

VARIABLES PUNTERO

Todo dato con que trabaja la computadora (así como toda instrucción cuando se ejecuta) esta en la memoria principal .

Al momento de ejecutar un programa, para ese programa en particular se distinguen cuatro zonas distintas, todas ubicadas en la memoria principal de la computadora.

Cada zona es un 'segmento' y tiene una función determinada. Los segmentos son los siguientes :

Segmento de Código: aquí esta el programa que se esta ejecutando, ya traducido a lenguaje de maquina

Segmento de Datos: aquí se ubican las variables del programa principal (las que hemos declarado en main).

Segmento de Pila (o Stack): este segmento, del que hablaremos después es fundamental para que podamos llamar a funciones.

Segmento Extra (o heap o montículo): sirve para utilizar la memoria dinámica.

Zona de código del programa: aquí está el programa que se está ejecutando, ya traducido a lenguaje de máquina

Zona de datos (estáticos) del programa: aquí se ubican las variables del programa principal (las que hemos declarado en main).

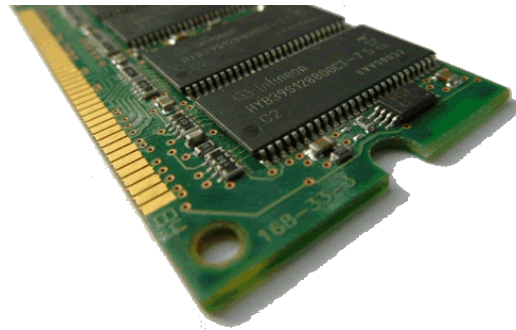
Zona de la pila del programa: este segmento, del que hablaremos después es fundamental para que podamos llamar a funciones.

Zona de la memoria dinámica (heap) del programa: sirve para utilizar la memoria dinámicamente.

Zona de la memoria principal usada por el programa durante su ejecución.

(El orden de los segmentos puede variar)

Vamos a considerar en particular la zona de los datos. De que modo se almacenan en la memoria?.

