

PROGRAMACIÓN

Unidad 6: Punteros

Lic. Mariela A. Velázquez

Temas a tratar

**P
A
R
T
E**

1

Definición de punteros

Conceptos básicos

Declaración de punteros

Operaciones de punteros

Punteros y Arreglos

Punteros y Funciones

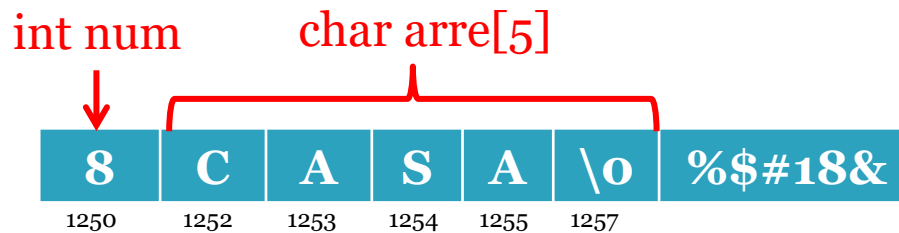
Pasaje de Parámetros

**P
A
R
T
E**

2

Introducción

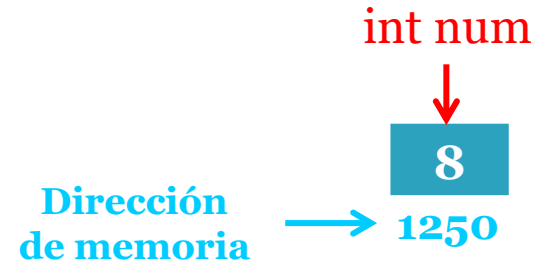
Cada vez que se declara una variable C



Representación de la memoria.

El compilador establece un área de memoria para almacenar el contenido de la variable

El espacio para esa variable se sitúa en una posición específica de la memoria, conocida como dirección de memoria



Se puede ganar en eficacia en el acceso a esta dirección de memoria utilizando un puntero

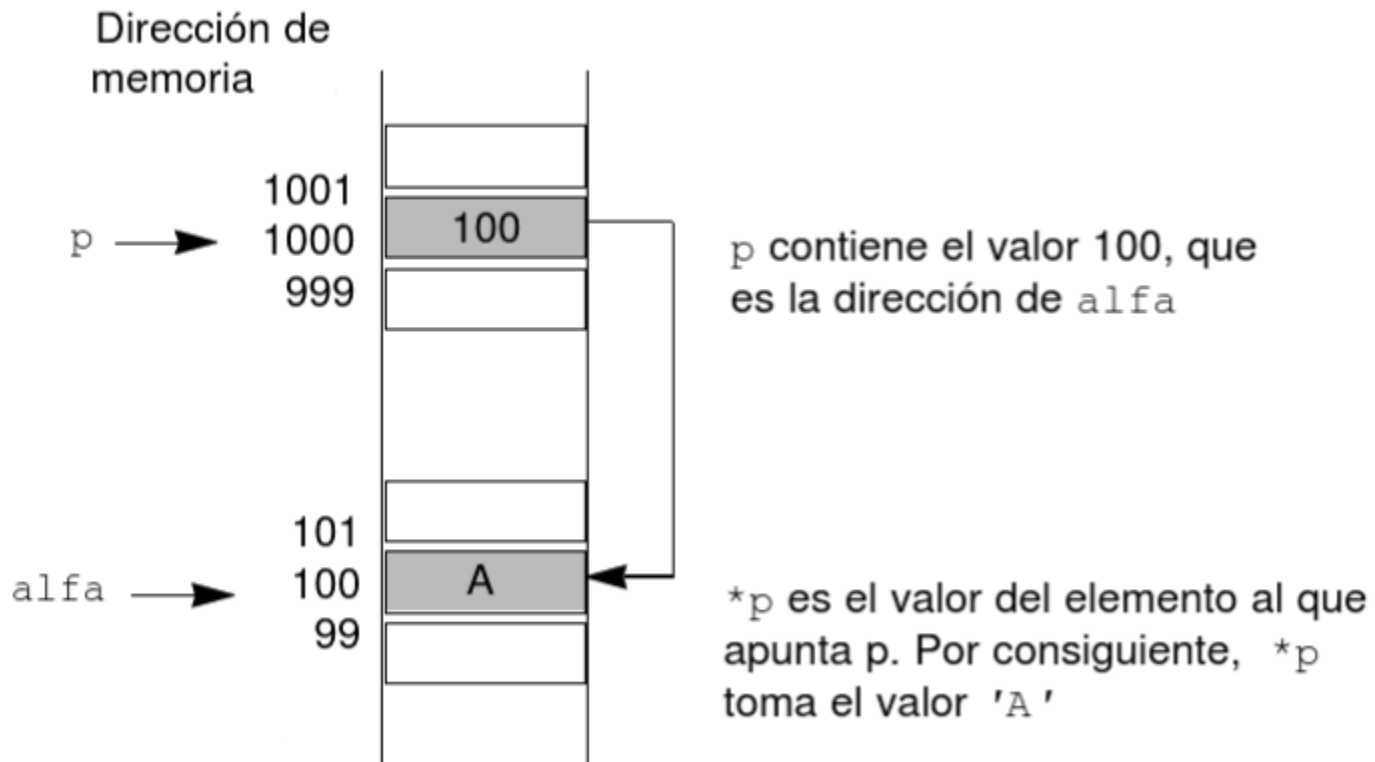
¿Qué es un puntero?

