

Punteros

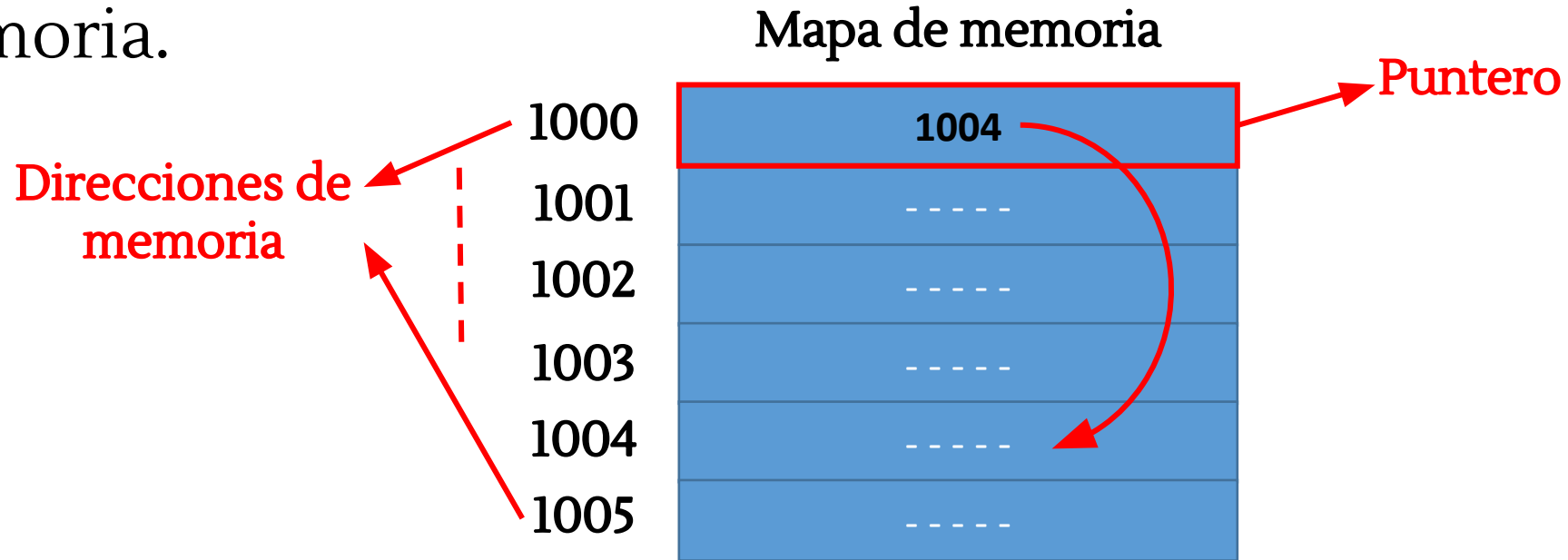
Mg. Ing. Facundo S. Larosa

Informática I

Instituto Universitario Aeronáutico
Universidad de la Defensa Nacional (UNDEF)

¿Qué es un puntero?

Un puntero es una variable destinada a guardar una dirección de memoria.



Así, el puntero de la imagen se dice que “apunta” a la dirección 1004. A través de los punteros se puede manipular directamente la memoria.

¿Cómo se declara un puntero?

La sintaxis de declaración de un puntero es:

```
[tipo de datos base] * [nombre del puntero];
```

Por ejemplo:

```
int    * p; // p es un puntero a entero  
float * q; // q es un puntero a flotante  
char  * m  // m es un puntero a char
```

El tipo de datos base modifica el funcionamiento de algunas de las operaciones realizadas con el puntero.

Operadores básicos

Existen dos operadores básicos del lenguaje que se utilizan frecuentemente con punteros:

- ***** (Operador de indirección): Se define como “**el contenido de lo apuntado por ...**”. No confundir el operador con el utilizado para la declaración del puntero.
- **&** (Operador de dirección): Se define como “**la dirección de memoria de ...**”

Ejemplo básico

//Analizamos el siguiente programa

```
int variable=5;
```

```
int * puntero;
```

```
puntero = &variable;
```

```
printf("variable=%d",variable);
```

```
printf("*puntero=%d",*puntero);
```

```
*puntero=*puntero+1;
```

```
printf("variable=%d",variable);
```

```
printf("*puntero=%d",*puntero);
```

Ejemplo básico

```
//Creo una variable entera y un puntero
```

```
//El puntero está "descontrolado" (no inicializado)
```

```
int variable=5;
```

```
int * puntero;
```

```
puntero = &variable;
```

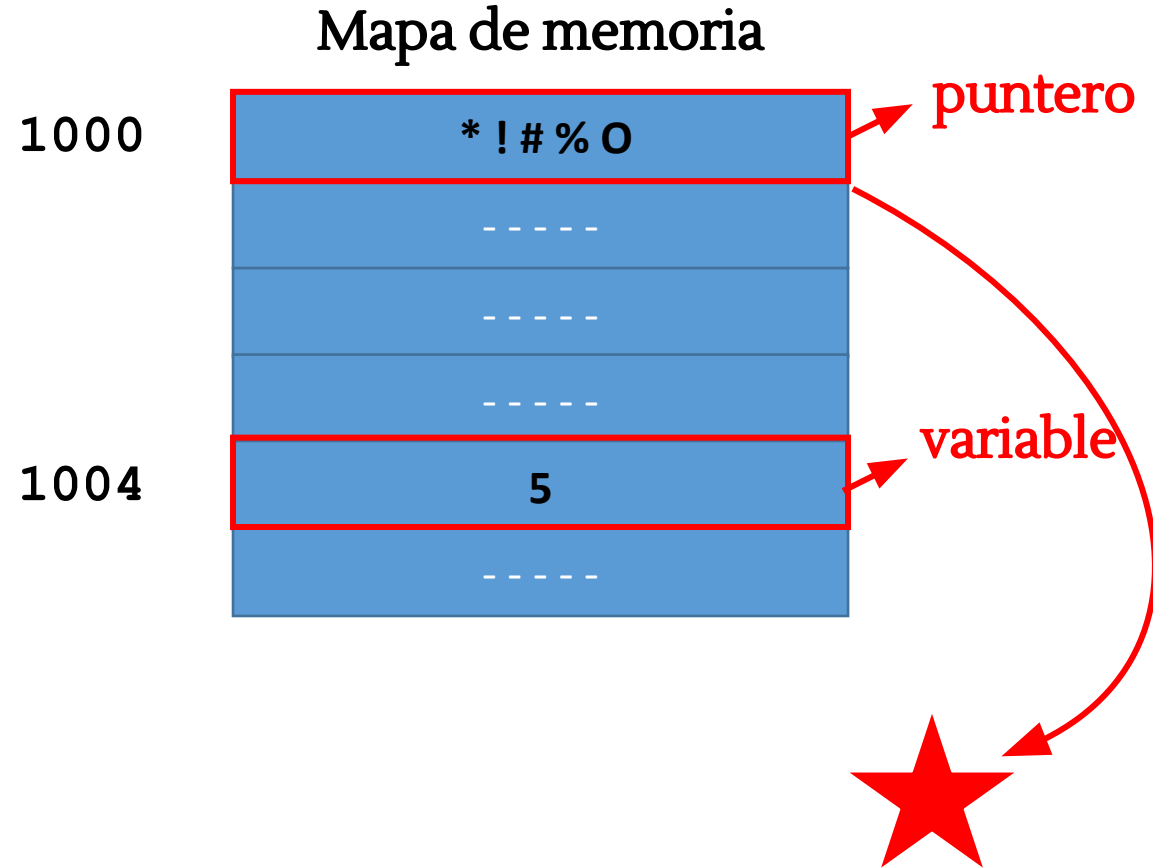
```
printf("variable=%d",variable);
```

```
printf("*puntero=%d",*puntero);
```

```
*puntero=*puntero+1;
```

```
printf("variable=%d",variable);
```

```
printf("*puntero=%d",*puntero);
```



