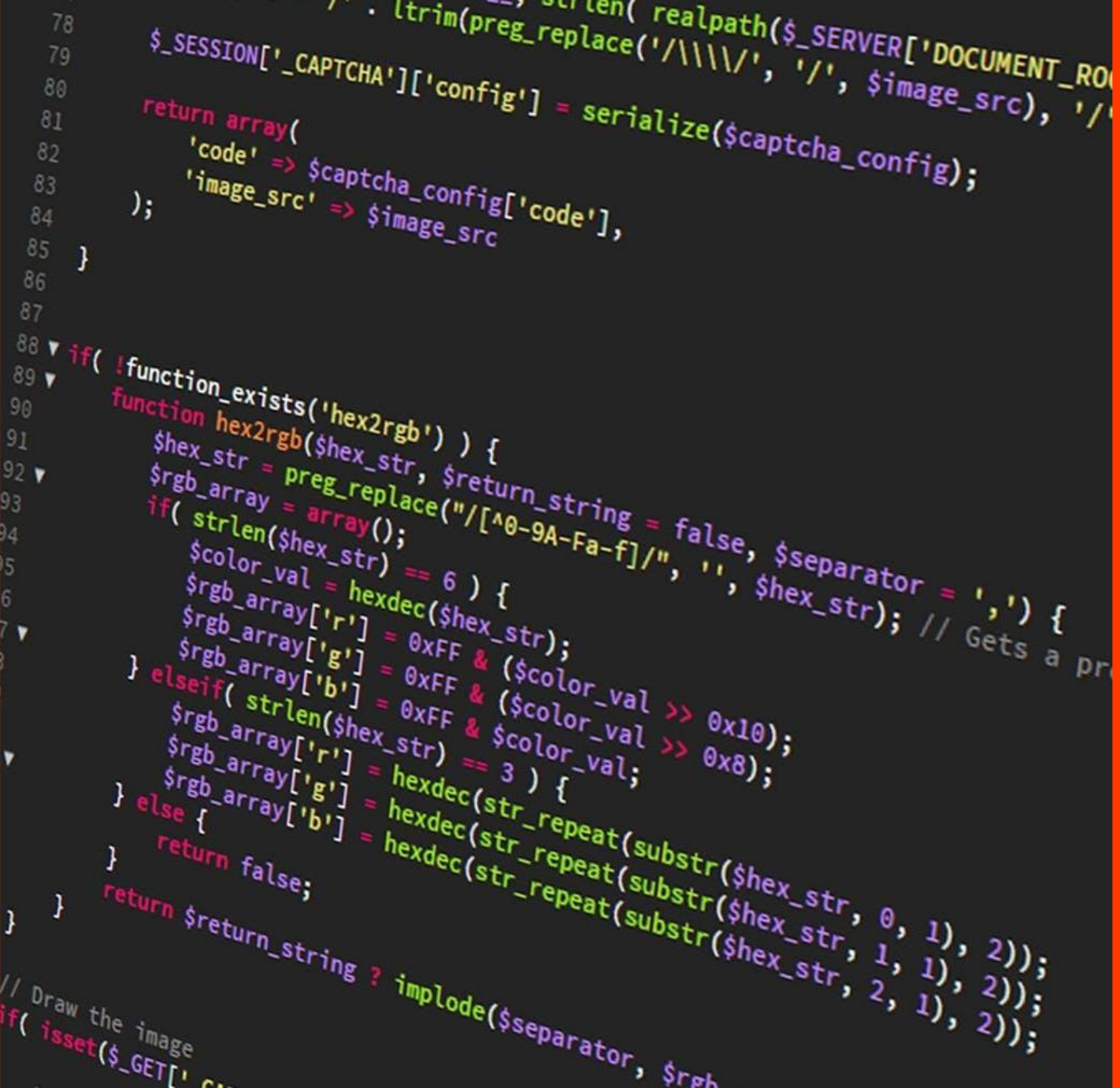
UNIVERSIDAD DE LIMA | 60 años

Paradigma Imperativo (p2)

