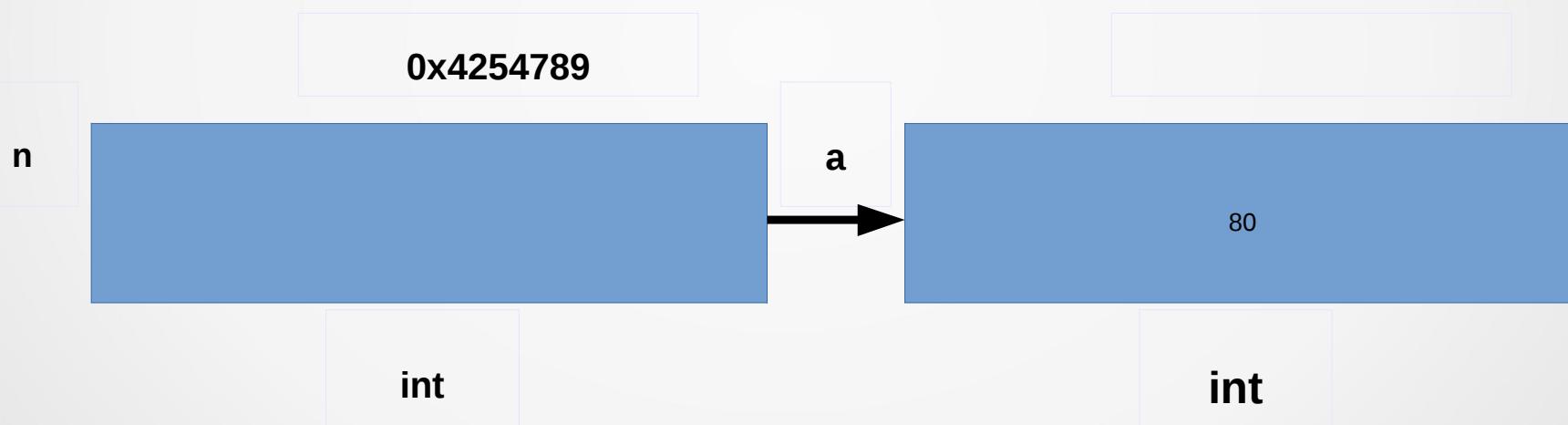


Punteros

Direcciones en memoria.

- Cuando una variable se declara, se asocian tres atributos fundamentales con la misma: su nombre, su tipo y su dirección en memoria.



Concepto de Puntero



El valor de un puntero es una dirección. La dirección depende del estado de la computadora en la cual se ejecuta el programa.

