

# Ejercicios. Punteros



Eva Lucrecia Gibaja Galindo  
Dpto. Informática y Análisis Numérico

# Ejercicio 1

- ¿Qué errores hay en la siguiente declaración de punteros?

```
int *p, &y;
char * b="Cadena larga";
char * c='C';
float x;
void * r = &x;
printf("%f", *r);
b[3]='\t';
```

# Ejercicio 1 (Solución)

- `int &y`
  - No es sintácticamente correcta. No tiene ningún sentido en C.
- `char * c = 'C' ;`
  - Cuando un carácter está rodeado de comillas simples es considerado como una constante de tipo *char*, no como una cadena
- `printf ("%f\n", *r) ;`
  - *r* es un puntero *void* y necesita que se le haga un *casting* al tipo concreto.
    - `printf ("%f\n", *(float *) r);`
- `b[3] = 't' ;`
  - Error en tiempo de ejecución por intentar cambiar una zona de memoria constante (un literal de cadena). Se puede recorrer elemento a elemento.

## Ejercicio 2

### ■ Salida del siguiente programa

```
int main () {
    int a = 5, *p;
    *p = *p * a;
    if (a == *p)
        printf("a es igual a *p");
    else
        printf("a es diferente a *p");
}
```

## Ejercicio 2 (Solución)

■ No es una asignación correcta

p no referencia  
ningún  
objeto válido

```
int main () {
    int a = 5, *p;
    *p = *p * a;
    if (a == *p)
        printf("a es igual a *p");
    else
        printf("a es diferente a *p");
}
```



## Ejercicio 3

### ■ Salida del siguiente programa

```
int main () {
    int a = 5, *p=&a;
    *p = *p * a;
    if (a == *p)
        printf("a es igual a *p");
    else
        printf("a es diferente a *p");
}
```

## Ejercicio 3 (Solución)

### ■ “*a es igual a \*p*”

- *p* referencia al objeto *a*, por lo que *\*p* nos da el contenido de la dirección de *a*.

```
int main () {
    int a = 5, *p=&a;
    *p = *p * a;
    if (a == *p)
        printf("a es igual a *p");
    else
        printf("a es diferente a *p");
}
```

## Ejercicio 4

### ■ Salida del siguiente programa

```
int main ()
{
    int a = 5, *p = &a, **p2 = &p;
    **p2 = *p + (**p2 / a);
    *p = a+1;
    a = **p2 / 2;
    printf("a es igual a: %d\n", a);
}
```



## Ejercicio 4 (Solución)

- $a$  es una variable de tipo entero, por eso el resultado es 3, y no 3.5

```
int main ()
{
    int a = 5, *p = &a, **p2 = &p;
    **p2 = *p + (**p2 / a);
    *p = a+1;
    a = **p2 / 2;
    printf("a es igual a: %d\n", a);
}
```

## Ejercicio 5

### ■ Salida del siguiente código:

```
int main () {
    int *p1, *p2, a, b;
    p1 = &a;
    *p1 = 42;
    p2 = p1;
    printf("%d y %d\n", *p1, *p2);
    *p2 = 53;
    printf("%d y %d\n", *p1, *p2);
    p1 = &b;
    *p1 = 88;
    printf("%d y %d\n", *p1, *p2);
}
```

## Ejercicio 5 (Solución)

■ Salida del siguiente código:

■ 42 y 42 53 y 53 88 y 53

```
int main () {
    int *p1, *p2, a, b;
    p1 = &a;
    *p1 = 42;
    p2 = p1;
    printf("%d y %d\n", *p1, *p2);
    *p2 = 53;
    printf("%d y %d\n", *p1, *p2);
    p1=&b;
    *p1 = 88;
    printf("%d y %d\n", *p1, *p2);
}
```

## Ejercicio 6

- Dadas las siguientes declaraciones, encontrar los errores en cada línea:

- `int *pta, *ptb, a, b;`

```
pta = *a;
ptb = &pta;
pta = 8;
ptb = ptb + 3;
ptb = &NULL;
b = 8;
*ptb = b;
ptb = a + 10;
printf("%d", pta + ptb);
```

## Ejercicio 6 (Solución)

- `pta = *a;`
  - `*a` no tiene sentido
- `ptb = &pta;`
  - Las expresiones no son del mismo tipo
- `pta = 8;`
  - A un puntero no le podemos asignar un entero
- `ptb = ptb + 3;`
  - Esta expresión tiene sentido solo con vectores
- `ptb = &NULL;`
  - `NULL` es una constante, no tiene dirección



## Ejercicio 6 (Solución)

- `b = 8;`
  - Correcto
- `*ptb = b;`
  - Ptb no apunta a nada
- `ptb = a + 10;`
  - No se puede apuntar un entero
- `printf("%d", pta + ptb);`
  - No se pueden sumar dos punteros

## Ejercicio 7

- Dadas las siguientes declaraciones

```
struct electrica
{
    char corriente[30];
    int voltios;
};
struct electrica a, b;
struct electrica *p=&a, *q=&b;
```

- ¿Qué hacen cada una de las siguientes sentencias?  
¿Hay alguna no válida?

## Ejercicio 7

- strcpy(p->corriente,"ALTERNA");
- strcpy(q->corriente,"ALTA");
- p->voltios = q->voltios;
- p->corriente = q->voltios;
- \*p = \*q;
- p = 54;
- p = q;
- \*q = p;
- \*p=72;
- scanf("%s", q->corriente);
- scanf("%d", &q->voltios);

## Ejercicio 7 (Solución)

- `strcpy(p->corriente, "ALTERNA");`
  - Asigna la cadena ALTERNA al campo corriente de *a*
- `strcpy(q->corriente, "ALTA");`
  - Asigna la cadena ALTA al campo corriente de *b*
- `p->voltios = q->voltios;`
  - Asigna el campo voltios de *b* al campo voltios de *a*
- `p->corriente = q->voltios;`
  - Asignación incorrecta
- `*p = *q;`
  - Asigna la estructura *b* a la estructura *a*

## Ejercicio 7 (Solución)

- `p = 54;`
  - Asignación incorrecta
- `p = q;`
  - `p` referencia a la estructura `b`
- `*q = p;`
  - Asignación incorrecta
- `*p=72;`
  - Asignación incorrecta, `*p` es un *struct electrica*
- `scanf("%s", q->corriente);`
  - Lee por teclado la cadena `q->corriente`
- `scanf("%d", &q->voltios);`
  - Lee por teclado el entero `q->voltios`



## Ejercicio 8

- Sean las siguientes declaraciones

```
int* pi, x=5, y;
```

```
int vector[]={1, 2, 3, 4, 5};
```

```
int** ppi;
```

- Explicar el significado de las siguientes sentencias:

## Ejercicio 8

- `pi=&x;`
- `y=*pi;`
- `*pi=0;`
- `*pi=vector[3];`
- `pi=vector;`
- `pi=&vector[0];`
- `y=*(pi+1);`
- `pi=vector+3;`
- `y=*(vector+4);`
- `x=*(pi+1);`
- `ppi=&pi;`
- `**ppi=8;`
- `*ppi=vector+1`
- `*( *ppi+2) = *(vector+3)+1;`
- `*(pi+2) = **ppi+2`

## Ejercicio 8 (Solución)

- `pi=&x;`
  - `pi` almacena la dirección de memoria de `x`
- `y=*pi;`
  - A `y` se le asigna el contenido del objeto referenciado por `pi` (`x`). Es equivalente a `y=x`
- `*pi=0;`
  - Al objeto referenciado por `pi` (`x`) se le cambia el valor a 0. Es equivalente a `x=0`
- `*pi=vector[3];`
  - Al objeto referenciado por `pi` (`x`) se le cambia el valor por el valor que tenga `vector[3]`. Es equivalente a `x=vector[3]`
- `pi=vector;`
  - El puntero `pi` almacena la dirección de memoria del primer elemento del vector

