Instituto Politécnico Nacional
La Técnica al Servicio de la Patria

Unidad Profesional Interdisciplinaria de Ingeniería Campus Tlaxcala UPIIT

Fundamentos de Programación

Esaú Eliezer Escobar Juárez

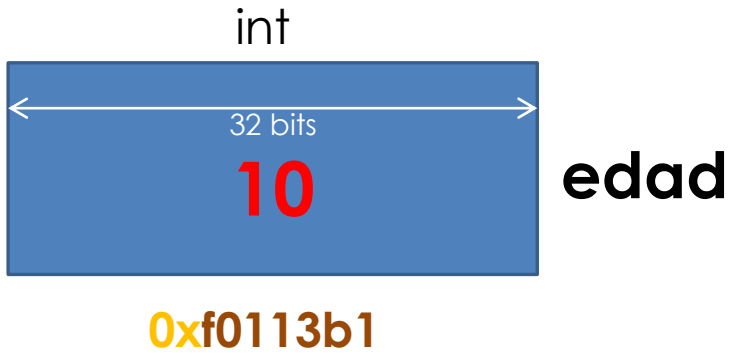
Apuntadores

- Hablar de lenguaje C, es como hablar de apuntadores.
- Este en todas partes del lenguaje y en todas las librerías estándar.
- Casi todo tiene que ver con ellos de una u otra forma.
- Se requiere una buena disciplina para usarlos

Repasando

- La variables tienen 5 elementos que debemos tener presente:

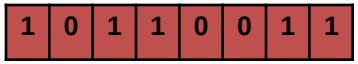
1. Nombre
2. Tipo de dato
3. Tamaño
4. Valor
5. **Dirección de memoria**



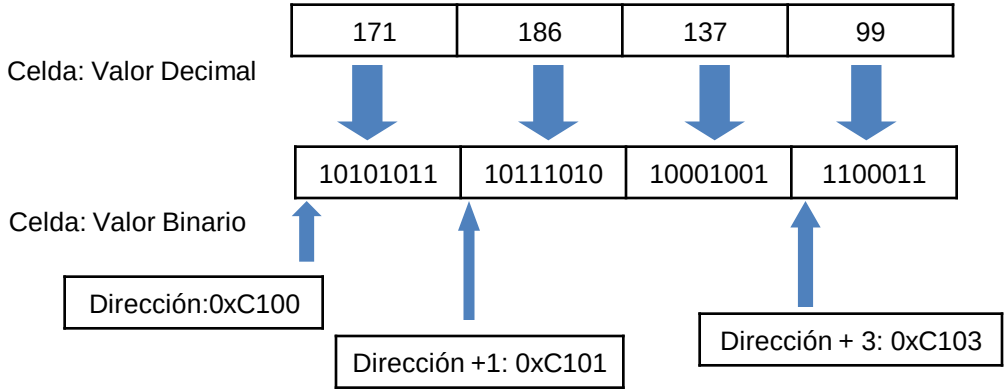
Memoria.

- La memoria es un conjunto de celdas contiguas donde se almacenan datos.

- La unidad de memoria más pequeña es el bit.
- El byte es un conjunto de 8 bits. Unidad de memoria.



- El lugar (ubicación) de cada byte es único y es su dirección.
- Si los bytes son consecutivos la dirección se ira incrementando secuencialmente.



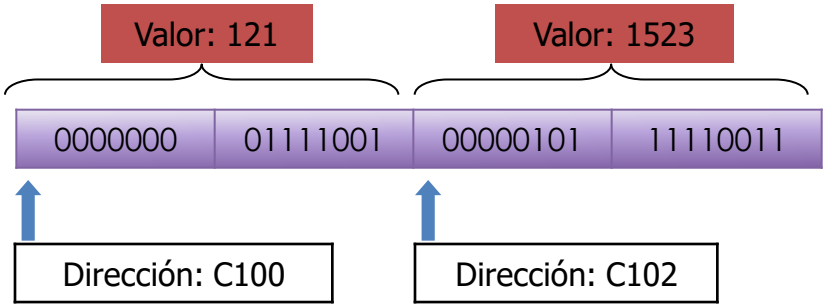
- Cada celda tiene dos valores asociados: Dirección y Contenido.

Variables

- Una variable es una porción de memoria identificada por un nombre.
- El tipo de dato define su representación binaria y longitud.
- Tiene tres valores asociados: Nombre, Contenido y Dirección.