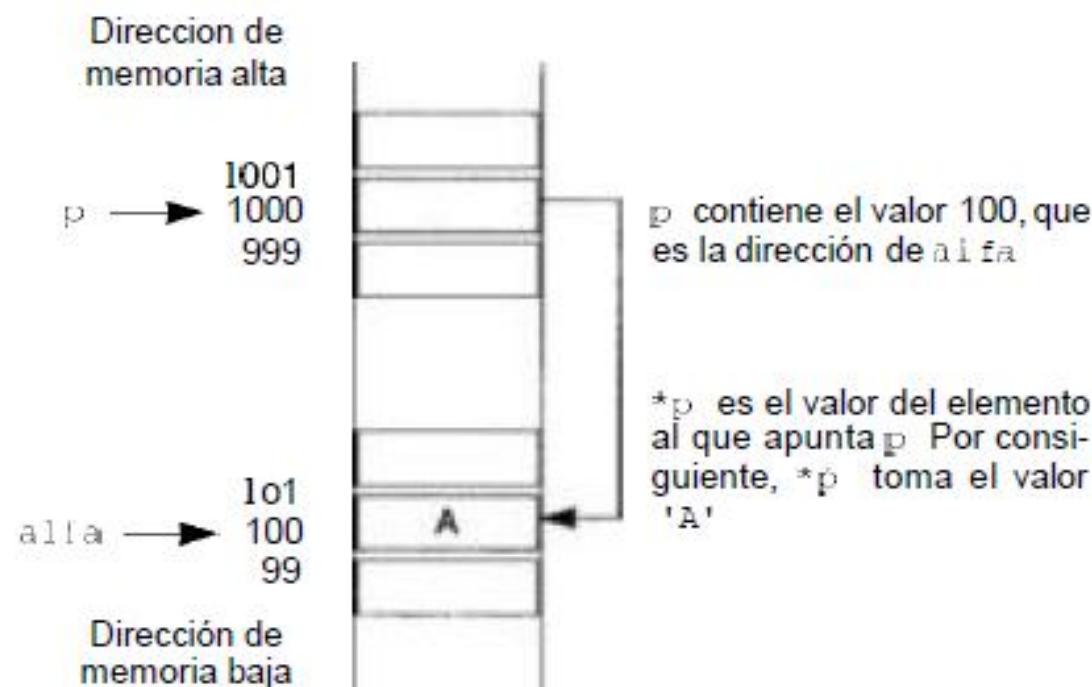


Figura 10.1. Relaciones entre $*p$ y el valor de p (dirección de `alfa`).

Declaración de puntero

<tipo de dato apuntado> *<identificador de puntero>

Algunos ejemplos de variables punteros:

int* ptr1; /*Puntero a un tipo de dato entero (int)*/

long* ptr2; /*Puntero a un tipo de dato entero largo (long int)*/

char*ptr3; /*Puntero a un tipo de dato char*/

float *f; /*Puntero a un tipo de dato float*/

Inicialización de un puntero

- + Asignar memoria (estáticamente) definiendo una variable y a continuación hacer que el puntero apunte al valor de la variable.

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

- + Asignar un valor a la dirección de memoria.

```
*p=50;
```

Punteros y verificación de tipos

```
float*fp;
```

```
char c;
```

```
fp=&c;
```

C requiere que las variables puntero direccion en realmente variables del mismo **tipo** de dato que está ligado a los punteros en sus declaraciones.

Punteros NULL

```
#define NULL
```

Un sistema de inicializar una variable puntero a nulo es:

```
char *p=NULL;
```

Los punteros nulos se utilizan con frecuencia en programas con arrays de punteros. Cada posición del array se inicializa a NULL; después se reserva memoria dinámicamente y se asigna a la posición correspondiente del array, la dirección de la memoria.

Puntero void

En C se puede declarar un puntero de modo que apunte a cualquier tipo de dato, es decir, no se asigna a un tipo de dato específico. El método es declarar el puntero como un punterovold*,denominado puntero genérico

```
void *ptr; /*declara un puntero void,puntero genérico*/
```

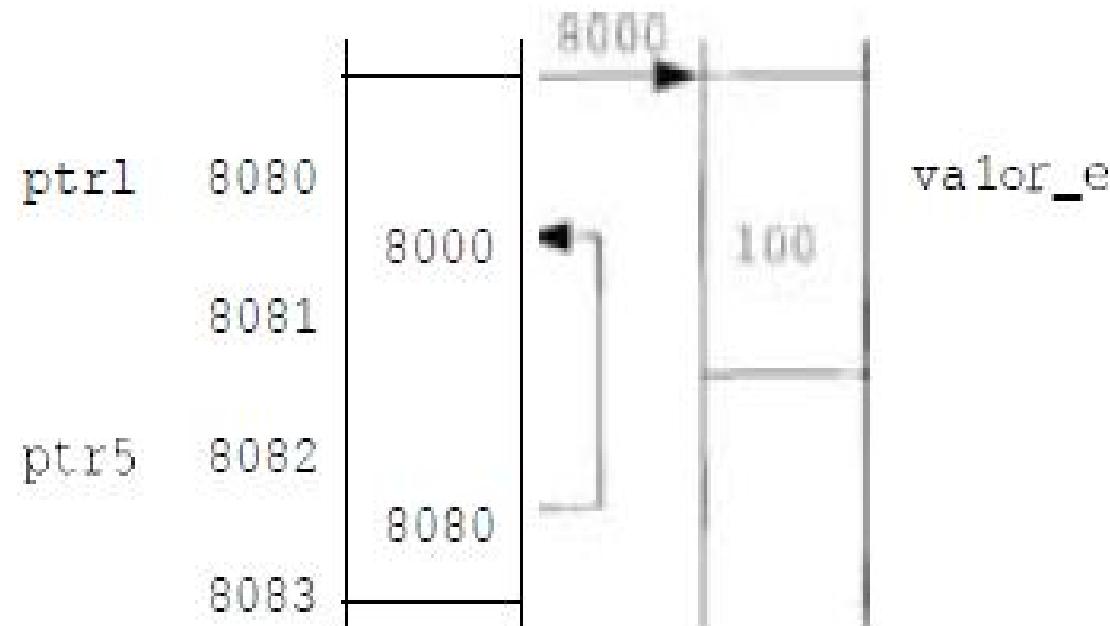
El puntero ptr puede direccionar cualquier posición en memoria, pero el puntero no está unido a un tipo de dato específico. De modo similar, los punteros void pueden direccionar una variable float,una char, o una posición arbitraria o una cadena.

