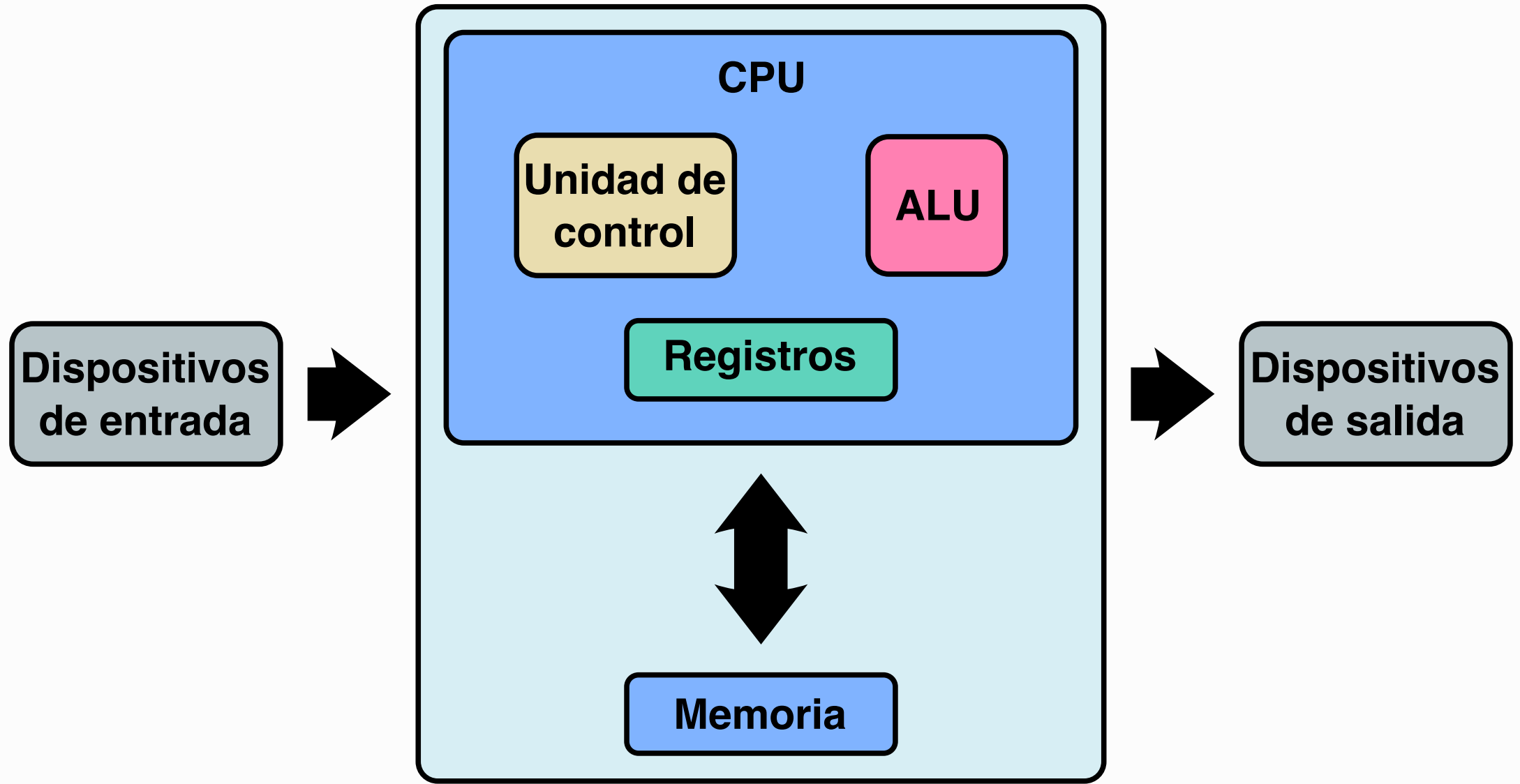


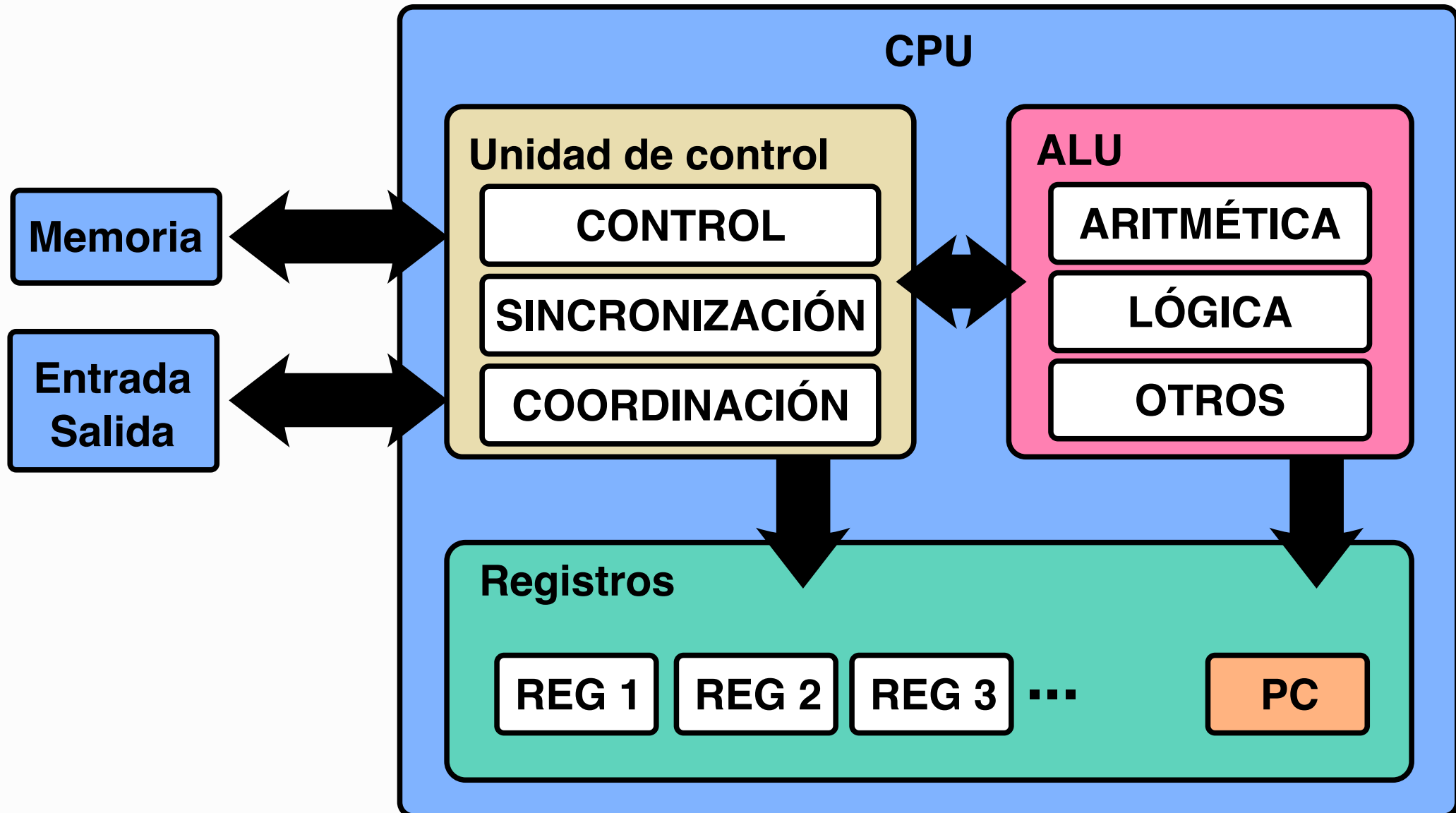
COMPILACIÓN, MEMORIA DINÁMICA (Y OTRAS CUESTIONES)

