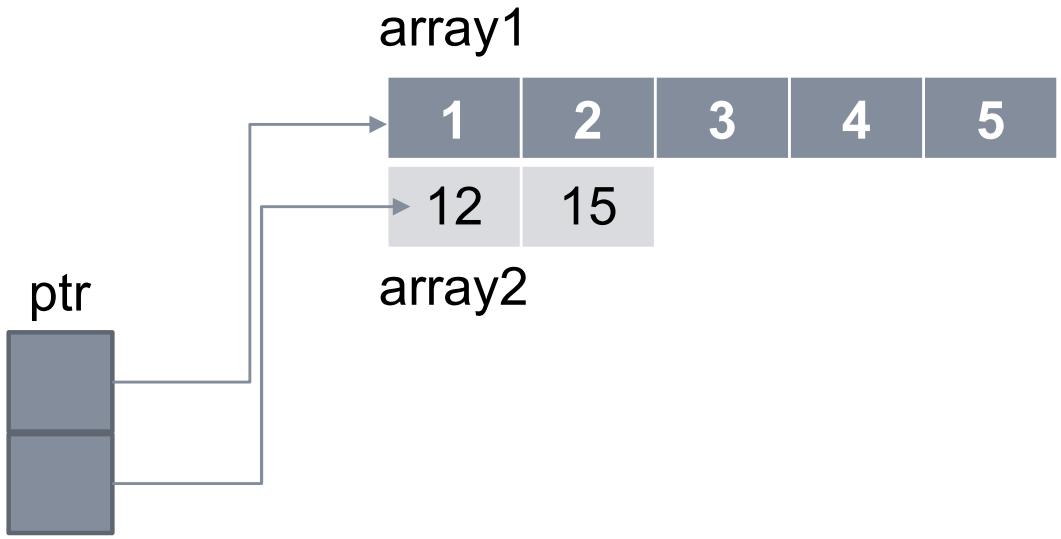
- Reserva un array de **10 variables puntero**.
- Cada elemento contiene una dirección que apunta a otros valores de la memoria
- Cada valor apuntado debe ser un entero

```
int edad = 18;
ptr [5] = &edad;   /* ptr [5] apunta a la dirección de edad */
ptr [4] = NULL;    /* ptr [4] no contiene ninguna dirección */
```

7.4 Punteros. Array de punteros

Ilustremos un ejemplo:

```
int array1 [5] = {1,2,3,4,5};
int array2 [2] = {12,15};
int *ptr [2];
ptr [0] = array1;
ptr [1] = array2;
```



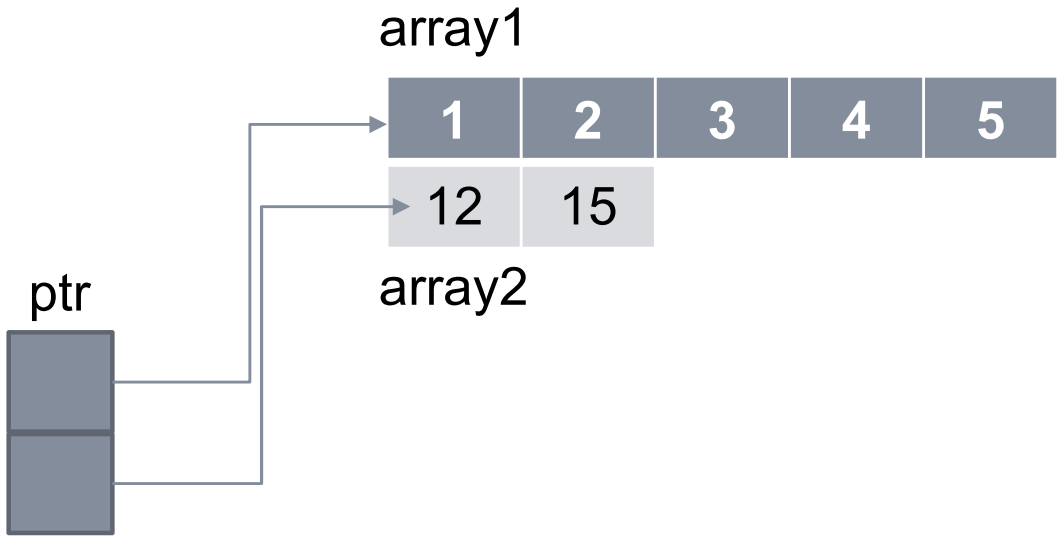
Que devolvería *ptr

Que devolvería *(*ptr)