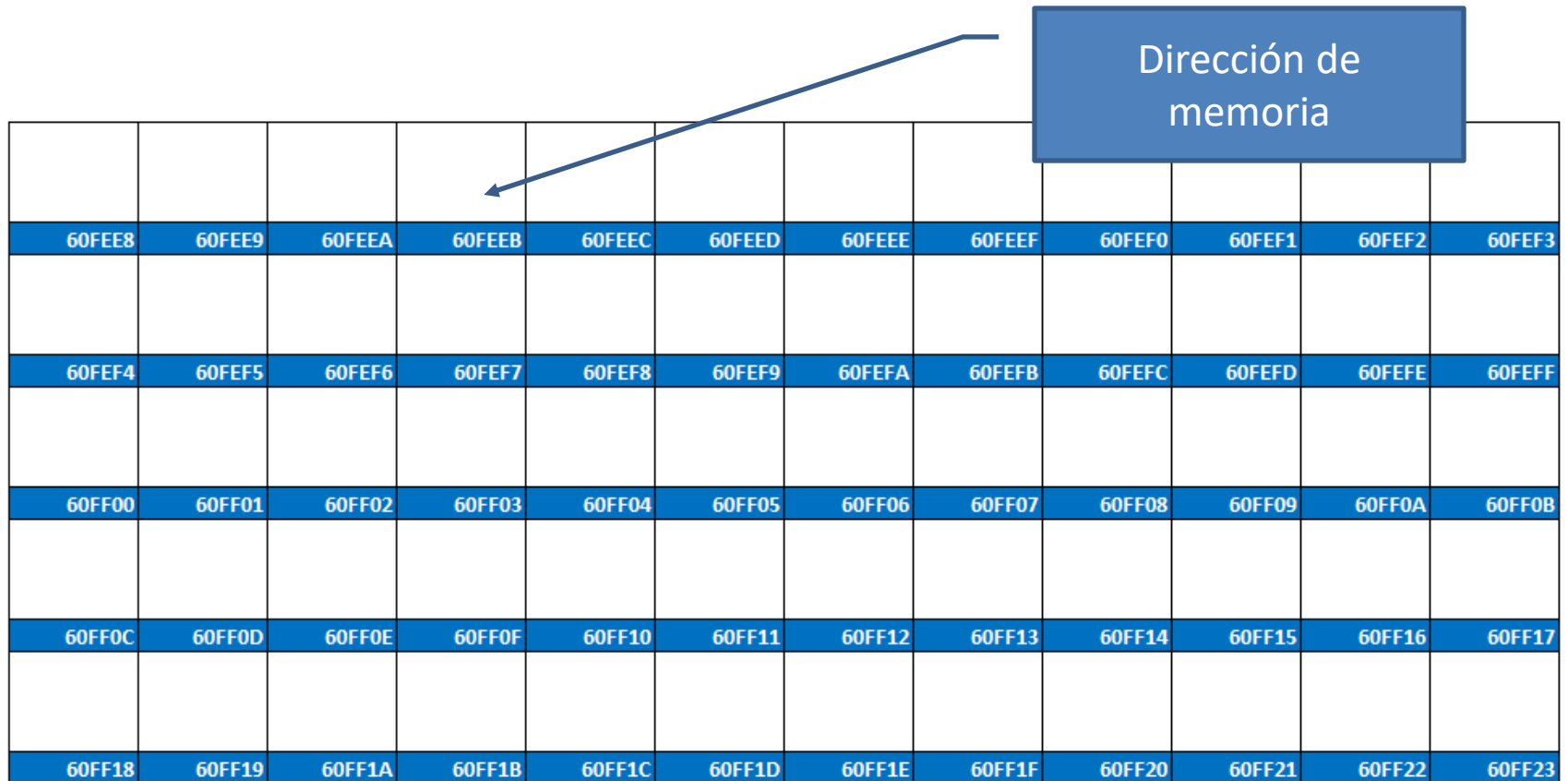


En principio, respecto de la estructura de la memoria, hay que tener en cuenta que se compone de celdas. Cada celda es 1 byte, es decir que esta formada por 8 bits.

Todas las celdas de la memoria tienen el mismo tamaño, y son idénticas entre si. Solo pueden diferenciarse por el número que tienen asignado. El número de posición de una celda es la dirección de la celda

Offset (d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0006356576	7A	C0	26	75	00	00	00	00	80	6D	2D	75	00	10	00	00
0006356592	38	A3	CB	DA	00	46	2D	75	80	12	40	00	00	90	37	00
0006356608	00	00	00	00	F5	6F	28	75	70	FE	60	00	98	FE	60	00
0006356624	CC	FF	60	00	C0	CC	27	75	E0	CF	87	AF	00	00	00	00
0006356640	B8	FE	60	00	3D	36	29	75	00	00	00	00	80	6D	2D	75
0006356656	00	10	00	00	80	12	40	00	D8	FE	60	00	4C	1D	40	00
0006356672	00	46	2D	75	B0	FE	60	00	CC	FF	60	00	C0	CC	27	75
0006356688	58	C4	87	AF	FE	FF	FF	FF	08	FF	60	00	2A	14	40	00
0006356704	2F	30	40	00	02	00	00	00	80	FF	60	00	CE	19	40	00
0006356720	70	19	40	00	7E	00	00	00	08	00	00	00	04	FF	60	00

Toda celda de la memoria tiene una dirección asignada. Esa dirección es única para cada celda y no puede ser cambiada. Las direcciones son números positivos.