Ejemplo básico

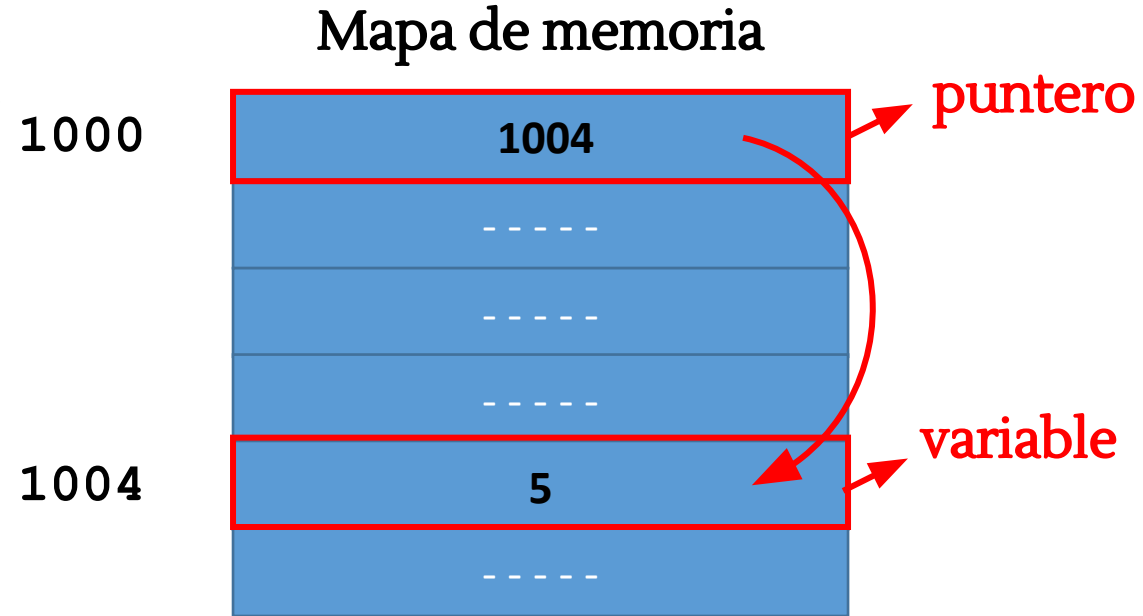
```
int variable=5;  
int * puntero;
```

```
// "Apunto" el puntero a 'variable' 1000  
puntero = &variable;
```

```
printf("variable=%d", variable);  
printf("*puntero=%d", *puntero);
```

```
*puntero=*puntero+1;
```

```
printf("variable=%d", variable);  
printf("*puntero=%d", *puntero);
```



Ejemplo básico

```
int variable=5;  
int * puntero;
```

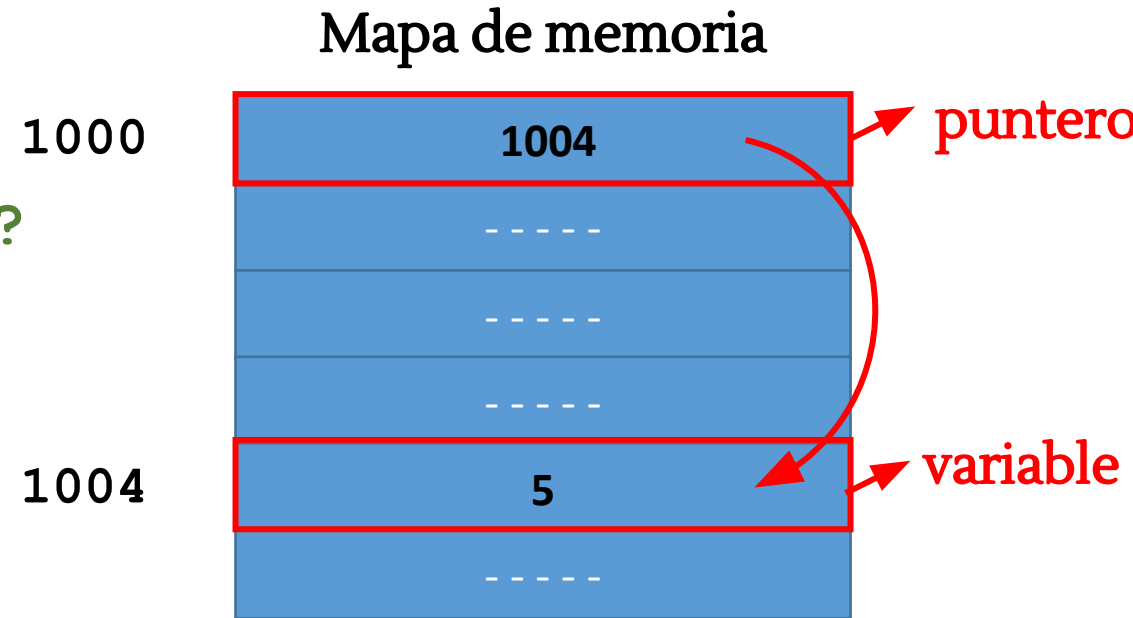
```
puntero = &variable;
```

```
//¿Qué imprime el siguiente código?
```

```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```

```
*puntero=*puntero+1;
```

```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```



Ejemplo básico

```
int variable=5;  
int * puntero;
```

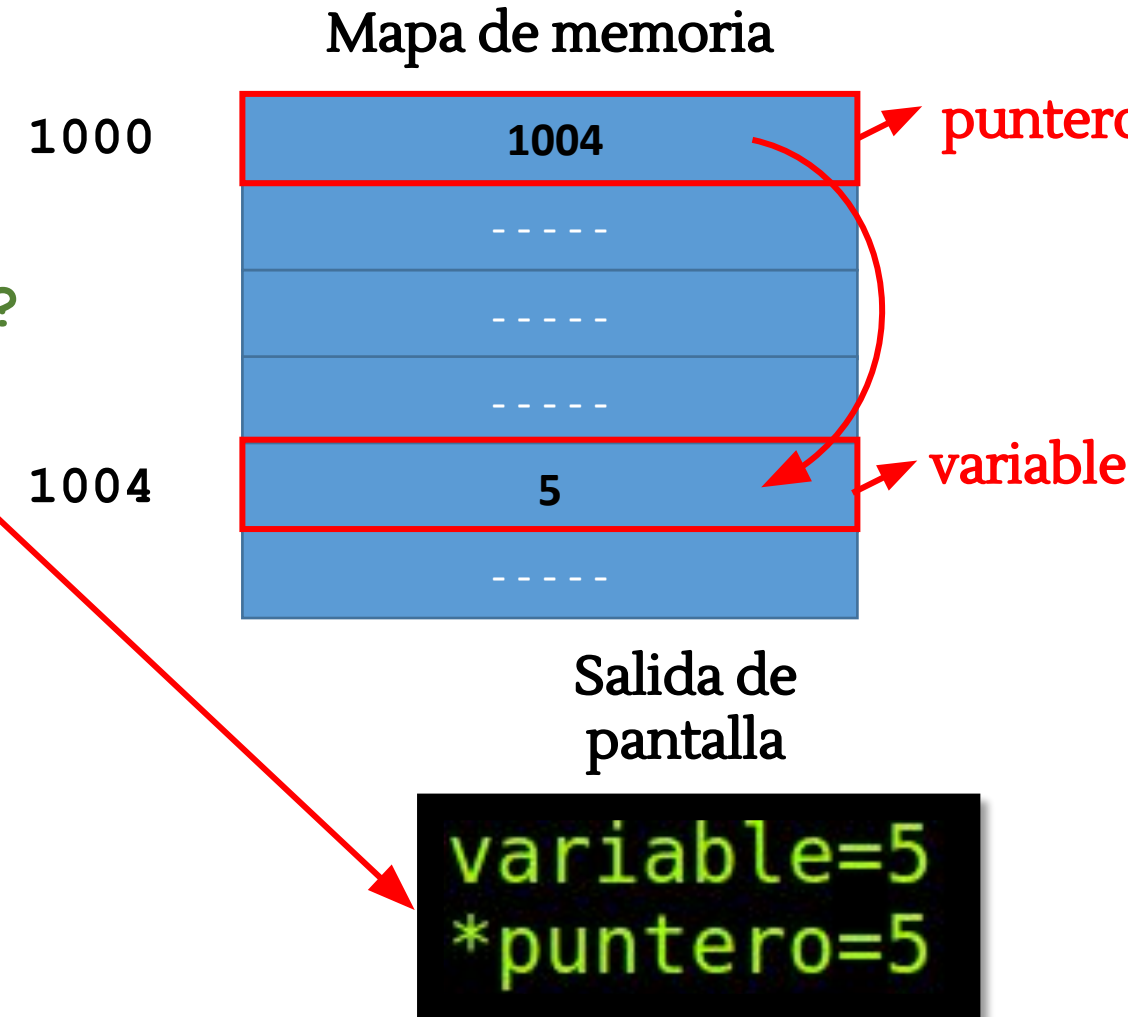
```
puntero = &variable;
```

//¿Qué imprime el siguiente código?

```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```

