

# Intro a C

# Punteros

With ❤ by @vichoeq & @KnowYourselfes

# Bases Numéricas #2



# Base Decimal

Los números que usamos normalmente son en base 10

The diagram illustrates the expansion of the decimal number 293706 into its place value components. On the left, the number 293706 is shown with each digit in a different color. Lines connect each digit to its corresponding power of 10: 2 to  $10^5$ , 9 to  $10^4$ , 3 to  $10^3$ , 7 to  $10^2$ , 0 to  $10^1$ , and 6 to  $10^0$ . Each connection is labeled with a multiplication sign 'x'. On the right, the expanded form is shown as a sum of these products, with each term aligned to the right. The terms are: 200000 (cyan), 90000 (purple), 3000 (pink), 700 (orange), 00 (yellow), and 6 (green). A horizontal line is drawn under the sum, and the final result, 293706, is shown below it in the same color-coded digits.

$$\begin{array}{r} 200000 + 90000 + 3000 + 700 + 00 + 6 \\ \hline 293706 \end{array}$$

# Base Hexadecimal

La base 16 usa los dígitos A, B, C, D, E y F para valores sobre 9

Diagram illustrating the conversion of the hexadecimal number  $0xC2AF3B$  to decimal by expanding it into powers of 16:

$\times 16^0$	11
$\times 16^1$	48
$\times 16^2$	3840
$\times 16^3$	40960
$\times 16^4$	131072
$\times 16^5$	12582912
	<hr/>
	12758843

Es muy útil para representar números extremadamente grandes.

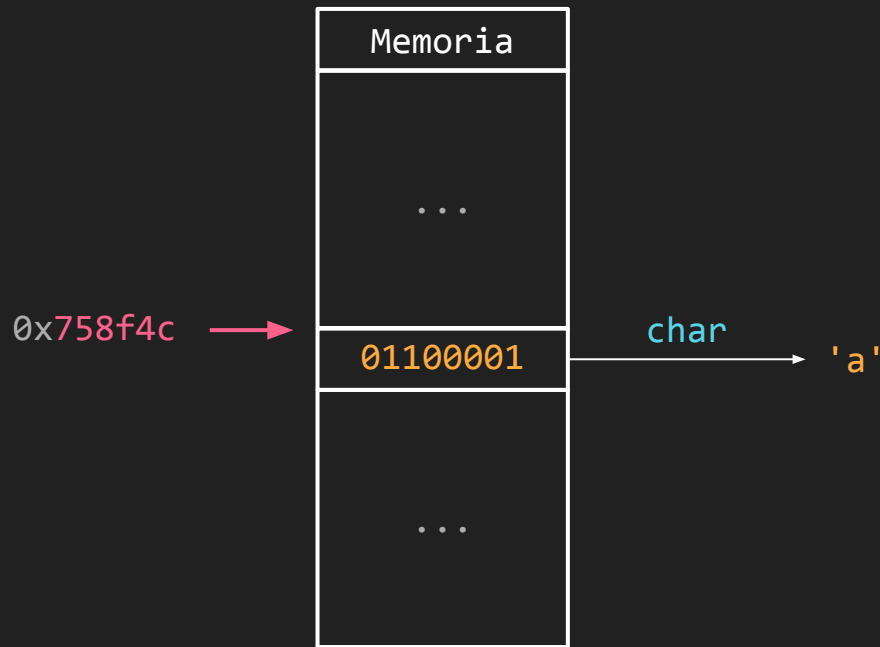
# Variables en un programa

Cada variable tiene:

- Una **dirección** de memoria
- Un **tipo**
- Un **valor**

El **tipo** indica dos cosas:

- Tamaño en memoria
- Interpretación



# Dirección de memoria

Cada variable tiene:

- Una **dirección** de memoria
- Un **tipo**
- Un **valor**

Puede pensarse en la memoria como una tabla gigante que asocia **direcciones** a **bytes**.

Memoria	
...	
0x7a58f8 →	01111010
0x7a58f9 →	11111011
0x7a58fa →	00100110
0x7a58fb →	11010011
0x7a58fc →	00011101
0x7a58fd →	11000011
0x7a58fe →	01011000
0x7a58ff →	00011101
0x7a5900 →	11011000
0x7a5901 →	11101001
...	

# Dirección de memoria

Las **direcciones** de memoria apuntan al comienzo de un bloque.

Para efectos de los diagramas, las flechas apuntarán al centro del primer **valor** de un bloque.

Además las direcciones se mostrarán con 6 dígitos hexadecimales en lugar de 12

Memoria	
...	
0x7a58f8 →	01111010
0x7a58f9 →	11111011
0x7a58fa →	00100110
0x7a58fb →	11010011
0x7a58fc →	00011101
0x7a58fd →	11000011
0x7a58fe →	01011000
0x7a58ff →	00011101
0x7a5900 →	11011000
0x7a5901 →	11101001
...	



Punteros

# Punteros

En **C** un puntero es una variable cuyo **valor** es la **dirección** de memoria de otra variable.

# &var - ¿Dónde está var?



```
int a = 5;
int b = 7;
printf("%p\n", &a);
printf("%p\n", &b);
```

```
$ gcc main.c -o main
$ ./main
0x7a85f8
0x7a85fc
```

El operador `&` nos retorna la **dirección** de memoria de una variable.

Para imprimir: `%p`

# &var - ¿Dónde está var?



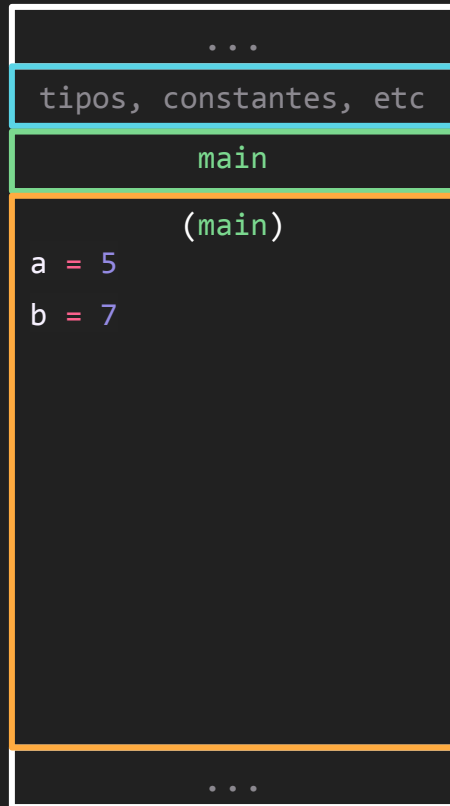
```
int a = 5;
int b = 7;
printf("%p\n", &a);
printf("%p\n", &b);
```

```
$ gcc main.c -o main
$ ./main
0x7a85f8
0x7a85fc
```

0x7a85f8 →

0x7a85fc →

RAM



# *type*\* - puntero a *type*



```
int c = 12;  
int* d = &c;  
printf("%p = %p\n", d, &c);
```

```
$ gcc main.c -o main  
$ ./main  
0x9b85f8 = 0x9b85f8
```

El símbolo *\** luego de un *tipo* se refiere a un *tipo* especial, un *puntero*.

Un *puntero* es sólo la *dirección* de memoria a donde apunta.

Tamaño: 64 bits (8 bytes)

# *type\** - puntero a *type*

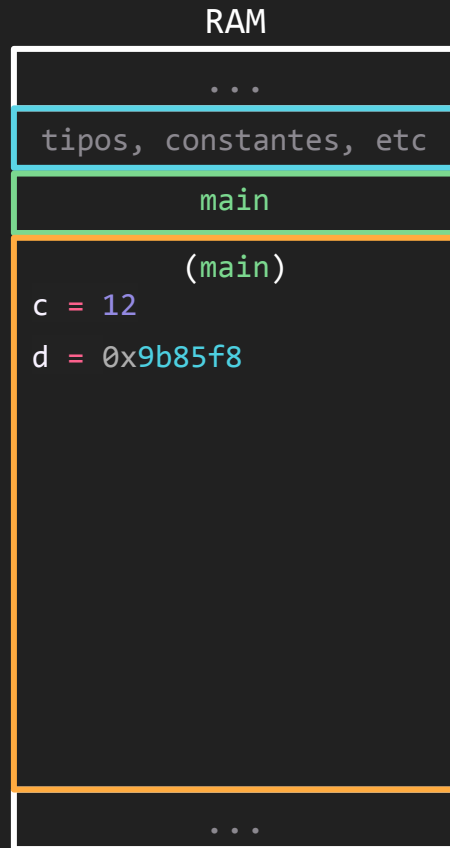


```
int c = 12;  
int* d = &c;  
printf("%p = %p\n", d, &c);
```

```
$ gcc main.c -o main  
$ ./main  
0x9b85f8 = 0x9b85f8
```

0x9b85f8 →

0x9b85fc →



`*ptr` - acceder a dirección ptr



```
int e = 3;
int* f = &e;
*f += 1;
printf("%d\n", e);
```

?

