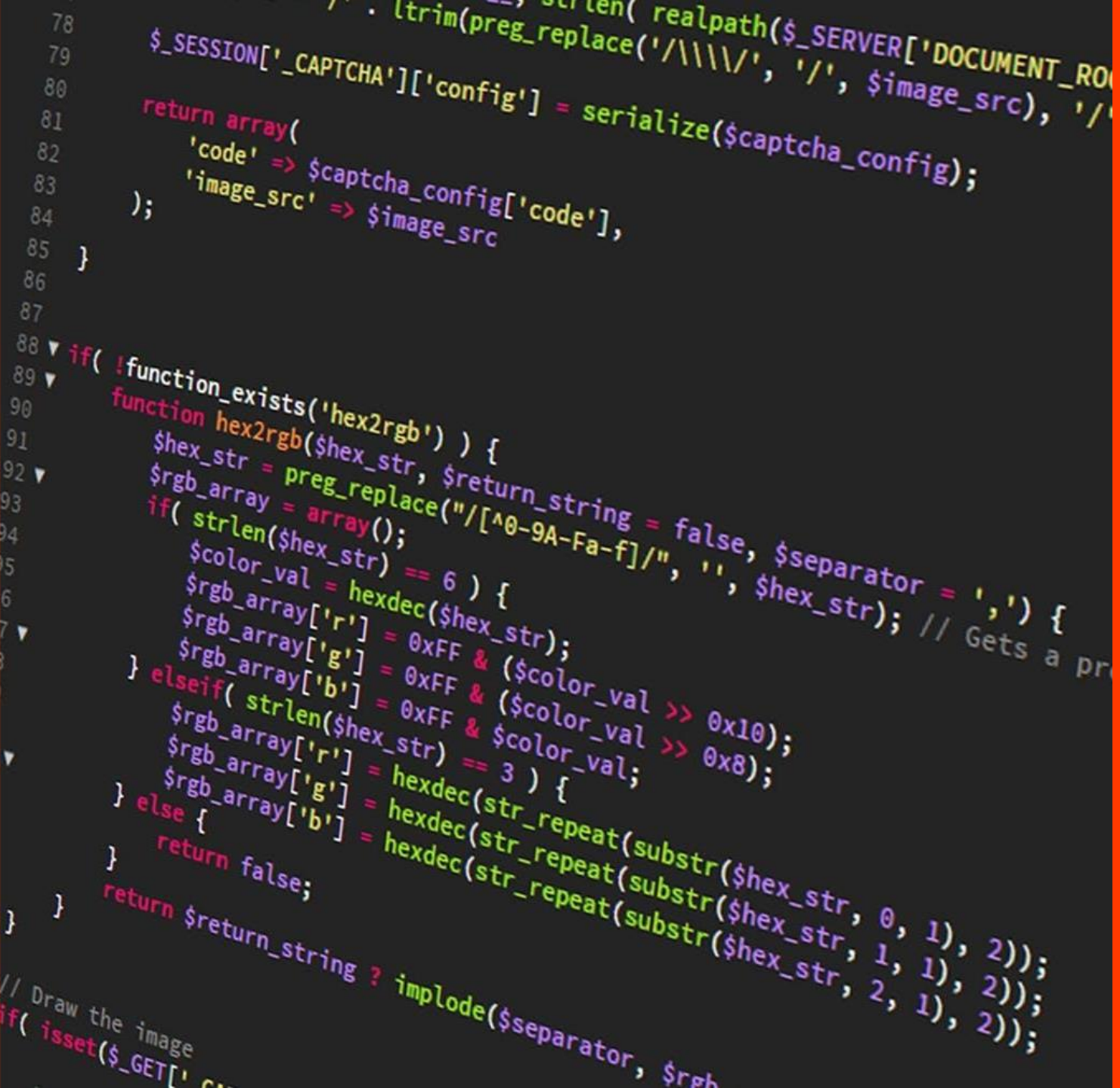
UNIVERSIDAD DE LIMA | 60 años

Paradigma Imperativo (p2)

Lenguajes de Programación


realpath($_SERVER['DOCUMENT_RO
trim(preg_replace('/\\\\/', '/', $image_src), '/
$_SESSION['_CAPTCHA']['config'] = serialize($captcha_config);
return array(
'code' => $captcha_config['code'],
'image_src' => $image_src
);
}
if(!function_exists('hex2rgb')) {
function hex2rgb($hex_str, $return_string = false, $separator = ',') {
$hex_str = preg_replace("/[^0-9A-Fa-f]/", '', $hex_str); // Gets a pr
$rgb_array = array();
if(strlen($hex_str) == 6) {
$color_val = hexdec($hex_str);
$rgb_array['r'] = 0xFF & ($color_val >> 0x10);
$rgb_array['g'] = 0xFF & ($color_val >> 0x8);
$rgb_array['b'] = 0xFF & $color_val;
} elseif(strlen($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['