```
*puntero=*puntero+1;
```

```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```



Ejemplo básico

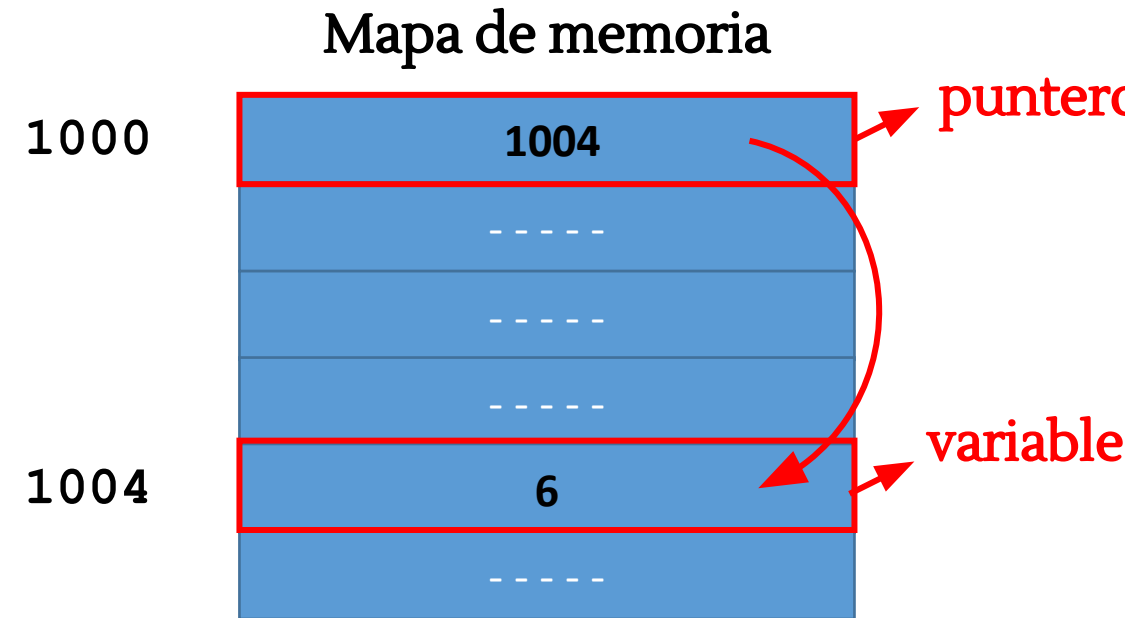
```
int variable=5;  
int * puntero;
```

```
puntero = &variable;
```

```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```

```
//El contenido de lo apuntado  
por 'puntero' se incrementa  
*puntero=*puntero+1;
```

```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```



Ejemplo básico

```
int variable=5;  
int * puntero;
```

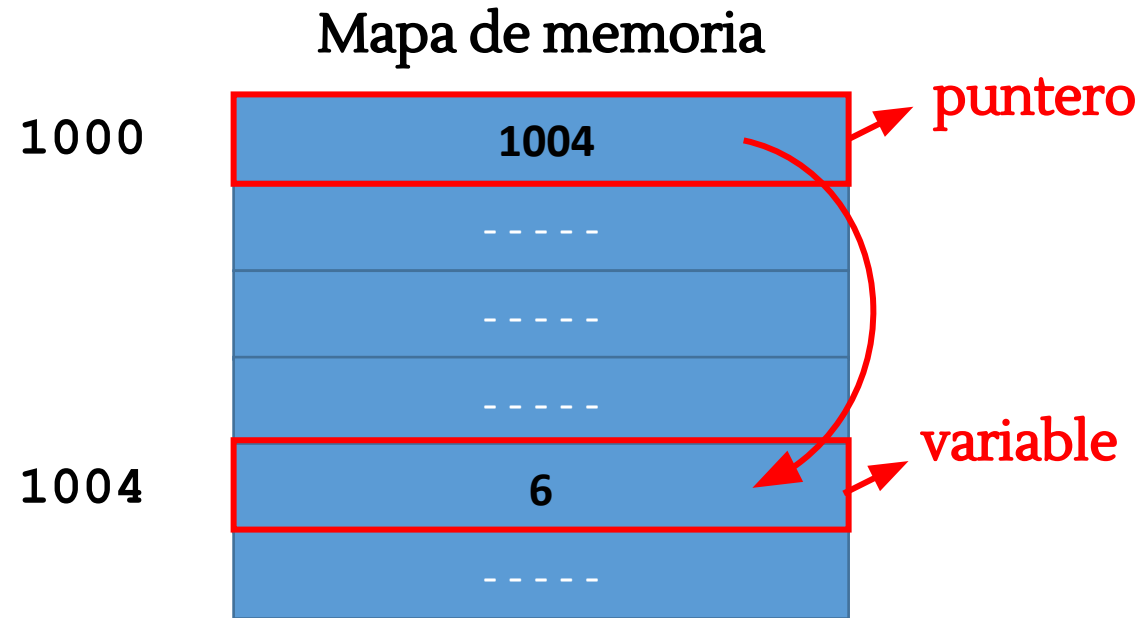
```
puntero = &variable;
```

```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```

```
*puntero=*puntero+1;
```

//¿Qué imprime el siguiente código?

```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```



Ejemplo básico

```
int variable=5;  
int * puntero;
```

```
puntero = &variable;
```

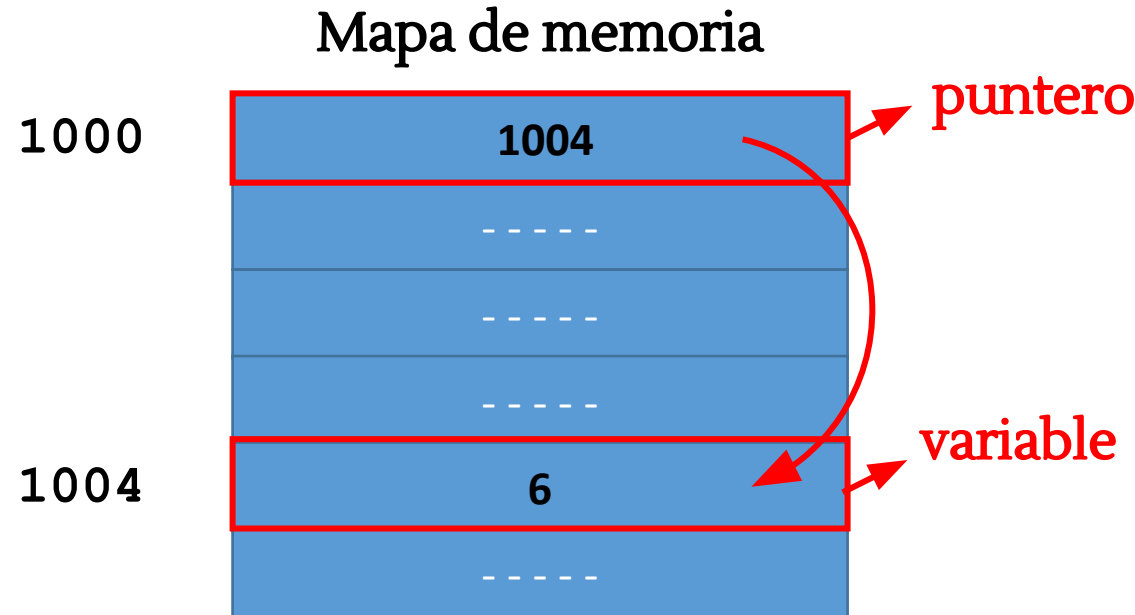
```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```

```
*puntero=*puntero+1;
```

//¿Qué imprime el siguiente código?

```
printf("variable=%d",variable);  
printf("*puntero=%d",*puntero);
```

¿Qué conclusiones podemos obtener?



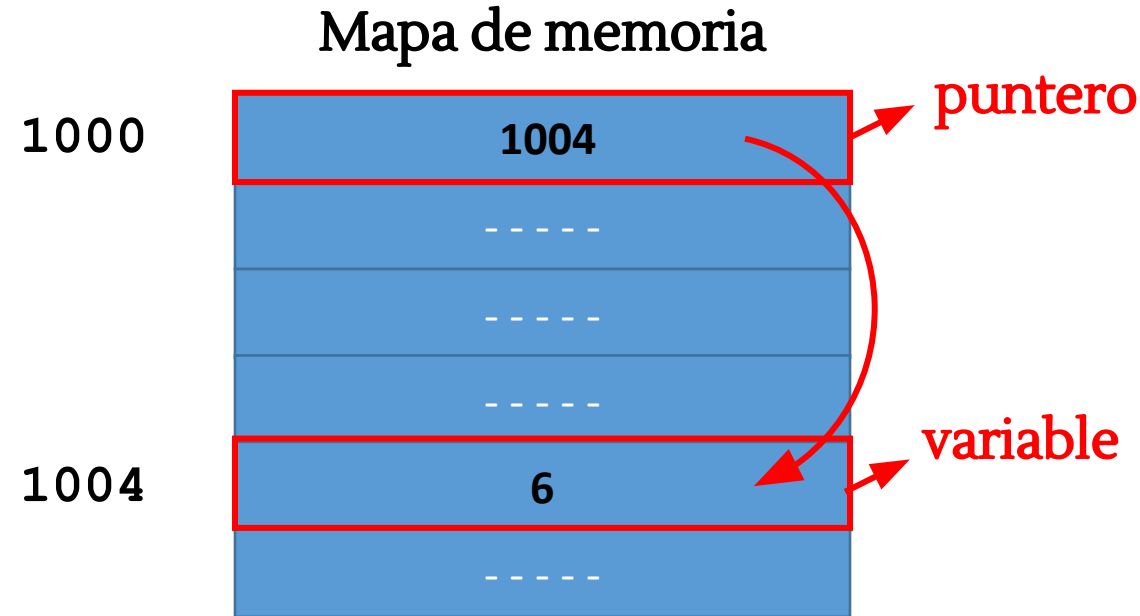
Salida de
pantalla

```
variable=6  
*puntero=6
```

Ejemplo básico



Dame la dirección de la variable, yo me encargo...



//¿Qué imprime el siguiente código?

