

# Introducción a variables de tipo Puntero (Apuntadores)

## Contenidos

1. Introducción a las variables puntero
2. Repaso:
  - operador de dirección: &
  - referencias
3. Declaración de variables de tipo puntero
4. Inicialización de variables de tipo puntero
  - El puntero nulo: NULL
5. Valor apuntado por un puntero: Indirección de punteros
6. Aritmética de punteros
7. Relación entre Arrays y punteros
8. Cadenas y punteros.
9. Punteros a estructuras.

1

# Introducción a variables de tipo Puntero (Apuntadores)

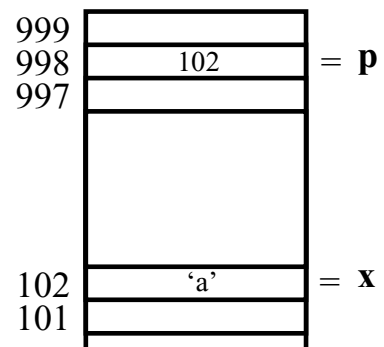
## Introducción

El puntero es una técnica muy potente que hace que la programación C++ sea tan utilizada. La técnica de punteros apoya a la programación orientada a objetos, que es una de las grandes diferencias entre C y C++.

**Definición:** Un *puntero* es una variable que almacena una dirección de memoria.

Hasta ahora hemos visto variables que almacenan datos; los punteros almacenan direcciones de otras variables.

**p** es una variable puntero



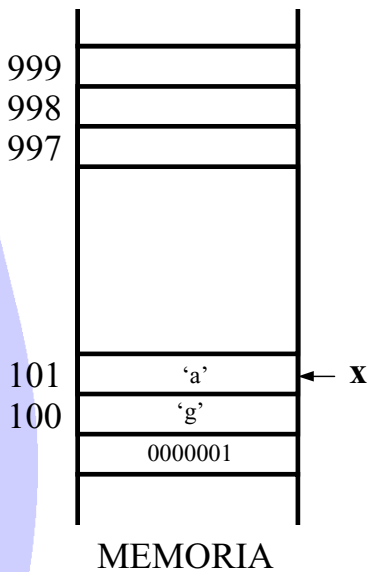
2

# Introducción a variables de tipo Puntero (Apuntadores)

## REPASO: Operador de dirección: &

Cuando se declara una variable, se le asocian tres características:

- nombre de la variable
- tipo de la variable
- dirección de memoria



```
...  
char x = 'a';  
...
```

El valor de la variable **x** es 'a'.  
La dirección de memoria es 101.

Para conocer la dirección de memoria donde se encuentra almacenada la variable, se puede utilizar el operador **&**.

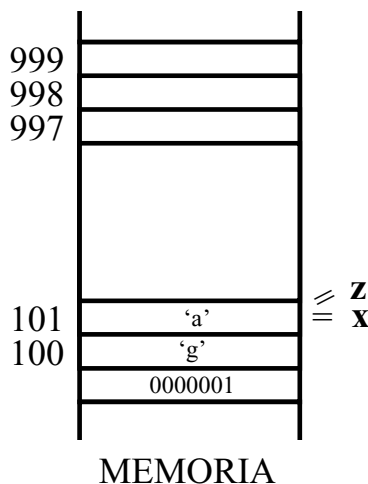
```
...  
cout << x; // Se visualiza el valor: 'a'  
cout << &x; // Se visualiza la dirección: 101  
...
```

# Introducción a variables de tipo Puntero (Apuntadores)

## REPASO: Referencias

Puede ser interesante declarar dos variables con la misma dirección de memoria. Para realizar ésta tarea también utilizamos el operador **&**.

```
<tipo> & <variable> = <variable>;
```



```
#include <iostream.h>
```

```
int main()  
{  
    char x = 'a';  
    char &z = x;  
    cout << x << z;  
    z = 'b';  
    cout << x << z;  
    ...  
    return 0;  
}
```

La dirección de la variable **z** es la misma que la dirección de la variable **x**

// x también vale 'b'

z y x son nombres diferentes para la misma variable

# Introducción a variables de tipo Puntero (Apuntadores)

## Declaración de una variable de tipo puntero

Para declarar una variable de tipo puntero, se utiliza el símbolo \*. Además, la declaración de una variable de tipo puntero debe indicar al compilador el tipo de dato al que apunta el puntero.