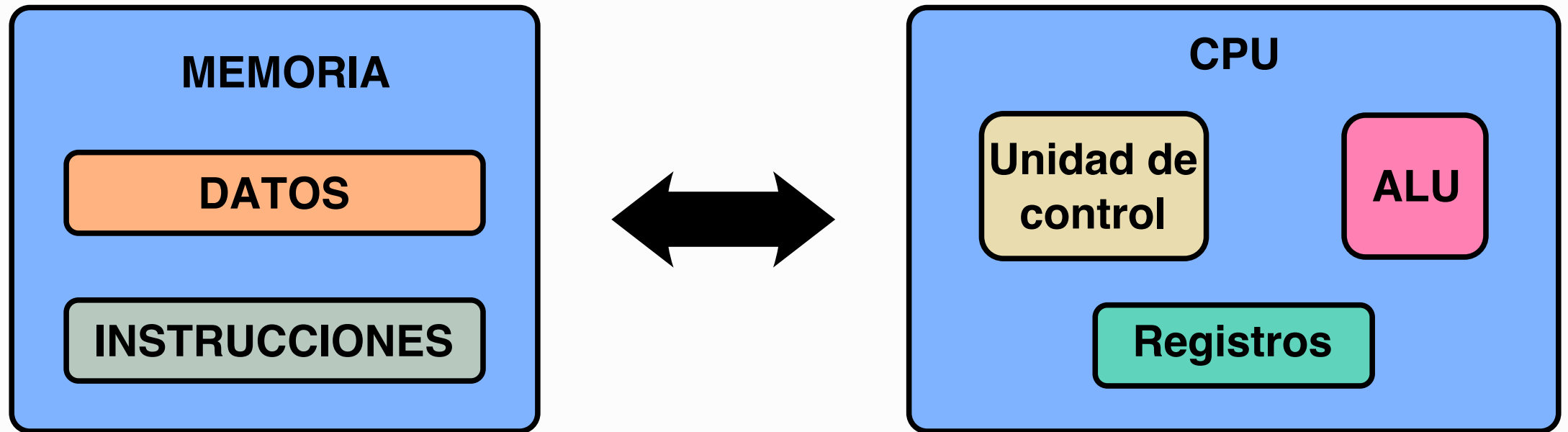
ALGORITMOS Y PROGRAMACIÓN II

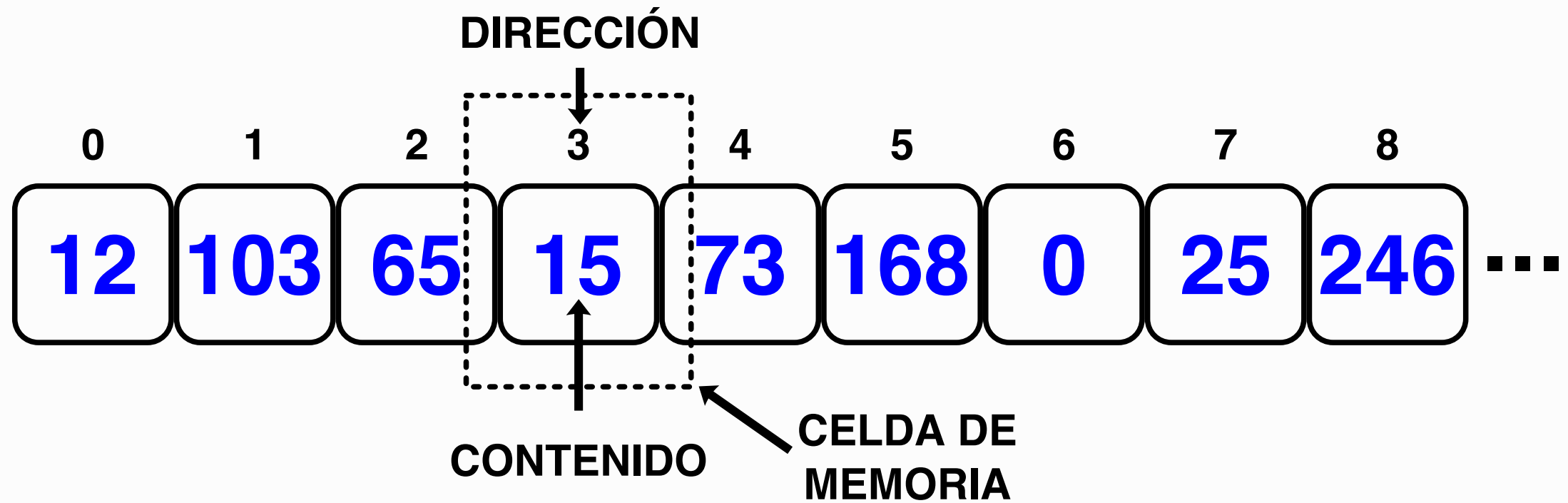


REPASO: PROCESADOR Y MEMORIA









DIRECCIÓN INICIAL

CONTENIDO

REPRESENTACIÓN ASCII

00000400	4a	74	21	e6	80	e4	64	3c	ff	75	06	66	49	75	06	eb	Jt!...d<.u.fIu..
00000410	13	a8	01	74	06	e6	80	e4	60	eb	e4	a8	02	75	e0	66	...t....`....u.f
00000420	31	c0	66	c3	66	83	c8	ff	66	c3	66	53	66	31	d2	8e	1.f.f...f.fSf1..
00000430	e2	66	83	ca	ff	8e	ea	64	66	8b	0e	00	02	66	89	ca	.f.....df....f..
00000440	67	66	8d	1c	08	66	39	da	74	16	66	42	64	66	89	16	gf...f9.t.fBdf..
00000450	00	02	e6	80	65	66	a1	10	02	66	31	d0	74	e7	eb	03ef...f1.t...
00000460	66	31	c0	64	66	89	0e	00	02	66	5b	66	c3	66	56	66	f1.df....f[f.fVf
00000470	53	66	83	ec	2c	66	bb	00	01	00	00	66	4b	0f	84	c2	Sf..,f.....fK...

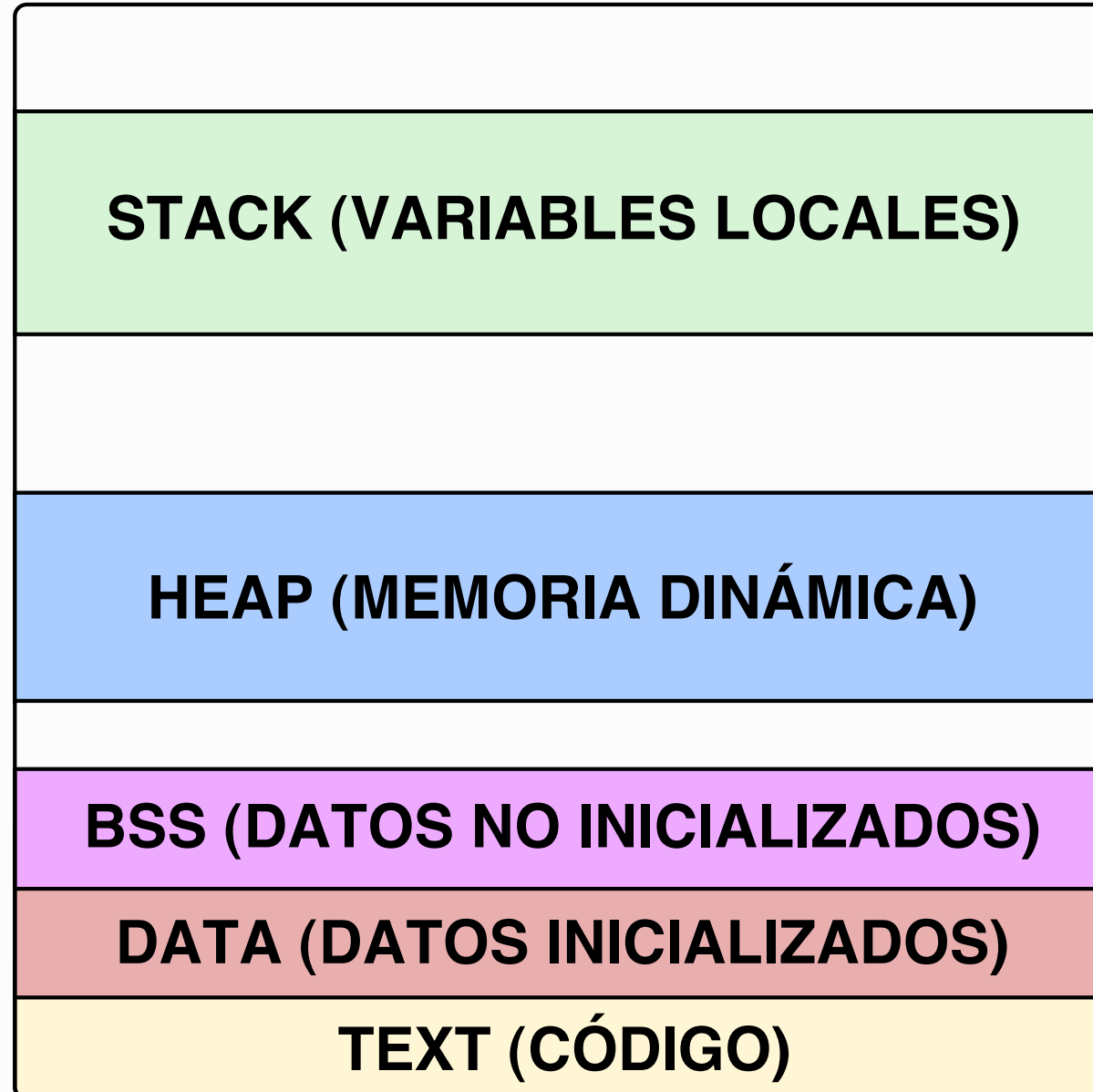
DIRECCIÓN: 470
CONTENIDO: 53
ASCII: S

DIRECCIÓN: 462
CONTENIDO: C0
ASCII: .

DIRECCIÓN: 454
CONTENIDO: 65
ASCII: e

DIRECCIÓN: 469
CONTENIDO: 66
ASCII: f

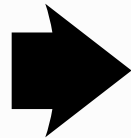
DIRECCIÓN: 47F
CONTENIDO: C2
ASCII: .