El operador `*` nos permite acceder a la variable guardada en una dirección de memoria.

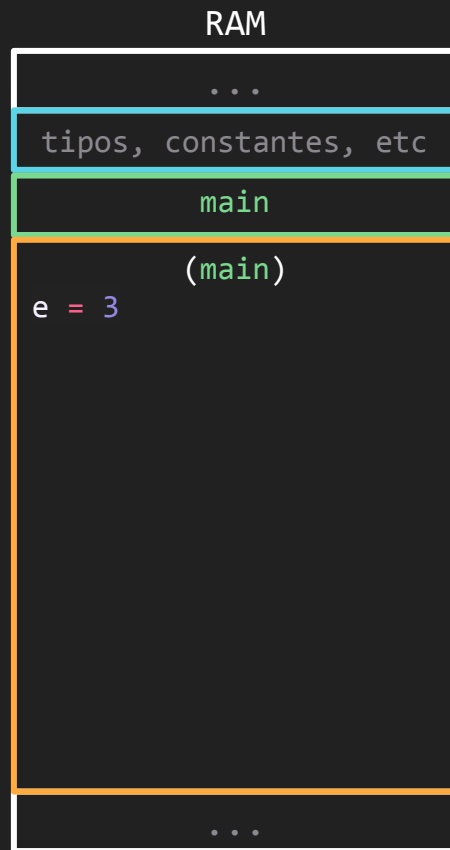
*\*ptr* - acceder a dirección ptr



```
int e = 3;  
int* f = &e;  
*f += 1;  
printf("%d\n", e);
```

```
$ gcc main.c -o main  
$ ./main
```

0x7b75f8 →





*\*ptr* - acceder a dirección ptr

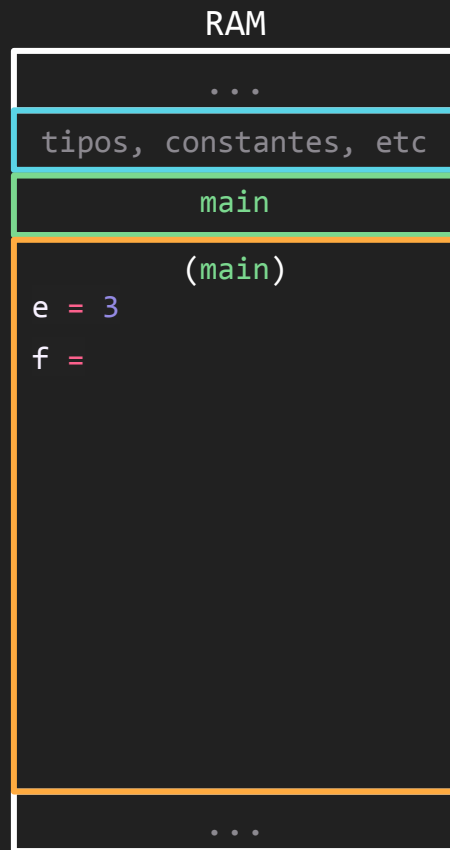


```
int e = 3;
int* f = &e;
*f += 1;
printf("%d\n", e);
```

```
$ gcc main.c -o main
$ ./main
```

0x7b75f8 →

0x7b75fc →



*\*ptr* - acceder a dirección ptr



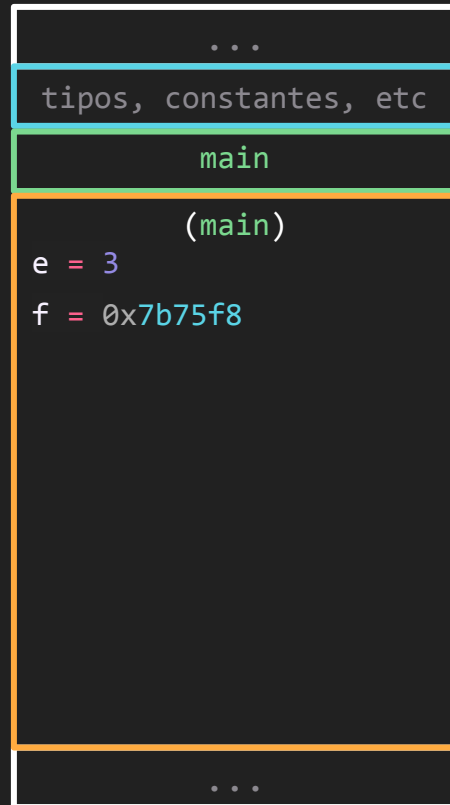
```
int e = 3;  
int* f = &e;  
*f += 1;  
printf("%d\n", e);
```

```
$ gcc main.c -o main  
$ ./main
```

0x7b75f8 →

0x7b75fc →

RAM



**\*ptr** - acceder a dirección ptr



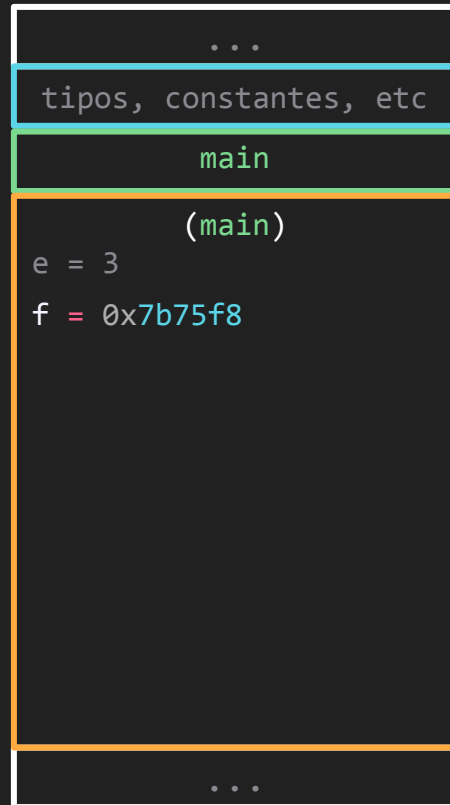
```
int e = 3;
int* f = &e;
*f += 1;
printf("%d\n", e);
```

```
$ gcc main.c -o main
$ ./main
```

0x7b75f8 →

0x7b75fc →

RAM



**\*ptr** - acceder a dirección ptr



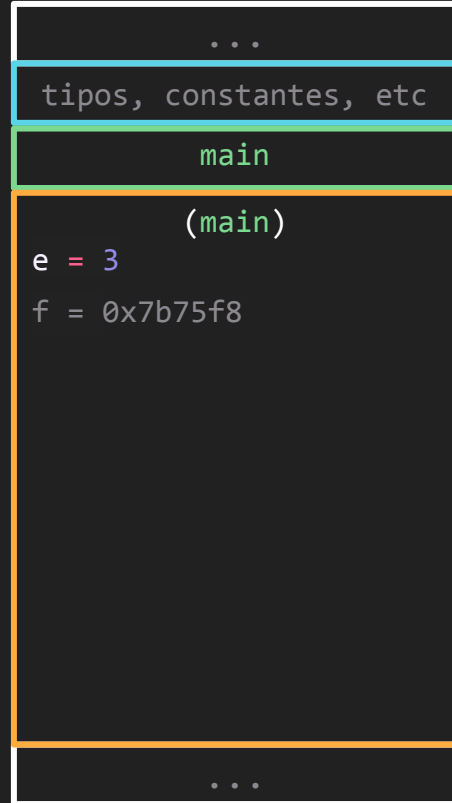
```
int e = 3;
int* f = &e;
*f += 1;
printf("%d\n", e);
```

```
$ gcc main.c -o main
$ ./main
```

0x7b75f8 →

0x7b75fc →

RAM



*\*ptr* - acceder a dirección ptr



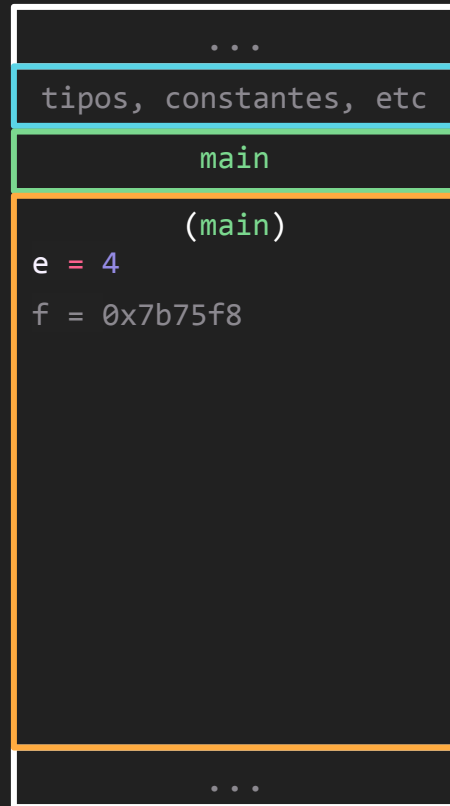
```
int e = 3;
int* f = &e;
*f += 1;
printf("%d\n", e);
```

```
$ gcc main.c -o main
$ ./main
```

