Programación, Programación I y Electrónica Programable

Área de Servicios - Depto. de Informática - FCFMyN

Universidad Nacional de San Luis

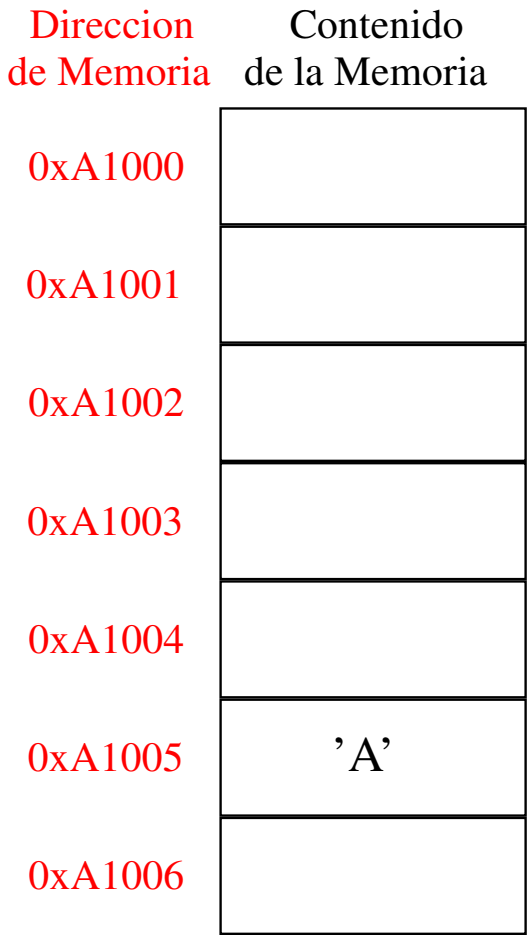
1er Cuatrimestre del 2017

Contenidos

1. Introducción
2. Variables Punteros
3. Operadores
4. Ejemplos
5. Conclusiones

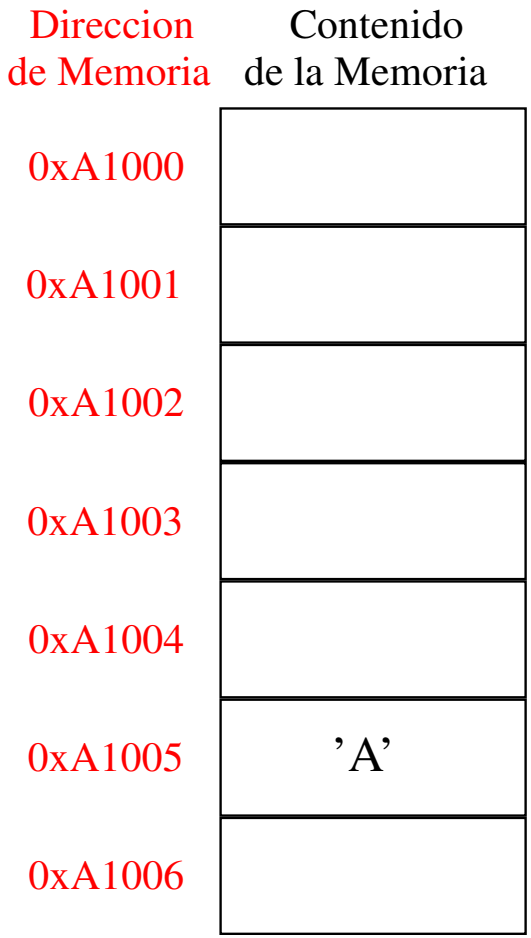
Memoria de una Computadora

La memoria de una computadora puede ser pensada como un arreglo de valores, donde cada una de sus celdas se encuentra referenciada o identificada por un número hexadecimal denominado **DIRECCIÓN DE MEMORIA**.



Memoria de una Computadora

La memoria de una computadora puede ser pensada como un arreglo de valores, donde cada una de sus celdas se encuentra referenciada o identificada por un número hexadecimal denominado **DIRECCIÓN DE MEMORIA**.