## Ejercicio 8 (Solución)

- $pi = \&vector[0];$ 
  - El puntero  $pi$  almacena la dirección de memoria del primer elemento del vector. Ídem al anterior
- $y = *(pi+1);$ 
  - A la variable  $y$  se le asigna el valor de la componente 1 del vector. Equivalente a  $y = v[1]$
- $pi = vector+3;$ 
  - $pi$  apunta al elemento 3 del vector
- $y = *(vector+4);$ 
  - A la variable  $y$  se le asigna el contenido de la componente 4 del vector. Equivalente a  $y = vector[4]$
- $x = *(pi+1)$ 
  - Asigna a  $x$  el contenido de la componente apuntada por  $pi+1$ . Como  $pi$  apuntaba a  $(vector+3)$  es equivalente a  $x = vector[4]$

## Ejercicio 8 (Solución)

- `ppi=&pi;`
  - *ppi* apunta a la variable *pi*. Referencia un objeto de tipo puntero a entero
- `**ppi=8;`
  - Como *pi* apunta a *vector[3]* equivale a *vector[3]=8*
- `*ppi=vector+1`
  - Pone a *pi* apuntando a *vector[1]*
- `*(*ppi+2) = *(vector+3)+1;`
  - Equivale a *vector[3] = vector[3]+1*
- `*(pi+2) = **ppi+2`
  - Equivale a *vector[3] = vector[3]+2*



## Ejercicio 9

- Determinar el resultado que almacenan las variables al final

```
int B[] = {3, 4, 1, 2, 7, 12, -4};
float f = 4.234, *ptf;
*(B+3) = *B + 15;
ptf = &f;
*B = (int) (*ptf);
f = *ptf + 20;
*(B + 5) = (int) (*ptf);
```

## Ejercicio 9 (Solución)

- Determinar el resultado que almacenan las variables al final:

- $f=24.213$ ,  $B=\{4,4,1,18,7,24, -4\}$

```
int B[] = {3, 4, 1, 2, 7, 12, -4};
float f = 4.234, *ptf;
*(B+3) = *B + 15;
ptf = &f;
*B = (int) (*ptf);
f = *ptf + 20;
*(B + 5) = (int) (*ptf);
```

## Ejercicio 10 (void\*)

- ¿Qué imprime el siguiente código en pantalla?

```
char car1='B', car2;  
void *ptg;  
ptg = &car1;  
car2 = *((char *)ptg) + 3;  
printf("%c", car2);
```

## Ejercicio 10 (Solución)

- ¿Qué imprime el siguiente código en pantalla?

```
char car1='B', car2;  
void *ptg;  
ptg = &car1;  
car2 = *((char *)ptg) + 3;  
printf("%c", car2);
```

Imprime una E

## Ejercicio 11 (void\*)

- ¿Qué imprime el siguiente código en pantalla?

```
char cad[20], car3;
void *ptg;
ptg = cad;
strcpy(cad, "Ejemplo");
car3 = *((char *) (ptg) + 3) + 5;
printf("%c", car3);
```



## Ejercicio 11 (Solución)

- ¿Qué imprime el siguiente código en pantalla?

```
char cad[20], car3;  
void *ptg;  
ptg = cad;  
strcpy(cad, "Ejemplo");  
car3 = *((char *) (ptg) + 3) + 5;  
printf("%c", car3);
```

Imprime una r

## Ejercicios 12, 13 y 14

- Escriba una función recursiva que determine si una palabra es o no un palíndromo, sin usar *string* y usando aritmética de punteros.
- Escriba una función que reciba un vector de enteros y calcule la media de los pares positivos y el mínimo de los impares negativos.
  - **mediaMinimo.c**
- Escribir un programa que permita calcular el área de diversas figuras: un triángulo, un cuadrado, un trapecio y un círculo. Utiliza un *array* de punteros a funciones, siendo las funciones las que permiten calcular el área.
  - **areas.c**