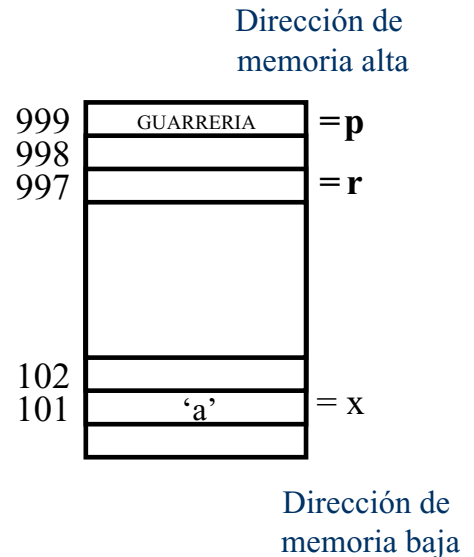
### Sintaxis

```
<tipo> * <nombre_del_puntero> ;
```

Así, podemos hablar de puntero a entero, puntero a char, puntero a float, etc.

```
char *p ;  
int *r ;
```

Como para el resto de variables, C++ no inicializa automáticamente los punteros.



# Introducción a variables de tipo Puntero (Apuntadores)

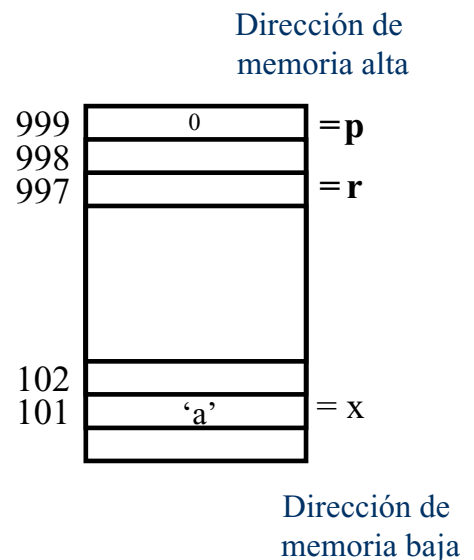
## Inicialización de una variable de tipo puntero

Es preciso inicializar las variables de tipo puntero antes de su uso. La inicialización proporciona al puntero una dirección de memoria a la que apuntar.

### A) Inicializar un puntero a una dirección nula.

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

- El puntero NULL apunta a la dirección 0.
- El puntero NULL no es una dirección de una variable en memoria.
- El puntero nulo proporciona al programa una manera de saber si un puntero apunta a una dirección válida.



## Introducción a variables de tipo Puntero (Apuntadores)

### Inicialización de una variable de tipo puntero

Para poder usar el puntero nulo NULL, podemos hacer una de éstas dos cosas:

- definirlo en la cabecera de nuestro programa:

```
# define NULL 0
```

- incluir alguno de los archivos de cabecera donde ya se encuentra definido:

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

```
# include <string.h>
```

7

## Introducción a variables de tipo Puntero (Apuntadores)

### Inicialización de una variable de tipo puntero

#### B) Inicializar un puntero a una dirección de memoria válida

Para asignar una dirección de memoria a un puntero, se utiliza el operador de dirección &.

El operador & aplicado a una variable, devuelve la dirección donde se almacena dicha variable.