```
printf("variable=%d", variable);  
printf("*puntero=%d", *puntero);
```

¿Qué conclusiones podemos obtener?

Salida de

```
variable=6  
*puntero=6
```

Conclusiones del ejemplo básico

- El puntero se utilizó para “apuntar” a una variable
- A través del puntero, el contenido de la variable se puede modificar directamente
- Se puede afirmar que en el ejemplo:

`*puntero ≡ variable`

es decir, que `*puntero` es un **alias** de `variable`

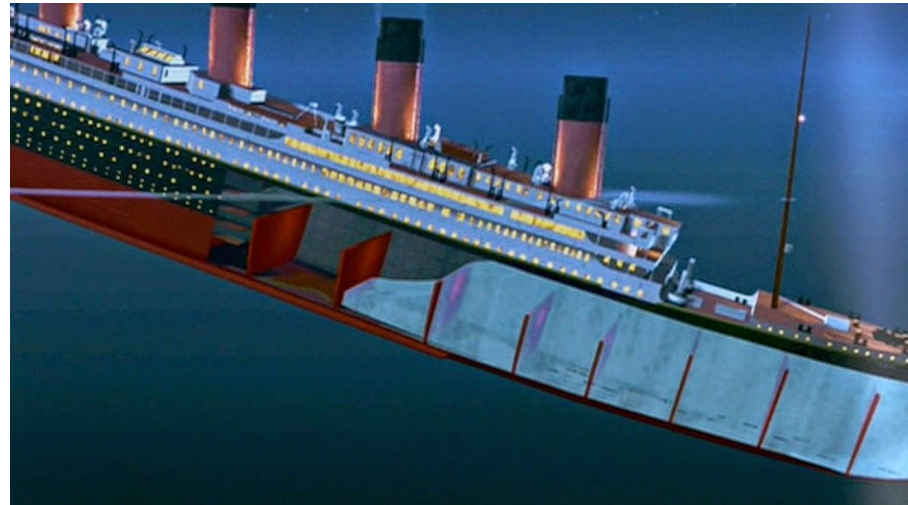
Aplicaciones de los punteros

- Acceder directamente a la memoria
- **Pasaje de argumentos a funciones “por referencia”**
- Utilización de memoria dinámica
- Optimización de algoritmos

Pasaje de argumentos por referencia

Hasta el momento vimos que en C, el valor de los argumentos se pasan a las funciones “por valor”, es decir, se copian los valores de los argumentos en la pila y éstos se reciben en la función por medio de los parámetros formales.

Esto asegura la “**compartimentalización**” de la información al pasarla a las funciones.



Recordatorio: Código de función

```
#include <stdio.h>

int suma (int,int);

int main (void)
{
    int a=5,b=3,c;

    c=suma(a,b);

    printf("c=%d=%d+%d",c,a,b);
}

int suma (int x, int y)
{
    int z;

    z=x+y;

    return z;
}
```

Pasaje por valor

5

3

Pasaje de argumentos por referencia

Sin embargo, en algunos casos puede ser deseable que la **función llamada** pueda modificar directamente valores de la **función llamante**, a esto se le llama “pasaje por referencia” y en C se realiza por medio de punteros.

No obstante, esto debe hacerse **sólo en los casos que sea justificable** para obtener alguna ventaja, ya que este enfoque destruye la “compartimentalización”.

Ejemplo: Pasaje de argumentos por referencia

```
#include <stdio.h>

void incrementar (int * p);

int main (void)
{
    int variable=5;
    printf("variable=%d",variable);

    incrementar(&variable);

    printf("variable=%d",variable);
}
```

```
void incrementar (int * p)
{
    *p=*p+1;
}
```

¿Qué hace el programa? ¿Qué
imprime?

Ejemplo: Pasaje de argumentos por referencia

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

```
void incrementar (int * p);
```

```
int main (void)  
{
```

```
    int variable=5;  
    printf("variable=%d",variable);
```

```
    //Se pasa la dirección de memoria de  
    //'variable' a la función incrementar  
    incrementar(&variable);
```

```
    printf("variable=%d",variable);
```

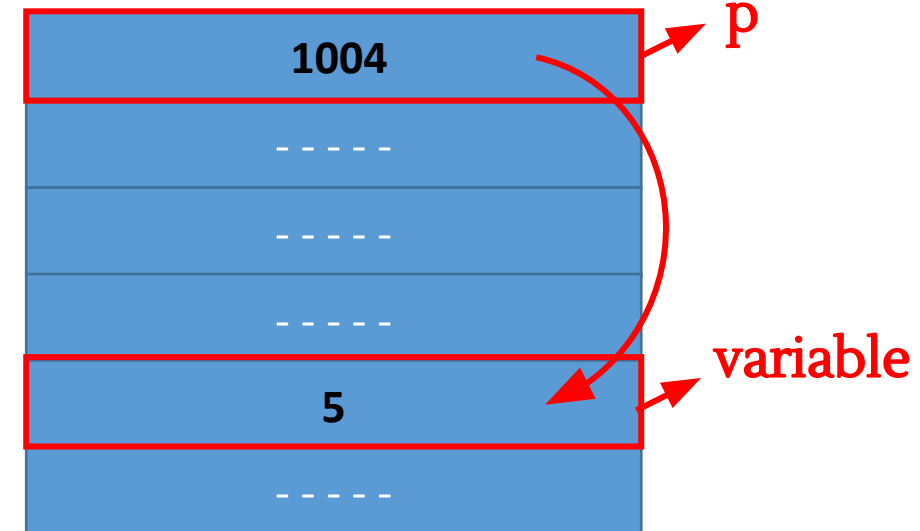
```
}
```

```
void incrementar (int * p)  
{  
    *p=*p+1;  
}
```

1004

Pasaje por
valor

Mapa de memoria



1004

Ejemplo: Pasaje de argumentos por referencia

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

```
void incrementar (int * p);
```

```
int main (void)
```

```
{
```

```
    int variable=5;
```

```
    printf("variable=%d",variable);
```

```
    incrementar(&variable);
```

```
    printf("variable=%d",variable);
```

```
}
```

```
void incrementar (int * p)
```

```
{
```

```
    //Se incrementa 'variable' por medio
```

```
    //del puntero 'p'
```

```
    *p=*p+1;
```

```
}
```

Mapa de memoria

1000

1004

p

1004

6

variable

Ejemplo: Pasaje de argumentos por referencia

```
#include <stdio.h>

void incrementar (int * p);

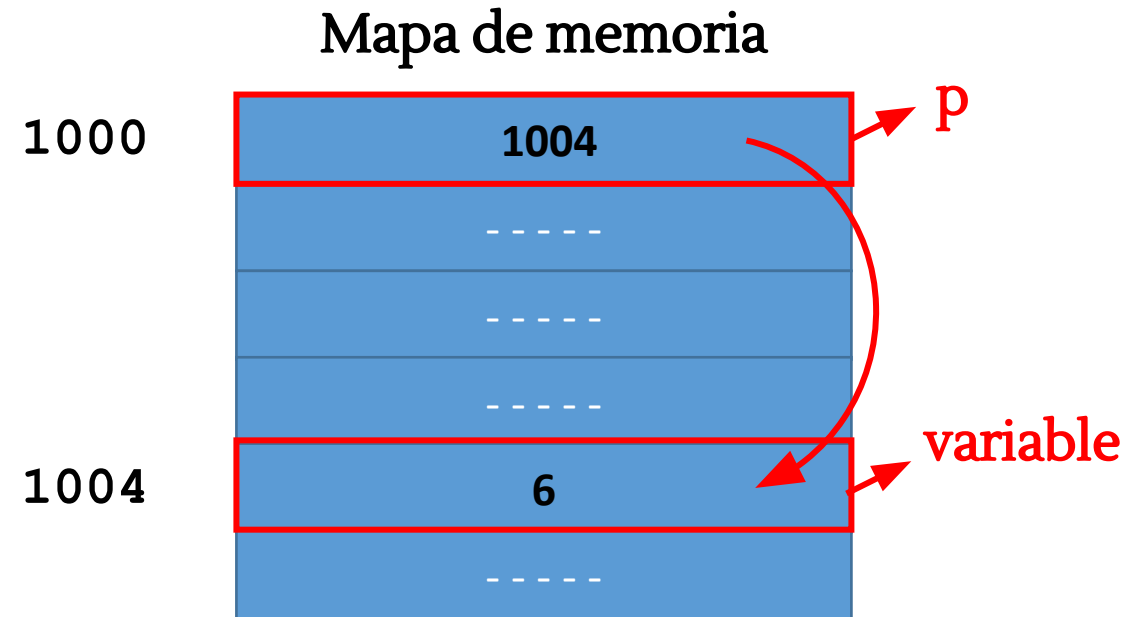
int main (void)
{
    int variable=5;
    printf("variable=%d",variable);

    incrementar(&variable);