- Declaramos: “int n=1523, m=121;”

- Contenido:
  - n ⇔ 1523
  - m ⇔ 121
- Dirección:
  - &n ⇔ C102
  - &m ⇔ C100



- Es contenido binario se codifica (bits) de acuerdo al tipo de variable.

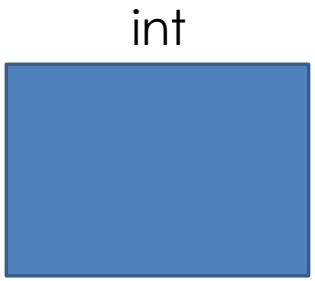
Tipos de Variables

- El tipo de variable establece:
  - Rango de valores que maneja.
  - Codificación binaria del dato.
  - Cantidad de memoria requerida.

Declaración	Bytes	Valor en Memoria Hexa	Valor en Memoria Binario
char c = 'X';	1	58	1011000
char w[ ] = "hola";	4+1	68 6F 6C 61 00	01101000 01101111 01101100 01100001 00000000
int n[5] = {555,444,333,222,111}	5x2=10	02 2B 01 BC 01 4D 00 DE 00 6F	00000010 00101011 00000001 10111100 00000001 01001101 00000000 11011110 00000000 01101111
long l[3] = {100,200,300}	3x4=12	00 00 00 64 00 00 00 C8 00 00 01 2C	00000000 00000000 00000000 01100100 00000000 00000000 00000000 11001000 00000000 00000000 00000001 00101100
float f = 3.141592	4	40 49 0F D8	10000000100100100001111111011000

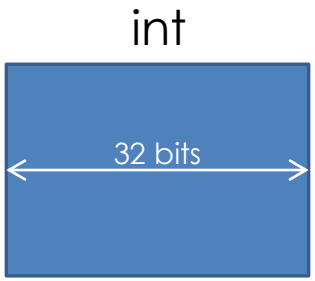
- En todos los casos la dirección de almacenaje es "&variable".

Usando variables



- **¿Cómo acceder a los elementos de una variable?**
- El tipo int y el nombre edad, no pueden ser accedidos .
- Somos responsables de conocer las variables que usamos y sus tipo de datos
- El tamaño, valor y dirección si pueden ser conocidos y accedidos.
- Saber como conocerlos es el primer paso para entender los punteros

Usando variables



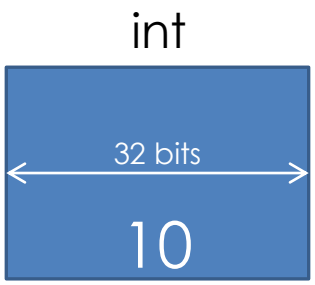
edad

- **¿Cómo saber el tamaño de una variable?**
- La función sizeof() nos el tamaño en bytes de una variable.
- Para saber el número de bits multiplicamos por 8.

```
int edad;
sizeof(edad);
```

Dev C++ 19

Usando variables

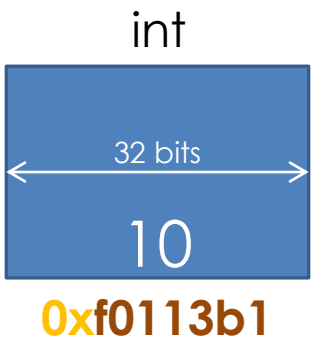


edad

- **¿Cómo ver o alterar el valor de una variable?**
- Por medio de su nombre.

```
int edad;
edad = 10;
printf("%d",edad);
```

Usando variables



edad

- **¿Cómo saber la dirección de memoria de una variable?**
- Utilizamos el signo de ampersand(&)

```
int edad=10;
printf("Dirección de int edad
        %p\n",&edad);
```

¡Atención!

Las direcciones de memoria
Cambian con cada ejecución

Dev C++ 20

Especificadores de formato de printf y scanf

Especificador	Descripción
%c	Carácter
%d	Número entero(int)
%i	Número entero(int)
%D	Número entero long(o también %ld)
%f	Punto flotante(float)
%e	Notación científica con e minúscula
%E	Notación científica con E mayúscula
%g	Formato para tipo punto flotante(float)
%G	Formato para tipo punto flotante(float)
%o	Número octal sin signo
%s	Cadena de texto
%u	Entero sin signo
%U	Entero sin signo long(o también %lu)
%x	Hexadecimal sin signo con minúsculas
%X	Hexadecimal sin signo con mayúsculas
%p	Puntero, dirección de memoria
%n	Número de caracteres
%o	Formato entero octal
%O	Formato entero octal long(o también %lo)
%lf	Formato double
%LF	Formato long double
%l	Formato double
%h	Formato double
%L	Formato long double

```c
#include <stdio.h>

int main()
{
   char nombre[8] = "Timoteo";
   int hermanos = 2, sobrinos = 4;

   printf( "%s tiene %d hermanos y %d sobrinos.",
           nombre, hermanos, sobrinos );

   return 0;
}
```

por pantalla se verá:

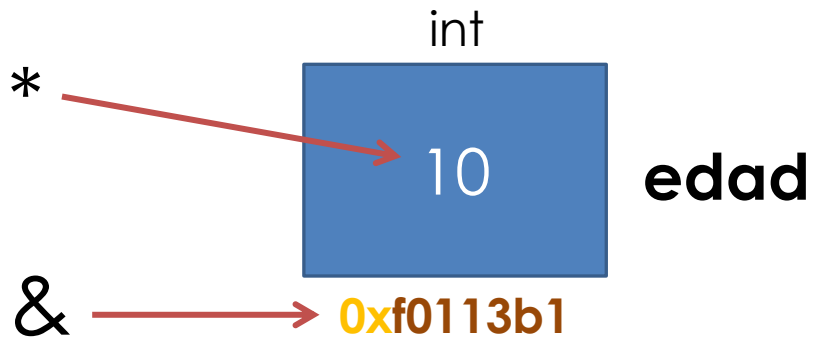
```
Timoteo tiene 2 hermanos y 4 sobrinos.
```

Los operadores & y *

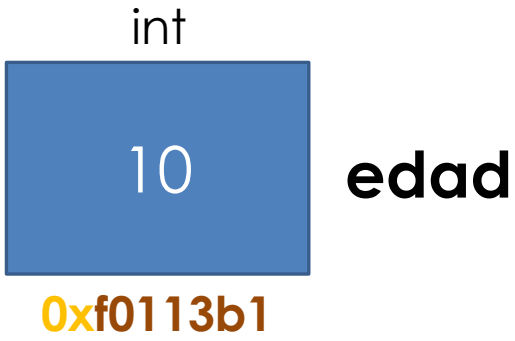
- **¿Qué significan estos símbolos?**

  & = referencia o dirección de memoria

  * = dereferencia, indirección, Valor en la dirección de memoria, valor apuntado



Los operadores & y *

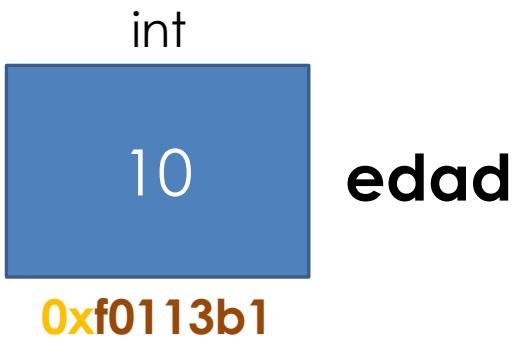


&edad ⟶ **0x7f5ff0ac**

Se pueden usar con cualquier objeto en memoria (variables, estructuras, arreglos, funciones), no aplican a expresiones o constantes

El operador & nos devolverá la dirección de memoria.

Los operadores & y *



&edad ⟶ **0x7f5ff0ac**

***0x7f5f0ac** ⟶ **10**

El operador * o indirección permite acceder al valor guardado en una dirección de memoria.

Es necesario que * se aplique a una dirección de memoria

Los operadores & y *

&edad ⟶ **0x7f5ff0ac**

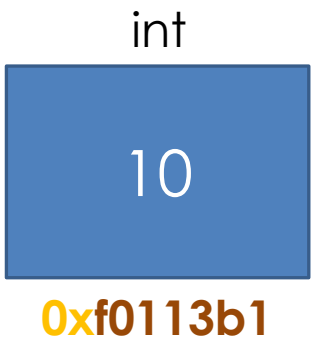
***&edad** ⟶ **10**

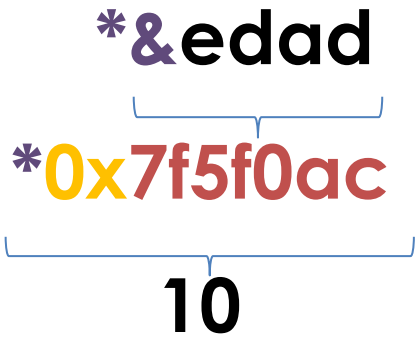
***0x7f5f0ac** ⟶ **10**

Podemos combinar ambos operadores.