7.4 Punteros. Array de punteros

Ilustremos un ejemplo:

```
int array1 [5] = {1,2,3,4,5};
int array2 [2] = {12,15};
int *ptr [2];
ptr [0] = array1;
ptr [1] = array2;
```



Que devolvería *ptr

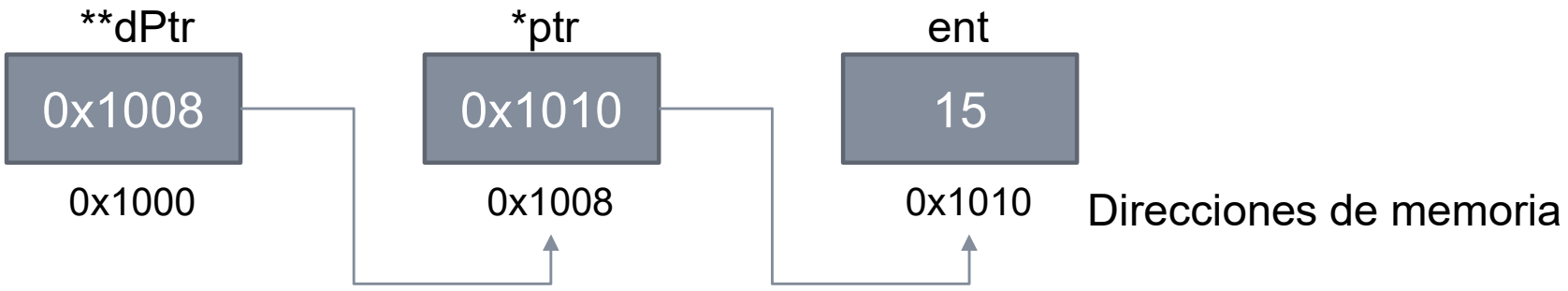
Que devolvería *(*ptr)

```
q=*ptr;          // array1
q1=*(*ptr);      // 1
```

7.4 Punteros. Punteros dobles

- Un puntero doble es aquel que apunta a otro u otros punteros.

```
int   ent;
int  *ptr;
int  **dPtr;
```