Ej.: 0x1500, 0xAB340, 0xD200

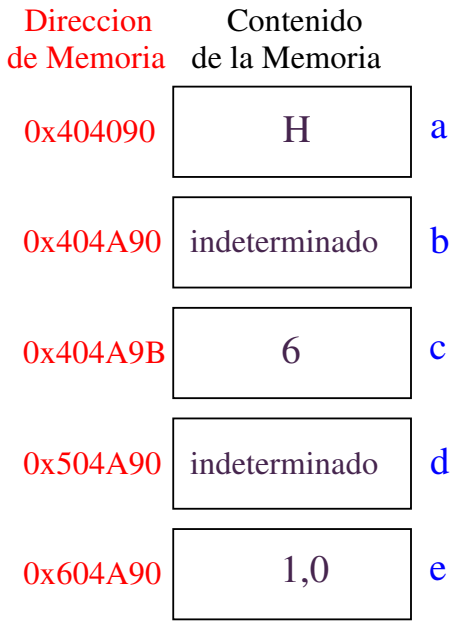
Variables en C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 char a = 'H',b;
4 int c = 6;
5 int main() {
6    int d;
7    float e = 1.0;
8    . . .
9    return 0;
10 }
```

Direccion de Memoria	Contenido de la Memoria	
0x404090	H	a
0x404A90	indeterminado	b
0x404A9B	6	c
0x504A90	indeterminado	d
0x604A90	1,0	e

Variables en C

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 char a = 'H',b;
4 int c = 6;
5 int main() {
6    int d;
7    float e = 1.0;
8    . . .
9    return 0;
10 }
```

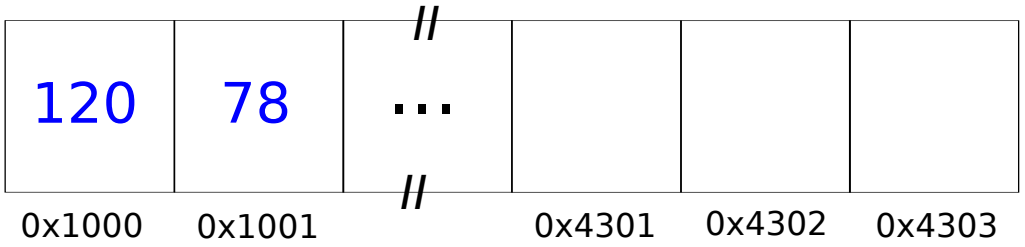


Variable: Lugar de memoria que puede mantener un valor.
Una variable posee tres atributos básicamente:

- *Identificador*
- *Tipo de dato*
- *Dirección de memoria*

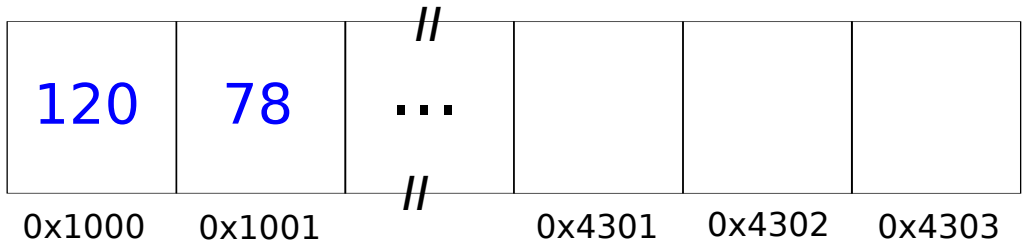
Ejemplo

Memoria:



Ejemplo

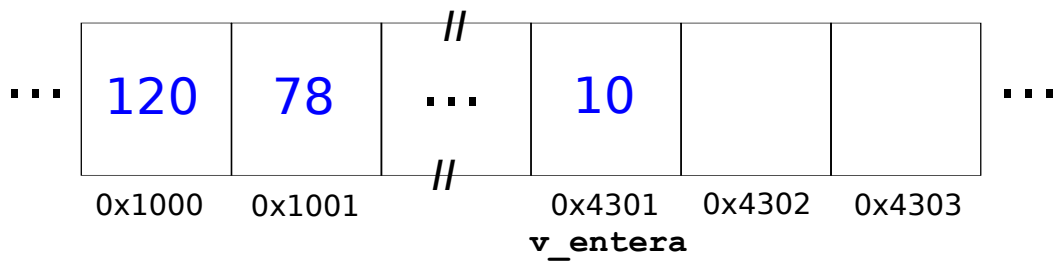
Memoria:



Esta modalidad de referenciar el contenido de una variable para luego operar con el mismo se denomina: **Modalidad Directa**.
Al valor o contenido de una variable se accede por medio de su nombre.

```
int v_entera;
v_entera = 10;
```

Memoria:



Introducción a las variables punteros

- Los **punteros** (o apuntadores) son otros tipos de datos que se utilizan en el lenguaje de programación C,
- Permiten a un programa ser más potente, dinámico y flexible,
- Su uso potencia al lenguaje C debido a que permite el acceso a memoria de una manera más eficiente. Sin embargo, una mala referencia a dicha memoria provocará en el programa una salida inesperada,
- El uso de punteros implica la no manipulación de las variables en sí, sino de manejar **direcciones de memoria** en las cuales residen los datos.