No confundir punteros void y NULL.

- + Un puntero nulo no dirige a ningún dato válido.
- + Un puntero void dirige datos de un tipo no especificado. Un puntero void se puede igualar a nulo si no se dirige a ningún dato válido.
- Nulo es un valor;
- + void es un tipo de dato.

En el ejemplo siguiente ptr5 es un puntero a un puntero.

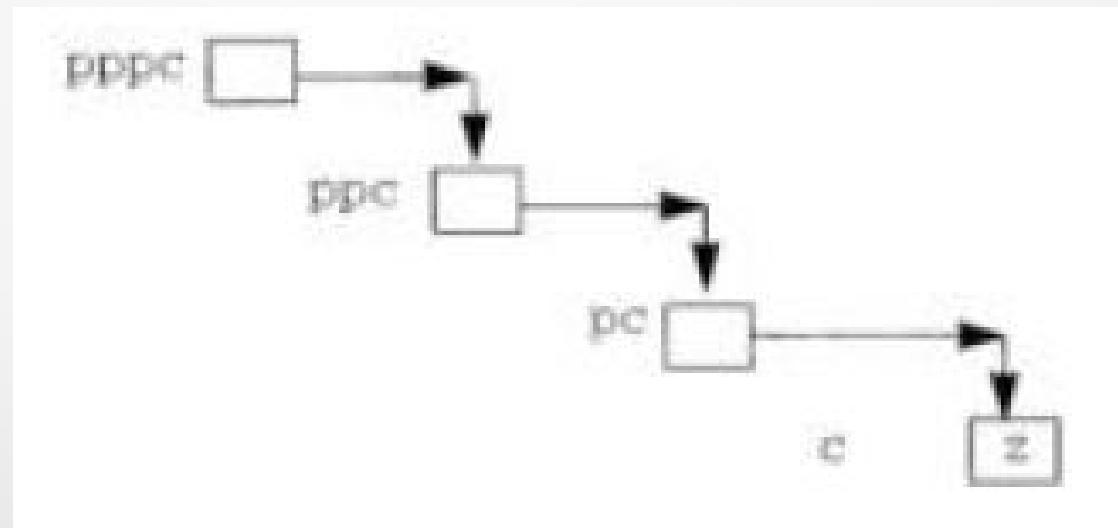
```
int valor_e=100;  
  
int *ptr1=&valor_e;  
  
int **ptr5=&ptr1;
```



Un puntero a un puntero.

Punteros a Punteros

```
char c='z';  
char* pc=&c;  
char** ppc=&pc;  
char*** pppc=&ppc;  
  
***pppc='m'; /*cambia el valor de z a 'm' */
```