Debido a que las direcciones se asignan en tiempo de ejecución, no es sencillo saber que dirección ocupará una variable.

Los operadores & y *





A nadie se le ocurriría utilizar la dirección de memoria en el código fuente.

Tampoco usar *&edad, para que si podemos usar simplemente el nombre de la variable

```
edad=10;              //Este es el mejor modo
*&edad=10;            //Nadie utilizaría esto
*0x7f5f0ac=10;        //Tampoco esto
```

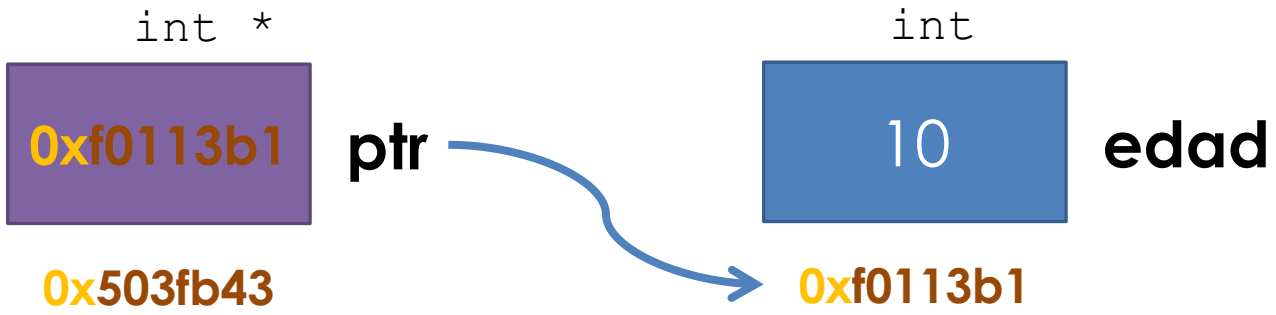
¿Qué es un apuntador?

¿Si nadie usaría ***0x7f5f0ac**, entonces que usamos?

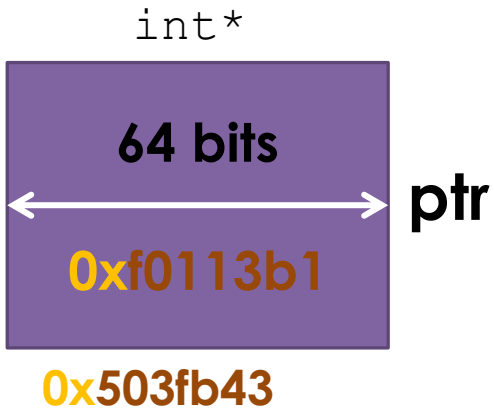
Un apuntador soluciona ese problema.

Conocer las direcciones de memoria brinda lo que hace al lenguaje C tan potente y requerido: Su velocidad.

Un apuntador es una variable especial capaz de guardar una dirección de memoria de tal forma que no tenemos que recurrir a escribirlas dentro del código.



¿Qué es un apuntador?

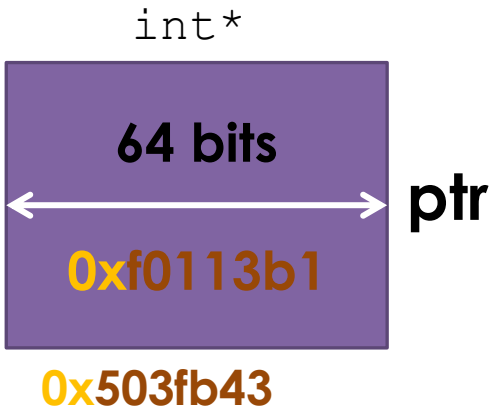


Un puntero es una variable, y tiene todos los elementos de cualquier variable

Su tipo de dato debe ir acompañado de *

Su tamaño debe ser suficientemente grande para guardar direcciones (8 bytes)

¿Qué es un apuntador?



Declaración de un puntero

int *ptr = &edad;

*ptr = 19;

Esto es lo que presenta dificultades para el programador, pues el operador * se usa para diferentes cosas.