0x7b75f8 →

0x7b75fc →

RAM



`*ptr` - acceder a dirección `ptr`



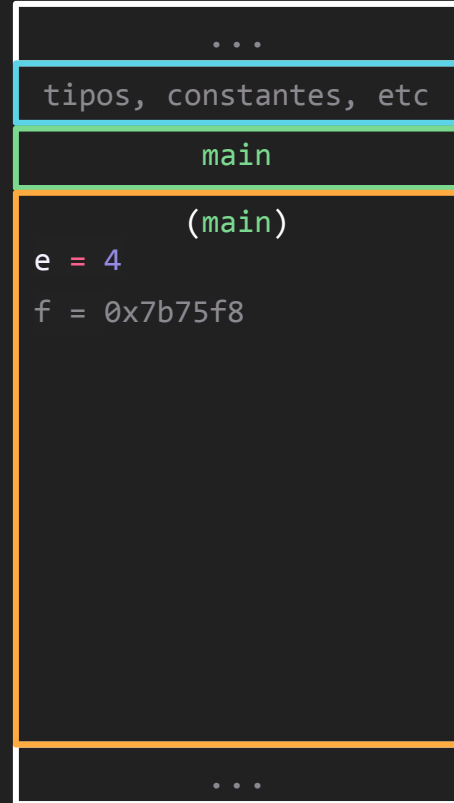
```
int e = 3;
int* f = &e;
*f += 1;
printf("%d\n", e);
```

```
$ gcc main.c -o main
$ ./main
4
```

0x7b75f8 →

0x7b75fc →

RAM



# Punteros de punteros



```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

?

Ya que son **variables**, podemos definir punteros a punteros.

El *tipo* de estos simplemente tiene un **\*** adicional.

Para **acceder** a ellos también se usa un **\*** adicional.

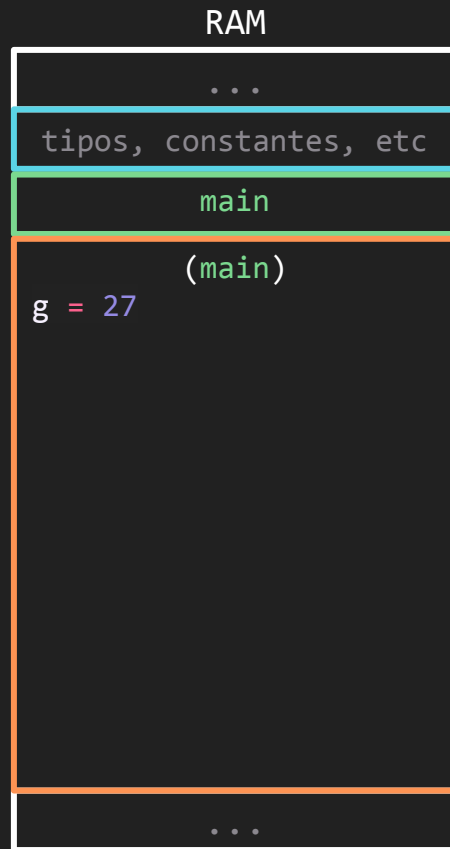
# Punteros de punteros



```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

```
$ gcc main.c -o main  
$ ./main
```

0x7c35f8 →





# Punteros de punteros



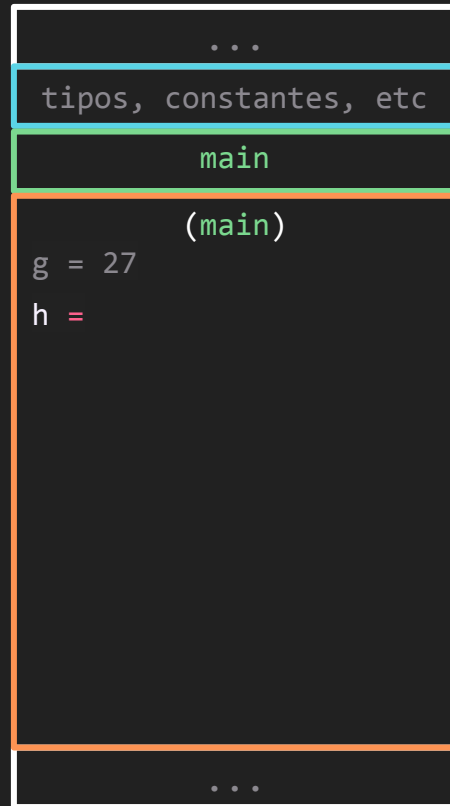
```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

```
$ gcc main.c -o main  
$ ./main
```

0x7c35f8 →

0x7c35fc →

RAM



# Punteros de punteros

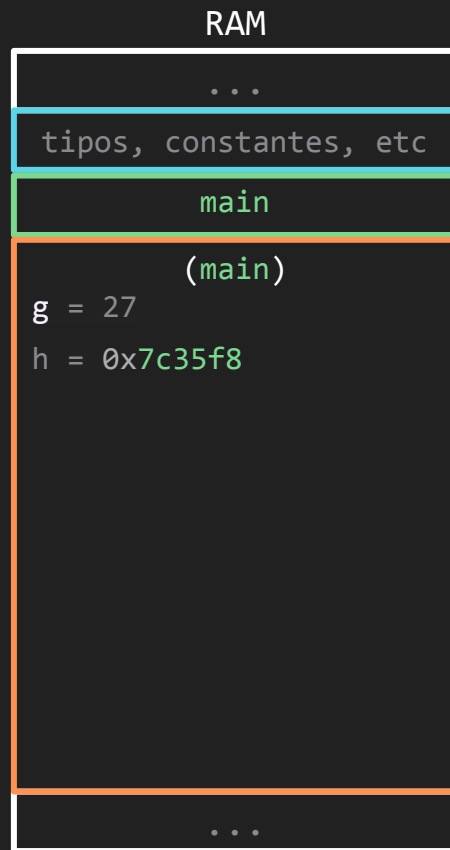


```
int g = 27;
int* h = &g;
int** i = &h;
printf("%d\n", g);
printf("%d\n", *h);
printf("%d\n", **i);
```

```
$ gcc main.c -o main
$ ./main
```

0x7c35f8 →

0x7c35fc →



# Punteros de punteros



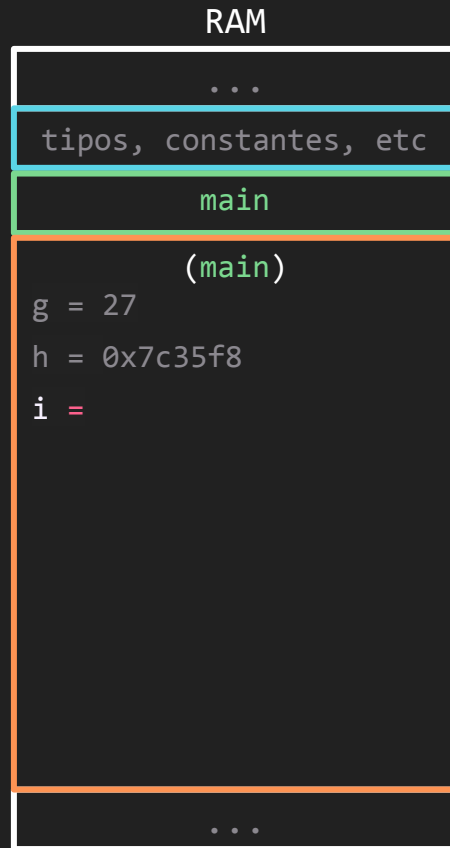
```
int g = 27;
int* h = &g;
int** i = &h;
printf("%d\n", g);
printf("%d\n", *h);
printf("%d\n", **i);
```

```
$ gcc main.c -o main
$ ./main
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



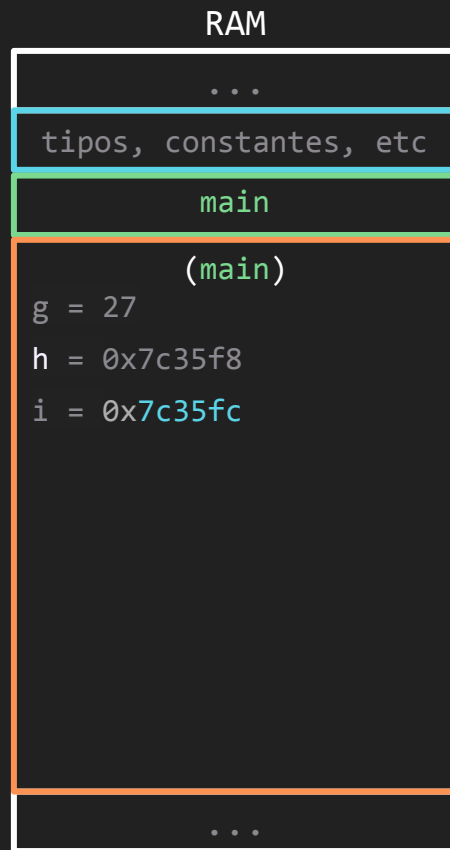
```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