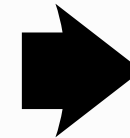
COMPILACIÓN



**ARCHIVO
ORIGINAL**



COMPILADOR



**ARCHIVO
COMPILADO**



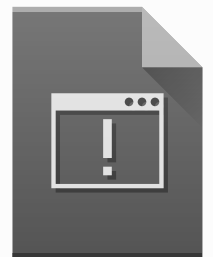
ARCHIVO
ORIGINAL

COMPILADOR



ARCHIVO
ASSEMBLY

ENSAMBLADOR



ARCHIVO
BINARIO

```
short main(){  
    short i=5, j=7, k;  
    k = i + j;  
    return k;  
}
```

```
pushq    %rbp  
movq     %rsp, %rbp  
movw     $0x5, -0x2(%rbp)  
movw     $0x7, -0x4(%rbp)  
movzwl   -0x2(%rbp), %edx  
movzwl   -0x4(%rbp), %eax  
addl     %edx, %eax  
movw     %ax, -0x6(%rbp)  
movzwl   -0x6(%rbp), %eax  
popq     %rbp  
retq
```

```
55 48 89 e5 66 c7  
45 fe 05 00 66 c7  
45 fc 07 00 0f b7  
55 fe 0f b7 45 fc  
01 d0 66 89 45 fa  
0f b7 45 fa 5d c3
```

DIRECCIÓN DE MEMORIA
DONDE ESTÁ LA INSTRUCCIÓN

CÓDIGO DE MÁQUINA
(INSTRUCCIONES)

REPRESENTACIÓN ASSEMBLY
DE LAS INSTRUCCIONES

0x401155 55

0x401156 48 89 e5

0x401159 66 c7 45 fe 05 00

0x40115f 66 c7 45 fc 07 00

0x401165 0f b7 55 fe

0x401169 0f b7 45 fc

0x40116d 01 d0

0x40116f 66 89 45 fa

0x401173 0f b7 45 fa

0x401177 5d

0x401178 c3

pushq %rbp

movq %rsp, %rbp

movw \$0x5, -0x2(%rbp)

movw \$0x7, -0x4(%rbp)

movzwl -0x2(%rbp), %edx

movzwl -0x4(%rbp), %eax

addl %edx, %eax

movw %ax, -0x6(%rbp)

movzwl -0x6(%rbp), %eax

popq %rbp

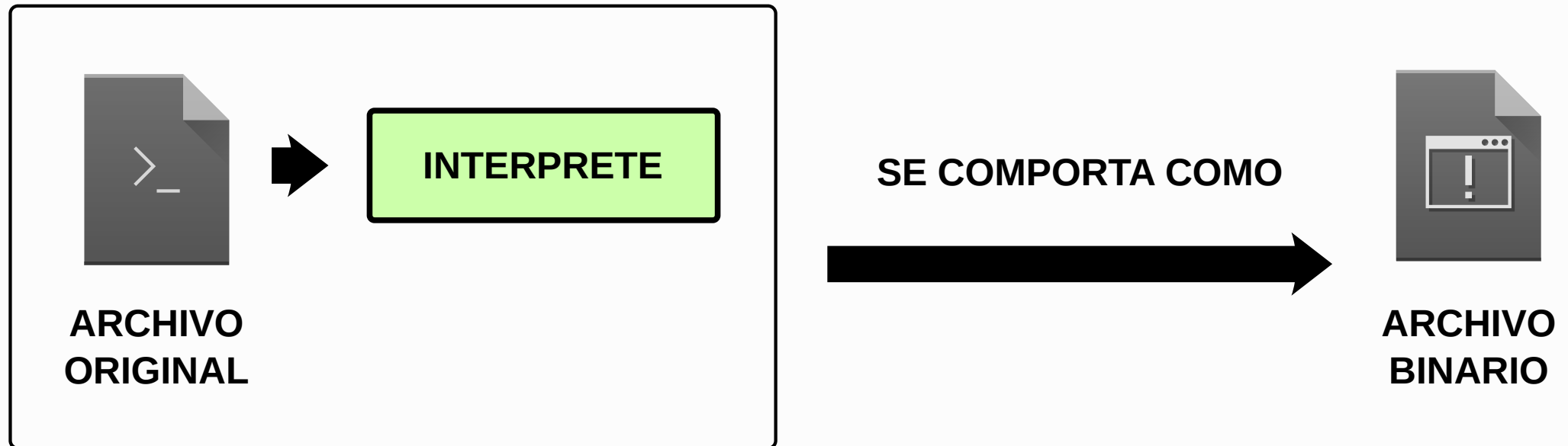
retq

OPCODE

OPERANDOS

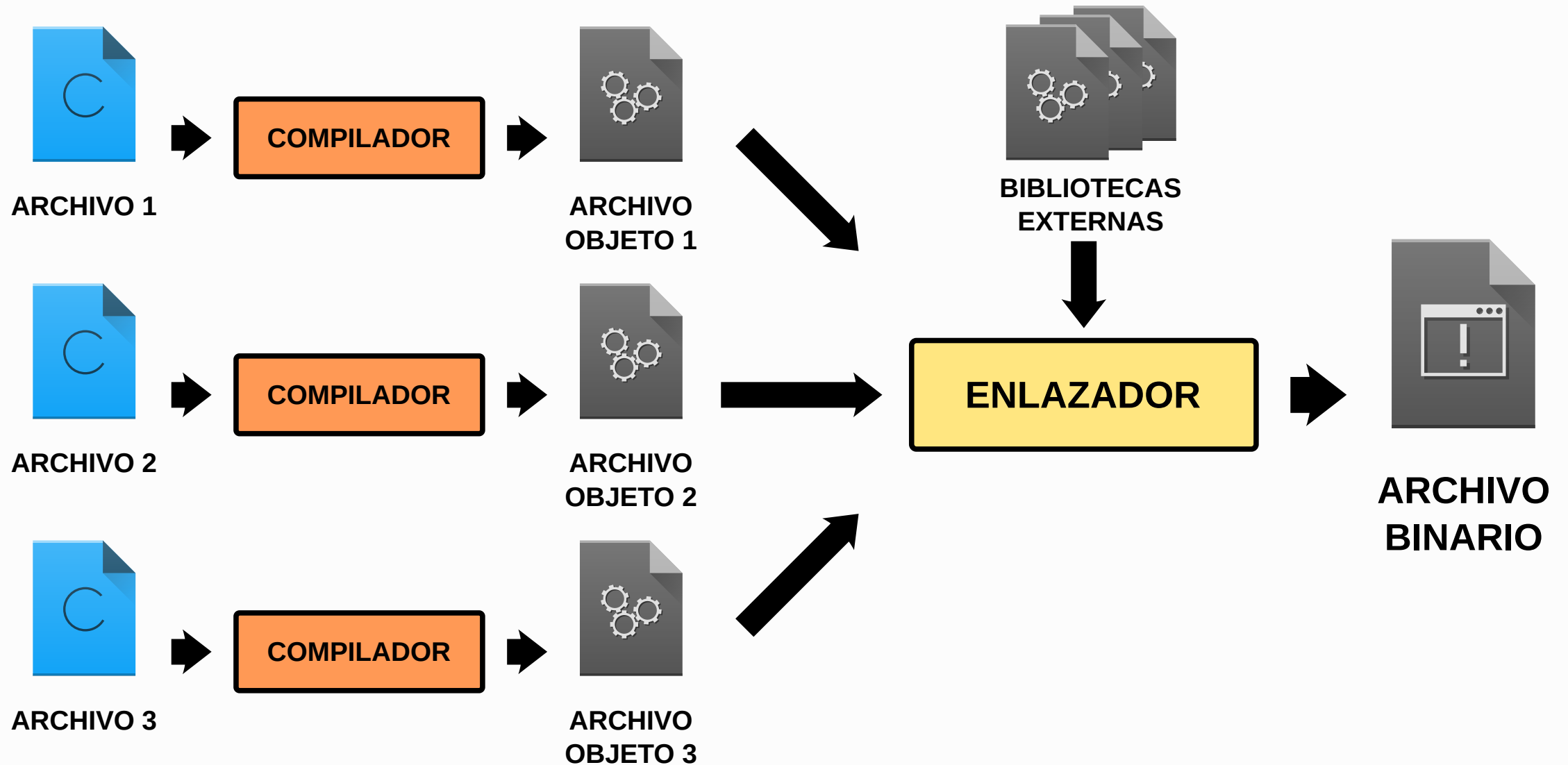
MNEMÓNICO

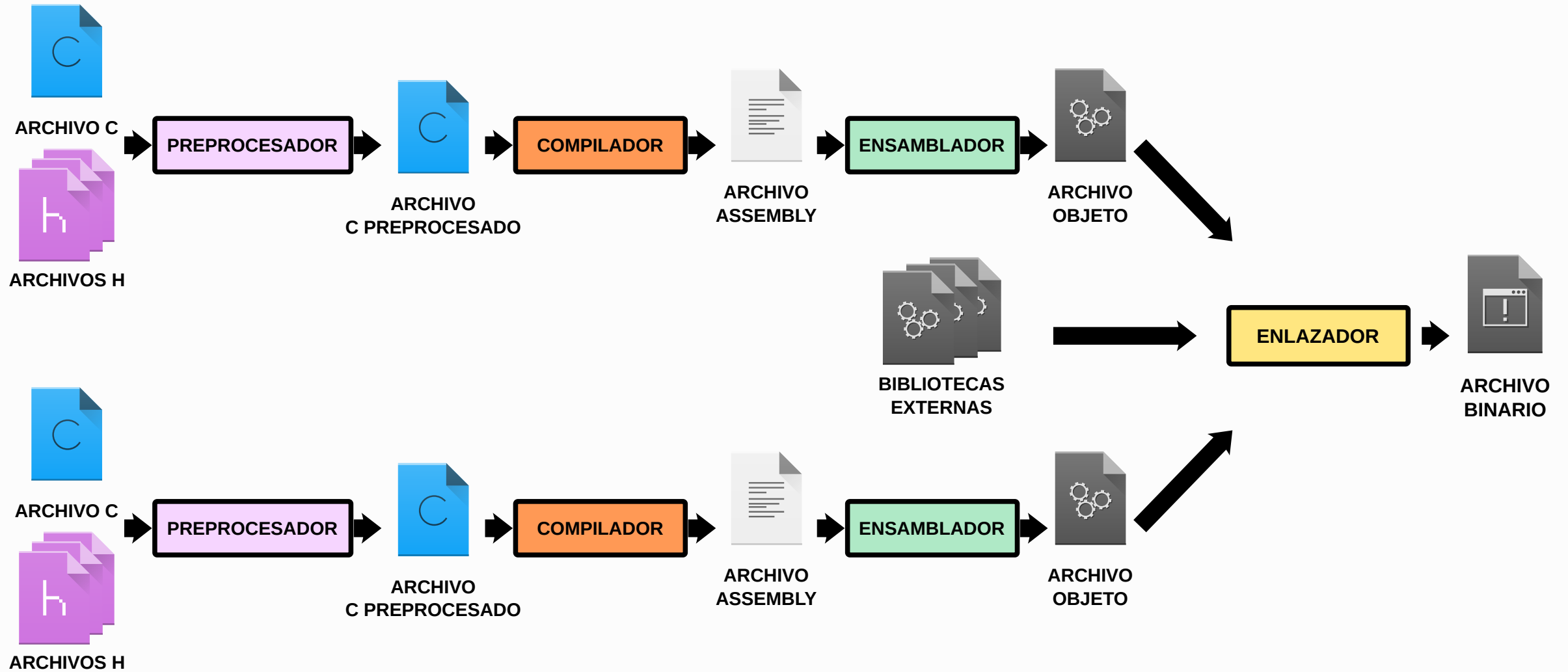
REGISTRO



LENGUAJE C: COMPILACIÓN







DEMO_PRE.C

```
#include "demo_pre1.h"
#define NUMERO (25*4/2)

int main(){

    return NUMERO;
}

#include "demo_pre2.h"
```