Lenguajes de Programación

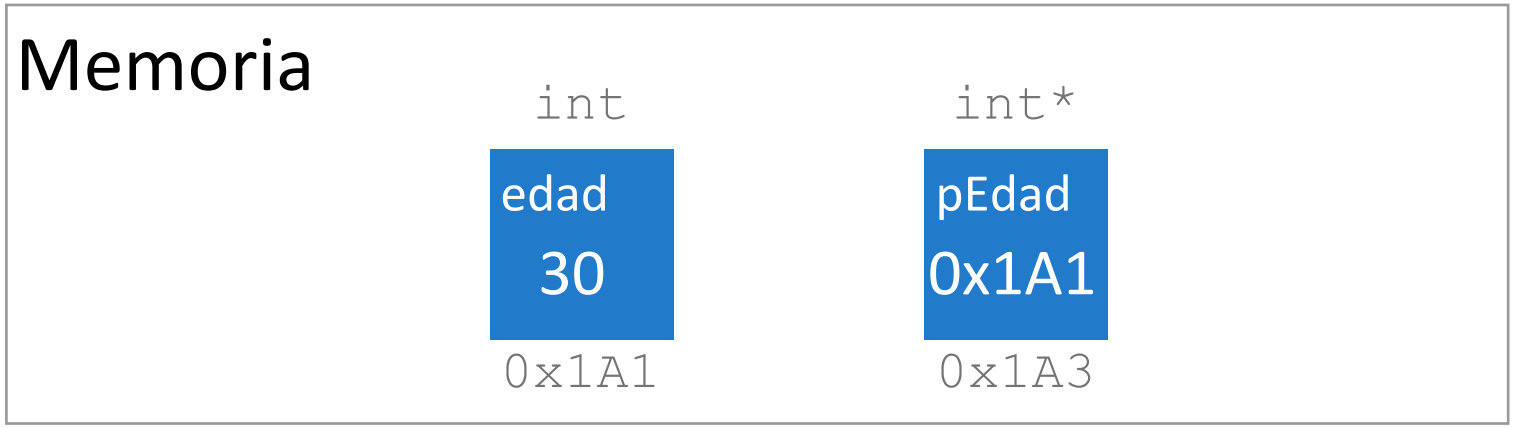

78 . trim(preg_replace('/\\\\/', '/', $image_src), '/'
79 $_SESSION['_CAPTCHA']['config'] = serialize($captcha_config);
80
81 return array(
82 'code' => $captcha_config['code'],
83 'image_src' => $image_src
84);
85 }
86
87
88 if(!function_exists('hex2rgb')) {
89 function hex2rgb($hex_str, $return_string = false, $separator = ',') {
90 $hex_str = preg_replace("/[^0-9A-Fa-f]/", '', $hex_str); // Gets a pr
91 $rgb_array = array();
92 if(strlen($hex_str) == 6) {
93 $color_val = hexdec($hex_str);
94 $rgb_array['r'] = 0xFF & ($color_val >> 0x10);
5 $rgb_array['g'] = 0xFF & ($color_val >> 0x8);
6 $rgb_array['b'] = 0xFF & $color_val;
7 } elseif(strlen($hex_str) == 3) {
$rgb_array['r'] = hexdec(str_repeat(substr($hex_str, 0, 1), 2));
$rgb_array['g'] = hexdec(str_repeat(substr($hex_str, 1, 1), 2));
$rgb_array['b'] = hexdec(str_repeat(substr($hex_str, 2, 1), 2));
} else {
return false;
}
return $return_string ? implode($separator, $rgb
}
// Draw the image
if(isset($_GET['