```
$ gcc main.c -o main  
$ ./main
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



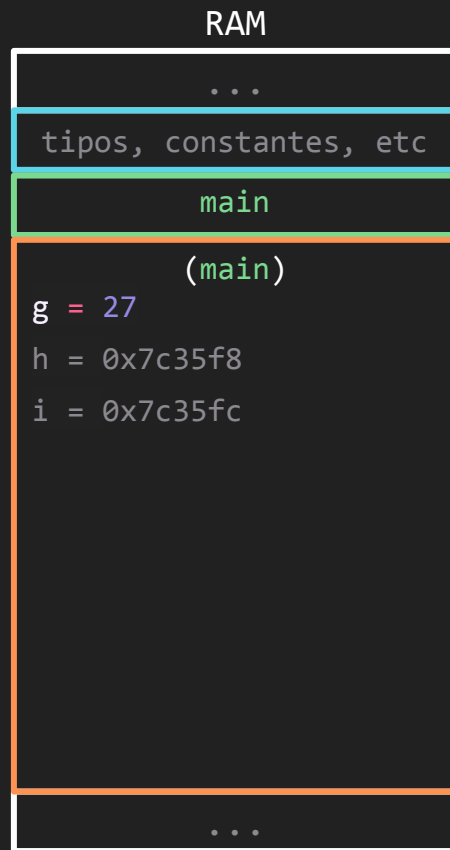
```
int g = 27;
int* h = &g;
int** i = &h;
printf("%d\n", g);
printf("%d\n", *h);
printf("%d\n", **i);
```

```
$ gcc main.c -o main
$ ./main
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



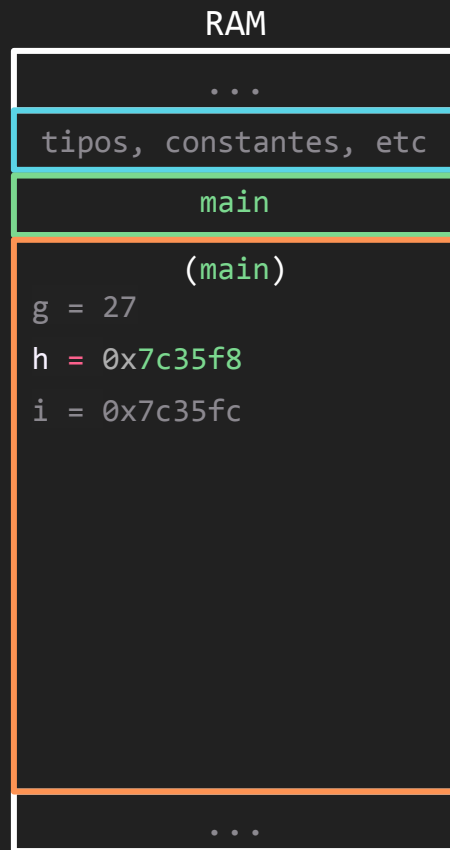
```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

```
$ gcc main.c -o main  
$ ./main  
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



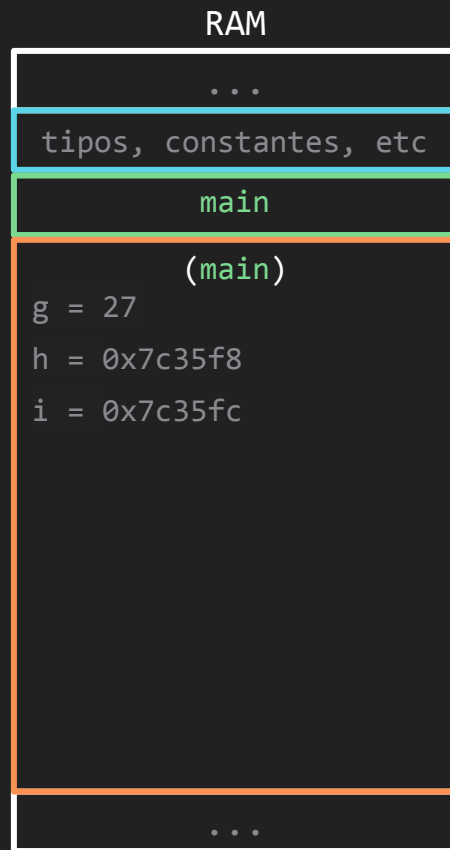
```
int g = 27;
int* h = &g;
int** i = &h;
printf("%d\n", g);
printf("%d\n", *h);
printf("%d\n", **i);
```

```
$ gcc main.c -o main
$ ./main
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



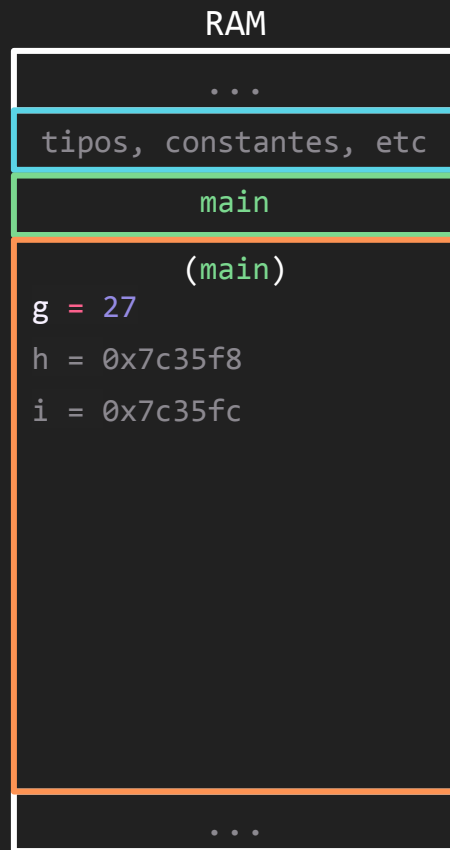
```
int g = 27;
int* h = &g;
int** i = &h;
printf("%d\n", g);
printf("%d\n", *h);
printf("%d\n", **i);
```

```
$ gcc main.c -o main
$ ./main
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →





# Punteros de punteros



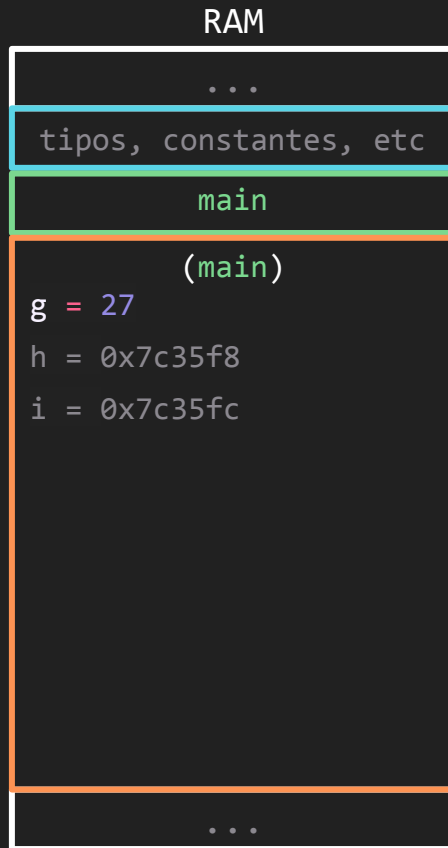
```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

```
$ gcc main.c -o main  
$ ./main  
27  
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



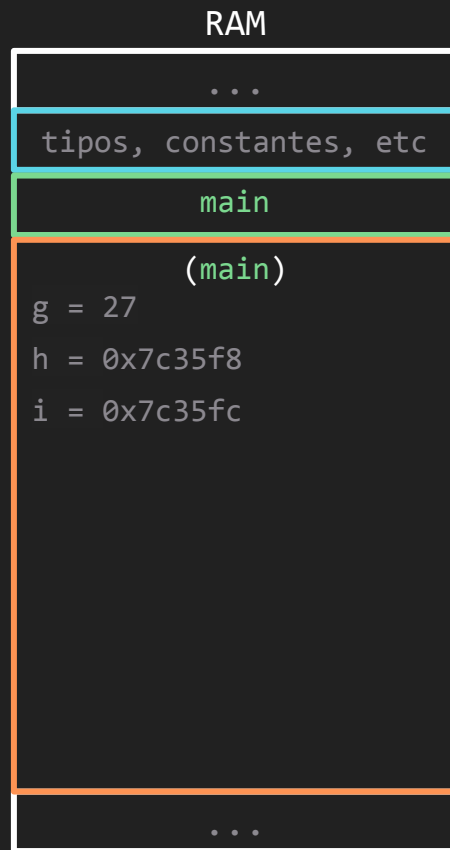
```
int g = 27;
int* h = &g;
int** i = &h;
printf("%d\n", g);
printf("%d\n", *h);
printf("%d\n", **i);
```

```
$ gcc main.c -o main
$ ./main
27
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