The diagram shows a grid of memory cells. A blue box labeled 'Dirección de memoria' has an arrow pointing to the cell at row 1, column 4. The grid consists of 6 rows and 12 columns. The addresses are as follows:

60FEE8	60FEE9	60FEEA	60FEEB	60FEED	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3		
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B
60FF0C	60FF0D	60FF0E	60FF0F	60FF10	60FF11	60FF12	60FF13	60FF14	60FF15	60FF16	60FF17
60FF18	60FF19	60FF1A	60FF1B	60FF1C	60FF1D	60FF1E	60FF1F	60FF20	60FF21	60FF22	60FF23

Cada dato contenido en la memoria ocupa una o más celdas, dependiendo del tamaño del mismo. Por ejemplo:

char	1 byte (una celda)
int	2 ó 4 bytes (2 ó 4 celdas)
float	4 bytes (4 celdas)
double	8 bytes
....	...

Y lo que siempre se verifica es que un dato, del tipo que sea, ocupa un conjunto de **celdas contiguas**.

No se debe confundir nunca la dirección de una variable con el contenido de la misma.

Ejemplo:

```
1  #include <stdio.h>
2
3  int main() {
4
5  int n;
6
7  n=8;
8
9  return 0;
10 }
```

Quando ejecutamos el programa y se llega a la línea: `int n`, se reserva memoria para el dato. La cantidad de memoria reservada corresponde al tipo de datos de la variable, para el ejemplo `int` o sea 4 bytes

60FEE8	60FEE9	60FEEA	60FEEB	60FEED	60FEED	60FEEF	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Ejemplo:

```
1  #include <stdio.h>
2
3  int main() {
4
5      int n;
6
7      n=8;
8
9      return 0;
10 }
```

En el momento que se asigna el dato (o se lee por teclado) , el mismo se guarda en la memoria reservada anteriormente.

60FEE8	60FEE9	60FEEA	60FEEB	60FEED	60FEED	60FEEF	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
								8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Ejemplo:

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
								8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Como se observa en la imagen, el dato int ocupa 4 bytes, o sea 4 celdas:

60FEFC

60FEFD

60FEFE

60FEFF

Que son contiguas. Cada una de éstas celdas representan 1 byte. El número asignado a cada una es la **dirección de cada celda**

Una variable puntero es una variable especial que sólo puede almacenar la dirección de otra variable. Su declaración tiene esta forma:

tipo de dato * identificador;

/ establece que identificador es una variable puntero a dato tipo */*

Ejemplo:

int * p;

/ indica que p es un puntero a entero */*

Ejemplo:

```
1  #include <stdio.h>
2
3  int main() {
4
5      int n;
6      int * p;
7
8      n=8;
9
10     return 0;
11 }
```

Declaración de una variable puntero en C

Una variable puntero debe declararse con el mismo tipo de dato de la variable cuya dirección guardará.

Ejemplo:

```
1  #include <stdio.h>
2
3  int main() {
4
5      int n;
6      int * p;
7
8      n=8;
9
10     return 0;
11 }
```

En el momento de la ejecución del programa , cuando llega a: `int n` , se reserva memoria para la variable `n`; y cuando se llega a: `int * p`, se reserva memoria para la variable puntero `p`

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B



& es el operador de dirección, indica
la dirección de una variable

Entonces, si en una variable puntero se puede guardar
una dirección....

Ejemplo:

```
1  #include <stdio.h>
2
3  int main() {
4
5      int n;
6      int * p;
7
8      p=&n;
9
10     return 0;
11 }
```

Una dirección de memoria sólo puede guardarse en una variable puntero. En el momento que se asigna la dirección de la variable, se guarda en la memoria reservada anteriormente para la variable puntero. La dirección que se guarda siempre es la primera.

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
				60FEFC							
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Ejemplo:

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEF0	60FEF1	60FEF2	60FEF3	
				60FEFC				8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Se puede dar valor a n antes o después de asignar la dirección a una variable puntero.

```
1  #include <stdio.h>
2
3  int main() {
4
5      int n;
6      int * p;
7
8      p=&n;
9
10     n=8;
11
12     return 0;
13 }
```

Entonces....

En una variable puntero se puede guardar una dirección de memoria de otra variable.

La variable puntero debe tener el mismo tipo de dato que la variable a la cuál apunta.

Se pueden declarar variables punteros a cualquier tipo de datos.