    //Al retornar de la función
    //'variable' se incrementó

    printf("variable=%d",variable);
}
```

```
void incrementar (int * p)
{
    *p=*p+1;
}
```



A programar...

- 1) Escribir una función que recibe un puntero a entero (int *) y un valor a entero. La función imprime la dirección de memoria de esa variable, su valor actual y lo reemplaza por el que se pasa como argumento. Comprobar su correcto funcionamiento.

```
void modificaVar (int * p, int valor);
```

- 2) Escribir una función que recibe un puntero a caracter (char *) y si está en minúscula lo pasa a mayúscula y si está en mayúscula lo deja como está. Comprobar su correcto funcionamiento.

```
void pasarMayuscula (char * p);
```

Aritmética de punteros

Se pueden realizar algunas operaciones sobre las variable de tipo puntero:

- Incremento
- Decremento
- Resta de punteros

En todos los casos, el resultado de las operaciones depende del tipo base del puntero.

Excursus: Operador 'sizeof'

El operador sizeof se aplica sobre una variable o un tipo de datos y devuelve su tamaño en bytes en tiempo de compilación.

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

```
int main (void)
```

```
{
```

```
    printf ("\n\nsizeof(char)  =%ld", sizeof(char));
```

```
    printf ("\n\nsizeof(int)   =%ld", sizeof(int));
```

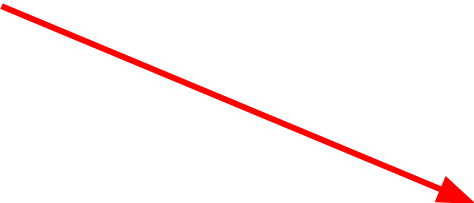
```
    printf ("\n\nsizeof(float)=%ld", sizeof(float));
```

```
    printf ("\n\n");
```

```
    return 0;
```

```
}
```

Salida de
pantalla



```
sizeof(char)  =1
sizeof(int)   =4
sizeof(float)=4
```

Aritmética de punteros: Incremento/decremento

Las operaciones de incremento/decremento sobre un puntero afectan su valor de acuerdo a su tipo base.

Veámoslo con dos ejemplos:

- Puntero a int (tamaño = 4 bytes)
- Puntero a char (tamaño = 1 byte)

Aritmética de punteros: Incremento puntero a int

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

```
int main (void)
```

```
{
```

```
    int * p=0x1000;
```

```
    int i;
```

```
    for(i=0;i<3;i++)
```


```
        printf("\n\npuntero+%d=%p", i, p+i);
```

```
    printf("\n\n");
```

```
    return 0;
```

```
}
```

Salida de
pantalla



```
puntero+0=0x1000
```

```
puntero+1=0x1004
```

```
puntero+2=0x1008
```

Aritmética de punteros: Incremento puntero a char

```
#include <stdio.h>


int main (void)
{
    char * p=0x1000;
    int i;

    for(i=0;i<3;i++)
        printf("\n\npuntero+%d=%p", i, p+i) ;

    printf("\n\n");

    return 0;
}
```

Salida de
pantalla



```
puntero+0=0x1000
puntero+1=0x1001
puntero+2=0x1002
```

¿En qué difieren ambos comportamientos?

Puntero a int

```
puntero+0=0x1000  
puntero+1=0x1004
```

Mapa de memoria

1000	-----	← puntero+0
1001	-----	
1002	-----	
1003	-----	
1004	-----	← puntero+1
1005	-----	
1006	-----	
1007	-----	

Puntero a char

```
puntero+0=0x1000  
puntero+1=0x1001
```

Mapa de memoria

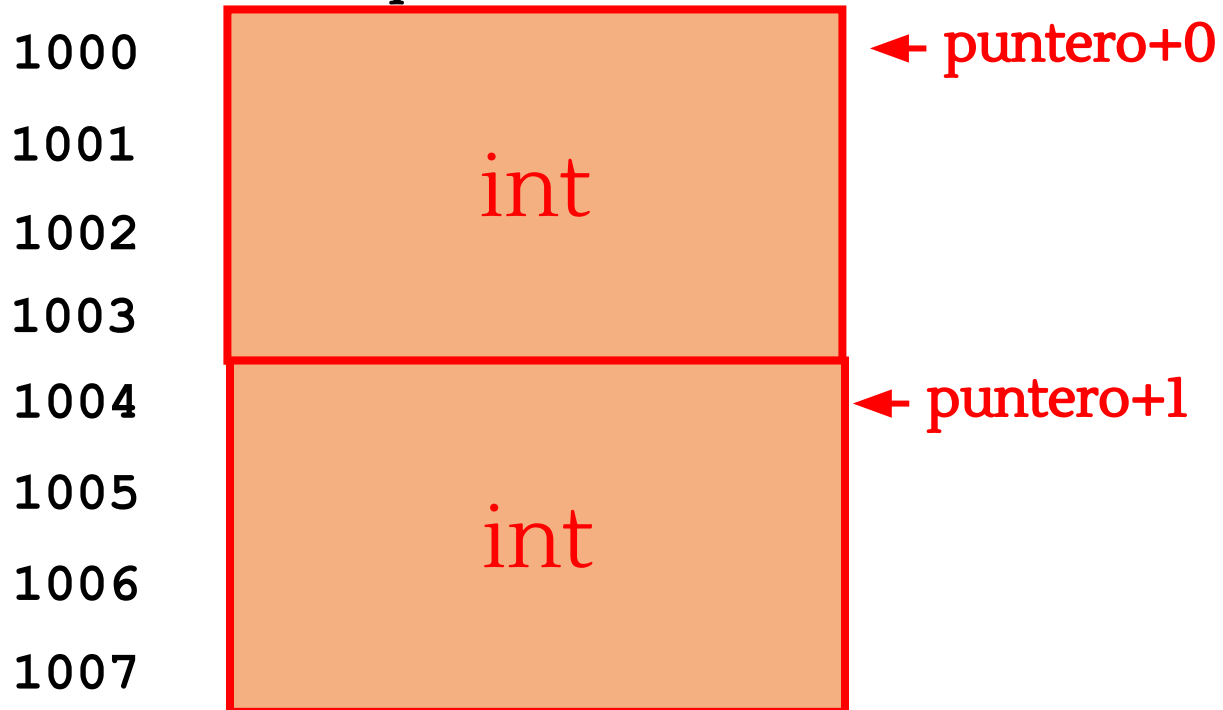
1000	-----	← puntero+0
1001	-----	← puntero+1
1002	-----	
1003	-----	
1004	-----	
1005	-----	
1006	-----	
1007	-----	

¿En qué difieren ambos comportamientos?

Puntero a int