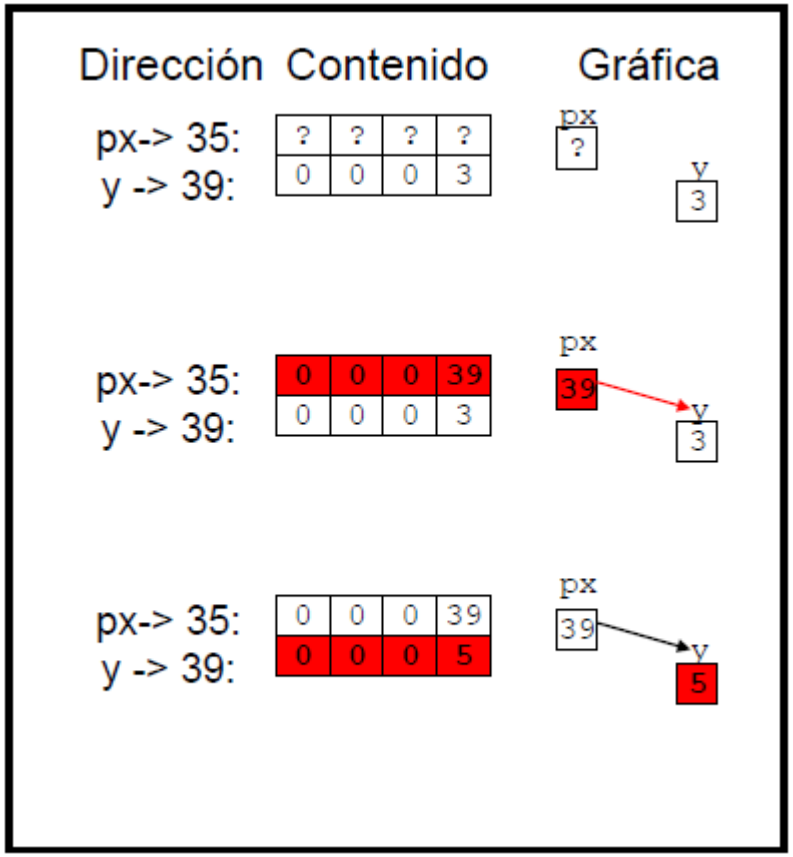
Dev C++ 22

Apuntadores

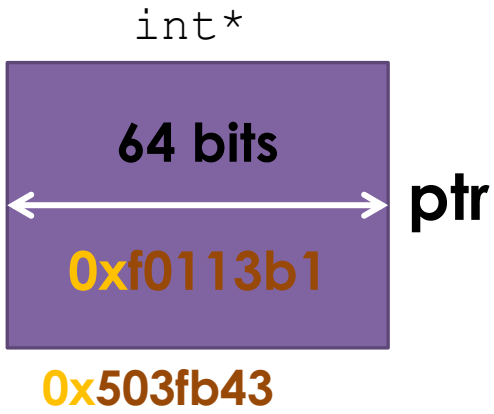
```
int main()
{
        int *px,y=3;

        px=&y;
        /* px apunta a y */

        *px=5;
        /* y vale 5 */
}
```



Apuntadores



¿Por qué los punteros tienen fama de ser complicados?

Errores comunes:

- Punteros no inicializados
- Asignación erronea
- Punteros con tipos distintos
- Uso incorrecto de punteros nulos

Apuntadores

Punteros no inicializados

Su valor debe ser la dirección de memoria de una variable o nulo antes de usarse

```
int edad = 10;
int *p;
*p = 19;
```

Si se usa produce una violación de segmento y el programa se interrumpe

Apuntadores

Inicialización de puntero errónea

Otro error común es olvidar colocar el signo & al inicializar un puntero

```
int edad = 10;
int *p = edad;
*p = 19;
```

La dirección a la que apunta p es 0xa, intentamos cambiar la dirección 0xa a 19, se produce el error.

Apuntadores

Punteros con tipos de datos distintos

Los punteros deben se declarados según el tipo de dato al que apuntaran.

```
char edad = 10;
int *p = &edad;
*p = 19;
```

ptr usará 4 bytes y no 1, lo que provoca una violación de segmento.

Apuntadores

Uso incorrecto de punteros nulos

Puntero nulificado

C permite nulificar un puntero para evitar acceder de forma accidental a otra dirección de memoria fuera de nuestro segmento permitido.

```
int edad = 10;
int *p = NULL;
*p = 19;
```

Primero hay que tener una dirección valida.