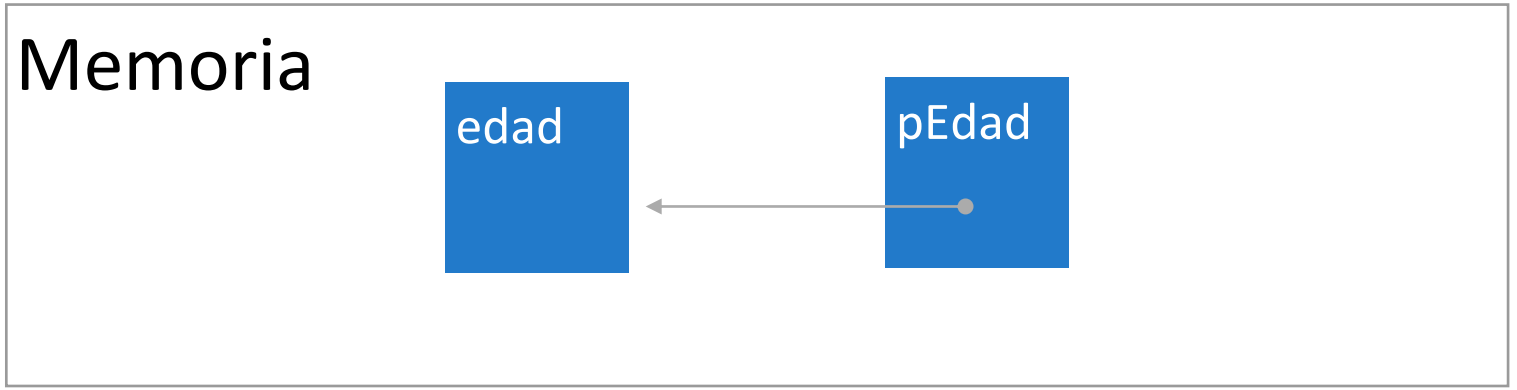
Punteros

Punteros (pointers)