```
puntero+0=0x1000  
puntero+1=0x1004
```

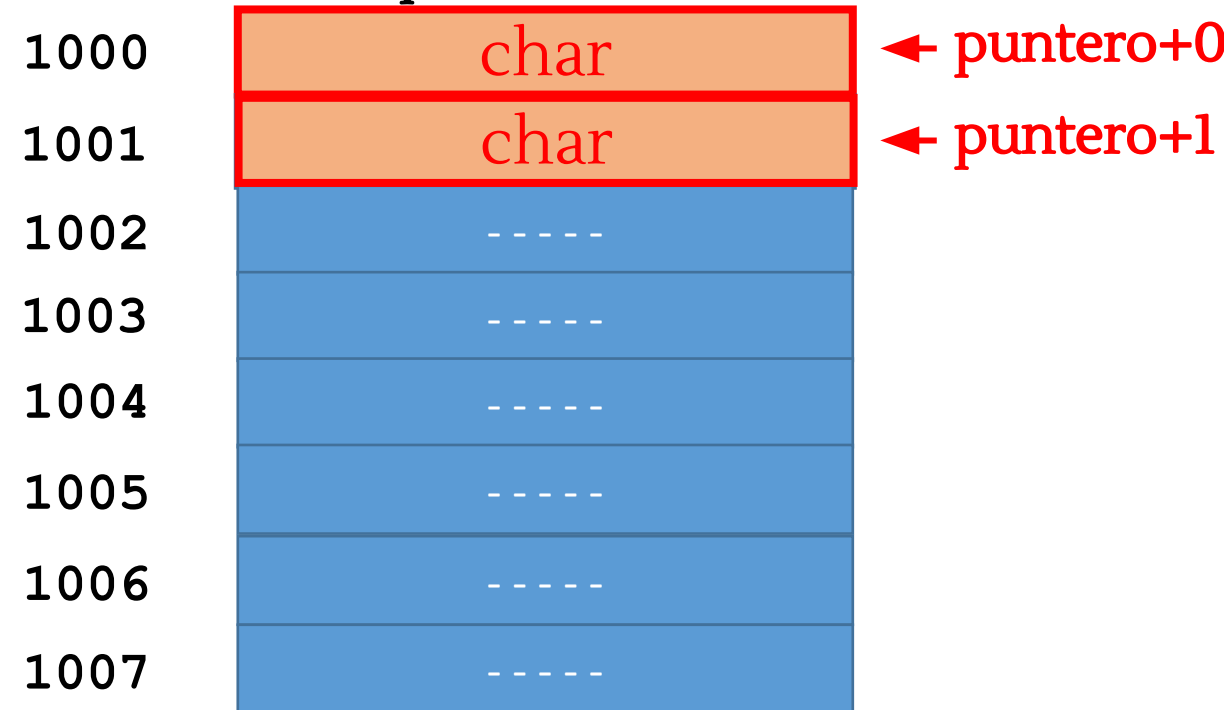
Mapa de memoria



Puntero a char

```
puntero+0=0x1000  
puntero+1=0x1001
```

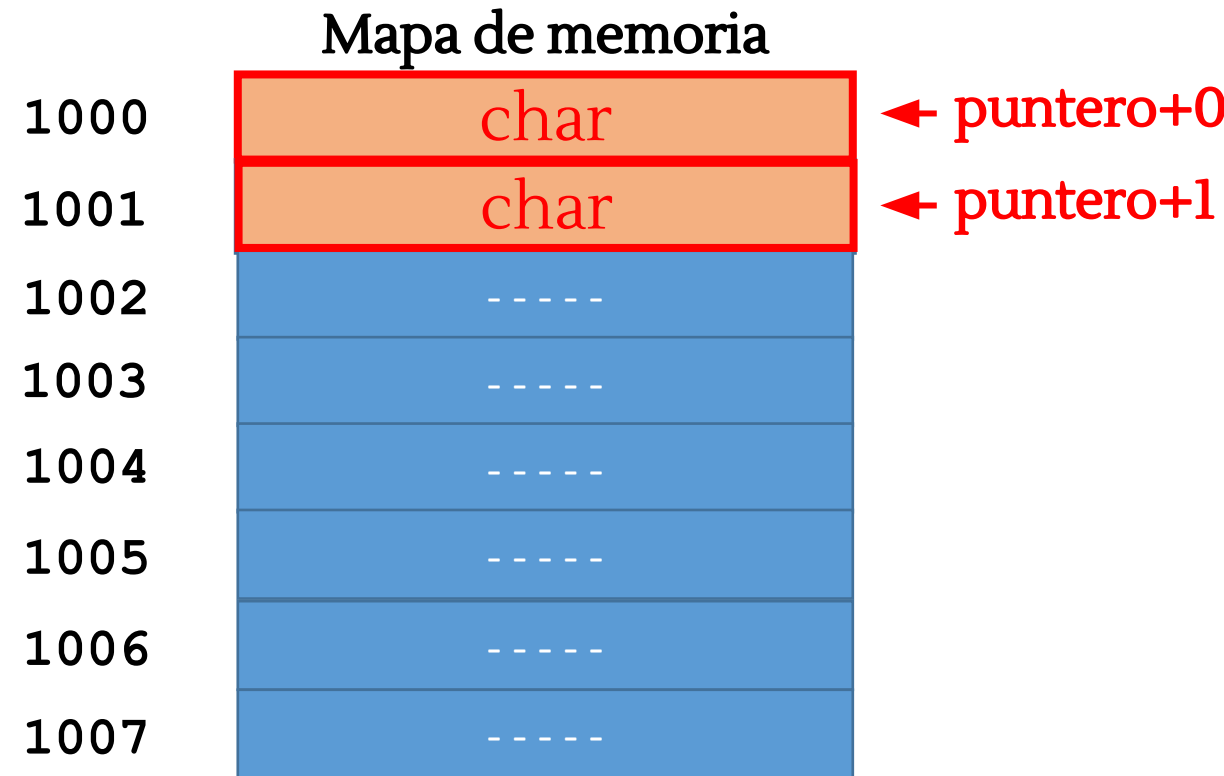
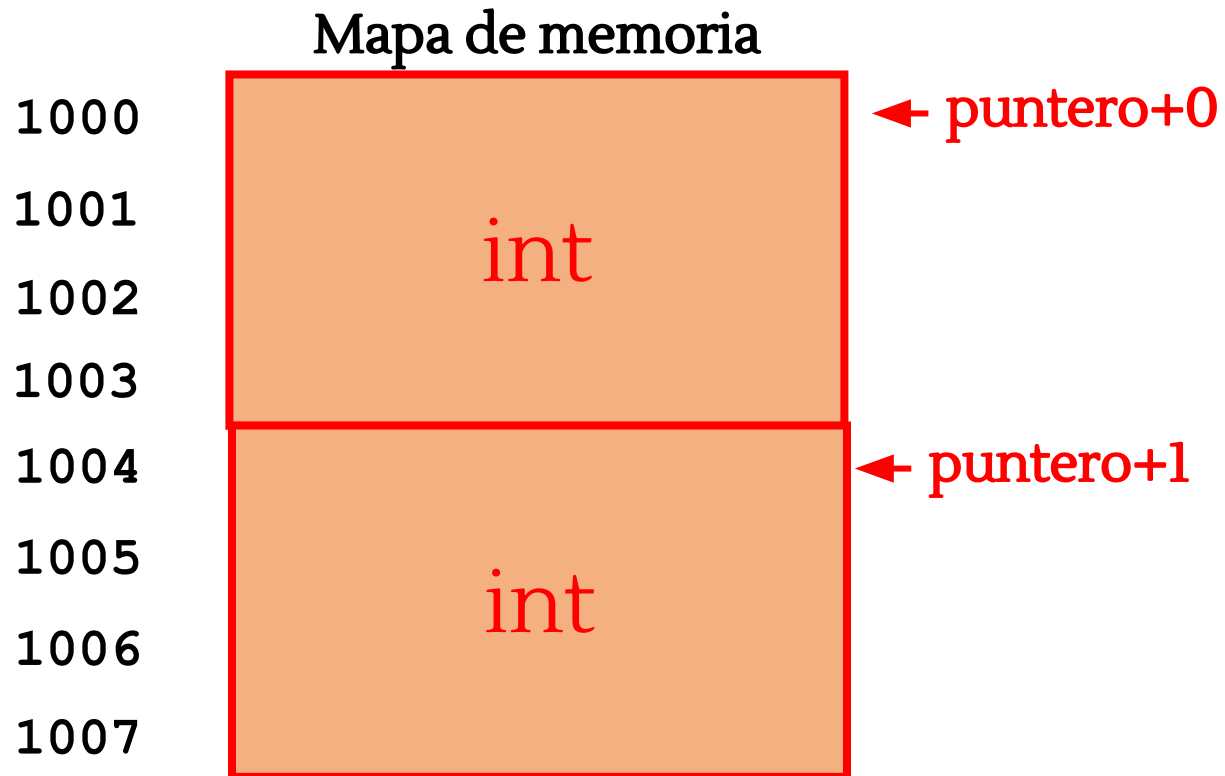
Mapa de memoria



¿En qué difieren ambos comportamientos?

Al incrementar/decrementar un puntero éste se modifica en base al tamaño del tipo de datos base.

Este comportamiento está estrechamente relacionado con el concepto de arreglos (*arrays*) o vectores de variables.



Aritmética de punteros: Resta de punteros

Las operaciones de resta de punteros dependen del tipo base de los punteros involucrados

Veámoslo con dos ejemplos:

- Puntero a int (tamaño = 4 bytes)
- Puntero a char (tamaño = 1 byte)

Aritmética de punteros: Resta de punteros a int

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