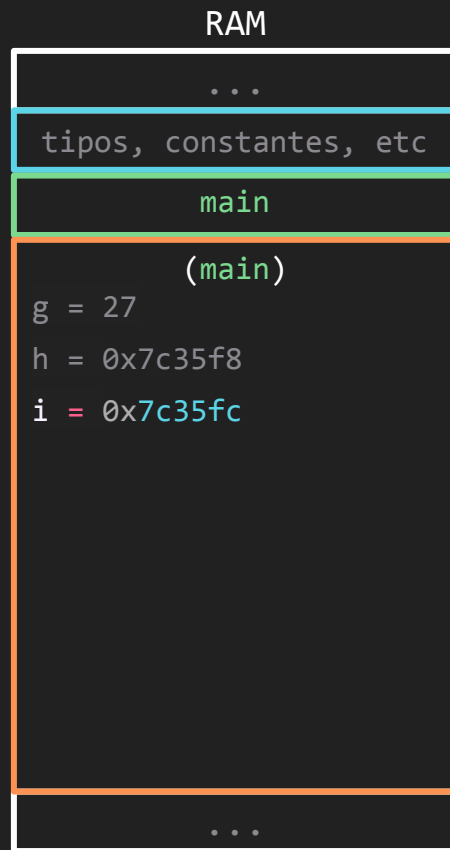
```
int g = 27;
int* h = &g;
int** i = &h;
printf("%d\n", g);
printf("%d\n", *h);
printf("%d\n", **i);
```

```
$ gcc main.c -o main
$ ./main
27
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



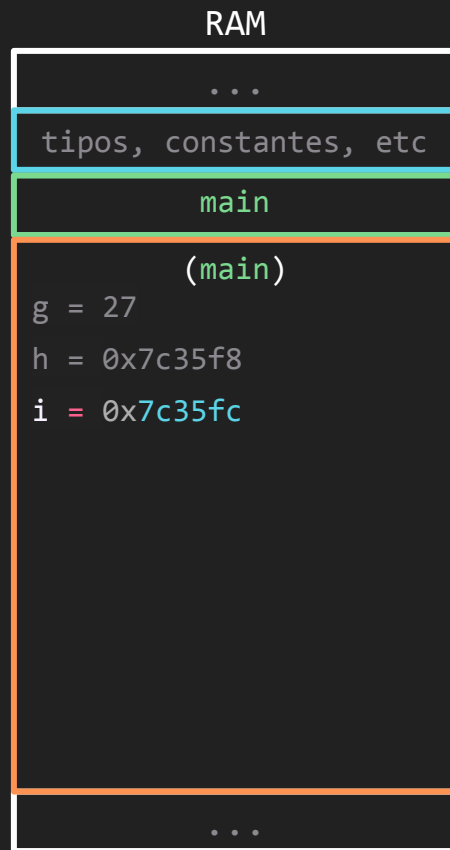
```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

```
$ gcc main.c -o main  
$ ./main  
27  
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

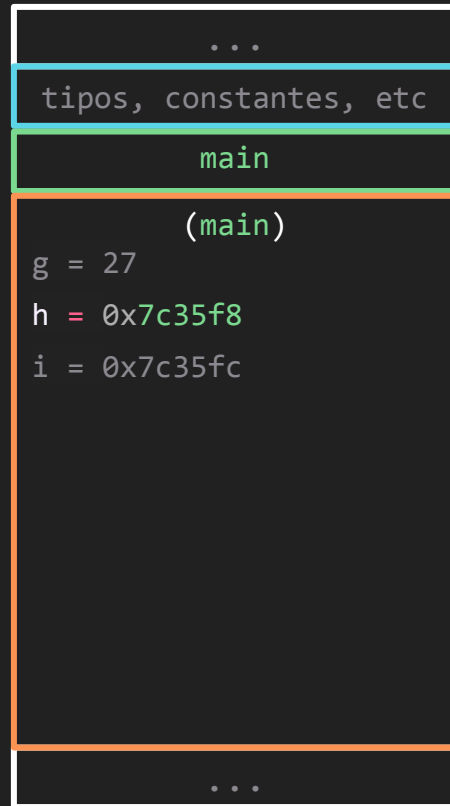
```
$ gcc main.c -o main  
$ ./main  
27  
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →

RAM



# Punteros de punteros



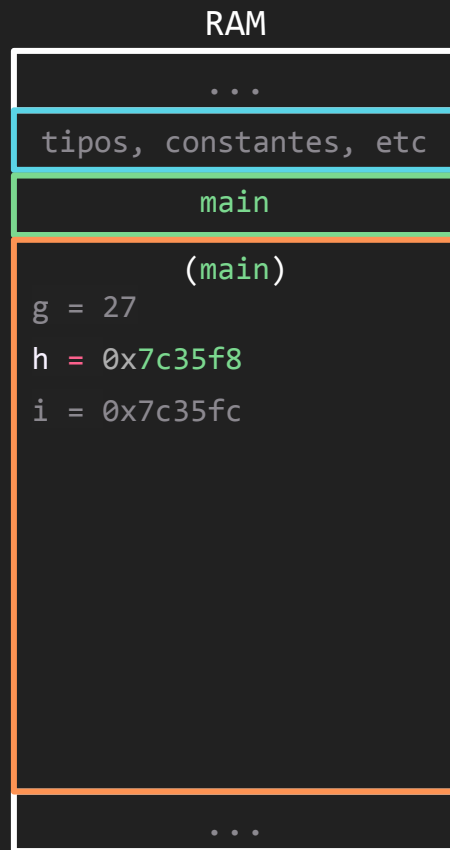
```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

```
$ gcc main.c -o main  
$ ./main  
27  
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



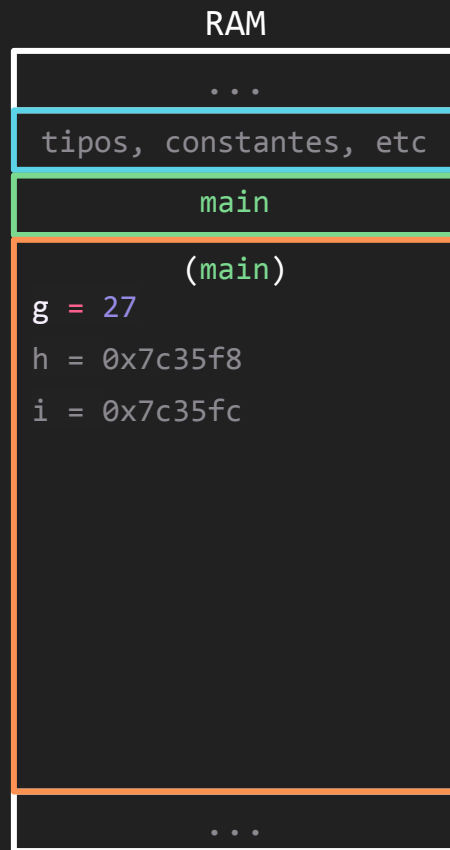
```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

```
$ gcc main.c -o main  
$ ./main  
27  
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# Punteros de punteros



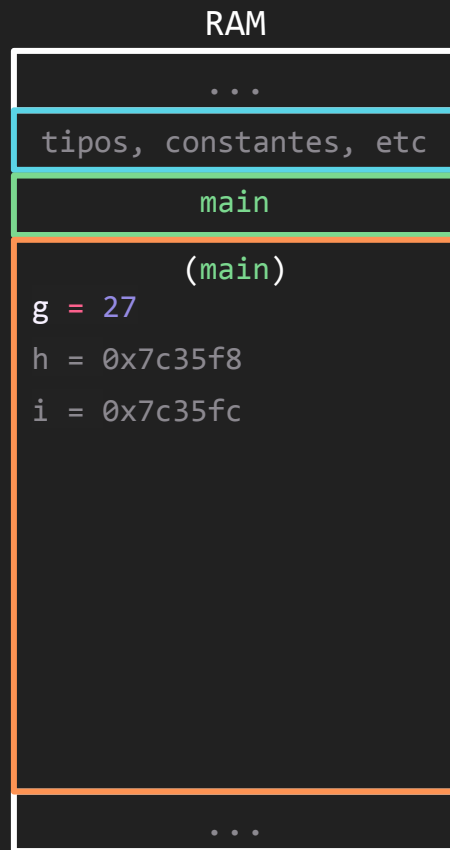
```
int g = 27;
int* h = &g;
int** i = &h;
printf("%d\n", g);
printf("%d\n", *h);
printf("%d\n", **i);
```

```
$ gcc main.c -o main
$ ./main
27
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →





# Punteros de punteros



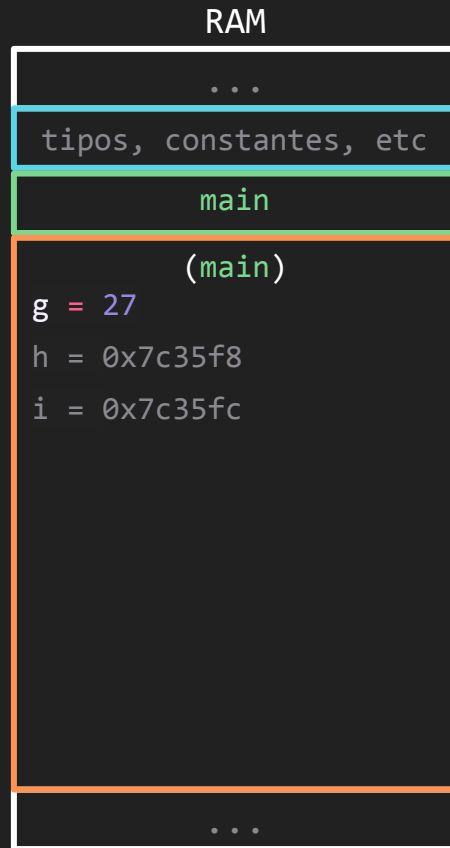
```
int g = 27;  
int* h = &g;  
int** i = &h;  
printf("%d\n", g);  
printf("%d\n", *h);  
printf("%d\n", **i);
```