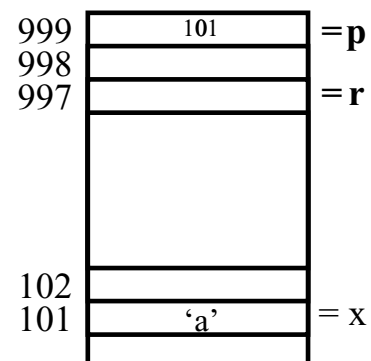
```
&x = 101
```

```
char *p = NULL;  
char x = 'a';  
p = &x;
```

**p apunta a x**



Dirección de memoria alta



8

## Introducción a variables de tipo Puntero (Apuntadores)

### Inicialización de una variable de tipo puntero

#### B) Inicializar un puntero a una dirección de memoria válida

La inicialización también se puede hacer en la declaración.

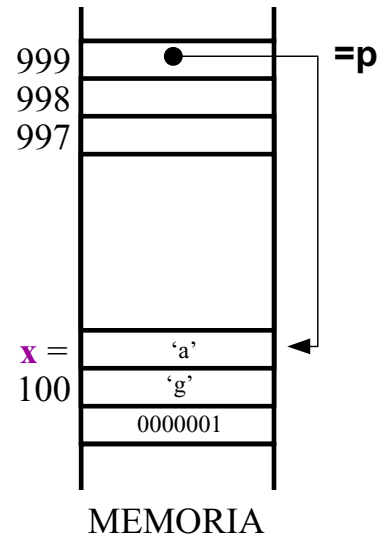
```
char x = 'a';  
char *p = &x;
```

Otra forma de representar punteros

Es importante tener en cuenta que el puntero debe apuntar a una variable del mismo tipo.

```
char x = 'a';  
int *p = &x;
```

Error !



## Introducción a variables de tipo Puntero (Apuntadores)

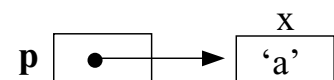
### Indirección de punteros

Una vez definida e inicializada una variable de tipo puntero, disponemos de un mecanismo para obtener el dato que almacena esa dirección de memoria.

El símbolo \* se llama también *operador de indirección* y sirve para acceder al valor al que apunta el puntero.

```
char x = 'a';  
char *p = &x;
```

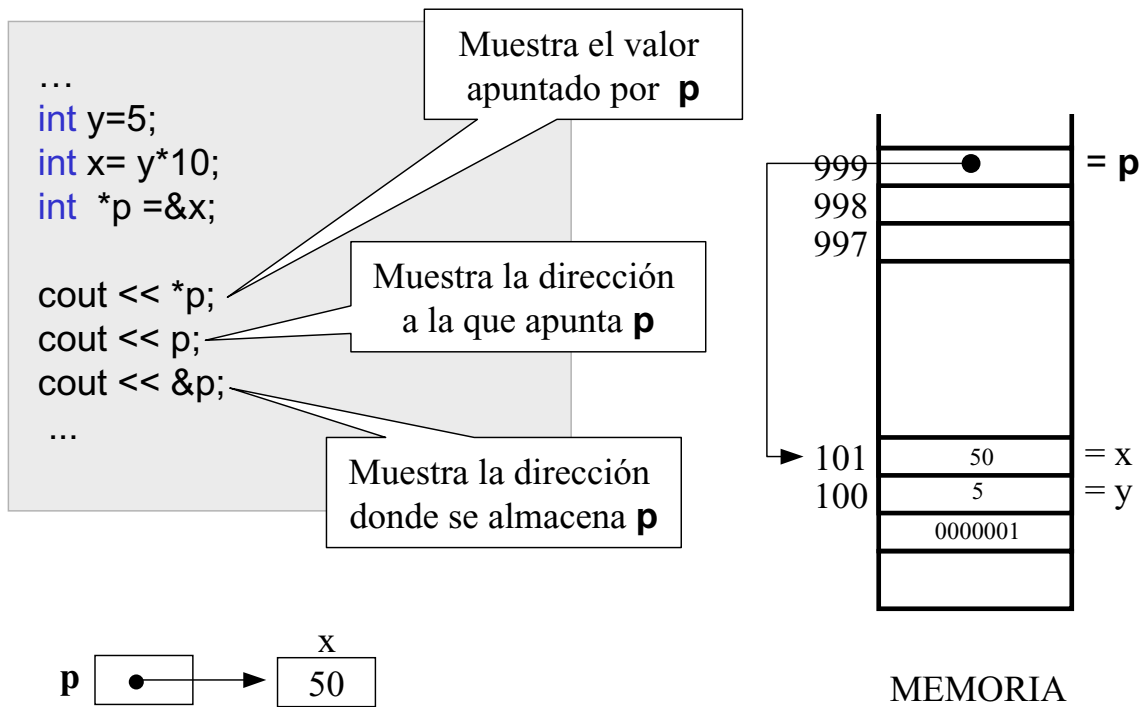
```
cout << *p; // muestra por pantalla la letra 'a'
```