```
int main (void)
```


```
{  
    int * p1=0x1000, *p2=0x1008;
```

```
    printf("\n\n p2-p1=%d \n\n",p2-p1);
```

```
    return 0;
```

```
}
```

Salida de
pantalla



p2 - p1 = 2

Aritmética de punteros: Resta de punteros a char

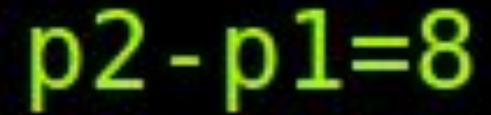

```
#include <stdio.h>

int main (void)
{
    char * p1=0x1000, *p2=0x1008;

    printf("\n\n p2-p1=%d \n\n",p2-p1);

    return 0;
}
```

Salida de
pantalla



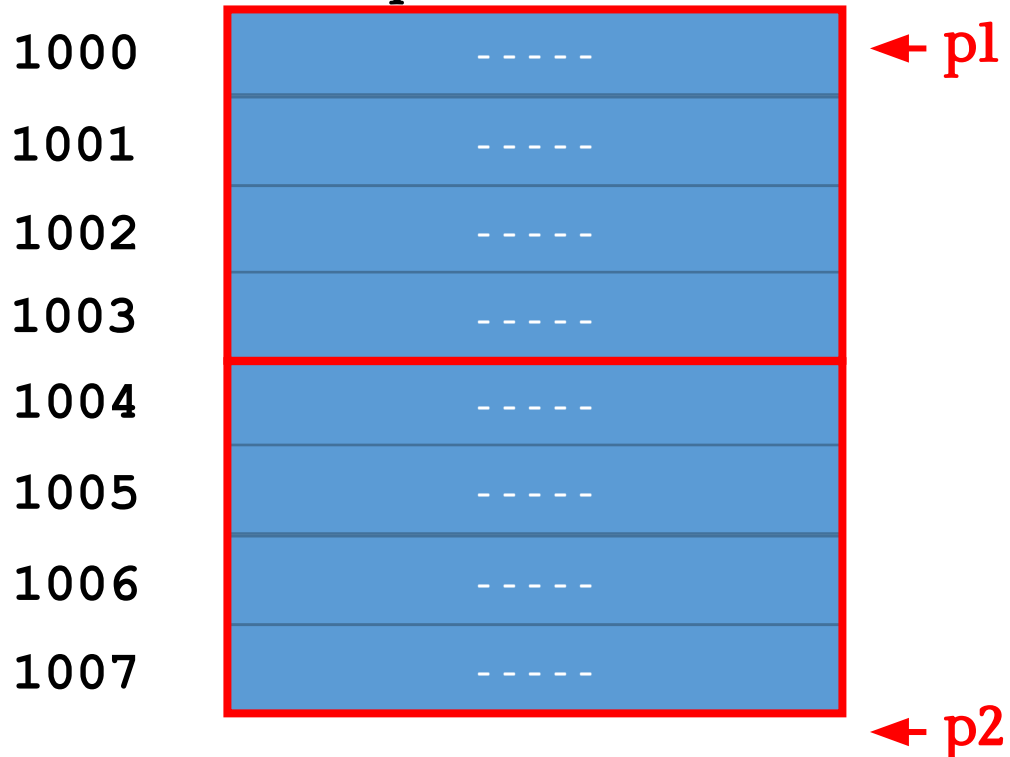
p2 - p1 = 8

¿En qué difieren ambos comportamientos?

Puntero a int

$p2 - p1 = 2$

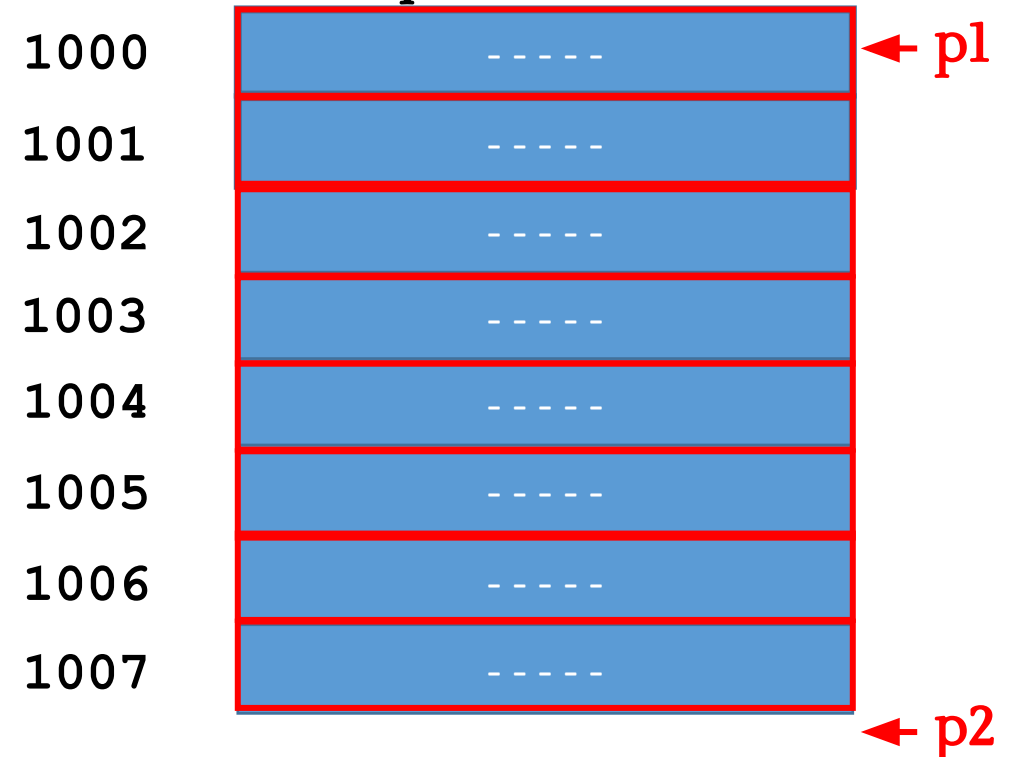
Mapa de memoria



Puntero a char

$p2 - p1 = 8$

Mapa de memoria

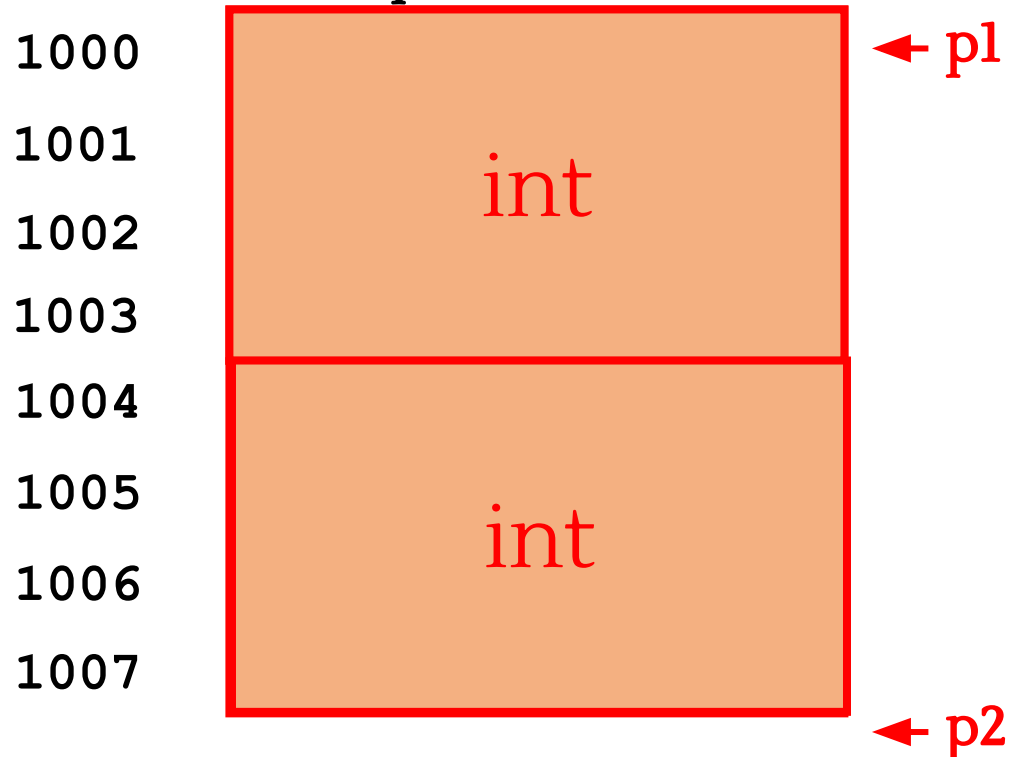


¿En qué difieren ambos comportamientos?

Puntero a int

$p2 - p1 = 2$

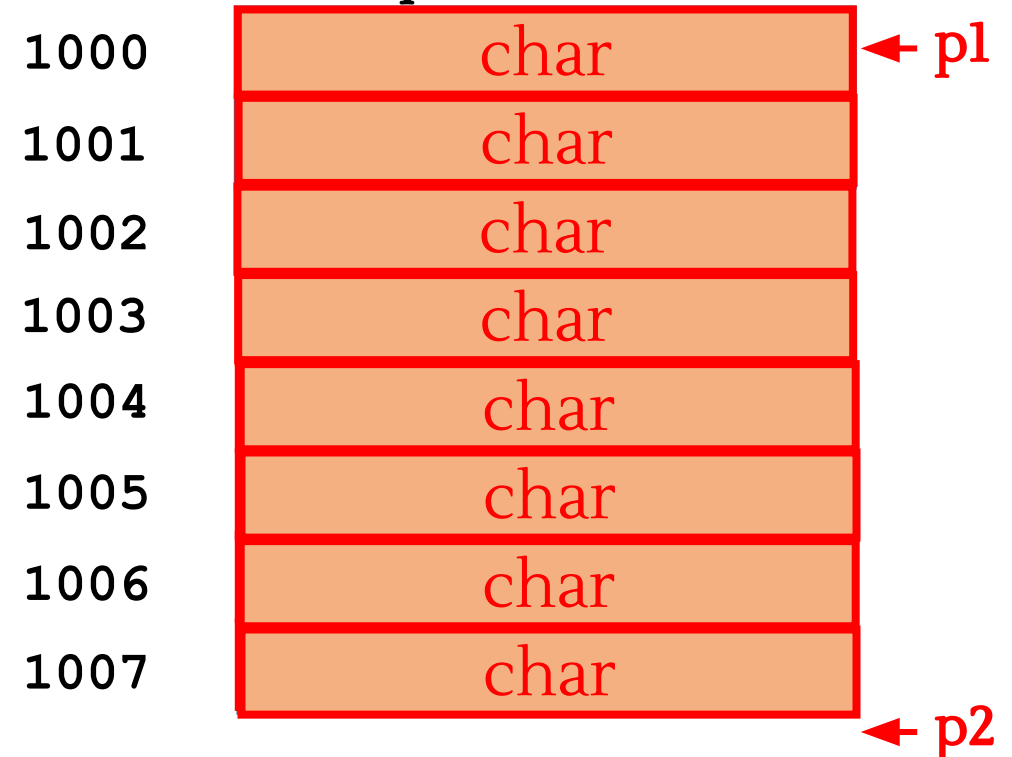