```
$ gcc main.c -o main  
$ ./main  
27  
27  
27
```

0x7c35f8 →

0x7c35fc →

0x7c3604 →



# NULL - Puntero vacío



```
int* a = NULL;
if (!a)
{
    printf("a no tiene un valor asignado\n");
}
```

```
$ gcc main.c -o main
$ ./main
a no tiene un valor asignado
```

NULL es una **dirección** de memoria especial, no apunta a ningún lado.

Es lógicamente igual a 0 (**false**).

NULL - ☢️ **WARNING! ACHTUNG! PELIGRO!** ☢️



```
int* d = NULL;  
  
printf("%i\n", *d);
```

```
$ gcc main.c -o main  
$ ./main  
Segmentation fault (core dumped)
```

La **dirección** NULL está protegida; acceder a ella causará un error en tu programa.

Aquí hay un [ejemplo](#).

# Propiedades de punteros

# ¿Por qué punteros?

En **C** todos los parámetros de **funciones** se pasan por **valor**.

Para que una **función** pueda modificar algo, necesitamos **referencias**.

# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

En C no existe el paso por referencia, pero usando punteros podemos simularlo y usar funciones para modificar objetos.

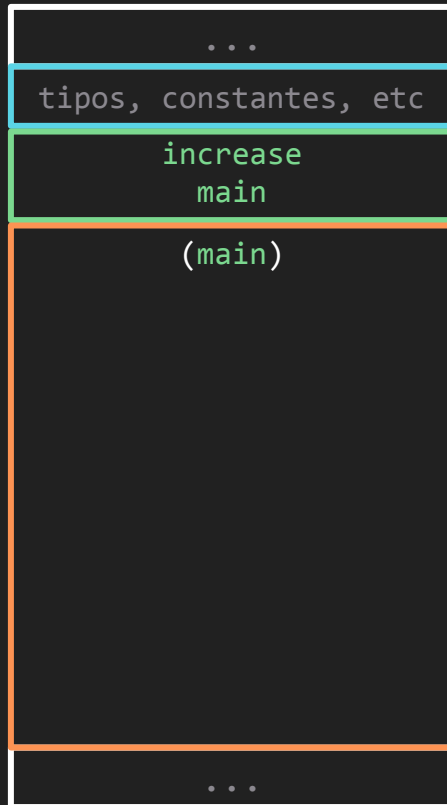
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

RAM



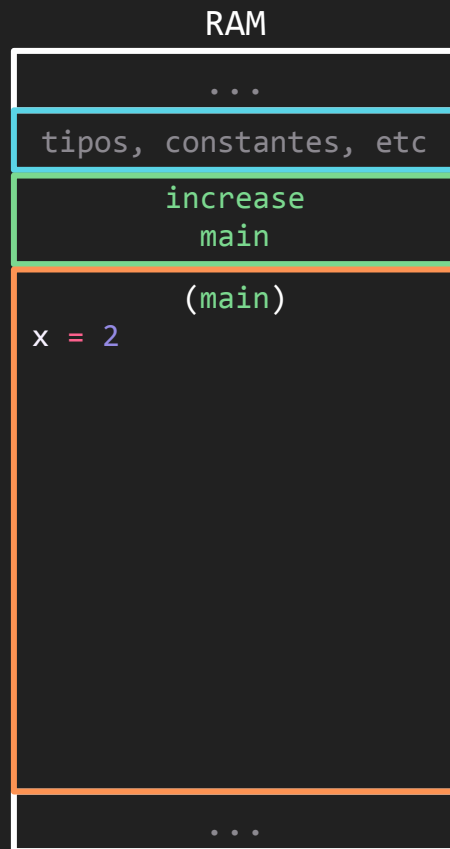
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

0x7c45f8 →





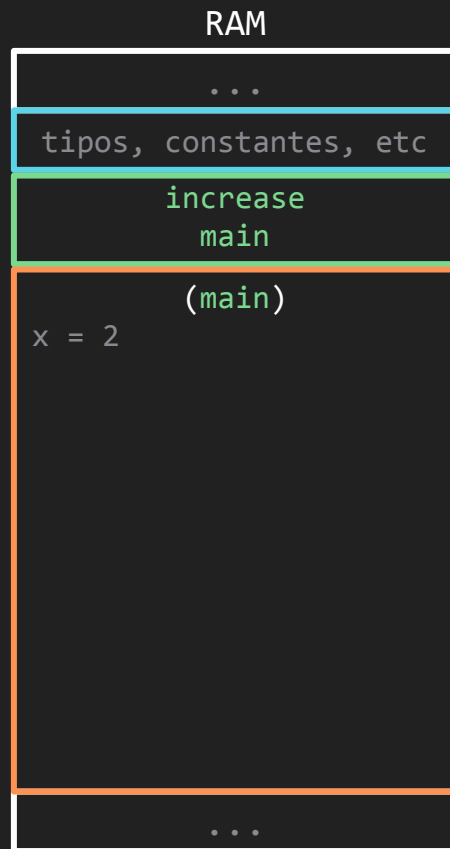
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

0x7c45f8 →



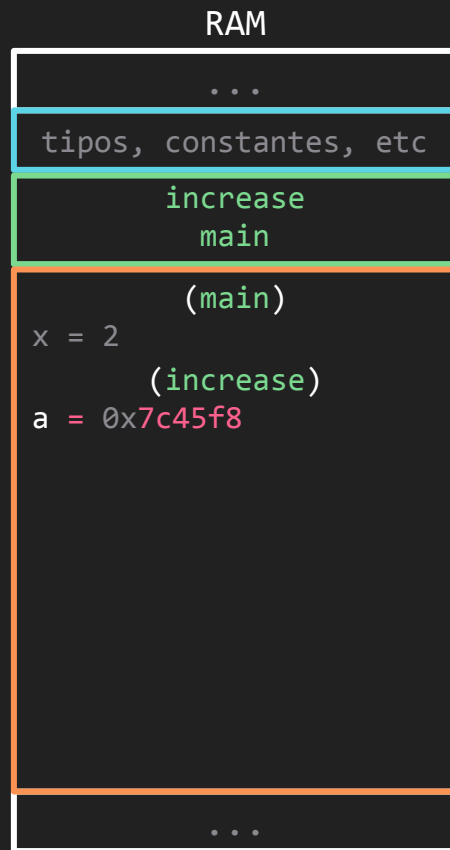
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

0x7c45f8 →



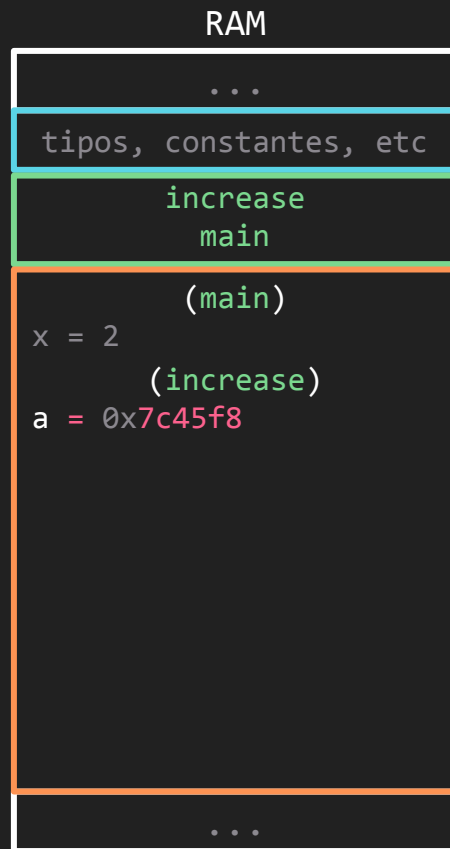
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

0x7c45f8 →



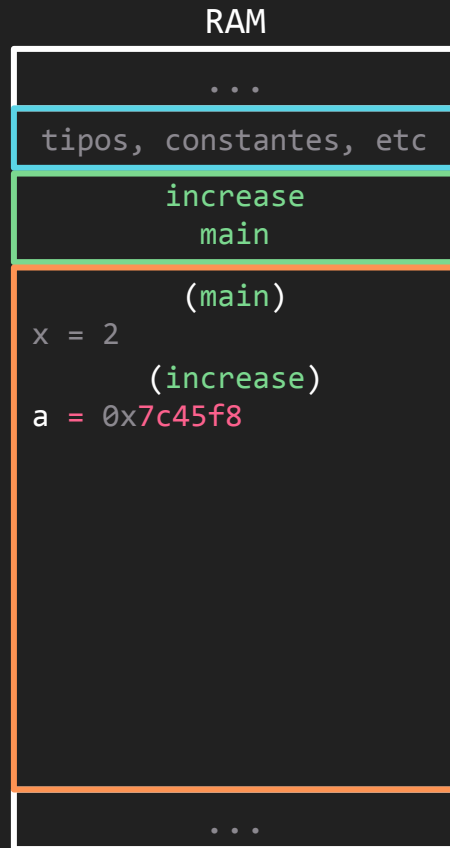
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

0x7c45f8 →



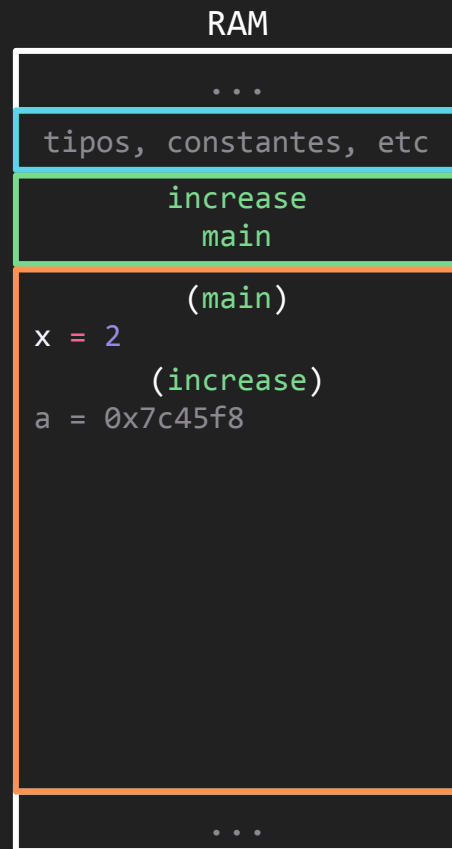
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

0x7c45f8 →



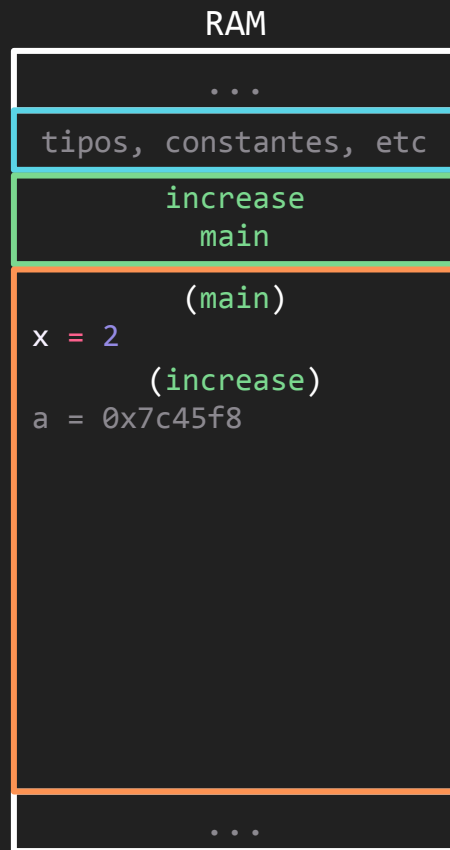
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

0x7c45f8 →



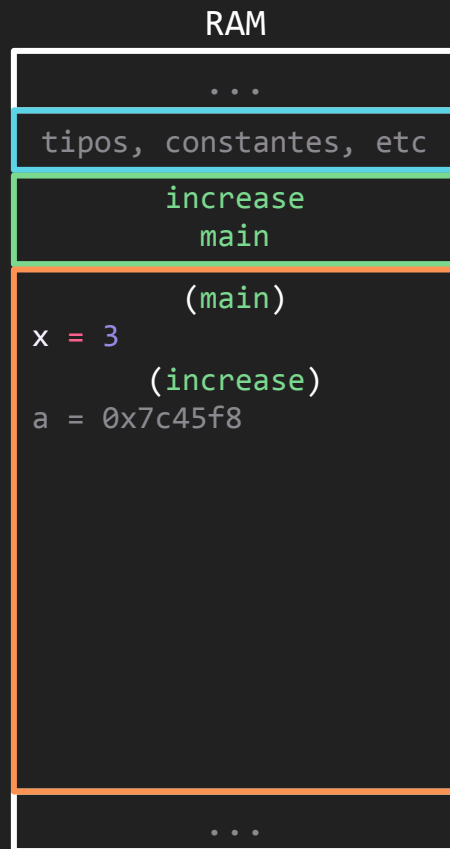
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

0x7c45f8 →



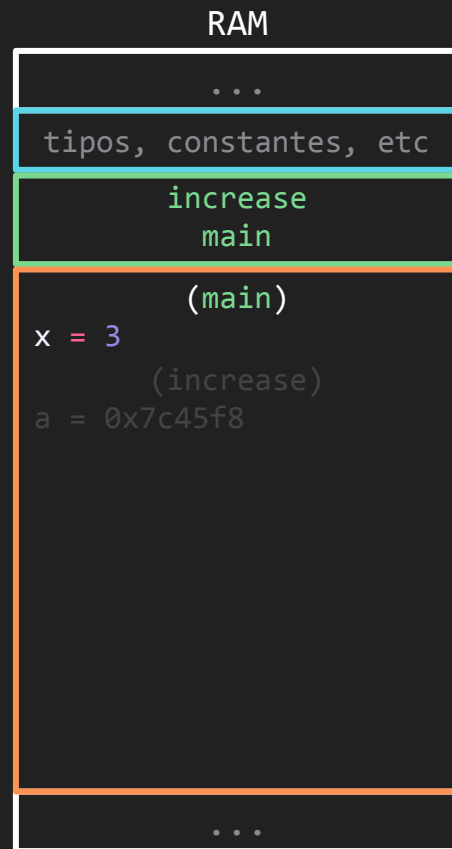
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x); // Imprime 3
    return 0;
}
```