Punteros y Arreglos

Nombres de arrays como punteros

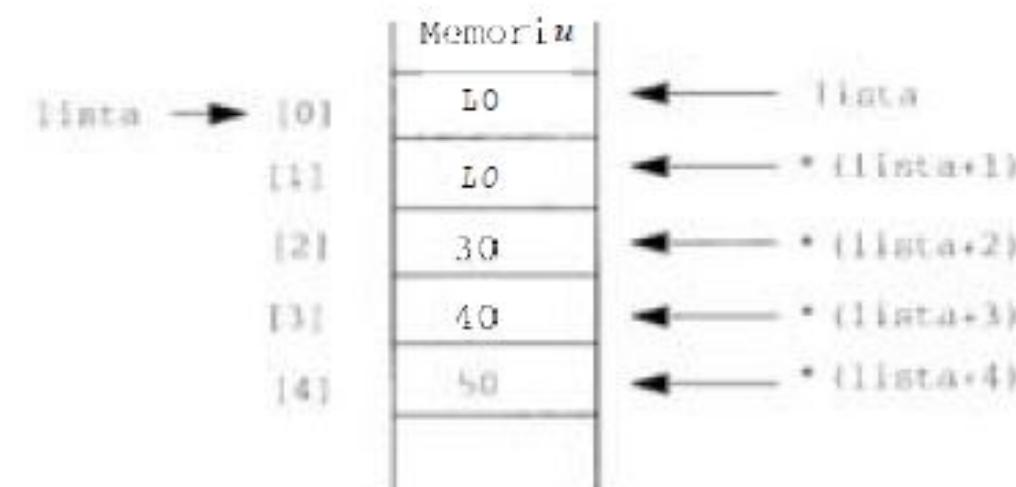
`lista +0 apunta a lista[0]`

`lista + 1 apunta a lista[1]`

`lista +2 apunta a lista[2]`

`lista + 3 apunta a lista[3]`

`lista + 4 apunta a lista[4]`



Un array almacenado en memoria.

Punteros y Arreglos

```
for ( j= 0; j<10; j++ )  
    *( v+j )=j"10.0 ;
```

equivale a :

```
for ( j= 0; j<10; j++ )  
    v[ j ]=j"10.0 ;
```

Array de punteros

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

```
char *puntos [25] /* array de 25 punteros a carácter */
```

(*ptr10) es un puntero, ptr10 es un nombre de variable.

(*ptr10)[1] es un puntero a un array

*(*ptr10)[1] es un puntero a un array de punteros

int *(*ptr10)[] es un puntero un array de punteros de variables int

Aritméticas de Punteros

- + A un puntero se le puede sumar o restar un entero n;

```
int v[10];
```

```
int *p;
```

```
p=v;
```

```
(v+4); /*apunta al 5º elemento*/
```

```
p=p+6; /*contiene la dirección del 7º elemento*/
```

Aritméticas de Punteros

- + A una variable puntero se le puede aplicar el operador ++,

```
float m[20];  
  
float *r;  
  
r=m;  
  
r++; /*contiene la dirección del elemento siguiente*/
```

OJO:Operaciones no validas con punteros

- + No se pueden sumar dos punteros.
- + No se pueden multiplicar dos punteros.
- + No se pueden dividir dos punteros.

Aritméticas de Punteros

```
char alfabeto [ 27 ] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ;  
char *p;  
int i;  
p = &alfabeto[0];  
for (i = 1; i < strlen(alfabeto); i++)  
{  
printf ("\\n%c", * p) ;  
p = p+1;  
};
```

Aritméticas de Punteros

```
char alfabeto [ 27 ] = "ABCDEFGHIJKLMNPQRSTUVWXYZ" ;  
char *p;  
int i;  
p = &alfabeto[0];  
for (i = 1; i < strlen(alfabeto); i++)  
{  
    printf ("%c", *p++);  
}
```

Aritméticas de Punteros

```
char alfabeto [ 27 ] = "ABCDEFGHIJKLMNPQRSTUVWXYZ" ;  
char *p;  
int i;  
p = &alfabeto[ 0 ];  
while ( *p )  
    printf ( "%c" , *p++ );
```