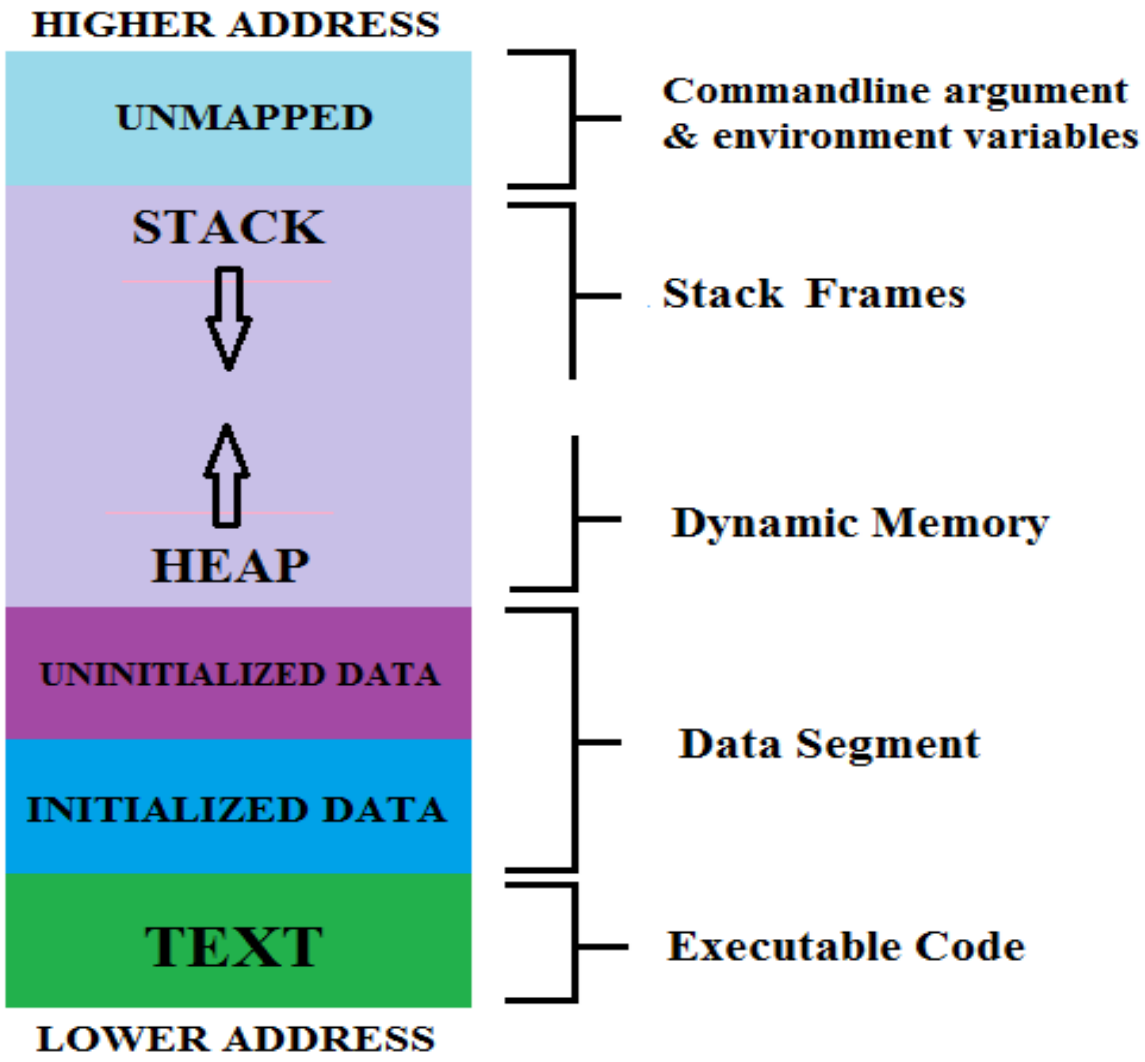
Memoria

Memoria en C++

- Existen 2(*) tipos diferentes de memoria:
  - Stack
  - Heap

- Cuando nuestro programa es ejecutado, el SO nos provee con un espacio de memoria, donde se almacenarán todas nuestras estructuras del programa.

Memoria en C++



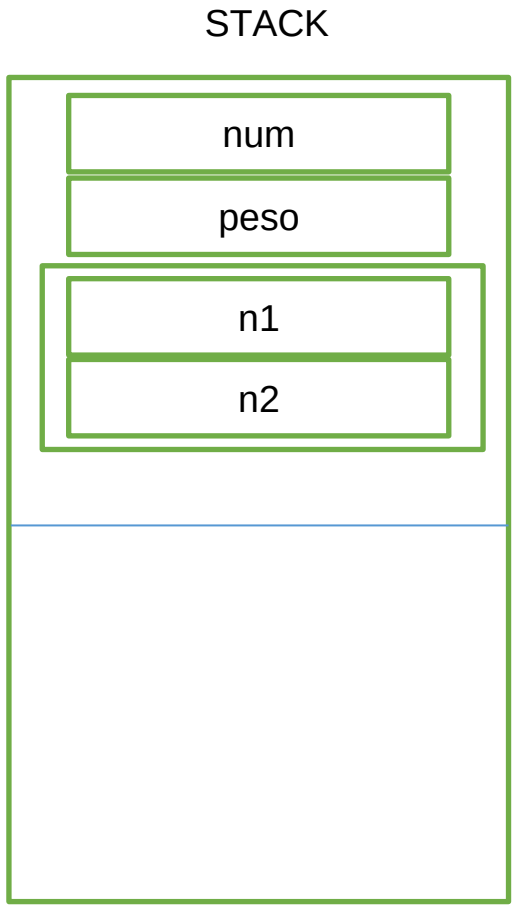
Se encuentra en la memoria RAM

UNIVERSIDAD DE LIMA

Asignación de Memoria

- **Stack:**

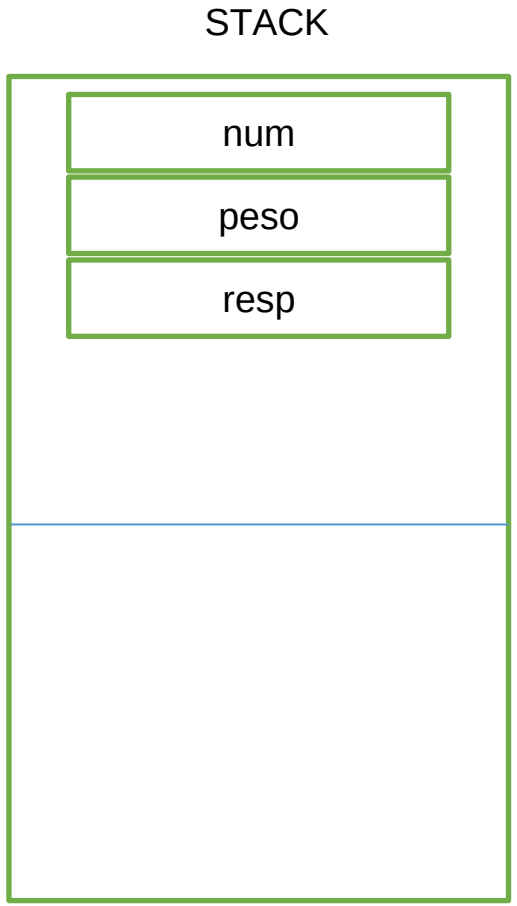
```
int sumar(int n1, int n2)
{
   return n1 + n2;
}

int main()
{
   int num = 20;
   float peso = 94.4f;
   int resp = sumar(10, 20);
}
```