En el contexto de declaración (ej: `int * p`) el asterisco (*) indica que la variable es un puntero.

La dirección de la variable que guarda siempre es la primera.

Para guardar una dirección se debe anteponer a la variable, el operador de dirección &

La variable puntero, al tener un tipo de dato, tiene registro de cuántas celdas ocupa ese tipo de datos y que esas celdas son contiguas, por eso sólo necesita guardar la primera dirección.

Una variable puntero siempre ocupa 4 bytes, no importa a qué tipo de datos apunte, se puede comprobar con el operador `sizeof`.

En una variable puntero no se puede guardar una dirección de un tipo de dato distinto al de la variable puntero.

Una variable puntero debe inicializarse con una dirección o con `NULL`.

No se pueden hacer operaciones aritméticas con direcciones de memoria pero sí aplicar la aritmética de punteros.

....Y qué se puede hacer con
una variable puntero?...

La variable puntero puede mostrar su contenido como cualquier variable.

Ejemplo:

```
#include <stdio.h>
int main()
{
    int n;
    int * p;

    n=8;
    p=&n;

    printf("La variable puntero p guarda la direccion de la variable n\n");
    printf("que es en hexa: %p\n", p);
    printf("que es en decimal: %d\n\n", p);

    printf("\nCompruebo la direccion con &n\n");
    printf("en hexa es: %p\n", &n);
    printf("en decimal es: %d\n\n", &n);

    getchar();
    return 0;
}
```

Obsérvese que una dirección puede emitirse con %p (en hexadecimal) o con %d (en decimal)

Se puede comprobar emitiendo directamente &n

```
La variable puntero p guarda la direccion de la variable n
que es en hexa: 0060FEFC
que es en decimal: 6356732
```

```
Compruebo la direccion con &n
en hexa es: 0060FEFC
en decimal es: 6356732
```



* (asterisco) es el operador de indirección y permite que la variable puntero acceda al valor de la variable

La variable puntero puede acceder al valor de la variable a la cual apunta.

Ejemplo:

```
#include <stdio.h>
int main() {
    int n;
    int * p;

    n=8;
    p=&n;

    printf("La variable puntero p accede al valor de n\n");
    printf("%d\n", *p);

    printf("\nCompruebo con n\n");
    printf("%d\n\n", n);

    getchar();
    return 0;
}
```

Obsérvese que para acceder al contenido es *p

Se puede comprobar emitiendo directamente el valor de la variable

```
La variable puntero p accede al valor de n
8

Compruebo con n
8
```

Al tener, la variable puntero p la dirección de n (se dice que apunta) puede acceder al contenido de n.

Al acceder puede emitir el valor, utilizarlo para operaciones e incluso modificarlo y para ello necesita el *

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
				p	60FEFC			n	8		
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Ejemplo:

```
#include <stdio.h>

int main() {
    int n, m;
    int * p;

    n=8;
    p=&n;

    m = *p + 3;
    printf("m: %d\n", m);

    m = m - *p;
    printf("m: %d\n", m);

    m = *p / *p;
    printf("m: %d\n", m);

    *p = *p * *p;
    printf("*p: %d\n", *p);

    printf("\nCuanto vale n ahora?\n");
    printf("...vale %d\n\n", n);

    getchar();
    return 0;
}
```

m: 11
m: 3
m: 1
*p: 64

Cuanto vale n ahora?
...vale 64

Ejemplo:

```
#include <stdio.h>

int main() {
    int n, m;
    int * p;

    n=8;
    p=&n;

    m = *p + 3;
    printf("m: %d\n", m);

    m = m - *p;
    printf("m: %d\n", m);

    m = *p / *p;
    printf("m: %d\n", m);

    *p = *p * *p;
    printf("*p: %d\n", *p);

    printf("\nCuanto vale n ahora?\n");
    printf("...vale %d\n\n", n);

    getchar();
    return 0;
}
```

El * en el contexto de declaración indica que la variable es puntero.