Declaración de una variable Puntero (I)

C provee otra modalidad de trabajo denominada **"Modalidad Indirecta"** a través del uso de **PUNTEROS**.

Los punteros son variables cuyo contenido es una dirección de memoria. En particular, estos son utilizados para referenciar direcciones de memoria de otras variables (*int*, *float*, *char*, *etc.*).

Declaración de una variable Puntero (I)

C provee otra modalidad de trabajo denominada **"Modalidad Indirecta"** a través del uso de **PUNTEROS**.

Los punteros son variables cuyo contenido es una dirección de memoria. En particular, estos son utilizados para referenciar direcciones de memoria de otras variables (*int*, *float*, *char*, *etc.*).

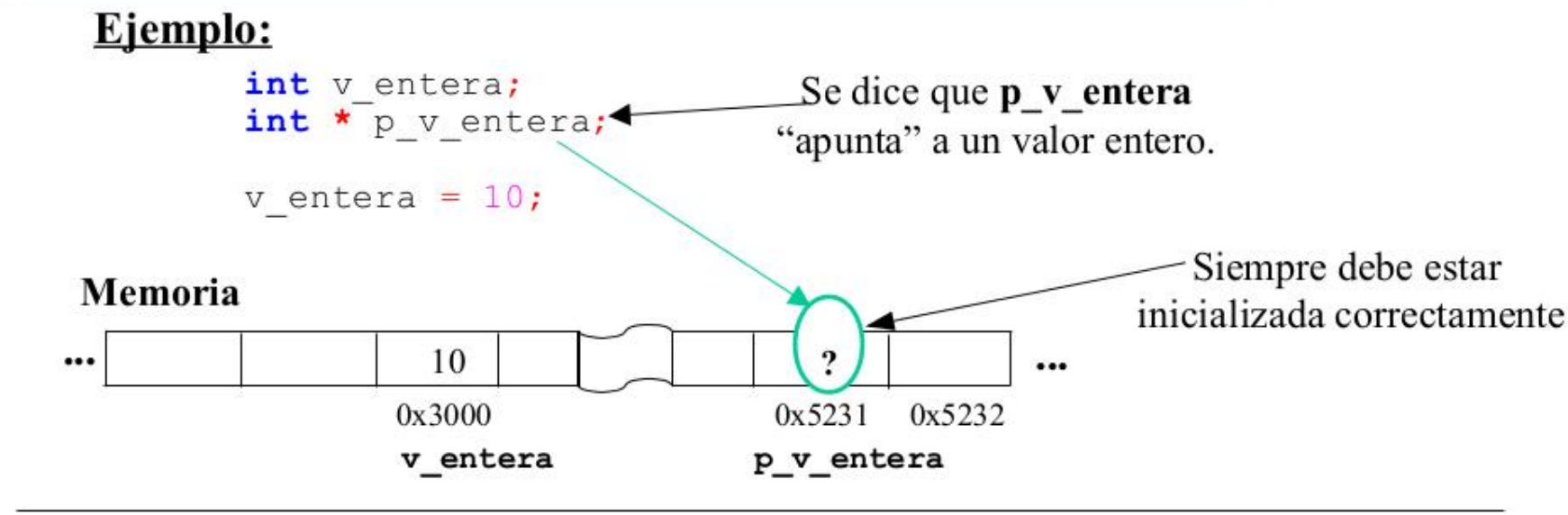
Sintaxis:

$< tipo > * < nombre\_de\_variable >;$

La variable que se declara es "puntero" y contendrá la dirección de una variable que contiene información del tipo $< tipo >$.