Asignación de Memoria

- **Stack:**

```
int sumar(int n1, int n2)
{
   return n1 + n2;
}

int main()
{
   int num = 20;
   float peso = 94.4f;
   int resp = sumar(10, 20);
}
```



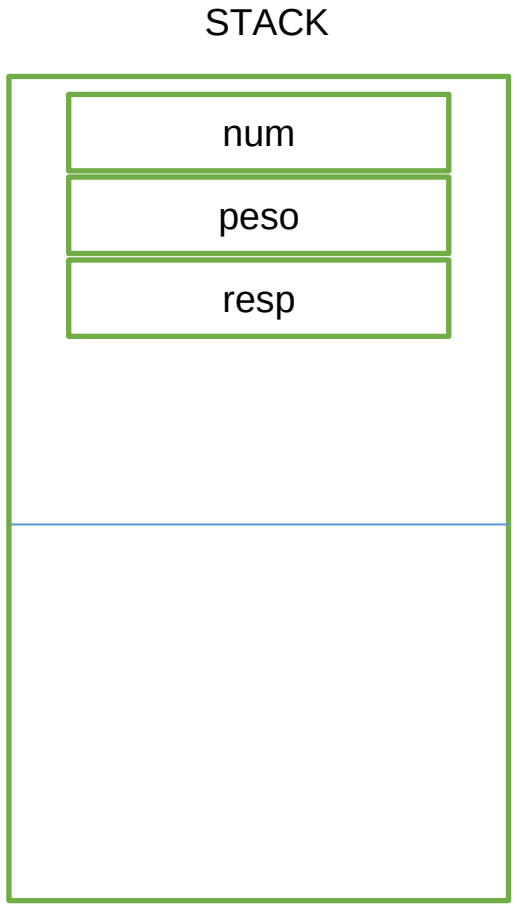
Asignación de Memoria

- **Stack:**

La asignación / liberación de las variables dependen de su *scope* (ámbito de vida).

Características:

- Gestión automática, pero estática.
- El tamaño disponible no es muy grande (**Stackoverflow**).
- Toda variable que definamos, se le asigna un espacio en el stack.

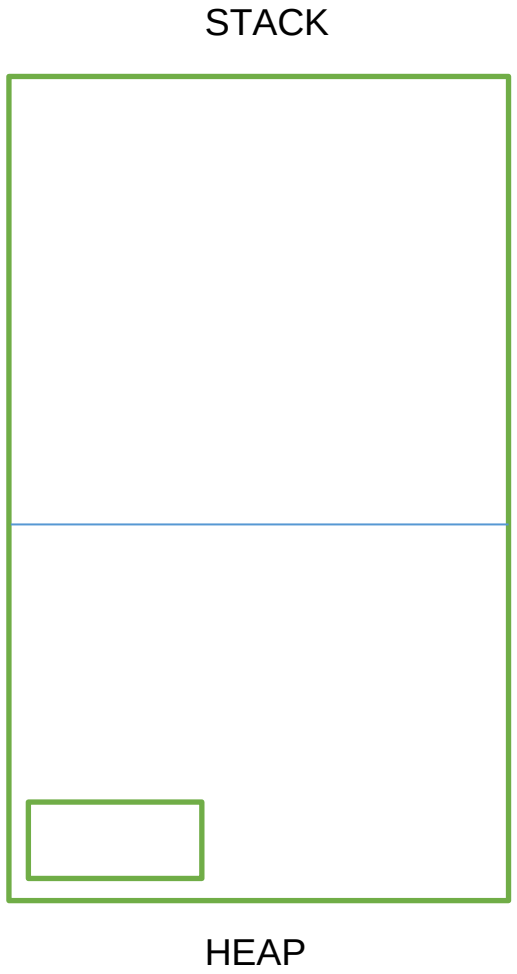


UNIVERSIDAD DE LIMA

Asignación de Memoria

- **Heap:**

```
int main()
{
    char* car = new char;
    int* pNum = new int[2]; // Asignacion dinamica
}
```

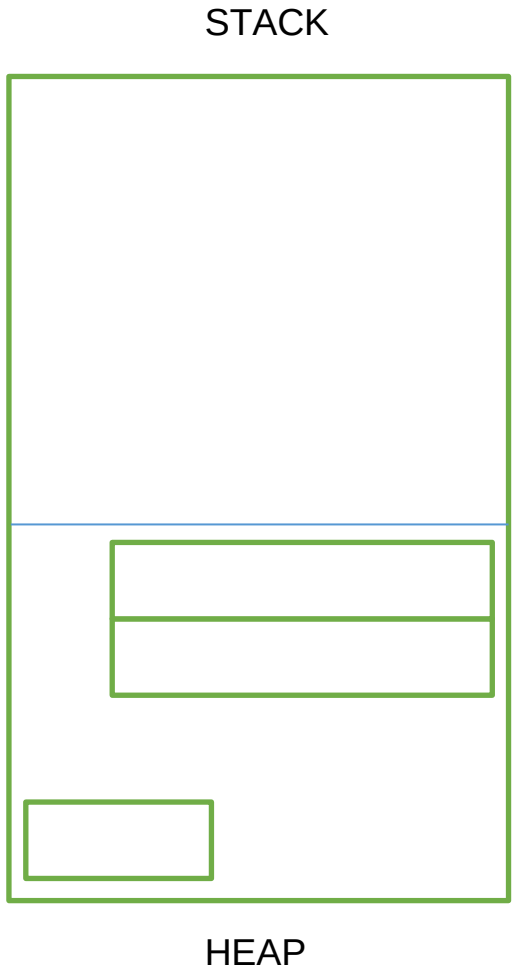


UNIVERSIDAD DE LIMA

Asignación de Memoria

- **Heap:**

```
int main()
{
    char* car = new char;
    int* pNum = new int[2]; // Asignacion dinamica
}
```