El * en el contexto de una operación o emisión de resultado indica acceso al valor o contenido de la variable a la apunta.

En esta instrucción tenemos dos significados del * :

$*p = *p * *p;$

Además de los mencionados, el destacado en color rojo significa multiplicación

Entonces...

Se puede acceder al contenido de la variable puntero donde se visualizará la dirección que guarda

Se puede acceder al valor de la variable a la cual apunta con el operador de indirección *

Anteponiendo el * a la variable puntero, se pueden realizar operaciones aritméticas

Se puede modificar el valor de la variable original a la que apunta la variable puntero

El * tiene significados diferentes según el contexto dónde se aplique

....Se puede hacer algo más
con las variables punteros?...

Ejemplo 1:

```
#include <stdio.h>

int main() {
    int n, m;
    int * p;

    n=8;
    p=&n;
    printf("p es %p\n\n", p);
    printf("&n es %p\n\n", &n);
    printf("n es %d\n\n", n);
    printf("&p es %p\n\n", &p);
    printf("*p es %d\n\n", *p);

    getchar();
    return 0;
}
```

p es 0060FEFC

&n es 0060FEFC

n es 8

&p es 0060FEF8

*p es 8

Las variables punteros también tiene su propia dirección. Para acceder se debe anteponer el operador &

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
				60FEFC				8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Ejemplo 2:

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

```
int main() {
    int n, m;
    int * p;

    n=8;
    p=&n;
    printf(" \n\n p es %p\n\n", p);
    printf(" &n es %p\n\n", &n);
    printf(" n es %d\n\n", n);
    printf(" &p es %p\n\n", &p);
    printf(" *p es %d\n\n", *p);

    p++;
    printf(" \n\n despues de p++, p es %p\n\n", p);
    printf(" *p es %d\n\n", *p);

    getchar();
    return 0;
}
```

p es 0060FEFC

&n es 0060FEFC

n es 8

&p es 0060FEF8

*p es 8

despues de p++, p es 0060FF00

*p es 129

Al incrementar el contenido de la variable puntero (p++), este suma -al valor que guarda- la cantidad de bytes correspondiente al tipo de datos al que apunta (en el ejemplo int = 4 bytes), perdiendo la dirección original a la cual apuntaba. Al acceder a ese nuevo lugar de la memoria podemos obtener un valor residual , en el ejemplo 129

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEF0	60FEF1	60FEF2	60FEF3	
				60FF00				8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
129											
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Ejemplo 3:

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

```
int main() {  
    int n, m;  
    int * p;
```

```
    n=8;  
    p=&n;
```

```
    printf(" \n\n p es %p\n\n", p);  
    printf(" &n es %p\n\n", &n);  
    printf(" n es %d\n\n", n);  
    printf(" &p es %p\n\n", &p);  
    printf(" *p es %d\n\n", *p);
```

```
    (*p)++;  
    printf(" \n\n despues de (*p)++, *p es %d\n\n", *p);  
  
    getchar();  
    return 0;  
}
```

```
p es 0060FEFC  
&n es 0060FEFC  
n es 8  
&p es 0060FEF8  
*p es 8
```

```
despues de (*p)++, *p es 9
```

Al ejecutarse la instrucción `(*p)++`, se incrementa el valor de la variable a la cual apunta la variable puntero

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
				60FEFC				9			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Ejemplo 4:

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

```
int main() {
    int n, m;
    int * p;

    n=8;
    p=&n;

    printf(" \n\n p es %p\n\n", p);
    printf(" &n es %p\n\n", &n);
    printf(" n es %d\n\n", n);
    printf(" &p es %p\n\n", &p);
    printf(" *p es %d\n\n", *p);

    *p++;
    printf(" \n despues de *p++, p es %p\n", p);
    printf(" \n despues de *p++, *p es %d\n", *p);

    getchar();
    return 0;
}
```

p es 0060FEFC

&n es 0060FEFC

n es 8

&p es 0060FEF8

*p es 8

despues de *p++, p es 0060FF00

despues de *p++, *p es 129

Al ejecutar la instrucción *p++, el resultado es que incrementa en 4 bytes la dirección guardada perdiendo la dirección original para apuntar a otra. El operador ++ actúa antes que *

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
				60FF00				9			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
129											
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Entonces...