0x7c45f8 →





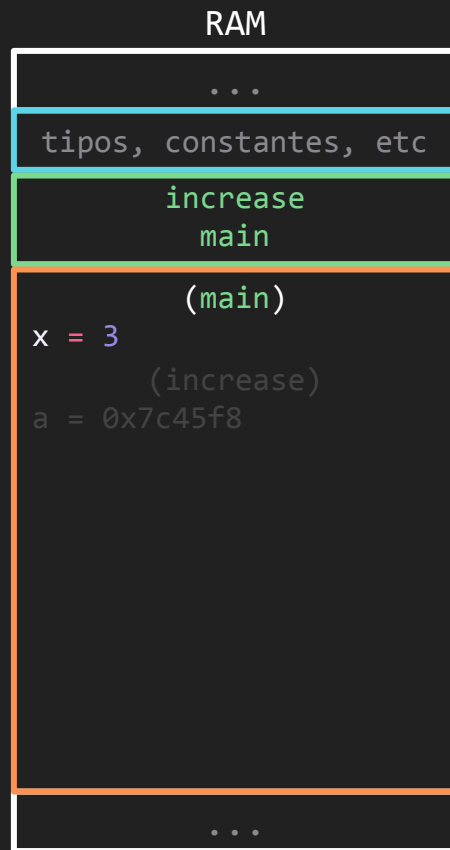
# Paso por "referencia"



```
void increase(int* a)
{
    *a += 1;
}

int main()
{
    int x = 2;
    increase(&x);
    printf("%i\n", x);
    return 0;
}
```

0x7c45f8 →



"Alquimia" de punteros



# Aritmética de punteros



# Aritmética de punteros



```
int j = 7;  
int* k = &j;  
printf("%p\n", k);  
printf("%p\n", k + 1);
```

?

0x7c35f4 →

0x7c35f8 →

RAM

...

tipos, constantes, etc

main

(main)

j = 7

k = 0x7c35f4

...