ASIGNACION DE PUNTEROS

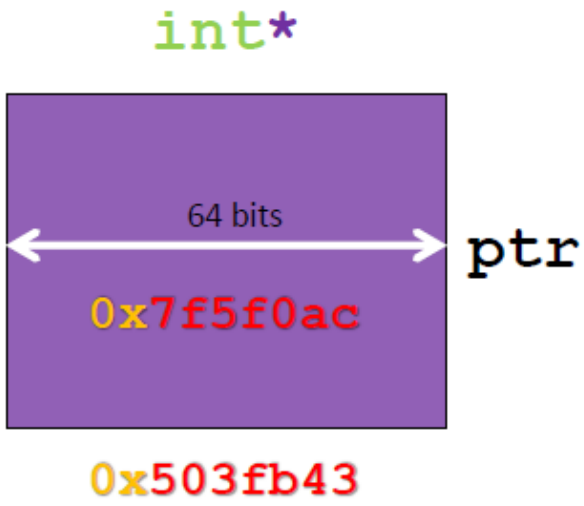
La clave en el uso de todo puntero es su correcta asignación (inicialización).

La asignación de punteros es una operación que permite indicar que dirección de memoria tomará un puntero determinado.

Hay 2 tipos de asignación de punteros

- Asignación de variable a puntero
- Asignación de puntero a puntero

La primera es la habitual, que permite apuntar a una variable determinada.



ASIGNACION DE PUNTEROS

Asignación de variable a puntero

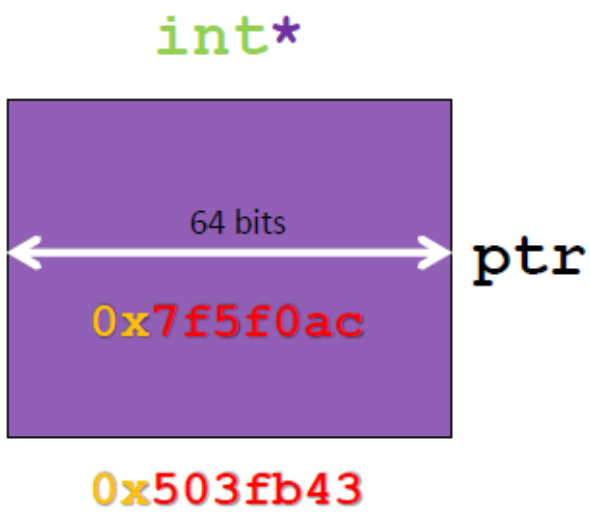
Para asignar una variable a un puntero podemos usar la declaración del puntero

```
int edad = 10;
int* ptr = &edad;
```

Sin embargo, también podemos usar esta forma

```
int edad = 10;
int* ptr;
ptr = &edad;
```

La 2da forma no se recomienda por ser peligrosa



Asignación de puntero a puntero

Más de un puntero puede apuntar a la misma variable

```
int edad = 10;
int *ptr1 = &edad;
int *ptr2 = ptr1;
*ptr2 = 19;
```

Lo que cambia en un puntero también cambia en el otro.

OPERACIONES CON LOS PUNTEROS

– Incremento, decremento: los valores de tipo puntero se pueden incrementar y decrementar, siempre en valores enteros. Se admiten los operadores '+' , '–' , '++' y '––'.

- Incremento de apuntadores

```
int *dir;
....
dir + 1; /* Ok */
dir += 10; /* incrementa dir para apuntar 10
  elmtos. mas adelante */
```

- Cuando se incrementa un apuntador se incrementa en un bloque de memoria (depende del tipo). Ejem: `p++;`
  - Un apuntador a carácter sumará un byte a la dirección.
  - Un apuntador a entero o float sumará 4 bytes a la dirección

OPERACIONES CON LOS PUNTEROS

Aritmética de punteros:

Sustracción de punteros:

- 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.