Los punteros referencian (apuntan) posiciones de memoria donde se encuentra la data (valores).

```
int edad;
int* pEdad;

edad = 30;

pEdad = &edad;

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





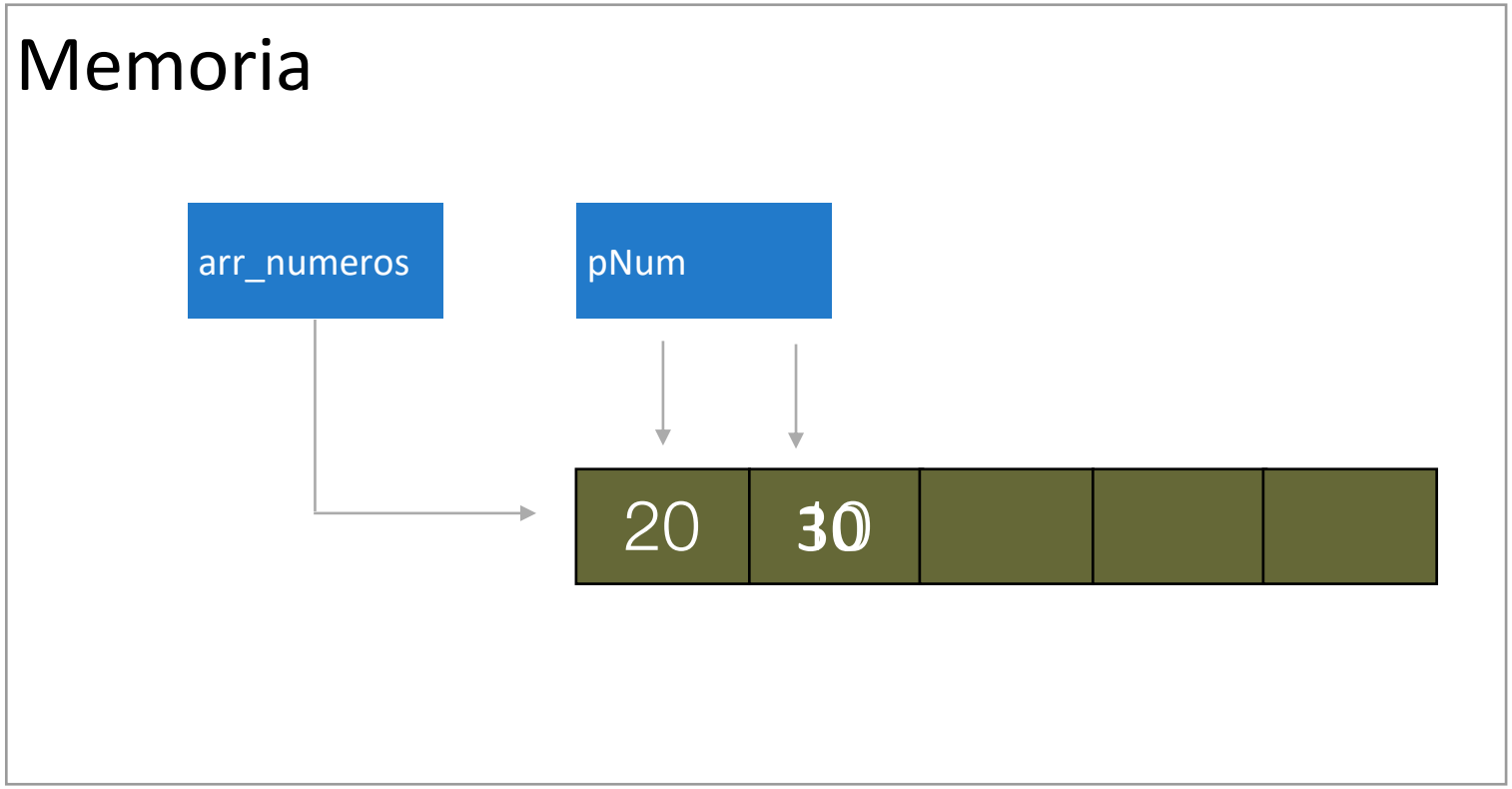
* : Ver el valor al que está apuntando un puntero.

& : Ver el puntero (dirección de memoria) del valor.

UNIVERSIDAD DE LIMA

Punteros a arreglos

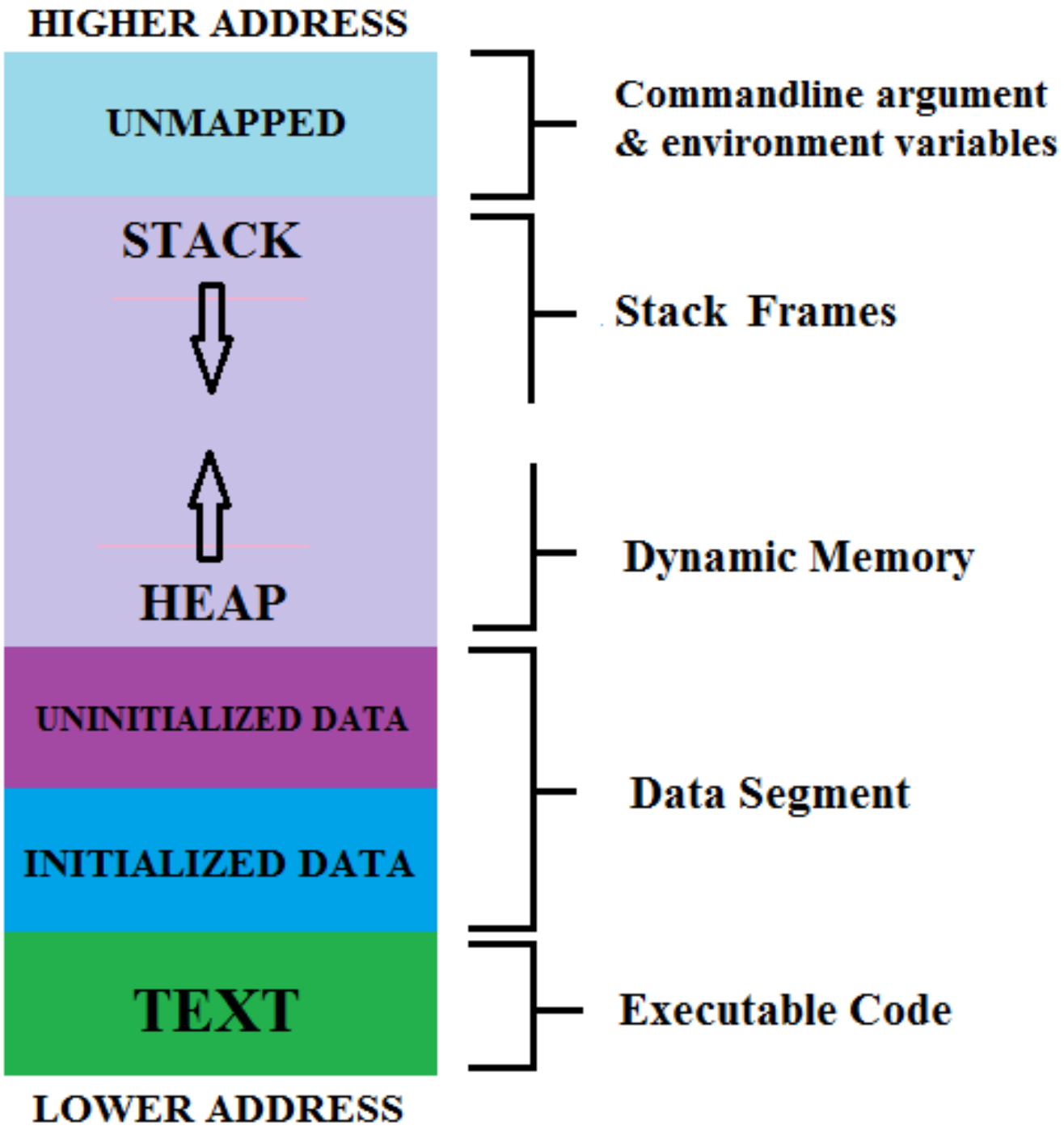
```
int arr_numeros[5];
int* pNum;