Punteros como argumentos de función

- + Llamada

```
void Incrementari(int *i)  
{  
    *i+=5;  
}
```

- + El paso

```
incrementari(&k);
```

Punteros a Estructuras

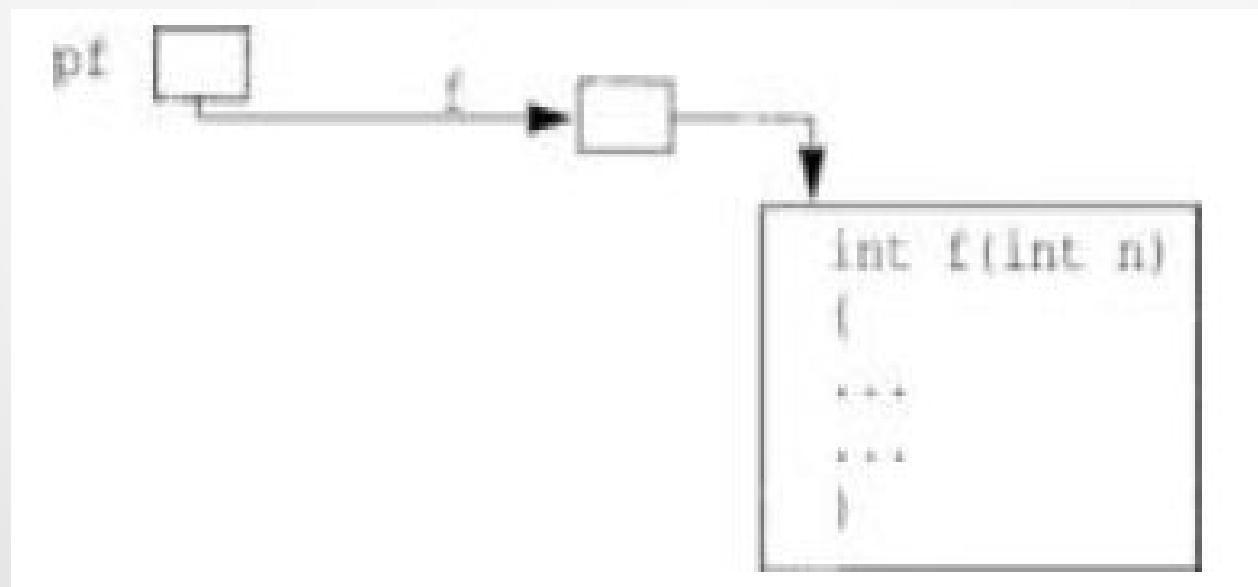
```
struct t_persona {  
    char nombre[30] ;  
    int edad; int altura; int peso; };  
typedef struct t_persona persona;  
void mostrar_persona(persona *ptr);  
void main(){  
    int i;  
    persona empleados [ ] = { {"Mortiér, Pepe", 47, 182, 851},  
    {"García, Luis", 39, 170, 75},  
    {"Jiménez, Tomás", 18, 175, 801} };  
  
    printf ("\nNombre : %s", empleados[1].nombre) ;  
  
    persona *p; /* puntero a estructura */  
    p = empleados;  
    for (i = 0; i < 3; i++, p++)  
        mostrar_persona(p);  
  
    return 0; }
```

```
void mostrar_persona(persona *ptr)  
{  
    printf ("\nNombre : %s",ptr -> nombre) ;  
    printf ("\tEdad: %d ",ptr -> edad);  
    printf ("\tAltura: %d ",ptr -> altura) ;  
    printf ("\tPeso: %d\n",ptr -> peso);  
}
```

Punteros a función

La sintaxis general para la declaración de un puntero a una función es:

Tipo-de-retorno (*PunteroFuncion) (<lista de parámetros>);



Punteros a función

+ Ejemplos :

```
int f(int); /*declara la función f */
```

```
int (*pf)(int); /*define puntero pf a una función con un argumento int */
```

```
pf=f; /*asigna la dirección de f a pf */
```

Punteros a función

+ Utilidad :

Los punteros a funciones también permiten pasar una función como un argumento a otra función.

Para pasar el nombre de una función como un argumento función,

se especifica el nombre de la función como argumento.

Supongamos que se desea pasar la función

miFunc() a la función **sufunc()**.

Funciones como argumentos

- + En C se pueden pasar funciones como argumentos a otras funciones.
- + Cuando una función acepta el nombre de otra como argumento,
la declaración formal debe identificar este argumento como un puntero a otra función.
- + Ejemplo:

Funciones como argumentos

```
#define FALSE 0
```

```
#define TRUE 1
```

```
int sumar(int a, int b)
```

```
{
```

```
return(a+b);
```

```
}
```

```
int restar(int a, int b)
```

```
{
```

```
return(a-b);
```

```
}
```

Funciones como argumentos

```
main() {  
    int a, b, oper, error, res;  
  
    printf("a, b: ");  
  
    scanf("%d %d", &a, &b);  
  
    printf("Operacion (0=suma, 1=resta): ");  
  
    scanf("%d", &oper);  
  
    error = FALSE;  
  
    switch(oper) {  
  
        case 0: res = ejecutar(sumar, a, b); break;  
  
        case 1: res = ejecutar(restar, a, b); break;  
  
        default: error = TRUE; }  
  
    if (!error)  
  
        printf("Resultado = %d\n", res); }
```

```
int ejecutar(int (*f)(int a, int b), int a, int b)  
{  
    int res;  
  
    res = (*f)(a,b);  
  
    return(res);  
}
```