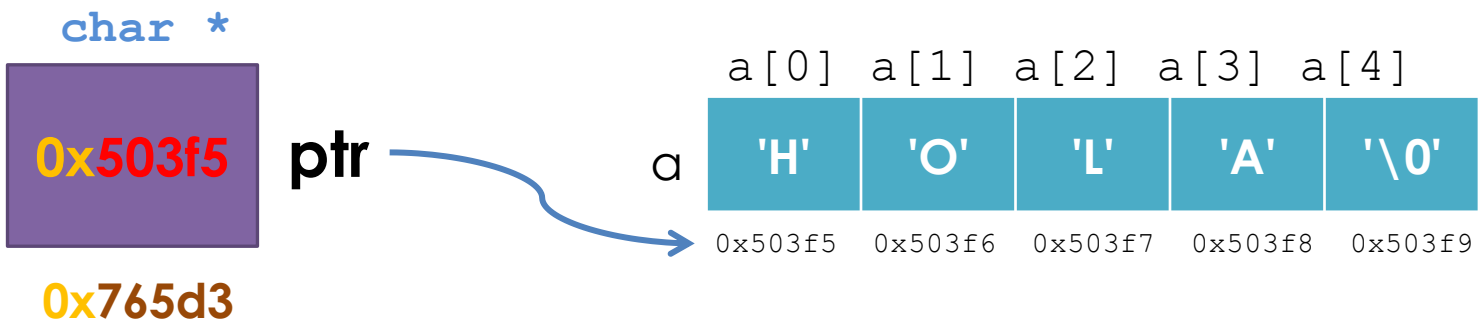
OPERACIONES PROHIBIDAS CON PUNTEROS

- SUMAR PUNTEROS
- MULTIPLICAR PUNTEROS
- DIVIDIR PUNTEROS

Punteros y Arreglos

- Relación estrecha entre apuntadores y arreglos.
- Lo que puede ser realizado con arreglos se puede hacer con apuntadores.

```
char a[4+1] = "HOLA";
char *ptr = &a[0];
```



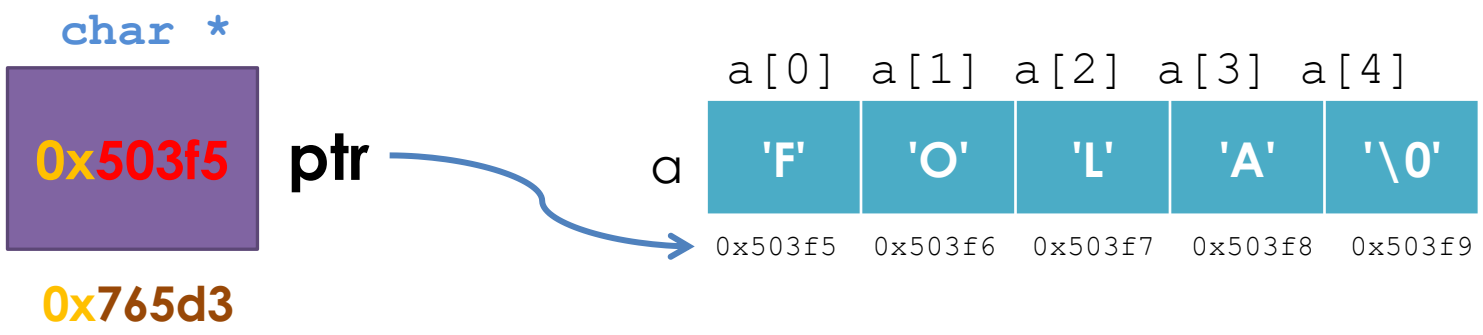
Punteros y Arreglos

- Operación con punteros

```
char a[4+1] = "HOLA";
char *ptr = &a[0];
*ptr = 'F';
```