Las variables punteros también tienen su propia dirección.

Se pueden mover en la memoria mediante ++ ó – y acceder al contenido siempre que esté inicializada, es lo que se denomina aritmética de punteros.

Tener en cuenta que ...

$p++ \neq (*p)++ \neq *p++$

....Y se puede guardar la
dirección de un puntero?...

Ejemplo:

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

```
int main() {
```

```
int n;
```

```
int * p;
```

```
int ** pp;
```

```
n=8;
```

```
p=&n;
```

```
printf(" \n p es %p\n", p);
```

```
printf(" \n *p es %d\n", *p);
```

```
getchar();
```

```
return 0;
```

```
}
```

p es 0060FEFC

*p es 8

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
				60FEFC				8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Ejemplo:

```
#include <stdio.h>

int main() {
    int n;
    int * p;
    int ** pp;

    n=8;
    p=&n;

    printf(" \n p es %p\n", p);
    printf(" \n *p es %d\n", *p);

    pp=&p;
    printf(" \n &pp es %p\n", &pp);
    printf(" \n pp es %p\n", pp);
    printf(" \n *pp es %p\n", *pp);
    printf(" \n **pp es %d\n", **pp);

    getchar();
    return 0;
}
```

Para guardar la dirección de un puntero se necesita declarar otra variable puntero a puntero.

Asignación de la dirección de la variable puntero a la variable puntero a puntero.

```
p es 0060FEFC
*p es 8
&pp es 0060FEF4
pp es 0060FEF8
*pp es 0060FEFC
**pp es 8
```

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
60FEF8				60FEFC				8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Se pueden acceder a todos los niveles con las variables punteros y puntero a puntero.

Ejemplo:

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
pp 60FEF8				p 60FEFC				n 8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

```
p es 0060FEFC
*p es 8
&pp es 0060FEF4
pp es 0060FEF8
*pp es 0060FEFC
**pp es 8
```

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
pp 60FEF8				p 60FEFC				n 8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03					60FF07	60FF08	60FF09	60FF0B

```

p es 0060FEFC
*p es 8
&pp es 0060FEF4
pp es 0060FEF8
*pp es 0060FEFC
**pp es 8

```

	&p = pp				&n = p = *pp				n = *p = ** pp			
60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3	
60FEF8				60FEFC				8				
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF	
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B	

Se pueden acceder a todos los niveles con las variables punteros y puntero a puntero.

60FEE8	60FEE9	60FEEA	60FEEB	60FEEC	60FEED	60FEEE	60FEEF	60FEF0	60FEF1	60FEF2	60FEF3
60FEF8				60FEFC				8			
60FEF4	60FEF5	60FEF6	60FEF7	60FEF8	60FEF9	60FEFA	60FEFB	60FEFC	60FEFD	60FEFE	60FEFF
60FF00	60FF01	60FF02	60FF03	60FF04	60FF05	60FF06	60FF07	60FF08	60FF09	60FF0A	60FF0B

Con `&pp`: obtengo la dirección de la variable puntero a puntero

Con `pp` : accedo a la dirección de la variable puntero

Con `*pp`: accedo al contenido de la variable puntero

Con `**pp`, accedo al contenido de la variable a la cual apunta la variable puntero que a su vez está apuntada por la variable puntero a puntero

Entonces...

Una variable puntero a puntero guarda la dirección de una variable puntero.

Se pueden declarar variables punteros de muchos niveles, con lo cual una variable puntero a puntero a puntero puede guardar la dirección de una variable puntero a puntero.

Se pueden realizar operaciones aritméticas teniendo en cuenta los niveles para acceder al valor de la variable original.