# Aritmética de punteros



```
int j = 7;  
int* k = &j;  
printf("%p\n", k);  
printf("%p\n", k + 1);
```

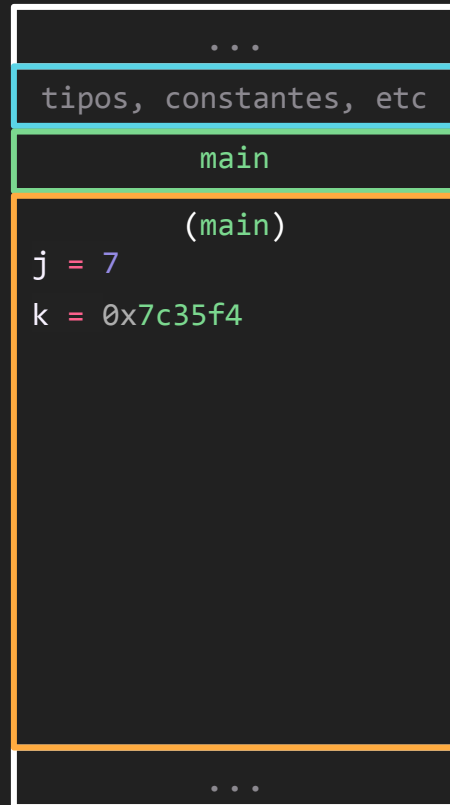
```
$ gcc main.c -o main  
$ ./main  
0x7c35f4  
0x7c35f8
```



0x7c35f4 →

0x7c35f8 →

RAM



# Aritmética de punteros

*type*\* A



A + i =

# Aritmética de punteros

*type*\* A



A + i = A + i \* sizeof(*type*)

# Aritmética de punteros



```
int j = 7;  
int* k = &j;  
printf("%p\n", k);  
printf("%p\n", k + 1);
```

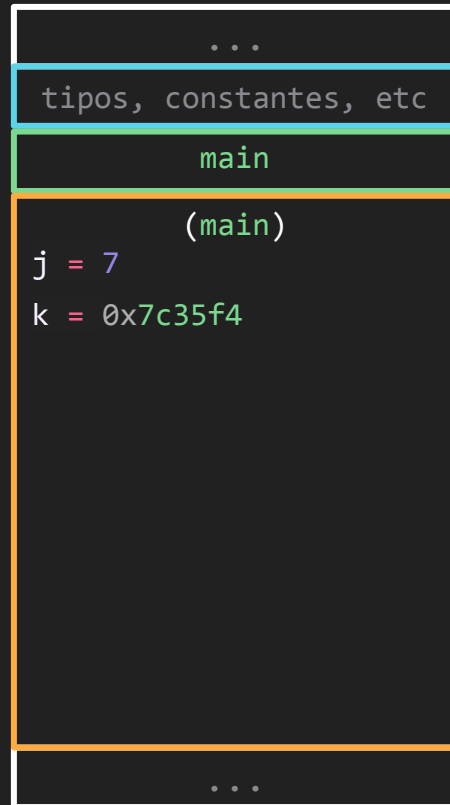
```
$ gcc main.c -o main  
$ ./main  
0x7c35f4  
0x7c35f8
```



0x7c35f4 →

0x7c35f8 →

RAM





# Aritmética de punteros



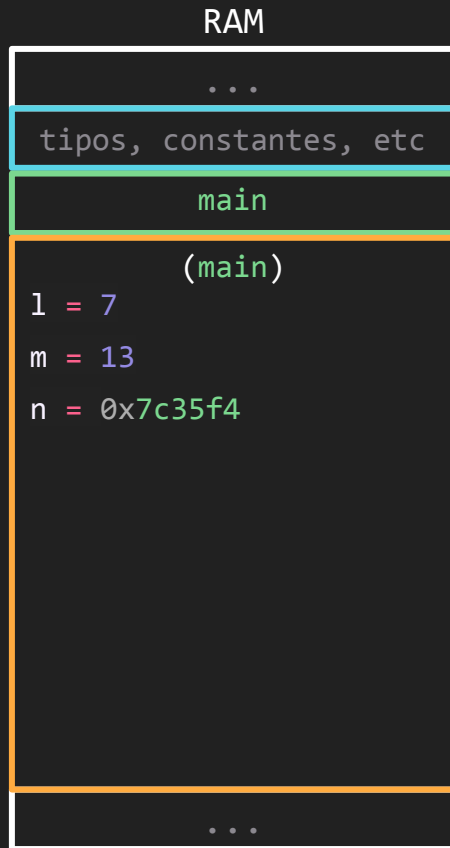
```
int l = 7;  
int m = 13;  
int* n = &l;  
printf("%p\n", n);  
printf("%p\n", n + 1);
```

```
$ gcc main.c -o main  
$ ./main  
0x7c35f4  
0x7c35f8
```

0x7c35f4 →

0x7c35f8 →

0x7c35fc →



# Aritmética de punteros



```
int l = 7;  
int m = 13;  
int* n = &l;  
printf("%d\n", *n);  
printf("%d\n", *(n + 1));
```

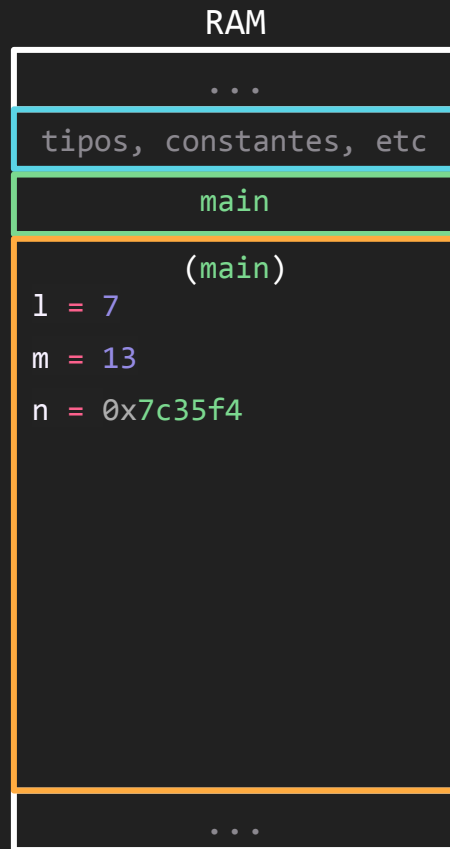
```
$ gcc main.c -o main  
$ ./main  
7  
13
```



0x7c35f4 →

0x7c35f8 →

0x7c35fc →



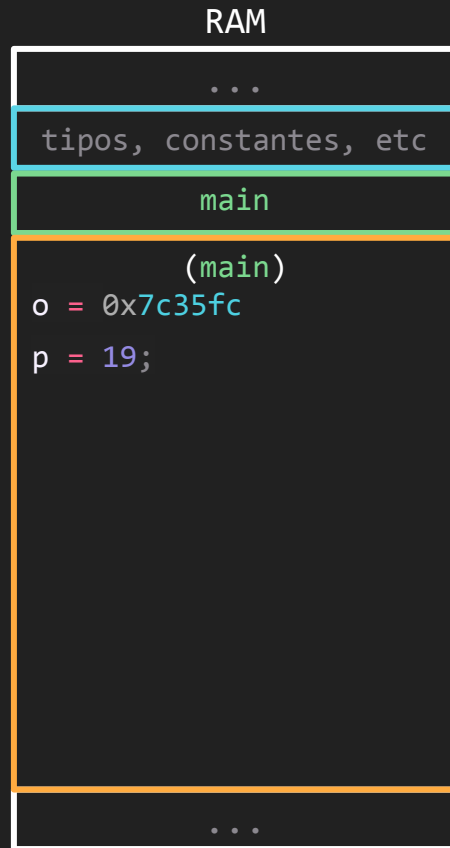
# Aritmética de punteros



```
int* o;  
int p = 19;  
o = &p;  
printf("%p\n", o);  
printf("%p\n", o + 1);
```

```
$ gcc main.c -o main  
$ ./main  
0x7c35fc  
0x7c3600
```

0x7c35f4 →  
0x7c35fc →  
0x7c3600 →



# Aritmética de punteros

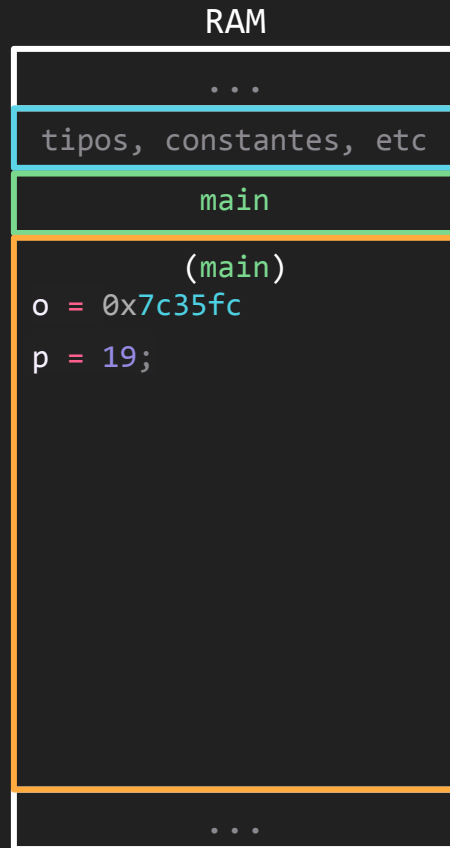


```
int* o;  
int p = 19;  
o = &p;  
printf("%d\n", *o);  
printf("%d\n", *(o + 1));
```

```
$ gcc main.c -o main  
$ ./main  
19  
-409884036
```



0x7c35f4 →  
0x7c35fc →  
0x7c3600 →



# ¡Muchas Gracias!



With ♥ by @vichoeq & @KnowYourselves