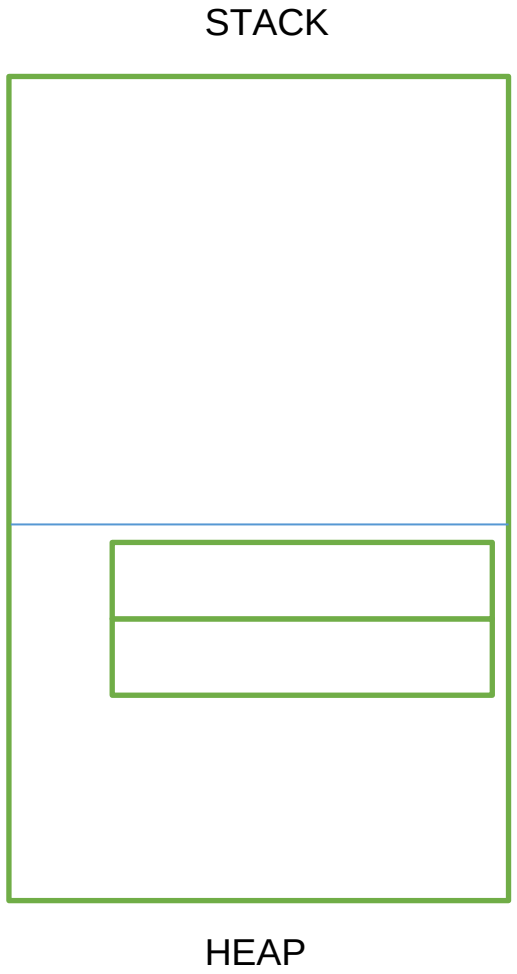


UNIVERSIDAD DE LIMA

Asignación de Memoria

- **Heap:**

```
int main()
{
   char* car = new char;
   int* pNum = new int[2]; // Asignacion dinámica
   delete car;
   delete[] pNum; // Liberacion
}
```

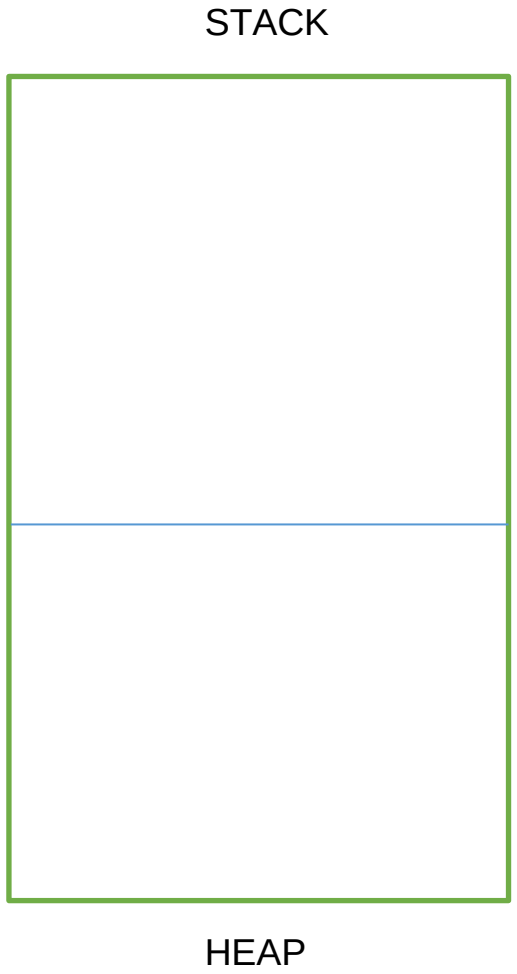


UNIVERSIDAD DE LIMA

Asignación de Memoria

- **Heap:**

```
int main()
{
   char* car = new char;
   int* pNum = new int[2]; // Asignacion dinámica
   delete car;
   delete[] pNum; // Liberacion
}
```



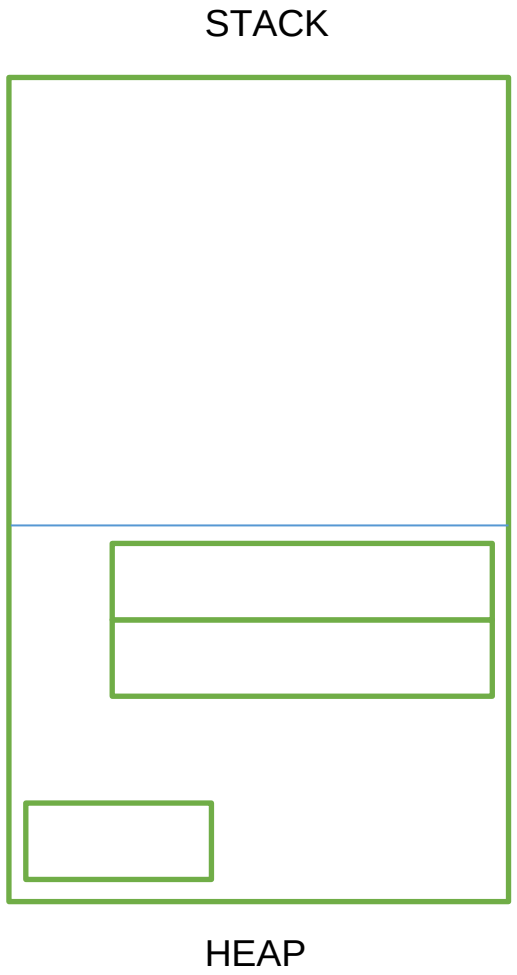
Asignación de Memoria

- **Heap:**

La asignación / liberación de las variables dependen de la lógica del código.

Características:

- Gestión manual, pero dinámica.
- **Más** espacio que en el STACK.
- El SO **busca** una posición libre en el HEAP y al encontrarla, devuelve su **dirección de memoria** (puntero). => Mayor costo computacional
- Se accede a este espacio via un **puntero**.