PREPROCESADOR



DEMO_PRE1.H

```
int i=0;

int hola(){
    return 0;
}
```

DEMO_PRE2.H

```
void chau(){
    int j=0;
    j++;
}
```

```
# 1 "demo_pre.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "demo_pre.c"
# 1 "demo_pre1.h" 1
int i=0;

int hola(){
    return 0;
}
# 2 "demo_pre.c" 2

int main(){

    return (25*4/2);
}

# 1 "demo_pre2.h" 1
void chau(){
    int j=0;
    j++;
}
# 10 "demo_pre.c" 2
```

INFORMACIÓN EXTRA
DE DEPURACIÓN

DEMO.C

```
#include <stdio.h>

int variable_global=5;
extern int variable_externa;

int main(){
    int variable_local=32;
    putchar(variable_externa+variable_local);
    return 0;
}
```

VARIABLE.C

```
int variable_externa=3;
```

CÓDIGO
OBJETO



DEMO.O

Name	Type	Addr	Section
main	FUNC	00000000	.text
putchar	NOTYPE		*UND*
variable_externa	NOTYPE		*UND*
variable_global	OBJECT	00000000	.data

CÓDIGO
OBJETO



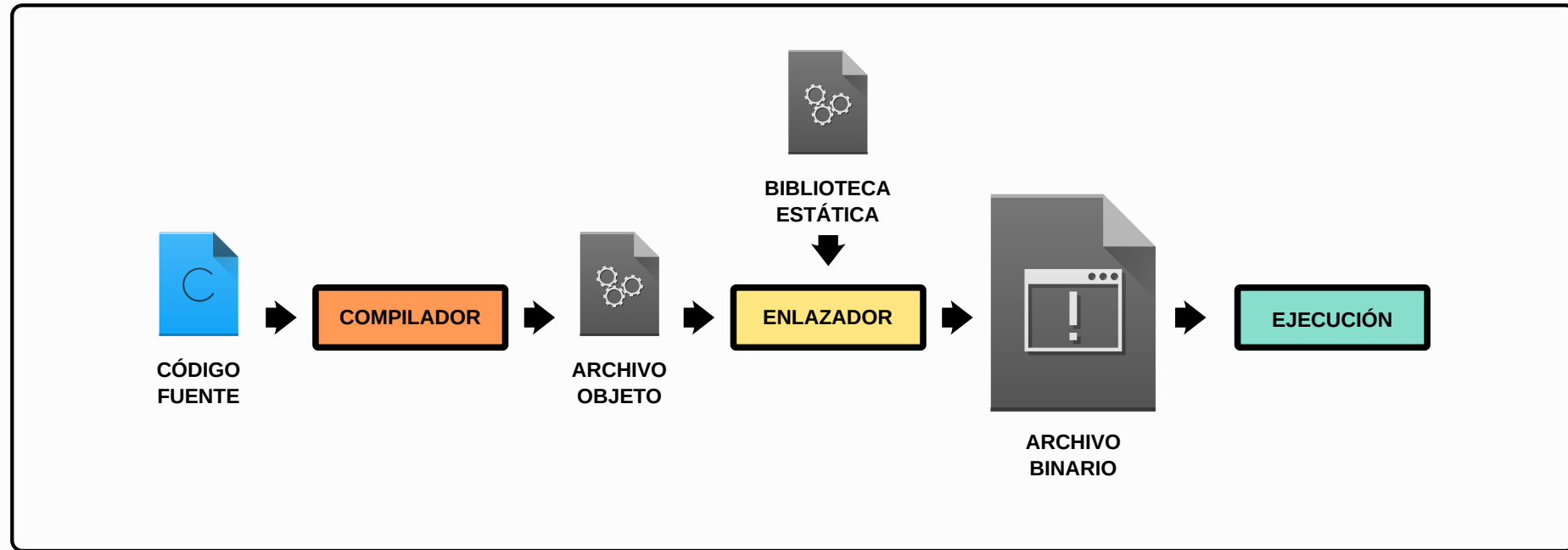
VARIABLE.O

Name	Type	Addr	Section
variable_externa	OBJECT	00000000	.data

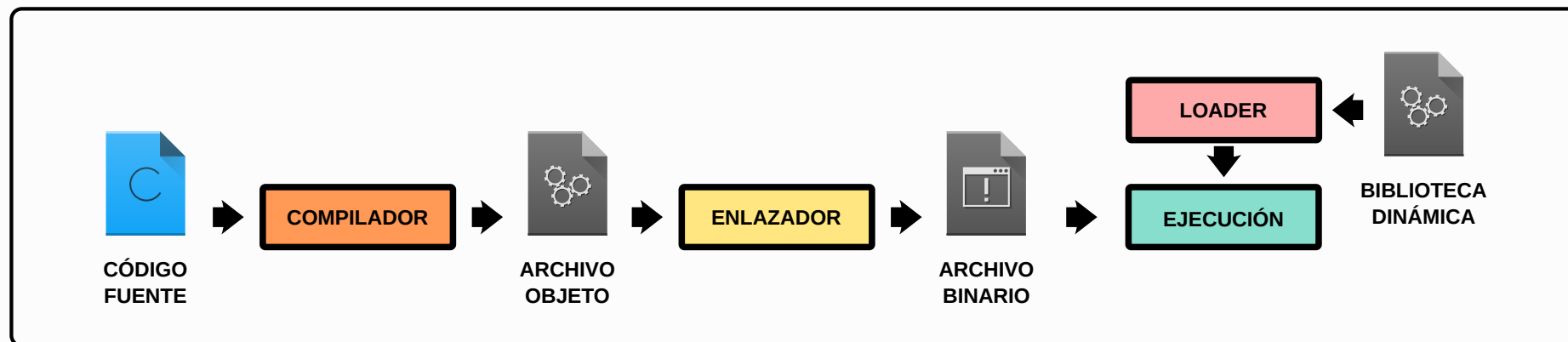
DEMO (ARCHIVO FINAL EJECUTABLE)

Name	Type	Addr	Section
...			
<muchos símbolos que no nos interesan>			
...			
main	FUNC	00401175	.text
putchar@@GLIBC_2.2.5			*UND*
variable_externa	OBJECT	00404034	.data
variable_global	OBJECT	00404030	.data