Declaración de una variable Puntero (II)

Cuando se declara una variable de tipo apuntador, el compilador también le asigna una dirección de memoria, sin embargo, en esa localidad de memoria no se almacena un dato, sino la dirección de una variable del mismo tipo que se declaró el puntero.
Ejemplo:



Operadores asociados al uso de punteros

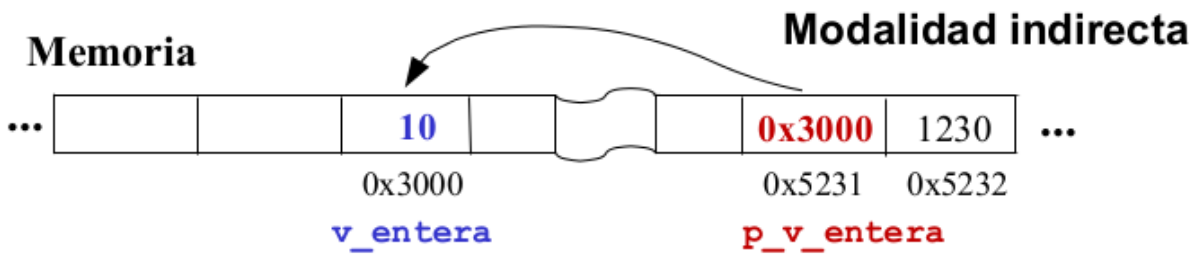
Operador	Descripción
&$< id >$	se asocia a cualquier variable $< id >$, devuelve la dirección de memoria donde comienza la variable $< id >$.
$* < ptr >$	se asocia al nombre de una variable puntero $< ptr >$; devuelve el contenido del objeto referenciado por el puntero $< ptr >$

Operadores asociados al uso de punteros

Operador	Descripción
&< *id* >	se asocia a cualquier variable < *id* >, devuelve la dirección de memoria donde comienza la variable < *id* >.
∗ < *ptr* >	se asocia al nombre de una variable puntero < *ptr* >; devuelve el contenido del objeto referenciado por el puntero < *ptr* >

```
int v_entera;
int * p_v_entera;   // Declaración del puntero

v_entera = 10;
p_v_entera = & v_entera; //Inicialización del puntero
```

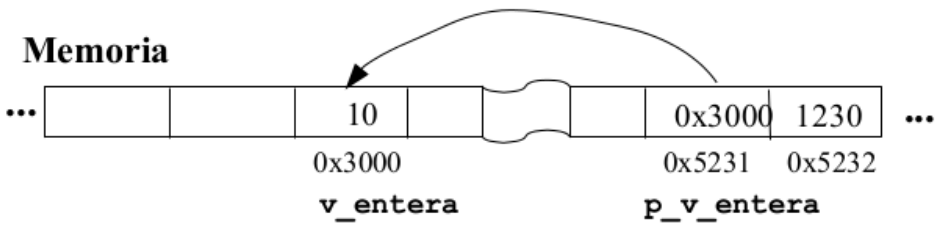


Ejemplo 2:

```c
#include <stdio.h>
int main (){
  int v_entera;
  int * p_v_entera;     // Declara la variable puntero.
  v_entera = 10;
  p_v_entera = & v_entera; // Inicializa la variable puntero.
  printf("El valor apuntado por p_v_entera es:");
  printf("%d",* p_v_entera); // Muestra lo apuntado por el puntero
  return 0;
}
```

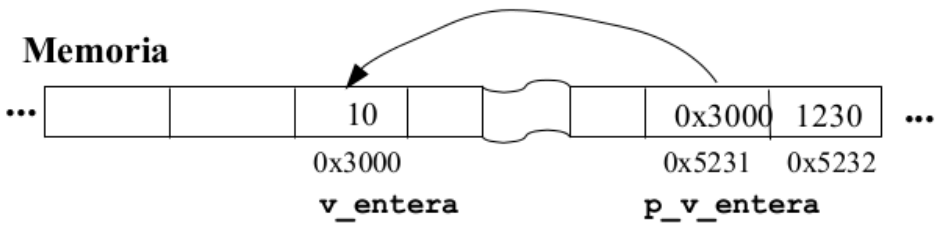
Ejemplo 2:

```
1 #include <stdio.h>
2 int main (){
3   int v_entera;
4   int * p_v_entera;     // Declara la variable
        puntero.
5   v_entera = 10;
6   p_v_entera = & v_entera; // Inicializa la
        variable puntero.
7   printf("El valor apuntado por p_v_entera es:"
        );
8   printf("%d",* p_v_entera); // Muestra lo
        apuntado por el puntero
9   return 0;
10 }
```



Ejemplo 2:

```
1 #include <stdio.h>
2 int main (){
3   int v_entera;
4   int * p_v_entera;     // Declara la variable
        puntero.
5   v_entera = 10;
6   p_v_entera = & v_entera; // Inicializa la
        variable puntero.
7   printf("El valor apuntado por p_v_entera es:"
        );
8   printf("%d",* p_v_entera); // Muestra lo
        apuntado por el puntero
9   return 0;
10 }
```