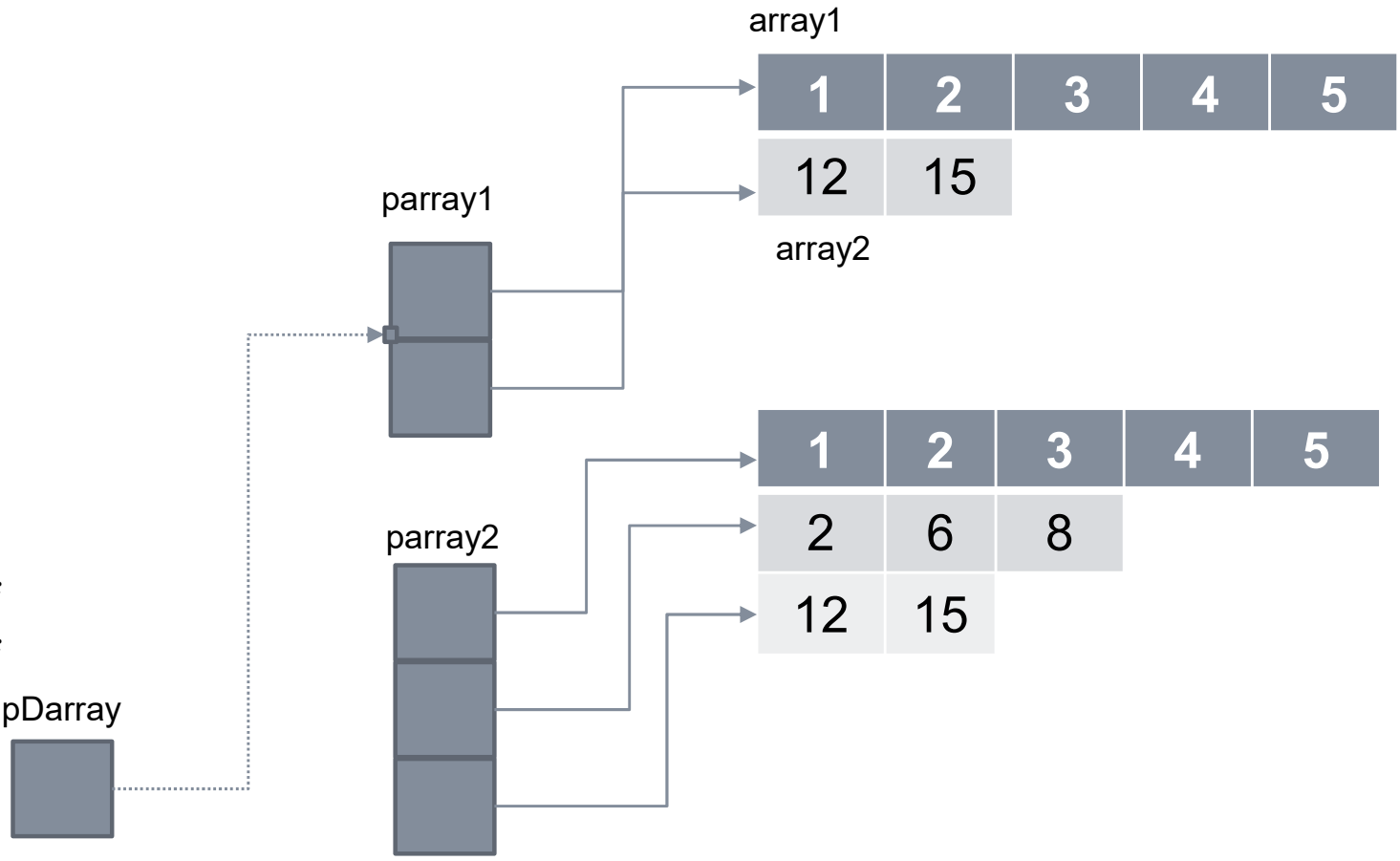


7.4 Punteros. Punteros dobles

- Un puntero doble es aquel que apunta a otro u otros punteros.

```
int **dPtr;
```

```
int array1 [5];
int array2 [10];

int *parray1 [2];
int *parray2 [3];
int  **pDarray;

parray1 [0] = array1;
parray1 [1] = array2;
pDarray = parray1;
```



7.4 Punteros. Punteros dobles. Ejercicio 3

- El siguiente código crea una matriz de 2 x 3. Completa el código para que muestre los valores de la matriz utilizando la aritmética de punteros

```
#include <stdio.h>
#define M 2
#define N 3
int main (){
   int matriz [M][N];
   int fila, columna;
   for (fila=0;fila<M;fila++)
      for (columna=0;columna<N;columna++)
         matriz [fila][columna] = fila *10 + columna;
...../// Imprimor esa matriz (sus elementos) con aritmética de punteros
}
```

7.4 Punteros. Punteros dobles. Ejercicio 3

- El siguiente código crea una matriz de 2 x 3. Completa el código para que muestre los valores de la matriz utilizando la aritmética de punteros

7.4 Punteros. Puntero a array de punteros

- Se puede declarar un puntero a un array de punteros a enteros

```
int *(*ptr)[10];
```

  - Paso a paso:

- `(*ptr)` es un puntero. `ptr` es un nombre de una variable puntero
- `(*ptr)[10]` es un puntero a un array
- `*(*ptr)[10]` es un puntero a un array de punteros
- `int *(*ptr)[10]` es un puntero a un array de punteros de variables `int`

7.4 Punteros. Punteros de cadenas

- Un puntero a char se puede inicializar para apuntar a constantes de cadena (string constant)

```
char *p = "Hola mundo";
```

- p no es un array sino un puntero, p guarda una dirección luego entonces ¿donde se almacena la cadena “Hola mundo”?
- El compilador de C crea lo que se llama una tabla de cadenas (string table), que almacena las constantes de cadena utilizadas por el programa. Por lo tanto, en la declaración anterior se coloca la dirección de "hola mundo", tal como se almacena en la tabla de cadenas, en el puntero p. En todo el programa, p se puede usar como cualquier otra cadena.

7.4 Punteros. Punteros de cadenas. Ejercicio 4

- Programa que calcule la longitud de una cadena utilizando punteros

7.4 Punteros. Punteros de cadenas. Ejercicio 5

- Programa que concatene dos cadenas de caracteres:
  - Utilizando un array
  - Utilizando punteros

7.4 Punteros. Punteros de cadenas. Ejercicio 5

- Programa que concatene dos cadenas de caracteres (array)

7.4 Punteros. Punteros de cadenas. Ejercicio 5

- Programa que concatene dos cadenas de caracteres (punteros)

Paso de Arrays como parámetros a una función.

- Funciones genéricas de paso de parámetros a una función de un array y de una matriz de dos dimensiones

```
imprimeArray(int *parray, int tamaño);

imprimeMatriz(int *pmatriz, int numFilas, int numCol);
```