- Equivalente en arreglo

```
char a[4+1] = "HOLA";
a[0] = 'F';
```



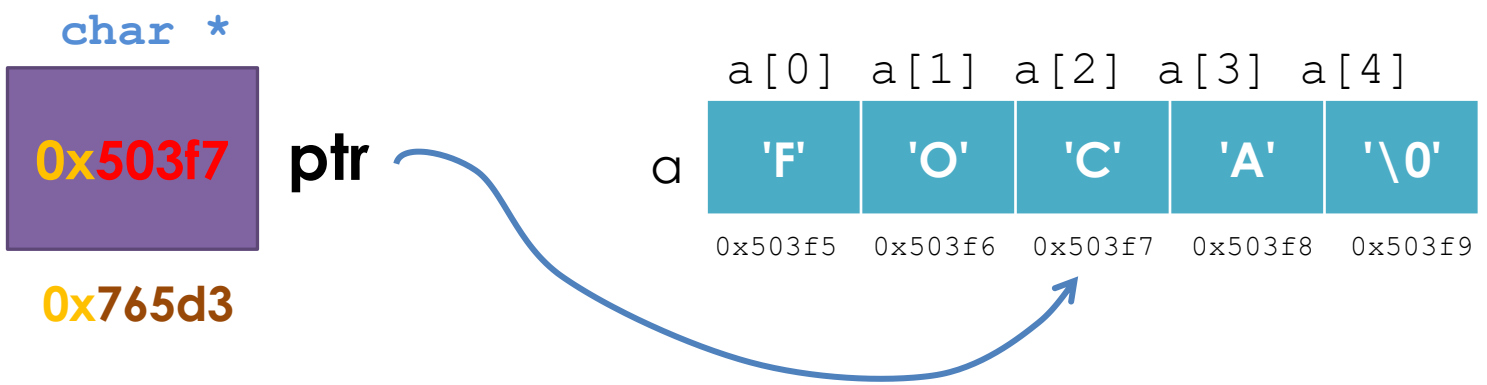
Punteros y Arreglos

- Operación con punteros

```
char a[4+1] = "HOLA";
char *ptr = &a[0];
*ptr = 'F';
ptr = ptr+2;
*ptr = 'C';
```

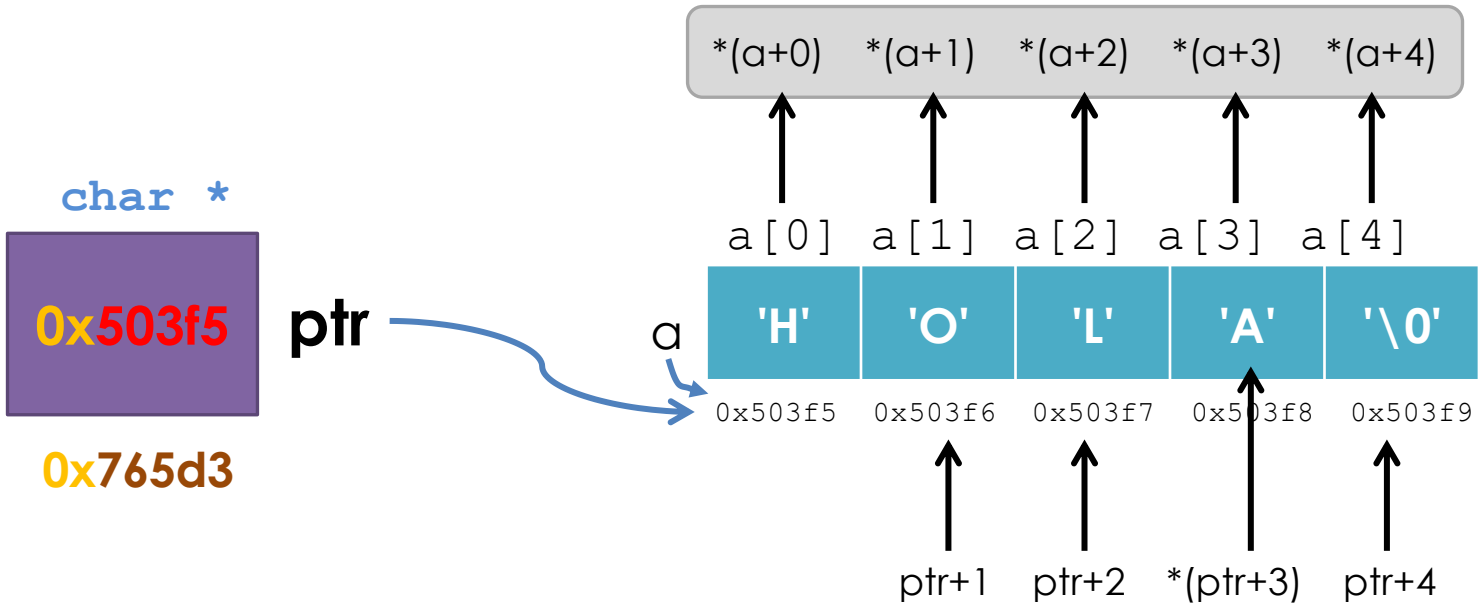
- Equivalente en arreglo

```
char a[4+1] = "HOLA";
a[0] = 'F';
a[2] = 'C';
```



Punteros y Arreglos

- a tiene el mismo valor que ptr
- ptr = a es equivalente a ptr = &a[0]
- a[i] puede ser escrita como *(a+i)



Punteros y Arreglos

- La diferencia entre un puntero y un arreglo es que ptr es una variable pero a no.
- No se permite a++ o a=ptr.
- Un nombre de un arreglo es un apuntador constante.