Salida en pantalla:

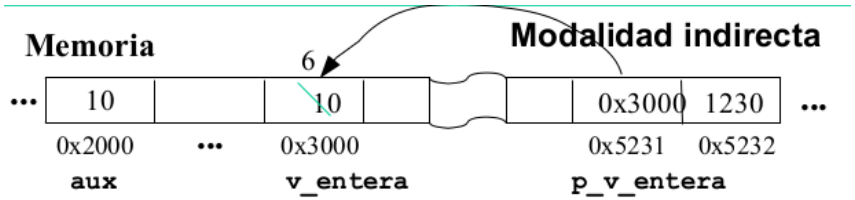
```
prompt $ El valor apuntado por p_v_entera es: 10
```

Ejemplo 3:

```c
#include <stdio.h>
int main (){
  int v_entera, aux;
  int * p_v_entera; // Variable puntero.
  v_entera = 10;
  p_v_entera = & v_entera; // Obtengo la dir de
      una variable
  printf("El valor apuntado por p_v_entera es:"
     );
  printf("%d \n", *p_v_entera);
  aux = * p_v_entera;
  * p_v_entera = 6;
  printf("aux: %d \n", aux);
  printf("El valor apuntado por p_v_entera es:"
     );
  printf("%d \n", *p_v_entera);
  printf("v_entera: %d \n", v_entera);
  return 0;
}
```

Ejemplo 3:

```
1 #include <stdio.h>
2 int main (){
3   int v_entera, aux;
4   int * p_v_entera; // Variable puntero.
5   v_entera = 10;
6   p_v_entera = & v_entera; // Obtengo la dir de
        una variable
7   printf("El valor apuntado por p_v_entera es:"
      );
8   printf("%d \n", *p_v_entera);
9   aux = * p_v_entera;
10  * p_v_entera = 6;
11  printf("aux: %d \n", aux);
12  printf("El valor apuntado por p_v_entera es:"
      );
13  printf("%d \n", *p_v_entera);
14  printf("v_entera: %d \n", v_entera);
15  return 0;
16 }
```

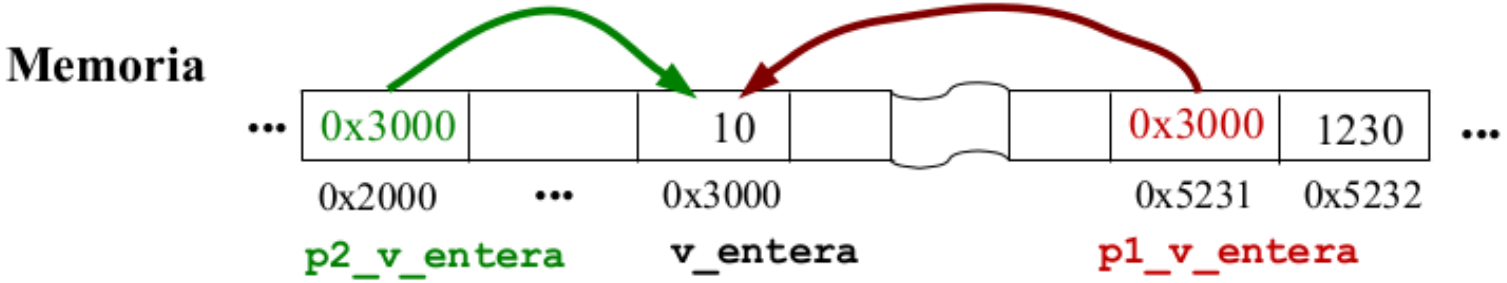


Salida en pantalla:

```
prompt $ El valor apuntado por p_v_entera es: 10
aux: 10
El valor apuntado por p_v_entera es: 6
v_entera: 6
```

Particularidad (I):