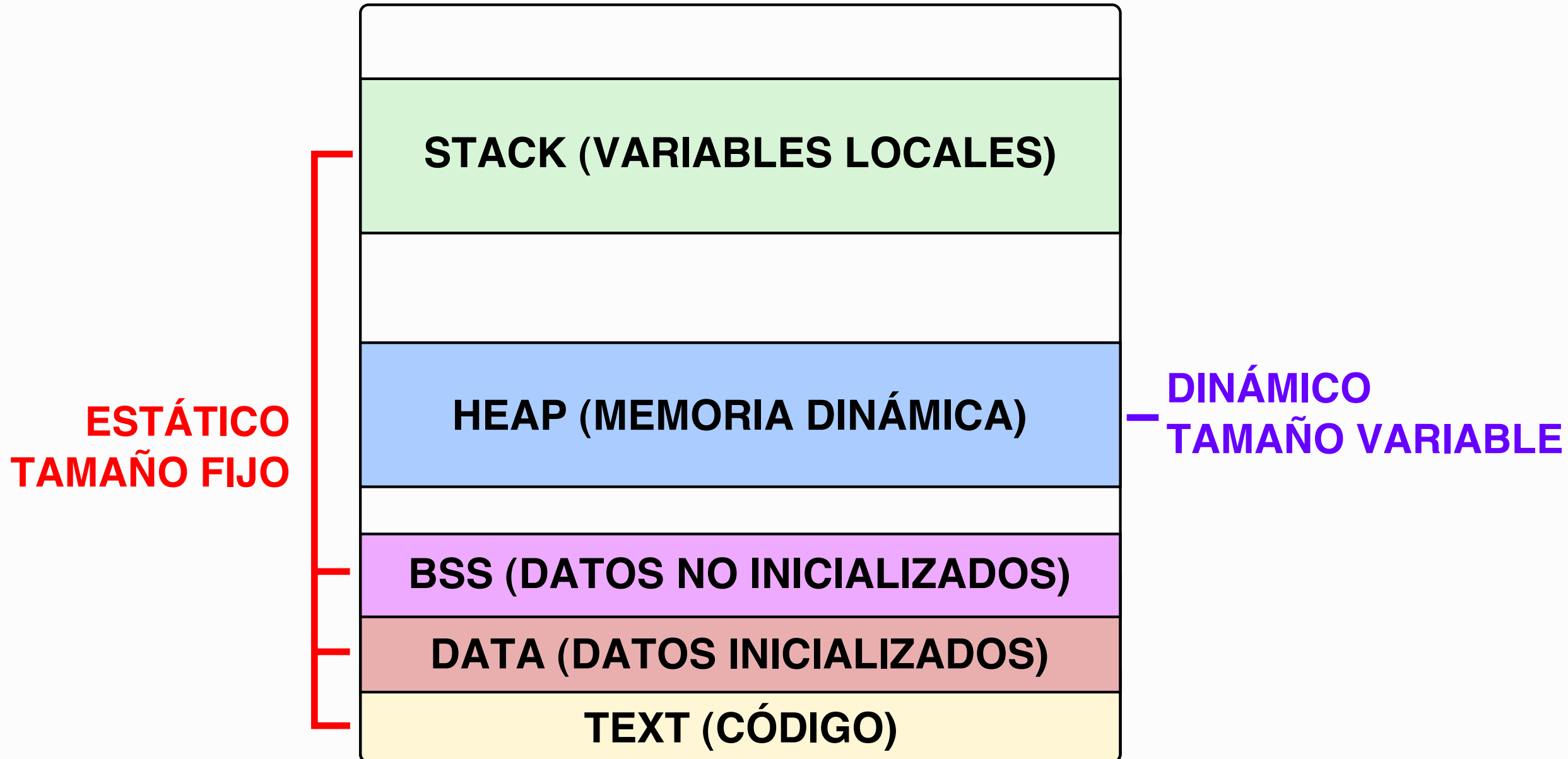
ENLAZADO ESTÁTICO



ENLAZADO DINÁMICO



LENGUAJE C: PUNTEROS Y MEMORIA DINÁMICA

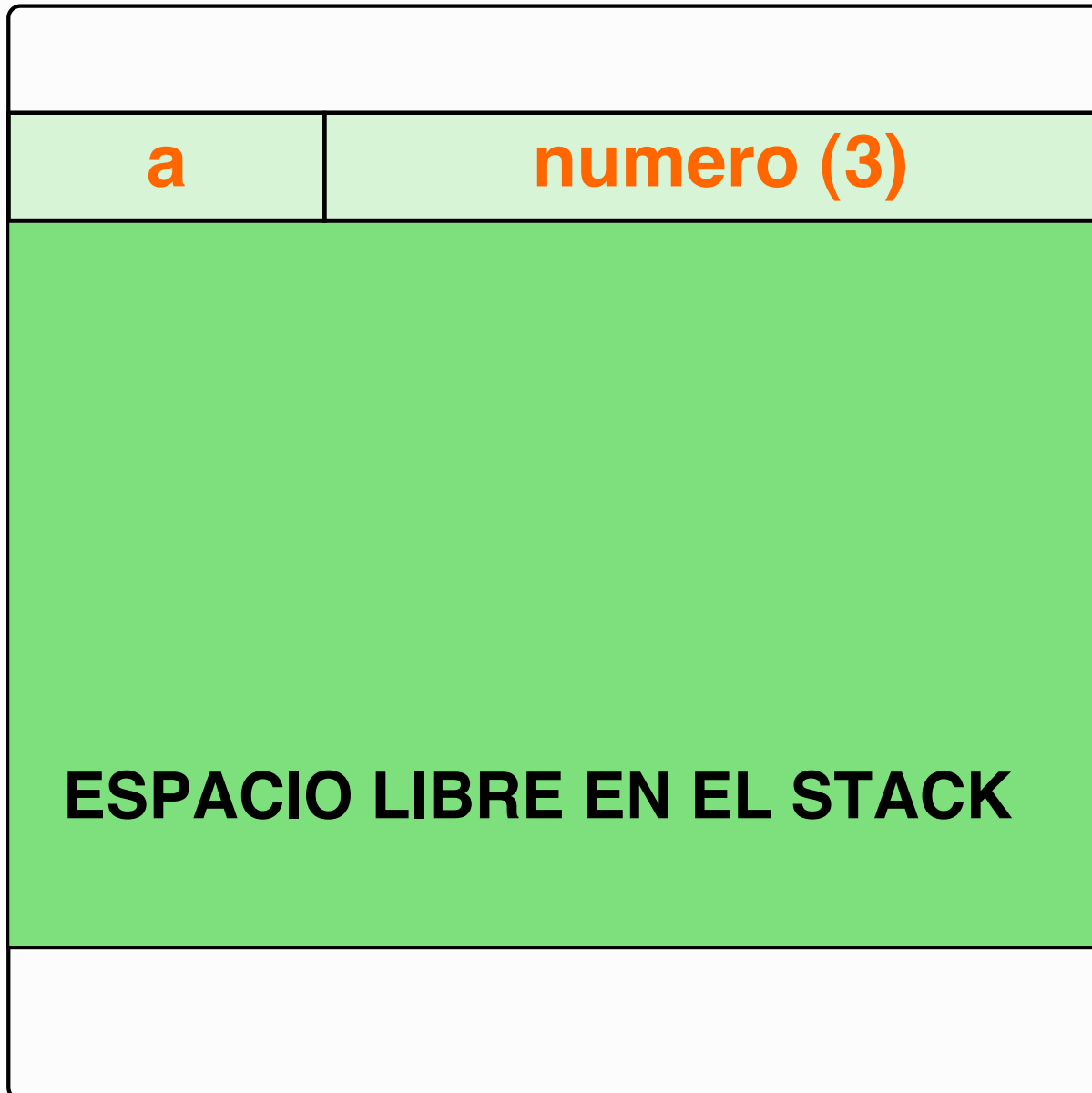


STACK (VACÍO)

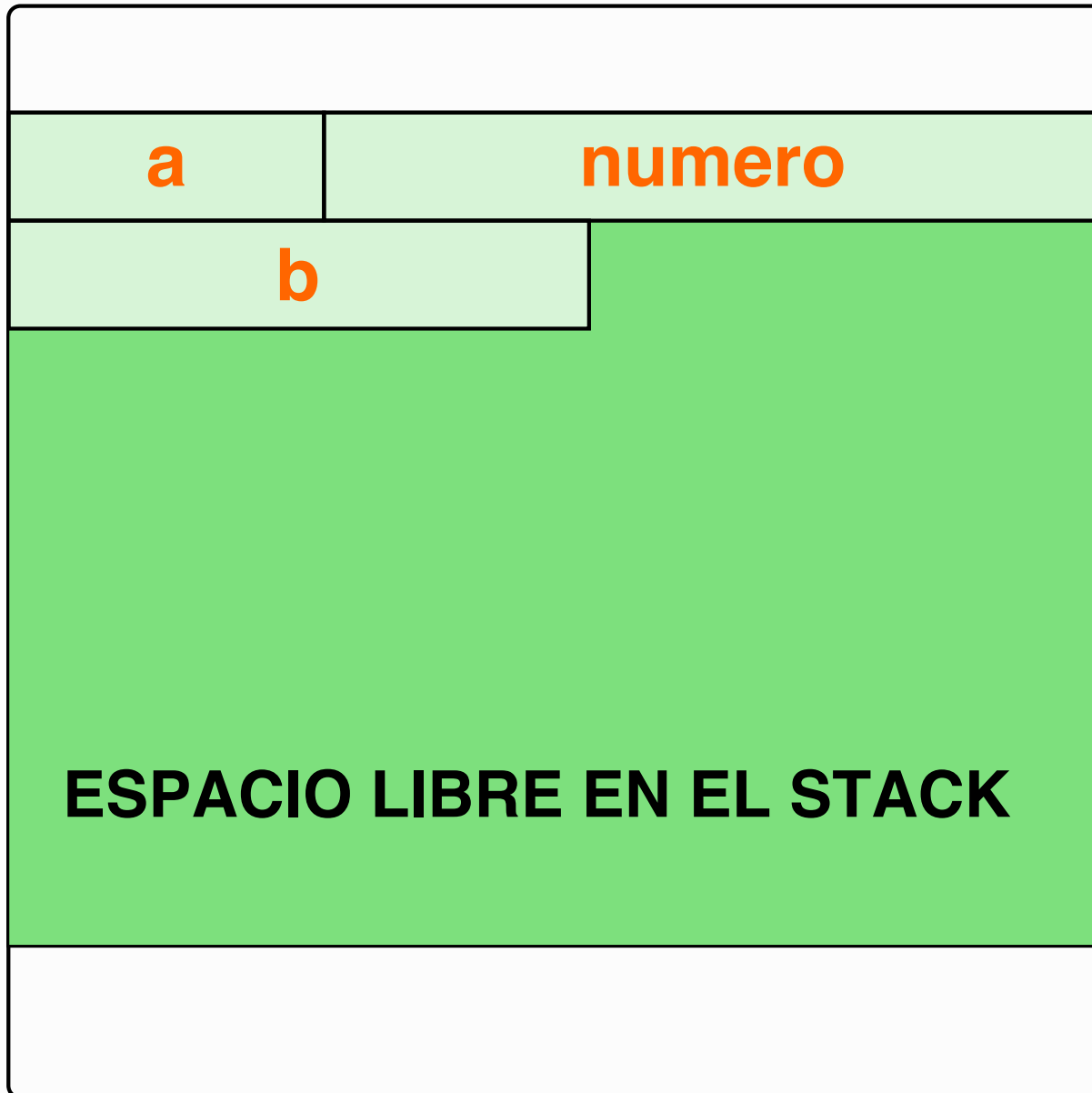
```
00 void mostrar_algo(int n){  
01     int m=25;  
02     printf("%i\n", m*n);  
03 }  
04  
05 int main(){  
06     char a;  
07     int numero=3;  
08     short b;  
09  
10     mostrar_algo(numero);  
11  
12     return 0;  
13 }
```



```
00 void mostrar_algo(int n){  
01     int m=25;  
02     printf("%i\n", m*n);  
03 }  
04  
05 int main(){  
06     ⇒ char a;  
07     int numero=3;  
08     short b;  
09  
10     mostrar_algo(numero);  
11  
12     return 0;  
13 }
```

```
00 void mostrar_algo(int n){
01     int m=25;
02     printf("%i\n", m*n);
03 }
04
05 int main(){
06     char a;
07     ⇒ int numero=3;
08     short b;
09
10     mostrar_algo(numero);
11
12     return 0;
13 }
```