- & **Operador de dirección:** Obtiene la dirección de una variable
- \* Declara una variable puntero.
- \* **Operador de indirección.** Obtiene el valor apuntado por el puntero.

## Introducción a variables de tipo Puntero (Apuntadores)

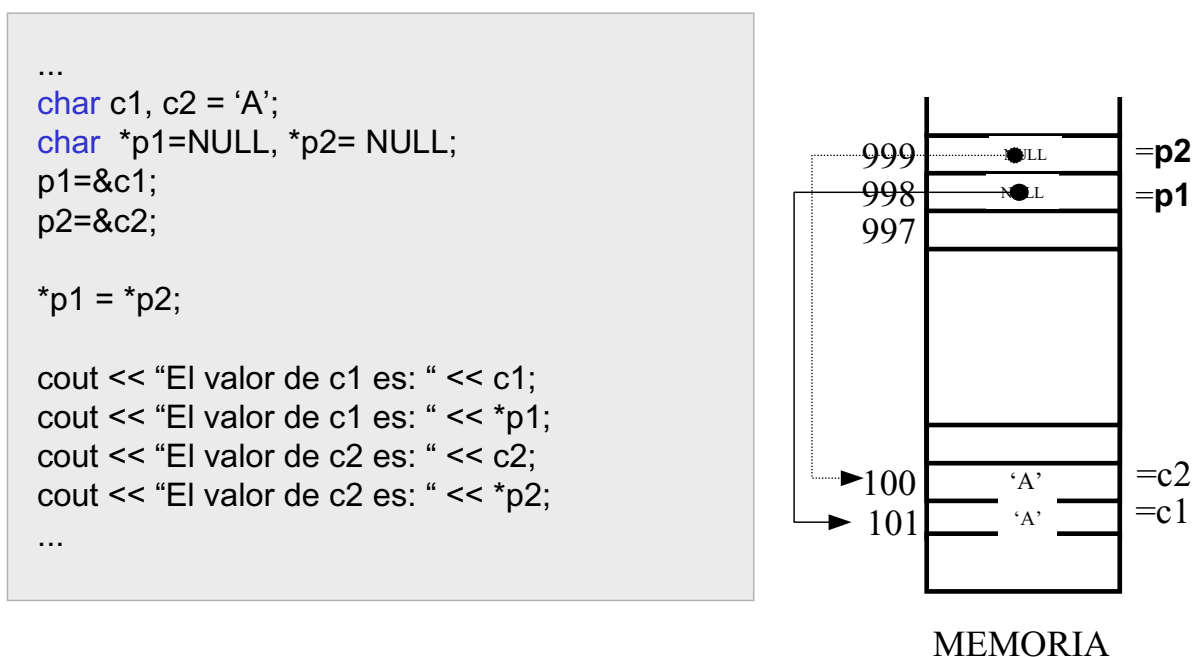
### Ejemplo:



11

## Introducción a variables de tipo Puntero (Apuntadores)

### Ejemplo:

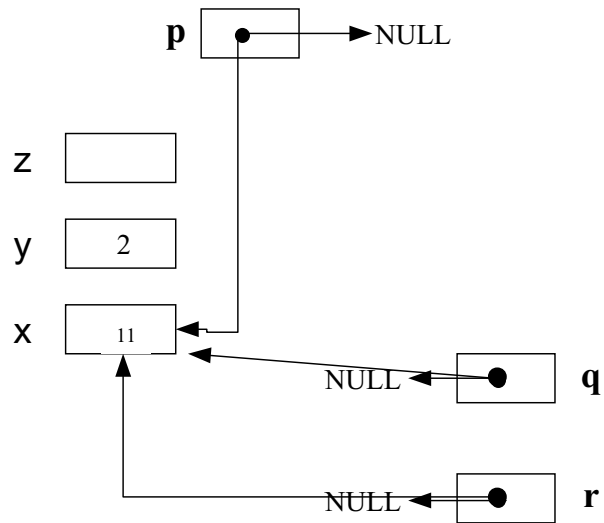


12

## Introducción a variables de tipo Puntero (Apuntadores)

### Ejemplo:

```
...
int *p=NULL;
int x, y, z;
p = &x;
x=2;
y=*p;
*p = 9;
int *q=NULL, *r =NULL;
q=p;
r=p;
*r=*r+y;
...
```



13

## Introducción a variables de tipo Puntero (Apuntadores)

### Ejemplo:

```
int a=10;
int *p;
cout << p;           //valor indefinido
cout << *p;         //valor indefinido
p=&a;                // ahora se inicializa, pero tarde.
cout << p;
cout << *p;
```

```
int a=10;
int *p = NULL;
*p = 9;
```

Es correcto ?

El puntero p se inicializa después de sus 2 primeros usos. Si en vez de hacer una lectura, hubiésemos hecho una escritura, **\*p=9**, el resultado podría haber sido fatal, ya que podríamos haber accedido a posiciones de memoria indebidas.

Hay que tener cuidado con no dejar un puntero sin inicializar

14

## Introducción a variables de tipo Puntero (Apuntadores)

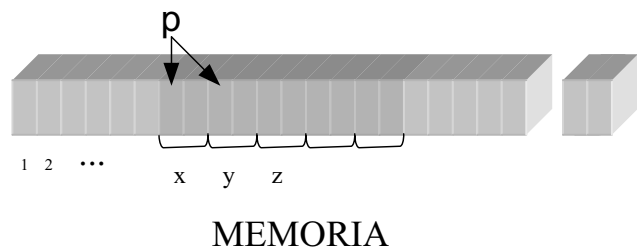
### Aritmética de punteros

Un puntero es una dirección de memoria. Las operaciones permitidas son:

- Incrementar y decrementar punteros,
- Sumar y restar un entero a un puntero

En éstas operaciones el tipo del puntero es fundamental, por ejemplo si el puntero es de tipo int, cuando sea incrementado, lo será en 2 bytes, para que el puntero apunte al siguiente entero, no al siguiente byte.

```
int *p=NULL;
int x, y, z;
p = &x;
p++;
```



## Introducción a variables de tipo Puntero (Apuntadores)

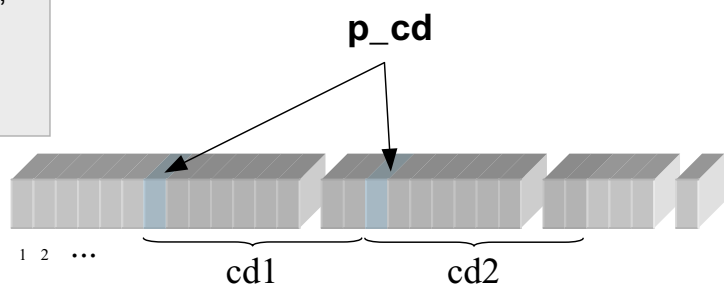
### Aritmética de punteros

De modo similar ocurre con estructuras más complejas. Por ejemplo si definimos la siguiente estructura:

```
struct disco
{
    char titulo[30];
    int num_canciones;
    float precio;
};
disco cd1 = { "El piano", 15, 18},
        cd2 = { "Dirty work", 10, 12};

disco *p_cd = &cd1;
p_cd = p_cd + 1;
```