Un puntero es una variable que contiene la dirección de memoria de un dato o de otra variable que contiene al dato. Quiere esto decir que el puntero apunta al espacio físico donde está el dato o la variable.

Los punteros se rigen por estas reglas básicas...

- Un puntero es una variable como cualquier otra.
- Una variable puntero contiene una dirección que apunta a otra posición en memoria.
- En esa posición se almacenan los datos a los que apunta el puntero.

Grafica de un puntero...



Declaración de punteros

La declaración de un puntero consiste en:

tipo *nombre-vble-puntero

Donde:

tipo: cualquier tipo de dato (int, char, float).

*: operador de puntero.

nombre-vble-puntero: identificador.

Declaración de punteros

Esta declaración significa que:

- La variable definida de tipo puntero apunta a objetos del tipo de dato especificado.
- La variable puntero es capaz de almacenar la dirección del objeto al cual apunta.
- El valor de un puntero es una dirección.
- El objeto al cual apunta un puntero se llama **objeto referenciado**. Los objetos referenciados son del tipo base especificado.

Los operadores de un puntero

& Operador de dirección

El **&**, devuelve la dirección de memoria de su operando.

p1 = &car; //p1 recibe la dirección de car.

Podemos ver que el **operador &** sirve para acceder a la dirección, NO al contenido del objeto referenciado.

Recuerdan:

```
scanf("%d", &numero);
```

Ahora podemos entender, que la función `scanf()`, tiene como parámetro la dirección de memoria de la variable en la cual queremos almacenar un valor.

Los operadores de un puntero

 Operador de indirección

El **operador *** es el complemento del operador **&**, devuelve el contenido del objeto referenciado.

Importante: Los operadores & y * tienen precedencia superior a todos los operadores aritméticos.

Ejemplos de uso de punteros

El siguiente programa cambiará los contenidos de las variables prim, seg.

```
#include <stdio.h>

int main()
{
    int prim = 10, seg=20, temp;
    int *punt_num;
    printf("Al inicio – prim: %d, seg:%d", prim, seg);
    punt_num = &prim;
    temp = *punt_num
    *punt_num = seg;
    seg = temp;

    printf("Despues - prim: %d, seg:%d", prim, seg);
    return 0;
}
```

Inicialización de punteros

Como otras variables en C, los punteros pueden ser inicializados en su definición. Por ejemplo:

```
int prim;  
int *punt_num = &prim
```

Operaciones con Punteros

- Asignaciones de Punteros.
- Aritméticas de Punteros.
- Comparaciones de Punteros.

Asignaciones con Punteros

Como cualquier variable, se puede usar un puntero en la parte derecha de una sentencia de asignación.