pNum = arr_numeros;

pNum[0] = 20;
pNum[1] = 10;

pNum++;

pNum[0] = 30;


printf("%d\n", pNum[0]); ??
```



Los identificadores de los arreglos son realmente punteros a ellos.

Si aplicamos operadores aritméticos, los punteros se "mueven" (realmente se suma el valor de las direcciones de memoria.

UNIVERSIDAD DE LIMA



Gestión Dinámica de Memoria

Asignación dinámica de memoria

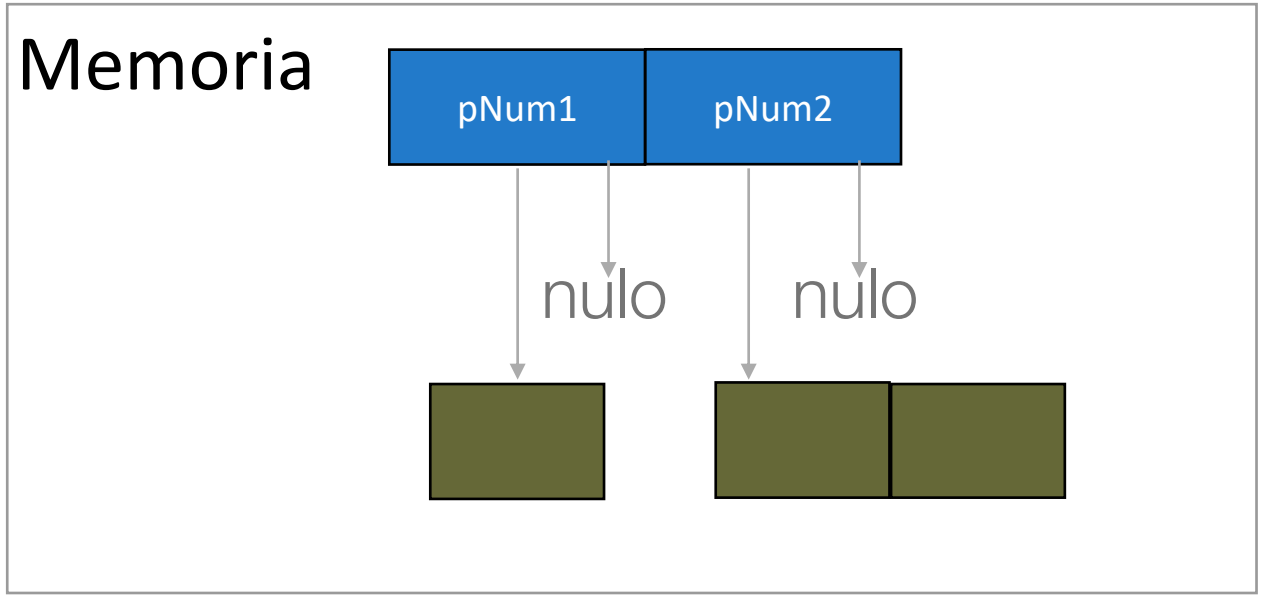
Hemos visto que podemos declarar variables de tipo **puntero**.

Estas variables almacenan una dirección de memoria (apuntan), pero si explícitamente no lo definimos, por defecto apuntan a nulo.