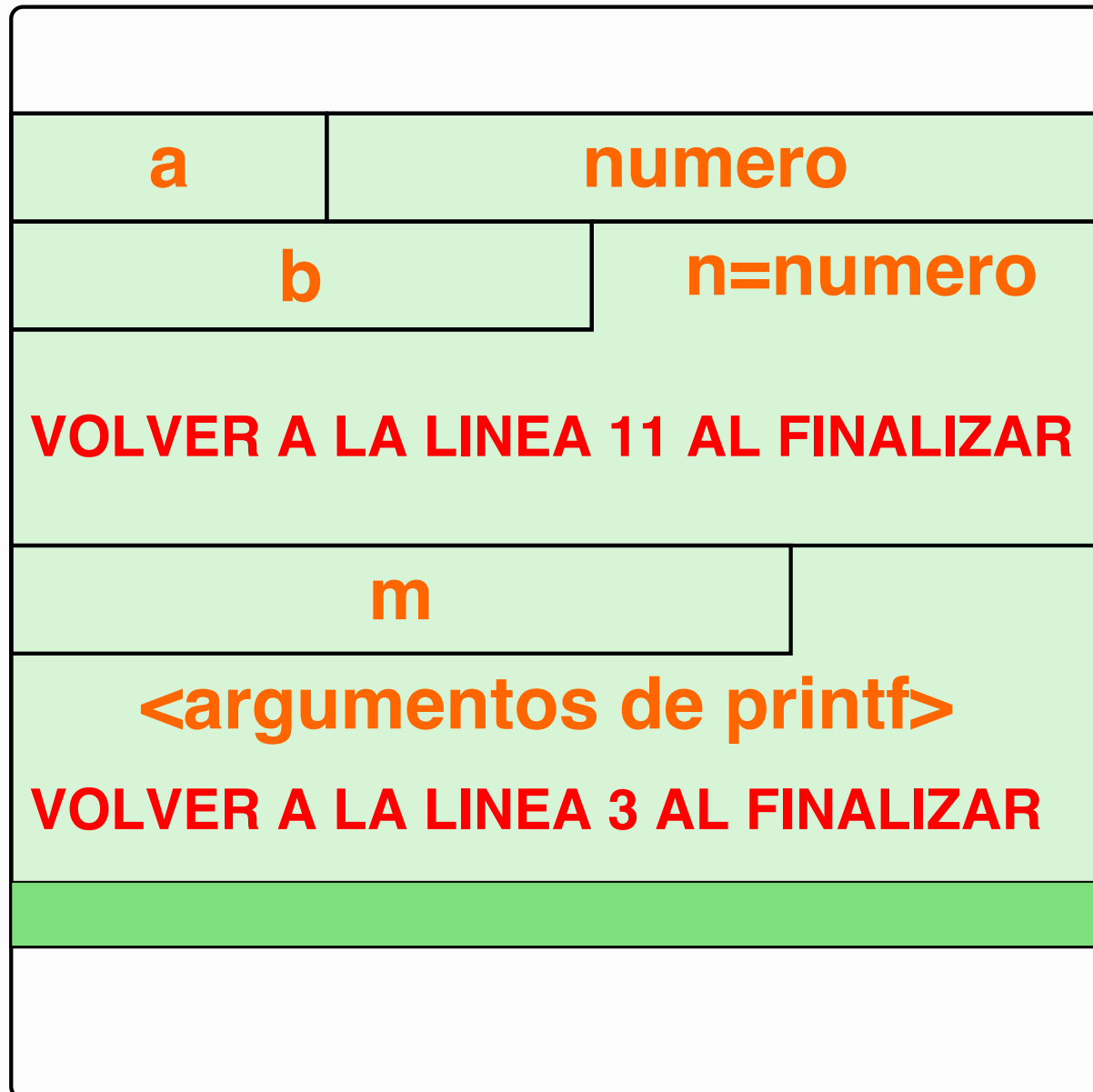
```
00 void mostrar_algo(int n){
01     int m=25;
02     printf("%i\n", m*n);
03 }
04
05 int main(){
06     char a;
07     int numero=3;
08     ⇒ short b;
09
10     mostrar_algo(numero);
11
12     return 0;
13 }
```



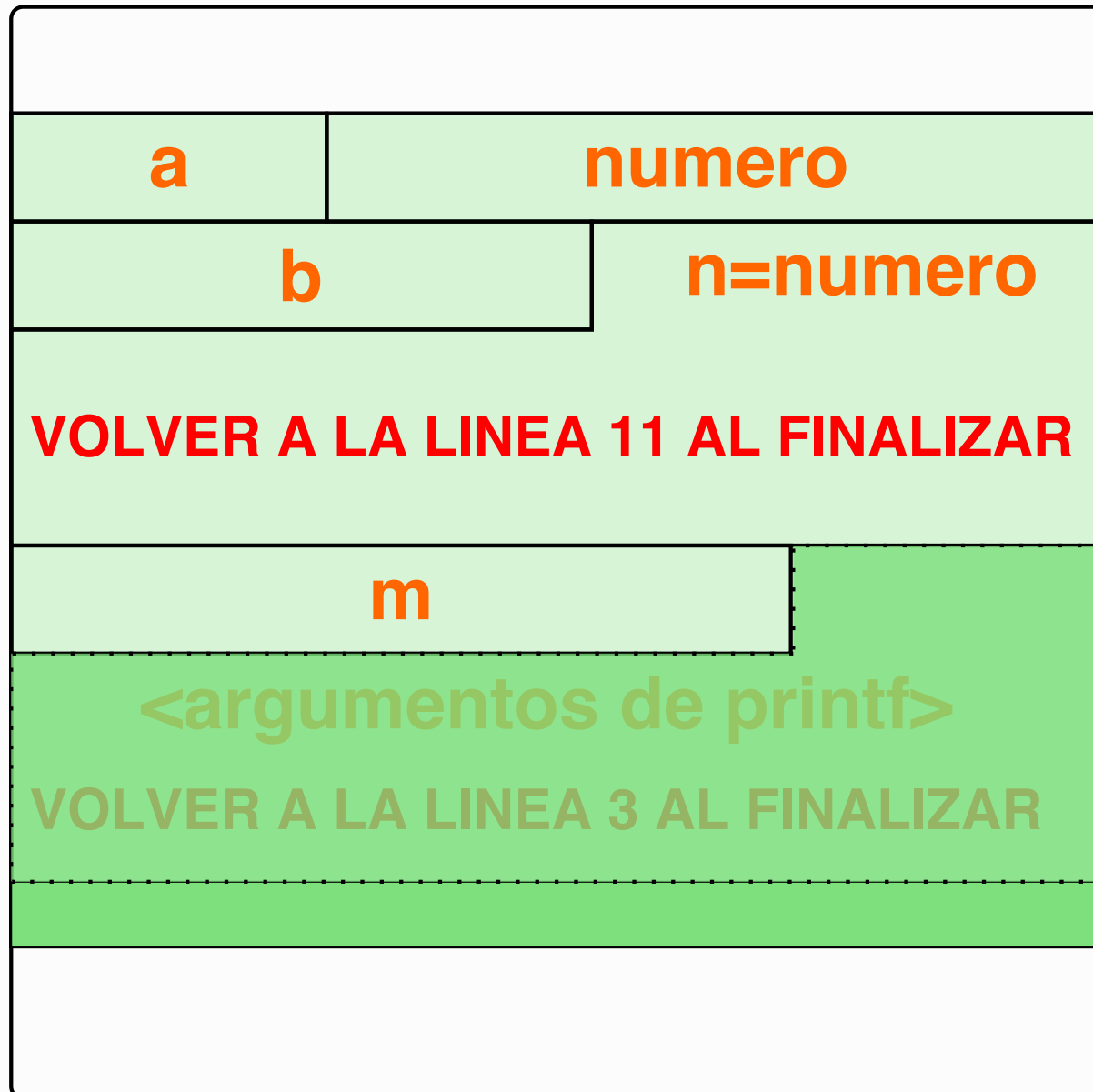
```
00 void mostrar_algo(int n){
01     int m=25;
02     printf("%i\n", m*n);
03 }
04
05 int main(){
06     char a;
07     int numero=3;
08     short b;
09
10     ⇒ mostrar_algo(numero);
11
12     return 0;
13 }
```