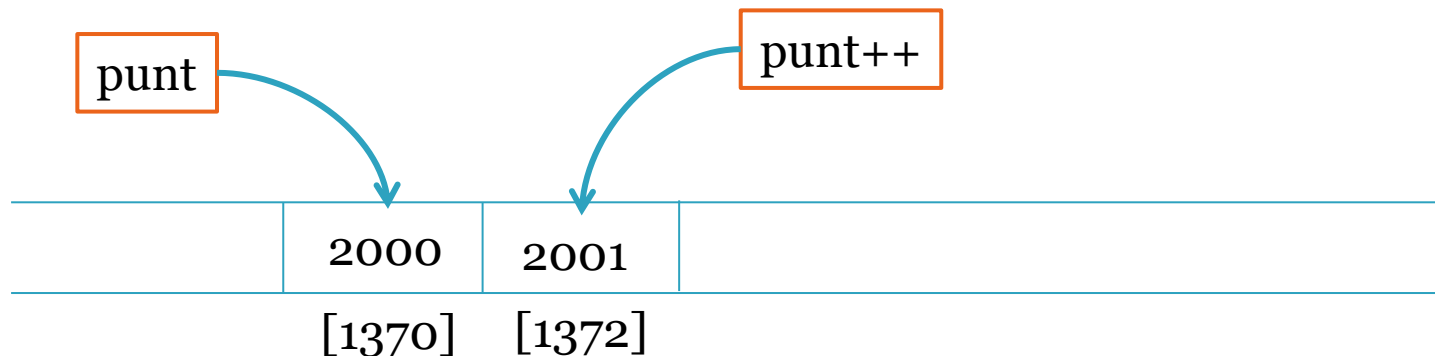
```
1
2  #include <stdio.h>
3
4  int main(void)
5  {
6      int num;
7      int *punt;
8
9      num = 101;
10     punt = &num;
11
12
13     printf("La dirección es: %p\n", punt);
14
15     printf("El valor de x es: %d", *punt);
16     return 0;
17 }
```

Aritméticas de Punteros (de Direcciones)

Solo se pueden realizar dos operaciones: +, -

Para entender que ocurre en la aritmética de direcciones, lo veremos mediante un ejemplo:

```
int *punt;
```



Aritméticas de Punteros (de Direcciones)

Cada vez que incrementamos el puntero `punt`, apuntará tantas direcciones más como bytes tenga el tipo base.

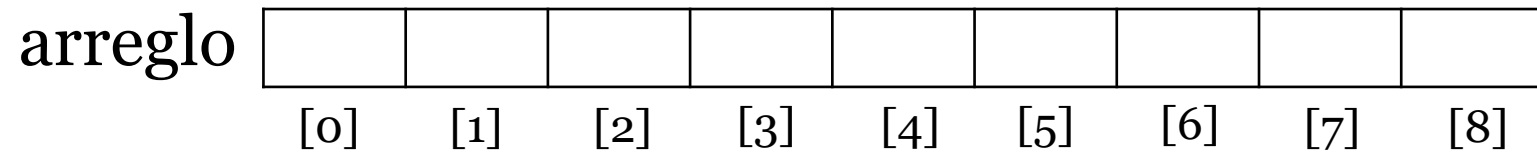
Pensemos un entero tiene 2 bytes de longitud, cuando se incrementa un puntero entero, su valor crece en dos. Es decir, el incremento o decremento se produce en función de su tipo base.

Punteros y Arreglos

- En C existe una fuerte relación entre arreglos y punteros.
- Cualquier operación que pueda lograrse por indexación de un arreglo, también puede realizarse con punteros.
- Las versiones con punteros son más rápidas.

Declaraciones

```
int arreglo[9]
```



```
int *p_arreglo;
```

```
p_arreglo = &arreglo[0]; //asignación
```

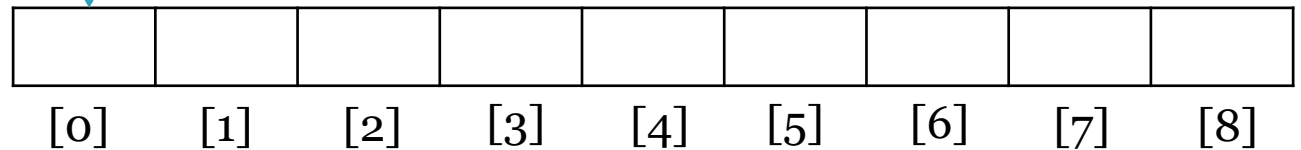
Asignación

p_arreglo



p_arreglo apunte al **elemento 0**
del arreglo

arreglo



Sabemos que, el nombre del un arreglo es sinónimo a la dirección del primer elemento del arreglo, por lo tanto la siguiente asignación es válida:

p_arreglo = arreglo;