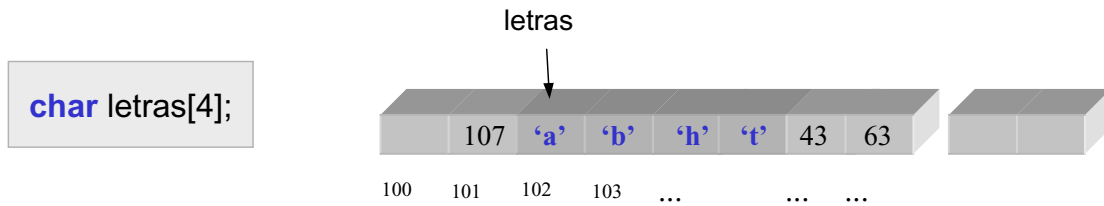
Cada variable de tipo **disco** ocupa 36 bytes de memoria



## Relación entre arrays y punteros

Existe una fuerte relación entre un vector y un puntero.

Un array es un puntero.



Cuando declaramos una variable de tipo array, lo que en realidad estamos haciendo es crear un *puntero constante* cuyo nombre no ocupa memoria.

```
cout << letras[0];  
cout << letras [1];  
cout << letras[3];
```

equivale a :

```
cout << *letras;  
cout << *(letras+1);  
cout << *(letras+3);
```

## Relación entre arrays y punteros. Ejemplo:

```
int v[10]={6,0,5, 7,4,3,2,2,4,9 };  
int *p;
```

(1) `p = v;` El puntero **p** apunta a donde apunta **v**

(2) `p = v+2;`

(3) `p = &v[1];`

`v = p;` **ERROR:** **v** es un puntero constante, y por tanto no puede apuntar a otro sitio

The diagram illustrates the memory stack. At the top, a pointer variable `p` is shown with a black dot representing its value. Below it, a vertical stack of memory cells represents the array `v`. The first cell contains the value 6, the second contains 0, the third contains 5, and the fourth contains 6. A bracket on the right groups the last three cells (0, 5, 6) as `v[1]`. The label `= v` is placed at the bottom of the stack. Three dashed lines with arrows labeled (1), (2), and (3) point to the first, second, and third cells of the array, respectively, corresponding to the code snippets.

### Relación entre vectores y punteros

El nombre del array sin índice se utiliza como la dirección de comienzo en la que está almacenado el array.

La aritmética de punteros se utiliza de forma habitual para acceder secuencialmente a los elementos de una array. Esta forma de acceso es más rápida que utilizar un índice.

Ejemplo:

```
...
int temperatura[ 10 ];
int *p;
p = temperatura;
for (int i = 0; i<10; i++)
{ *p = 0;
  p++;
}
...
```

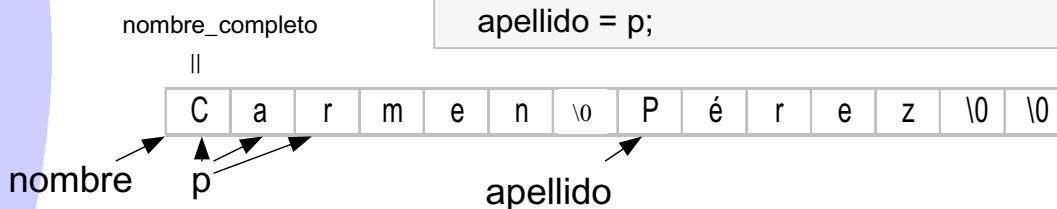
### Cadenas y punteros

También podemos utilizar aritmética de apuntadores para recorrer cadenas.