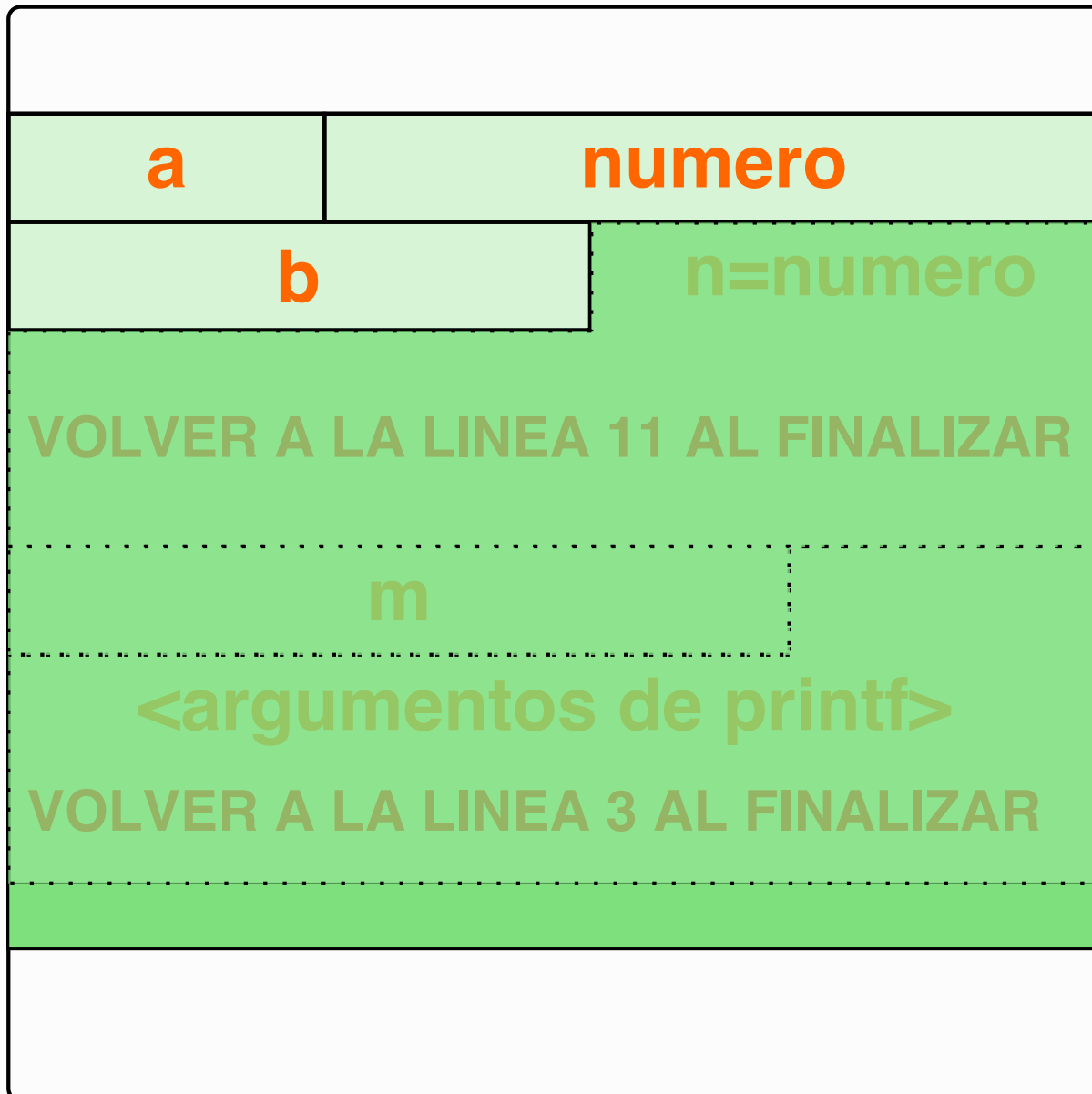
```
00 void mostrar_algo(int n){  
01     ⇒ int m=25;  
02     printf("%i\n", m*n);  
03 }  
04  
05 int main(){  
06     char a;  
07     int numero=3;  
08     short b;  
09  
10     mostrar_algo(numero);  
11  
12     return 0;  
13 }
```



```
00 void mostrar_algo(int n){
01     int m=25;
02     ⇒ printf("%i\n", m*n);
03 }
04
05 int main(){
06     char a;
07     int numero=3;
08     short b;
09
10     mostrar_algo(numero);
11
12     return 0;
13 }
```



```
00 void mostrar_algo(int n){
01     int m=25;
02     printf("%i\n", m*n);
03     ⇒}
04
05 int main(){
06     char a;
07     int numero=3;
08     short b;
09
10     mostrar_algo(numero);
11
12     return 0;
13 }
```



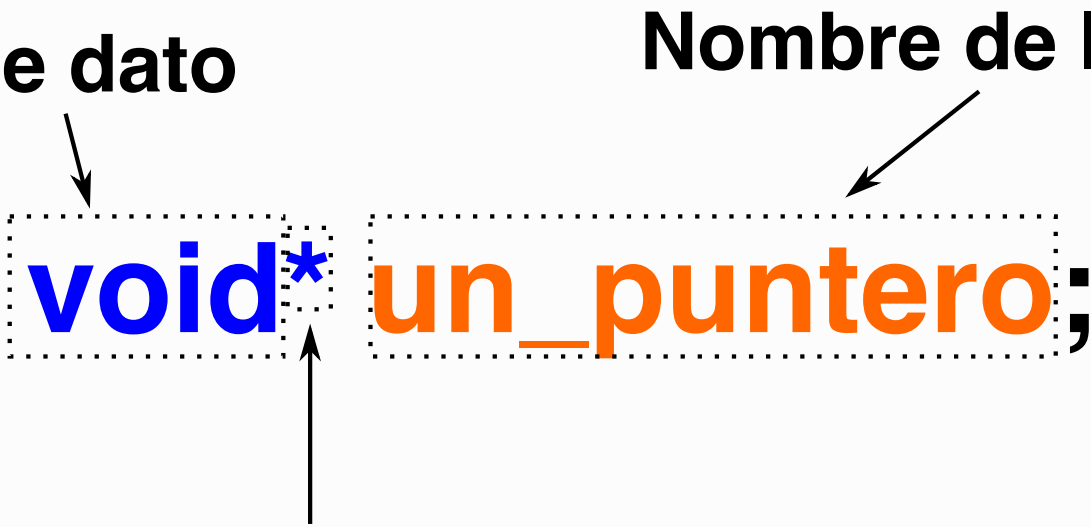
```
00 void mostrar_algo(int n){
01     int m=25;
02     printf("%i\n", m*n);
03 }
04
05 int main(){
06     char a;
07     int numero=3;
08     short b;
09
10     mostrar_algo(numero);
11     ⇒
12     return 0;
13 }
```