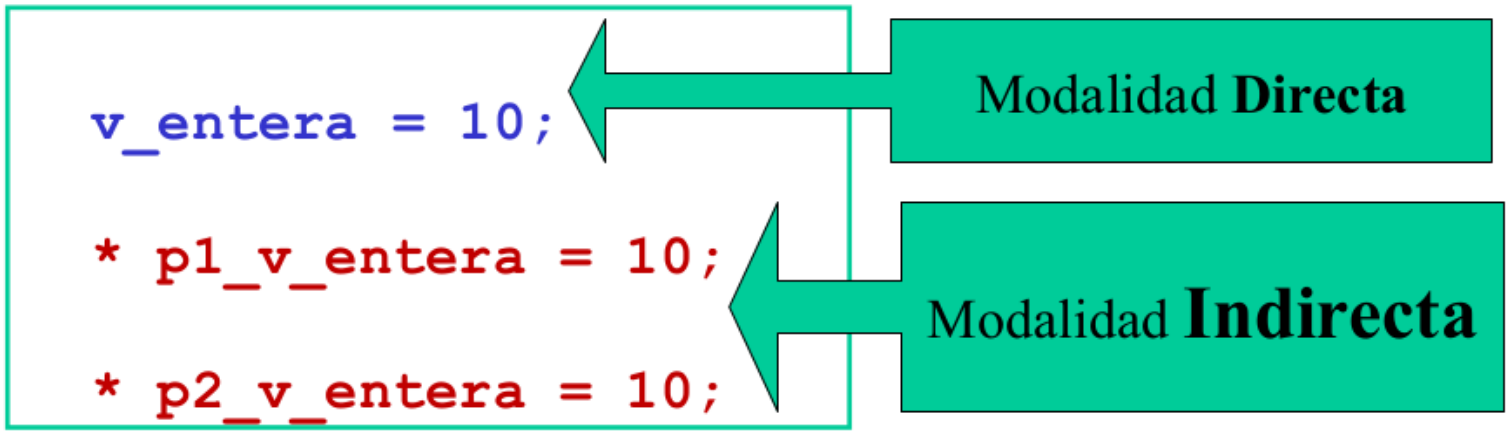
```
 1 #include <stdio.h>
 2 int main (){
 3     int v_entera;
 4     int * p1_v_entera, * p2_v_entera;
 5     v_entera = 10;
 6     p1_v_entera = & v_entera;
 7     p2_v_entera = p1_v_entera;
 8     printf("v_entera: %d \n", v_entera);
 9     printf("Apuntado por p1_v_entera: %d \n",
            * p1_v_entera);
10     printf("Apuntado por p2_v_entera: %d \n",
            * p2_v_entera);
11     return 0;
12 }
```



Salida en pantalla:

```
prompt $ v_entera: 10
Apuntado por p1_v_entera: 10
Apuntado por p2_v_entera: 10
```

Las 3 instrucciones producen el mismo resultado!!!



Particularidad (II):

```c
#include <stdio.h>
int main (){
  int v_entera;
  int * p1_v_entera, * p2_v_entera;
  v_entera = 10;
  p1_v_entera = & v_entera;
  p2_v_entera = p1_v_entera;
  printf("v_entera: %d \n", v_entera);
  printf("Apuntado por p1_v_entera: %d\n",
           *p1_v_entera);
  printf("Apuntado por p2_v_entera: %d\n",
           *p2_v_entera);
  printf("El contenido de p1_v_entera: %p  \n",
       p1_v_entera);
  printf("El contenido de p2_v_entera: %p  \n",
       p2_v_entera);
  return 0;
}
```

Particularidad (II):

```c
#include <stdio.h>
int main (){
  int v_entera;
  int * p1_v_entera, * p2_v_entera;
  v_entera = 10;
  p1_v_entera = & v_entera;
  p2_v_entera = p1_v_entera;
  printf("v_entera: %d \n", v_entera);
  printf("Apuntado por p1_v_entera: %d\n",
            *p1_v_entera);
  printf("Apuntado por p2_v_entera: %d\n",
            *p2_v_entera);
  printf("El contenido de p1_v_entera: %p  \n",
       p1_v_entera);
  printf("El contenido de p2_v_entera: %p  \n",
       p2_v_entera);
  return 0;
}
```

```
prompt $ v_entera: 10
Apuntado por p1_v_entera: 10
Apuntado por p2_v_entera: 10
El contenido de p1_v_entera: 0x3000
El contenido de p2_v_entera: 0x3000
```

Conclusiones

- las variables punteros son una de las razones fundamentales para que el lenguaje C sea tan potente,
- utilizando variables punteros un programa puede realizar muchas tareas que no sera posible slo utilizando tipos de datos estndar,
- se utiliza la palabra reservada VOID en la declaracin de apuntadores, cuando se quiere que el puntero pueda apuntar a cualquier tipo de variables,
  por ejm. int , char, float,....
- muchas veces se hace necesario expresar explcitamente que un apuntador declarado no este apuntando a "NINGÚN"lugar (en apuntadores quiere decir que el apuntador apunta a la direccin CERO)
  char * ptr = NULL;