Ejemplo:

```
char nombre_completo[20] = "Carmen Pérez" ;
char *p;
char *nombre;
char *apellido;

p = nombre_completo;
while ( *p != ' ' )
  p++;
*p = '\0';
nombre = nombre_completo;
p++;
apellido = p;
```



# Introducción a variables de tipo Puntero (Apuntadores)

## Punteros a estructuras

Podemos declarar un puntero a una estructuras tal y como se declara para cualquier otro tipo de dato.

```
struct disco
{
    char titulo[30];
    int num_canciones;
    float precio;
};
disco cd1 = { "El piano", 15, 18};

disco *p =&cd1;
```

Para acceder a los miembros, tenemos dos posibilidades:

A) Utilizando el operador punto (.) sobre el valor apuntado por el puntero

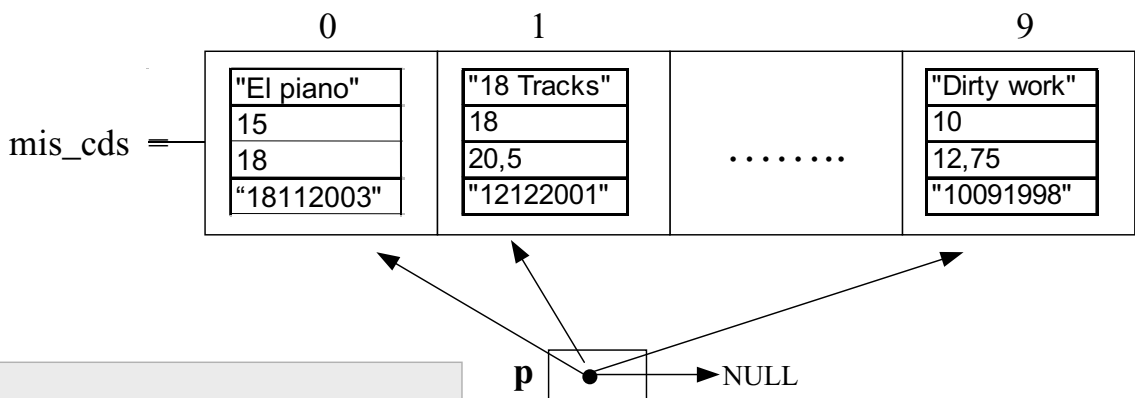
(\*p).titulo

B) Utilizando el operador →

p→titulo

# Introducción a variables de tipo Puntero (Apuntadores)

## Punteros a estructuras



```
disco *p = NULL;
p = mis_cds;
for (int i = 0; i<10; i++)
{
    cout << (*p).titulo;
    cout << (*p).precio;
    p++;
}
```

```
for (int i = 0; i<10; i++)
{
    cout << p->titulo;
    cout << p->precio;
    p++;
}
```

## Introducción a variables de tipo Puntero (Apuntadores)

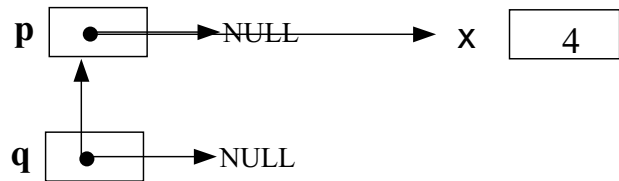
### Puntero a puntero

Un puntero puede apuntar a una variable de tipo puntero.

Esto se utiliza con mucha frecuencia en programas complejos de C++.

Para declararlo ponemos dos asteriscos (\*\*)

```
...  
int x = 4;  
int *p = NULL;  
int **q = NULL;  
p = &x;  
q = &p;  
...  
cout << *p;  
cout << **q;  
...
```



Estas dos instrucciones tienen el mismo efecto

## Introducción a variables de tipo Puntero (Apuntadores)

### Ejemplo:

```
int x = 10, y = 5, z;  
int *p, *q, *r;  
r = &z;  
*r = x;  
q = &x;  
*q = *r + 4;  
p = r;  
*p = *q + *r + y;  
  
cout << x;  
cout << y;  
cout << z;
```

¿Cuál es la salida del siguiente fragmento de código ?

## Resumen

- Un puntero es una variable que puede almacenar una dirección de memoria.
- El operador de dirección `&`, se utiliza para obtener la dirección de memoria de una variable.
- El símbolo `*` se utiliza para declarar una variable de tipo puntero.
- El operador de indirección `*` se utiliza para acceder al valor apuntado por un puntero. Se llama de indirección, porque se accede de manera indirecta al valor almacenado en una dirección de memoria.