STACK vs HEAP

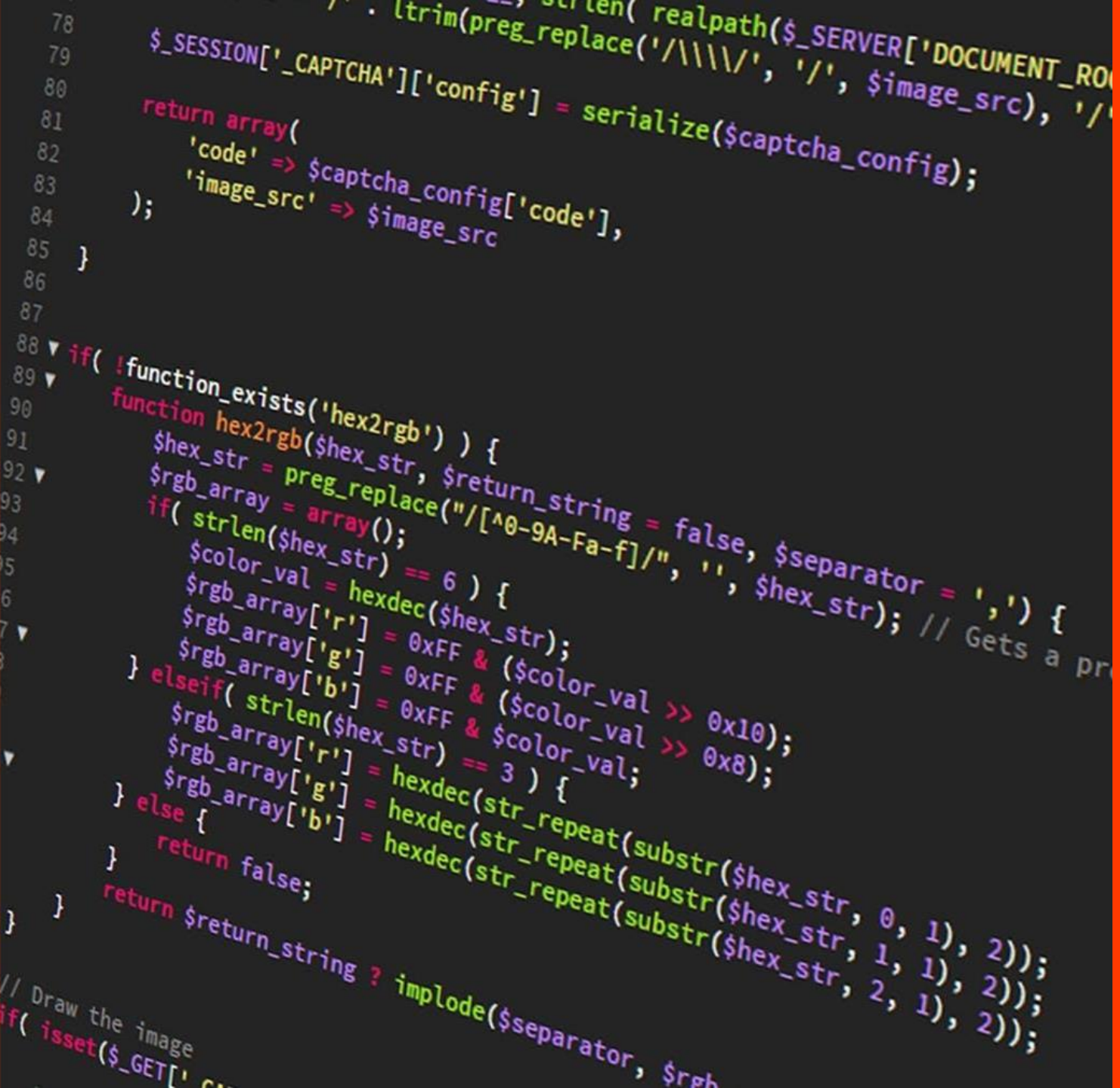
- **Stack:**

  - Espacio limitado.
  - Menor tiempo de asignación.
  - Se accede directamente mediante la variable.
  - Liberación de memoria automática (*scope*).

- **Heap:**

  - Espacio más grande.
  - Mayor tiempo de asignación.
  - Se accede mediante el uso de puntero.
  - Liberación de memoria por código (**delete** keyword).

TIP:
Si se requiere almacenar variables que ocupen mucho espacio (listado de objetos), preferir almacenar en el HEAP.
Caso contrario, almacenamiento en el STACK es suficiente.

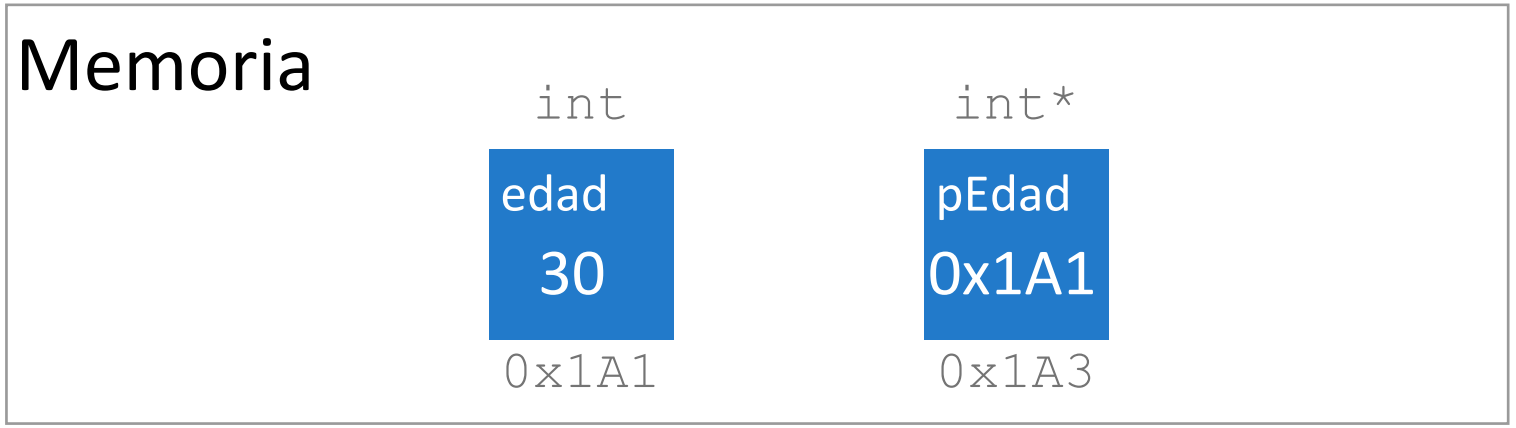

78 . trim(preg_replace('/\\\\/', '/', $image_src), '/'
79 $_SESSION['_CAPTCHA']['config'] = serialize($captcha_config);
80
81 return array(
82 'code' => $captcha_config['code'],
83 'image_src' => $image_src
84);
85 }
86
87
88 if(!function_exists('hex2rgb')) {
89 function hex2rgb($hex_str, $return_string = 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['