```
int* pNum1, pNum2;
pNum1 = new int();
pNum2 = new int();
```

La sentencia new se encarga de separar posiciones de memoria en el heap (o montón).

La cantidad de memoria separada depende del tipo que se está haciendo new.



UNIVERSIDAD DE LIMA

Liberación de memoria (explícita)

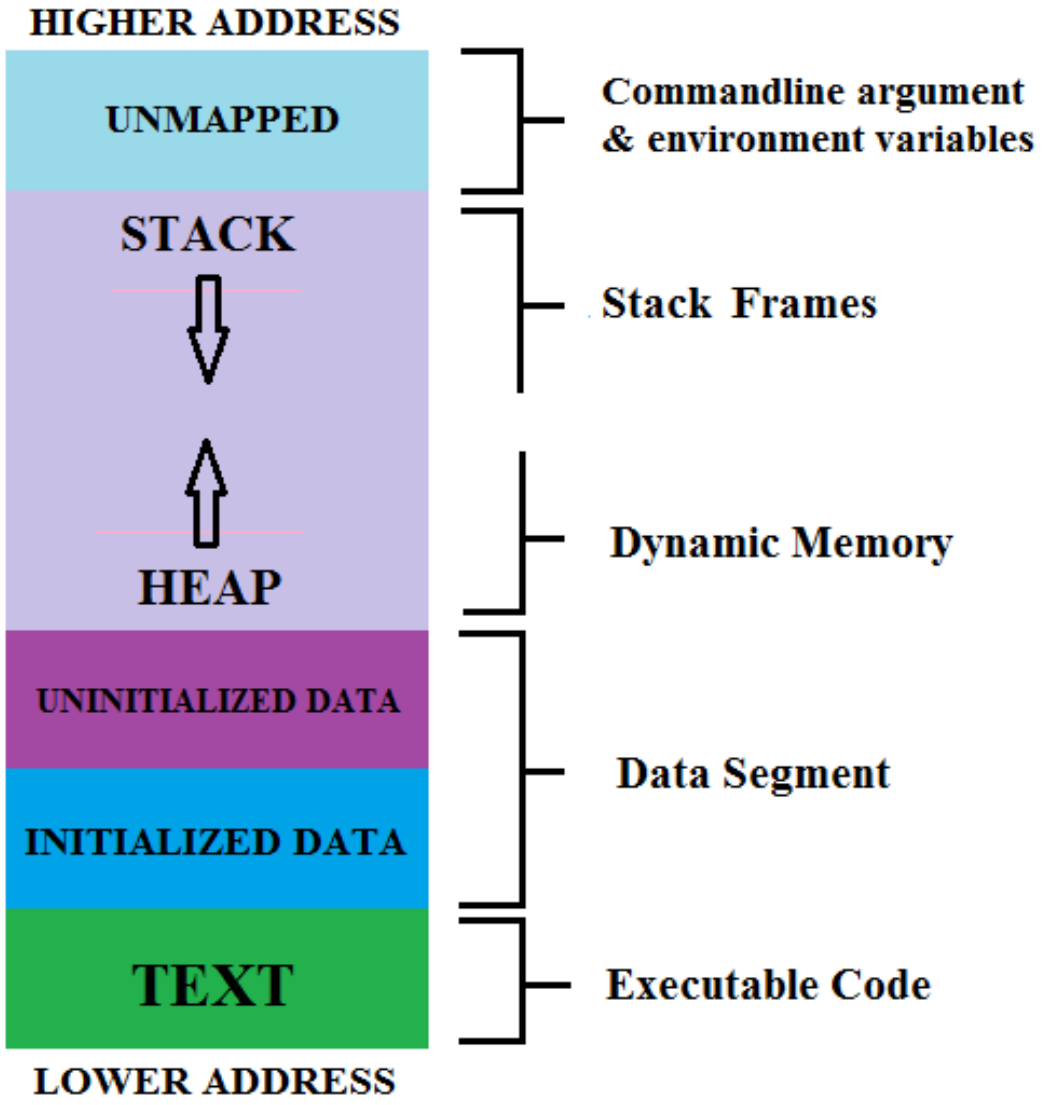
- Un gran problema de C++ es que automáticamente no se libera la memoria (a diferencia de otros lenguajes de programación). Esto es, que si se crean demasiadas variables en el heap podríamos **llenarlo** lo que ocasionaría que se caiga nuestro programa.

- Utilizaremos la función delete para liberar de manera explícita la memoria.

```
int* pNum1, pNum2;
pNum1 = new int();
pNum2 = new int();
delete pNum1;
delete pNum2;
```

UNIVERSIDAD DE LIMA

Mapa de la memoria asignada



UNIVERSIDAD DE LIMA