Mapa de memoria



Puntero a char

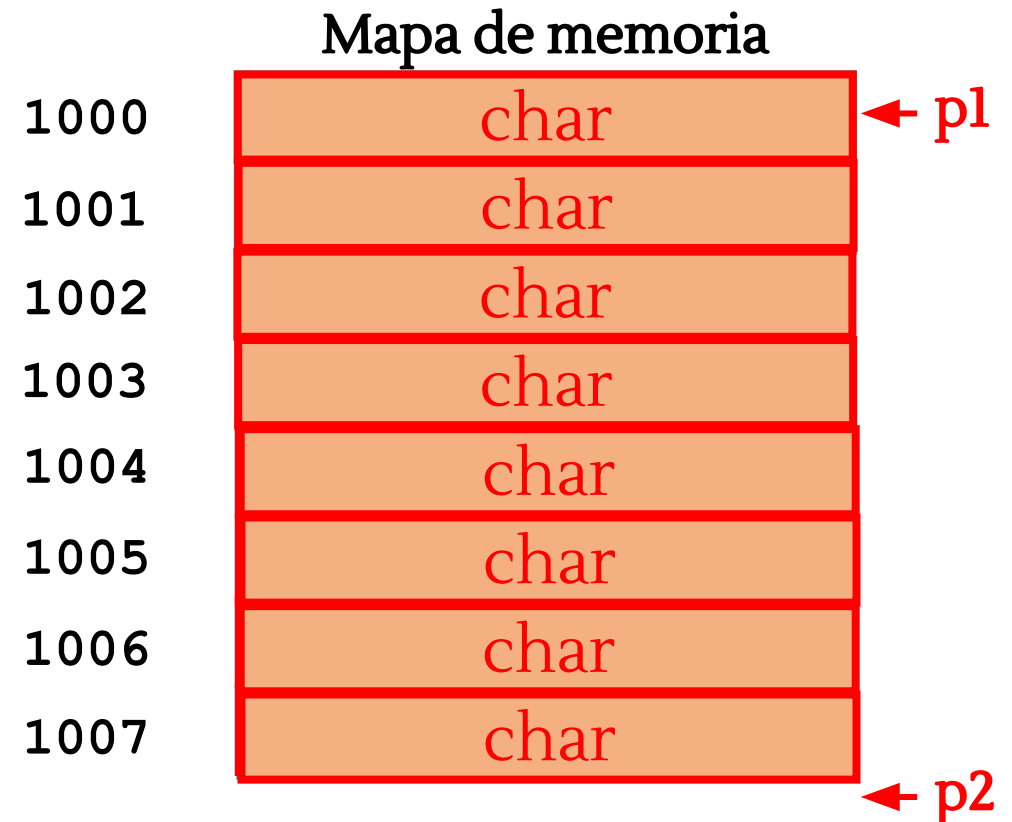
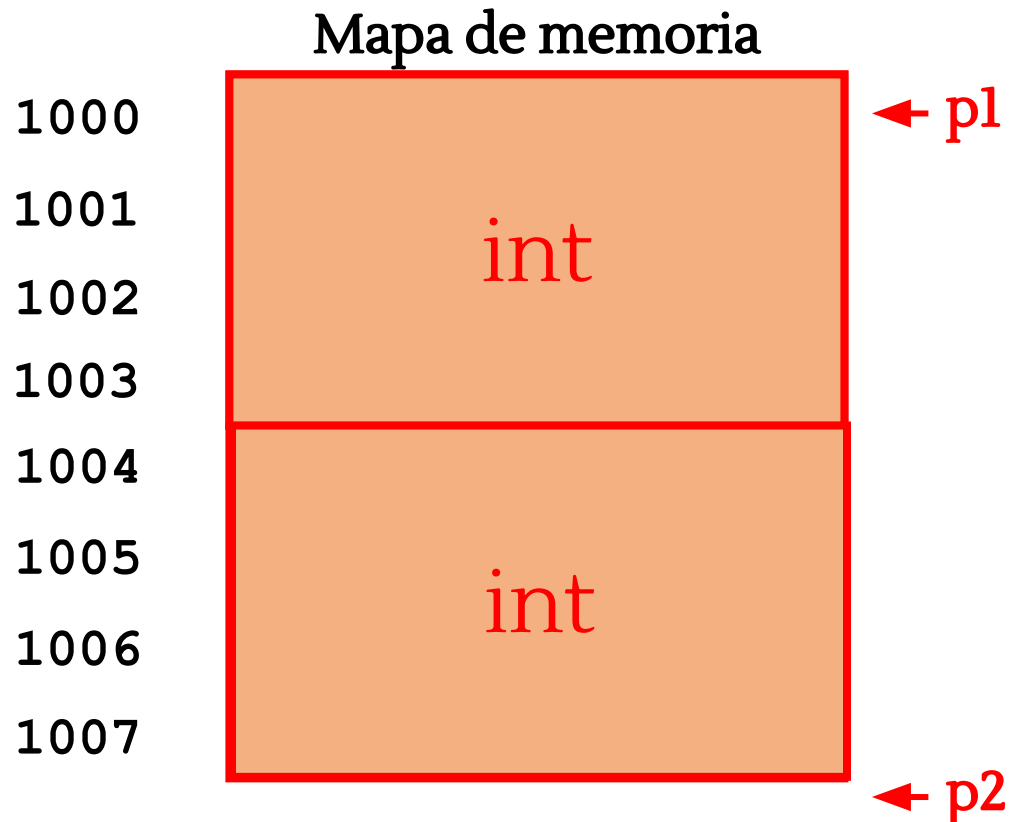
$p2 - p1 = 8$

Mapa de memoria



¿En qué difieren ambos comportamientos?

Al restar dos punteros, el resultado está dado por la “cantidad de variables del tipo base” contenidas entre las dos posiciones indicadas por los punteros. Este comportamiento está estrechamente relacionado con el concepto de arreglos (*arrays*) de variables.



Bibliografía

- Schildt, H. , “C Manual de referencia”, Capítulo 5
- Deitel, “Cómo programar en C/C++”, Capítulo 7
- Gottfried, B. , “Programación en C”, Capítulo 10
- Argibay J. , “C para Ingeniería Electrónica”, Capítulo 9
- Ceballos, F. , “Enciclopedia del lenguaje C”, Capítulo 6
- Kernighan B, Ritchie D. , “El lenguaje de programación C”, Capítulo 6