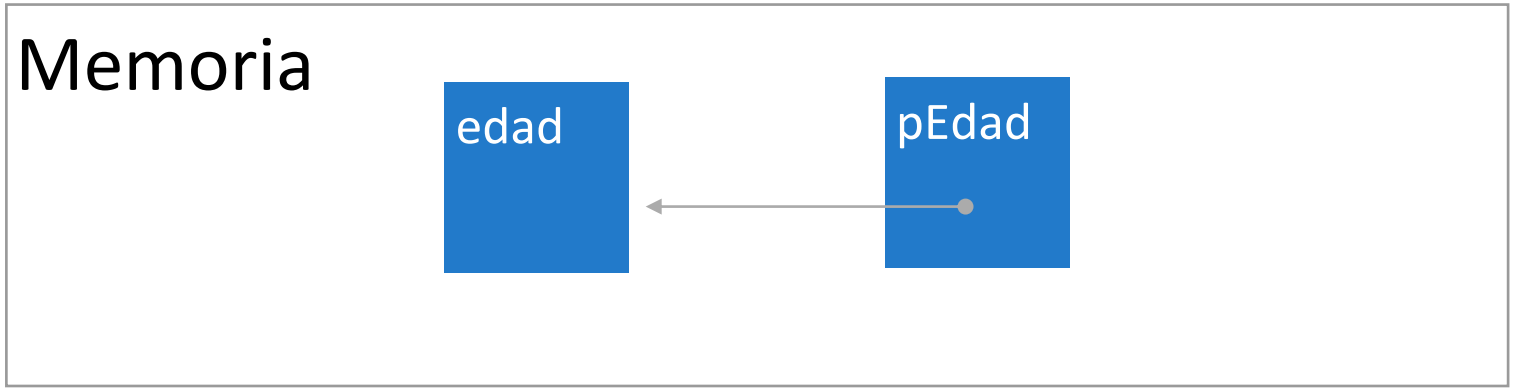
Punteros

Punteros (pointers)

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

```
int edad;
int* pEdad;

edad = 30;

pEdad = &edad;

std::cout << pEdad;

std::cout << *pEdad;
```





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

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

UNIVERSIDAD DE LIMA

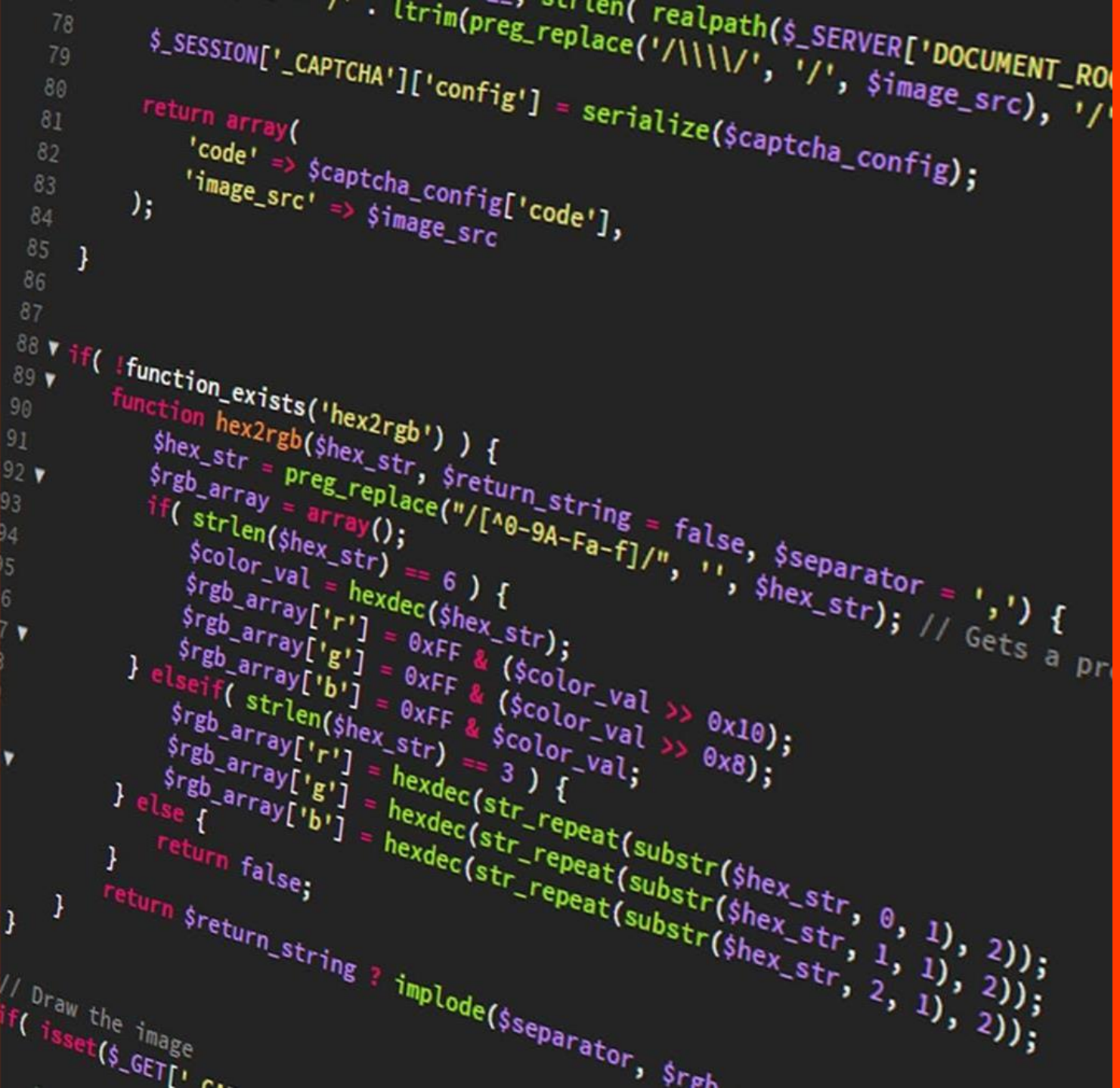
Formas de uso de punteros

- Se tienen 2 operadores principales:
  - Address_of (**&**): Nos devuelve la dirección de memoria (puntero) de una variable.

```
int* p = &num1;
```