Asignación

```
int aux;
```

```
int arreglo[9];
```

```
int *p_arreglo;
```

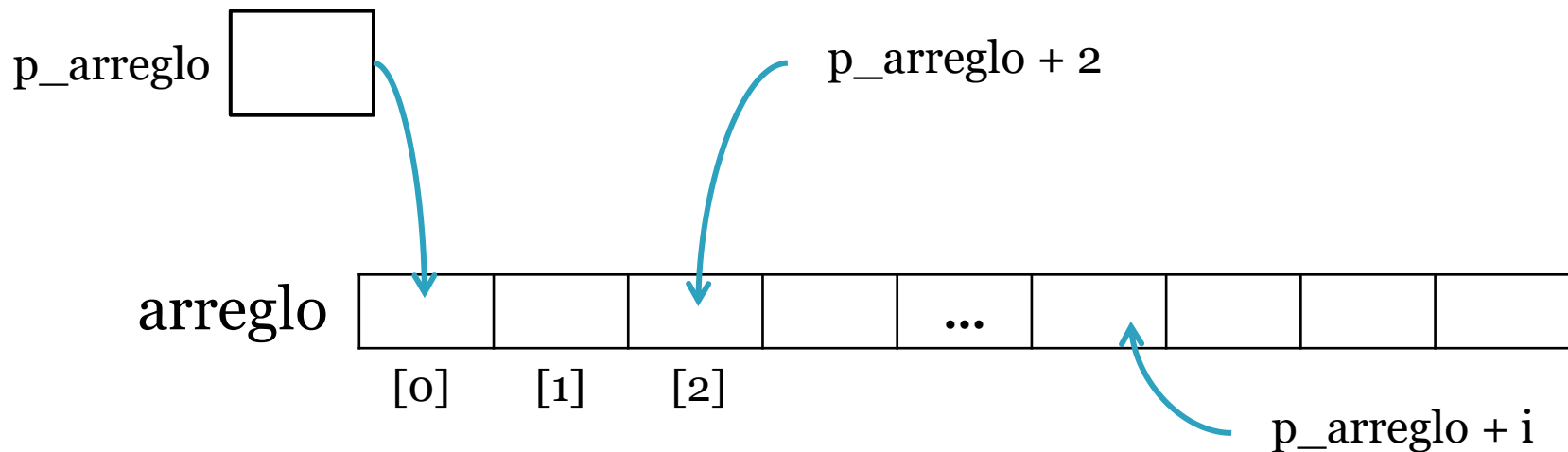
```
p_arreglo = arreglo;
```

```
aux = *p_arreglo[0]; // copia el contenido del arreglo[0] en aux
```

arreglo	1	1	2	3	5	7	12	19	31
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

En general

Si $p_arreglo$ apunta a un elemento en particular del arreglo,
 $p_arreglo+1$ apunta al siguiente elemento y $p_arreglo+i$ apunta
 i elementos después de la dirección de $arreglo[0]$



Ejemplo

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char cadena[80];
    int i;

    puts("Introducir una cadena en mayuscula");
    gets(cadena);

    puts("Esta es la cadena en miniscula");
    for(i=0; cadena[i]!='\0'; i++)
    {
        printf("%c", tolower(cadena[i]));
    }

    return 0;
}
```

ARREGLOS

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char cadena[80], *p_cadena;

    puts("Introducir una cadena en mayuscula");
    gets(cadena);

    puts("Esta es la cadena en miniscula");
    p_cadena = cadena;

    while(*p_cadena != '\0')
    {
        printf("%c", tolower(*(p_cadena++)) );
    }

    return 0;
}
```

PUNTEROS

La velocidad es un bien preciado en programación, se debe saber que entre ambas versiones, la de punteros es más rápida.

Existe una diferencia muy importante entre nombre del arreglo y un apuntador.

Un puntero es una variable, es lícito escribir:

```
p_cadena = cadena ;
```

```
p_cadena ++;
```

En tanto: un nombre de arreglo NO es una variable, por lo cual no es correcto escribir:

```
cadena++;
```

Indexando un puntero

```
#include <stdio.h>

int main(void)
{
    int numeros[5] = {1,2,3,4,5};
    int *p_numeros, i;
    p_numeros = numeros;

    for(i=0; i<5; i++)
    {
        printf("%d", p_numeros[i]);
    }
    return 0;
}
```

Cualquier expresión de arreglo e índice es equivalente a puntero y desplazamiento.

Terminamos...



...por ahora

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("Hasta la próxima clase!!\n");
```

```
    return 0;
```

```
}
```