  - Dereference (*): Nos devuelve el valor de lo que está apuntando el puntero (el valor de la dirección de memoria).

```
int num2 = *p;
```



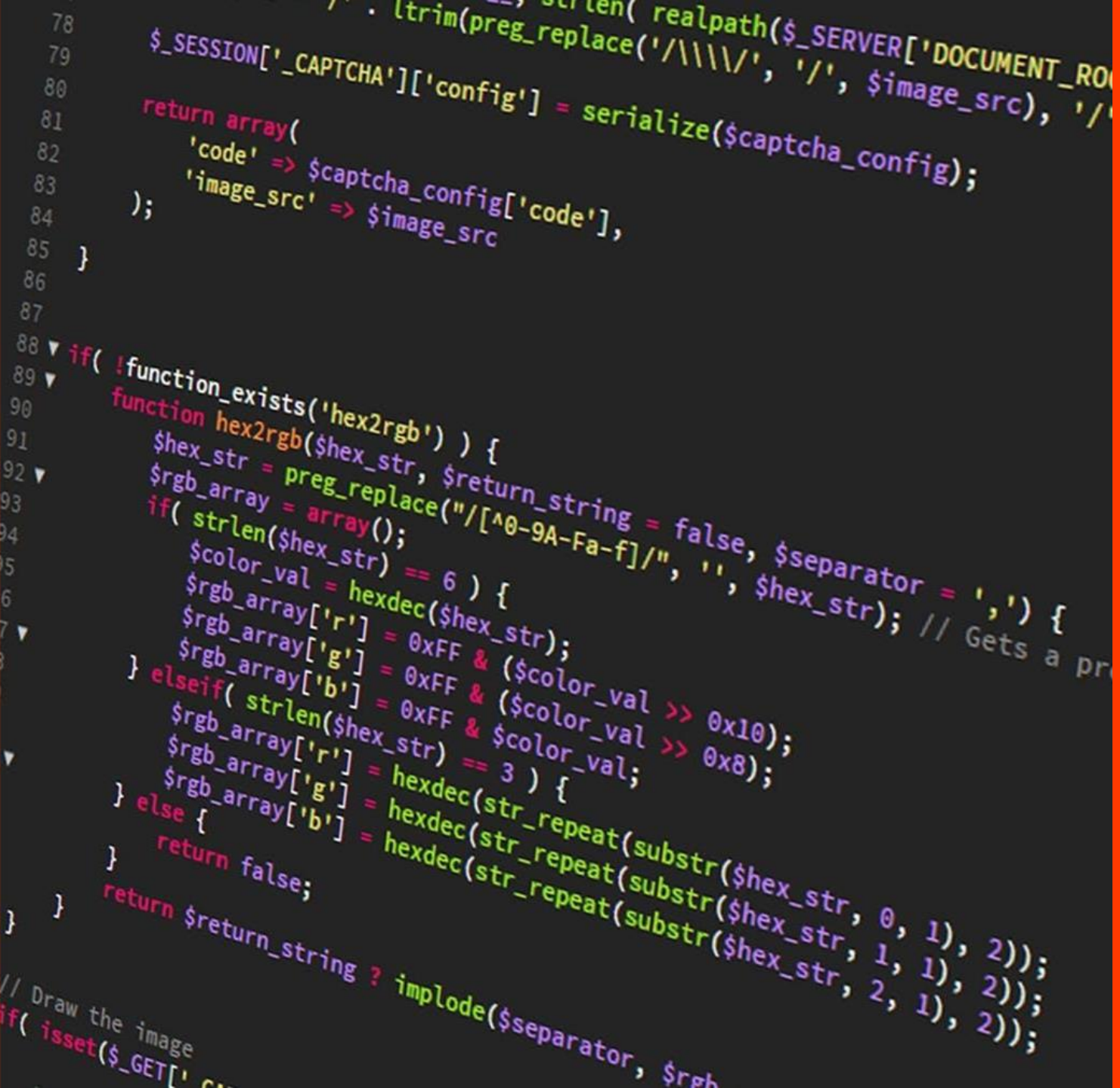
Referencias

Referencias

- Es una etiqueta o alias de una variable existente.
- Una vez que se asocia a una variable, no puede referirse a otra, ni volverse nula.

```
int x = 10;
int& ref = x; // 'ref' es una referencia a 'x'
```

- Cualquier operación que se haga sobre ref, afectarán también a x => Apuntan al mismo espacio de memoria.



Pase de argumento de entrada a funciones

Pase por referencia vs Pase por valor

- C++ nos permite decidir si pasar una **referencia** o una copia de la variable (argumentos de entrada de función).

- Esto nos permite realizar código mucho más eficiente en cuanto a tiempo computacional.

¿Pero... para que sirven los punteros?

```
struct Alumno
{
   std::string codigo;
   std::string nombre;
   int edad;
};

void Imprimir(Alumno al)
{
   std::cout << "Codigo: " << al.codigo << std::endl;
   std::cout << "Nombre: " << al.nombre << std:: endl;
   std::cout << "Edad: " << al.edad << std::endl;
}

int main()
{
   Alumno alumno;
   alumno.codigo = "20232123";
   alumno.nombre = "Dennis Ritchie";
   alumno.edad = 20;

   Imprimir(alumno);
}
```

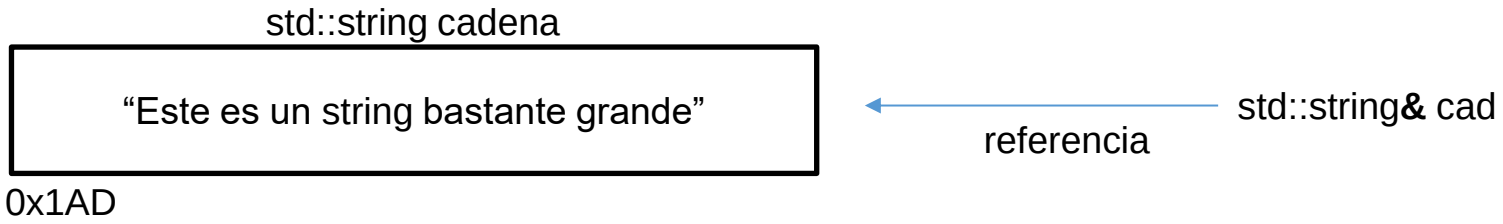
```
struct Alumno
{
   std::string codigo;
   std::string nombre;
   int edad;
};

void ImprimirPorReferencia(Alumno& al)
{
   std::cout << "Codigo: " << al.codigo << std::endl;
   std::cout << "Nombre: " << al.nombre << std:: endl;
   std::cout << "Edad: " << al.edad << std::endl;
}

int main()
{
   Alumno alumno;
   alumno.codigo = "20232123";
   alumno.nombre = "Dennis Ritchie";
   alumno.edad = 20;

   ImprimirPorReferencia(alumno);
}
```

UNIVERSIDAD DE LIMA

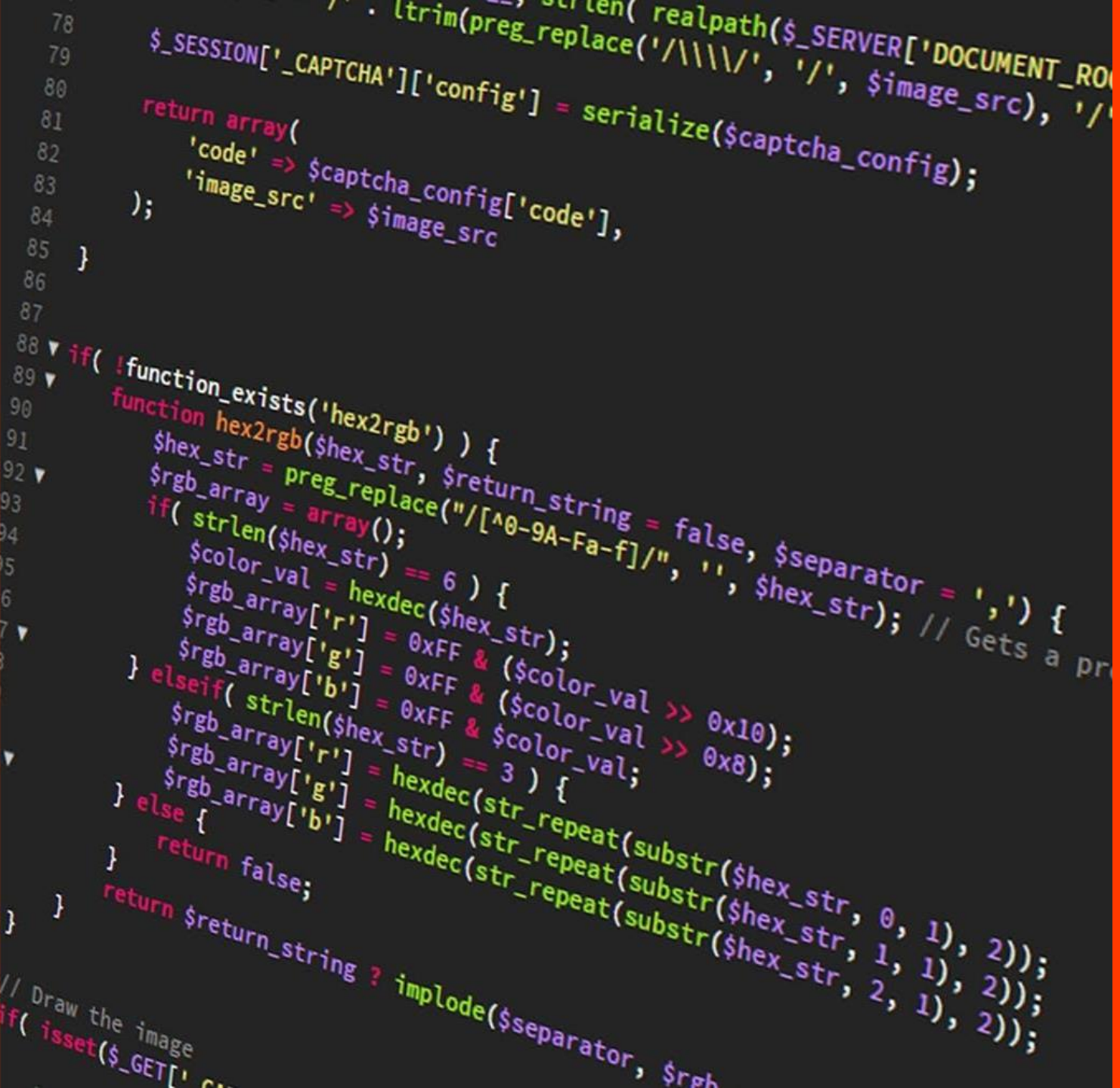
Pase por referencia vs Pase por valor

- Cuando se pasa por valor, se realiza una copia de lo que la variable almacena.

- Cuando se pasa por referencia, se pasa un **alias** a una variable, por lo tanto, al utilizar la variable dentro de la función, se está referenciando la variable creada por fuera.



UNIVERSIDAD DE LIMA



Inmutabilidad

Inmutabilidad

- Al igual que en el paradigma funcional, en C++ también podemos definir variables que son **inmutables**.

```
const bool esVerdad = true;
```

- Además, podemos definir argumentos de entrada de funciones inmutables.
  - Esto para asegurarnos de modificar **referencias**.

Inmutabilidad de argumentos de entrada

```
void ImprimirPorReferenciaMutable(Alumno& al)
{
   al.nombre = ""; // Puedo modificar el nombre, pero nodeberia
   std::cout << "Codigo: " << al.codigo << std::endl;
   std::cout << "Nombre: " << al.nombre << std:: endl;
   std::cout << "Edad: " << al.edad << std::endl;
}
```

```
void ImprimirPorReferenciaNoMutable(const Alumno& al)
{
   al.nombre = ""; // No nos permite. ERROR!
   std::cout << "Codigo: " << al.codigo << std::endl;
   std::cout << "Nombre: " << al.nombre << std:: endl;
   std::cout << "Edad: " << al.edad << std::endl;
}
```