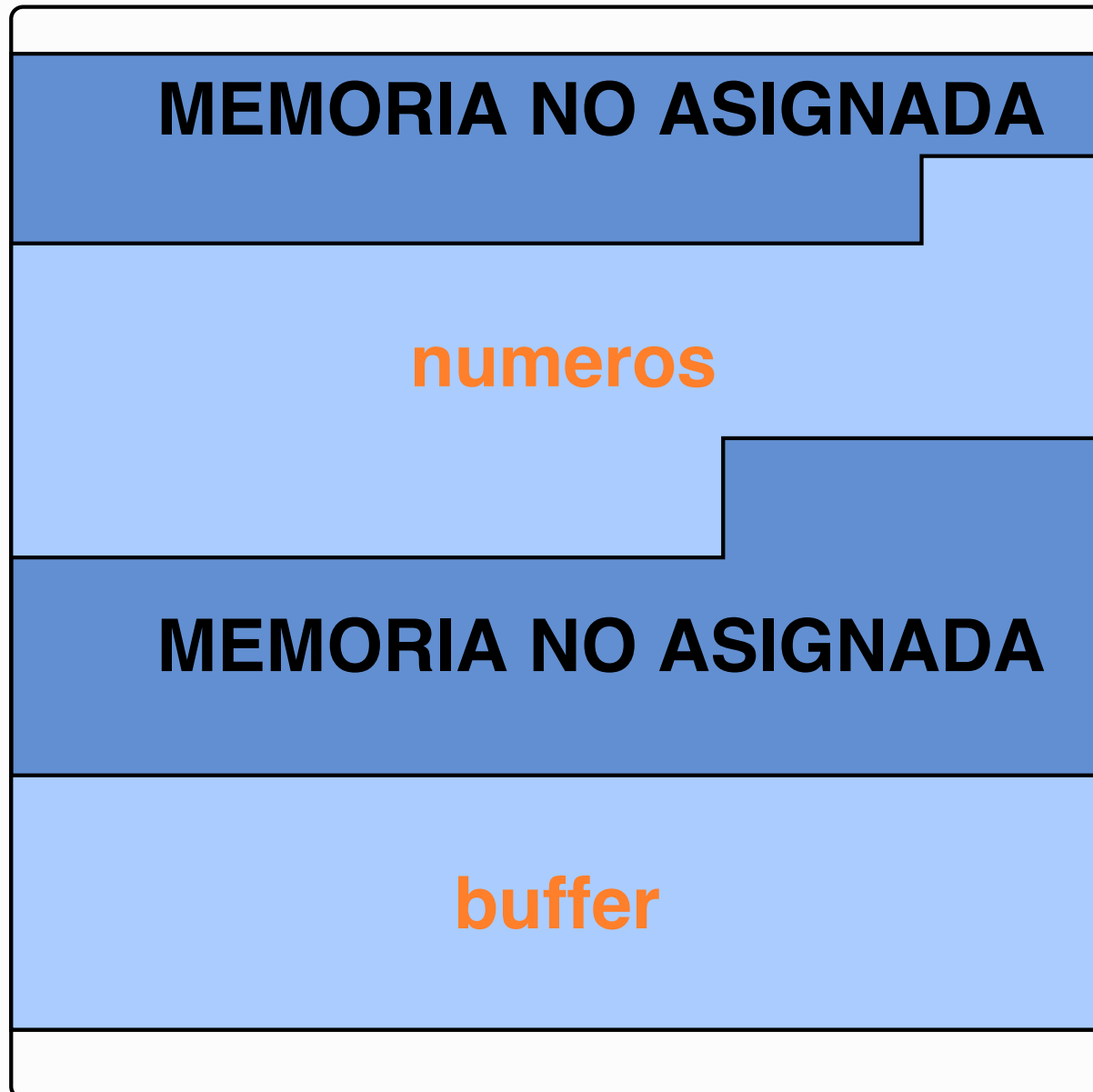
Tipo de dato

Nombre de la variable

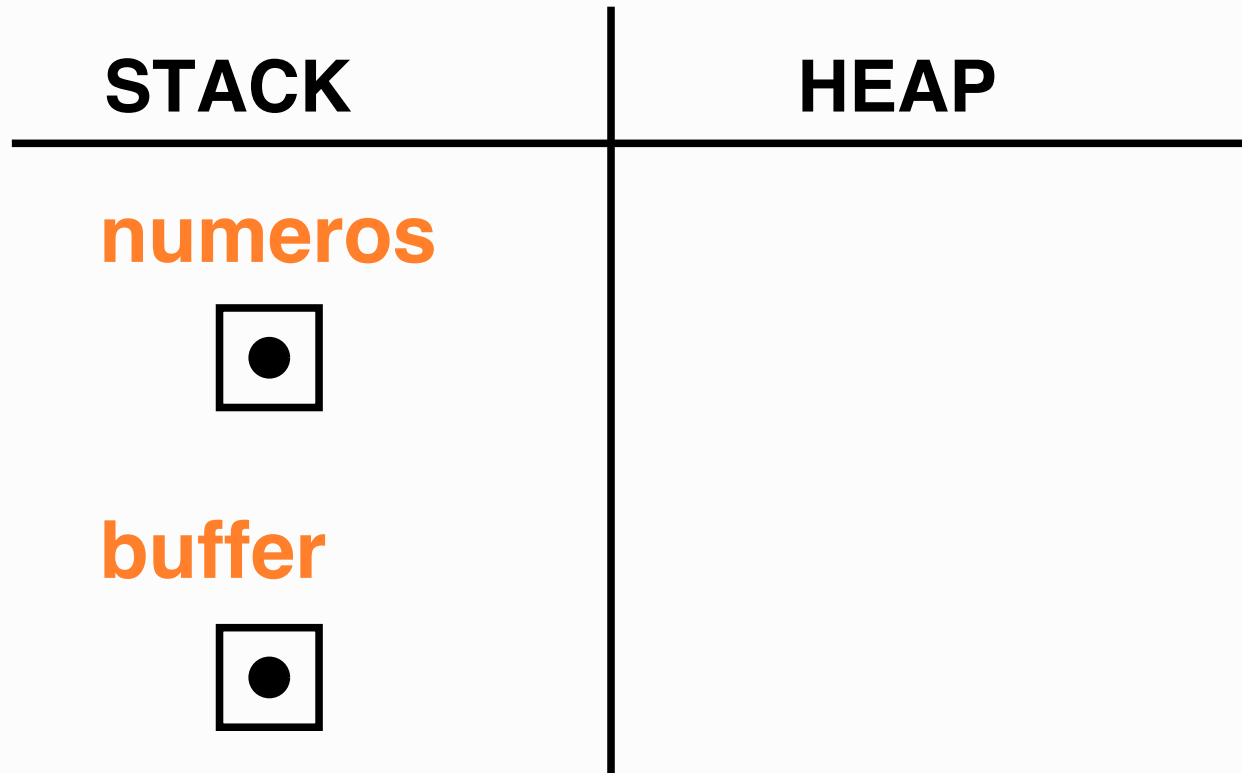
`void*` `un_puntero;`

Indica que es un puntero

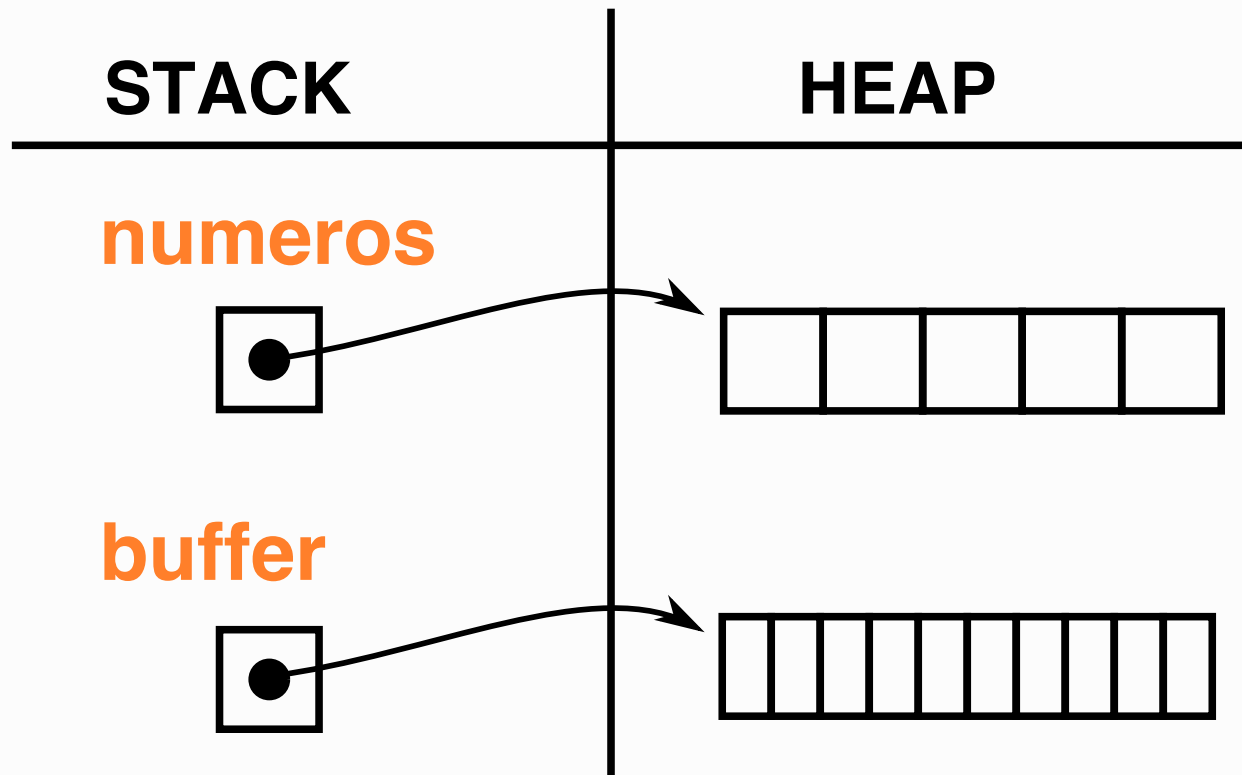




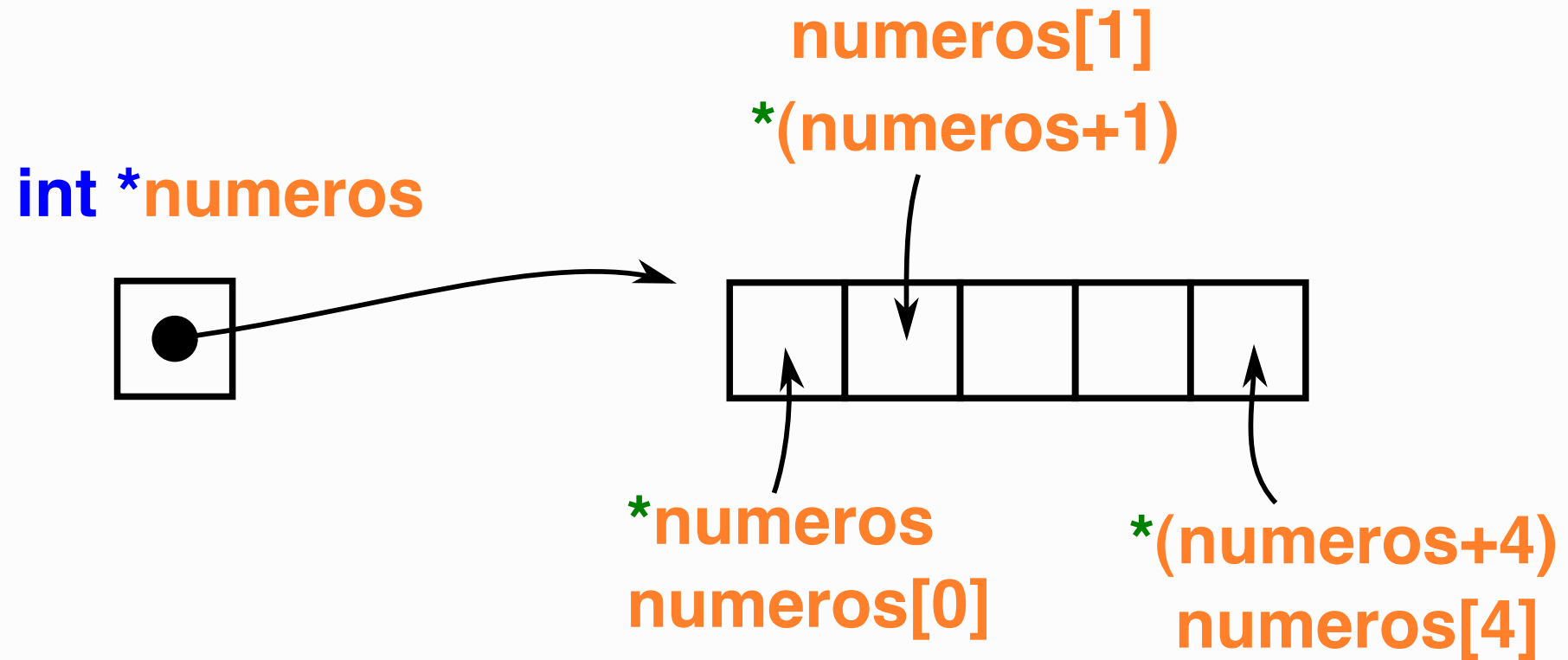
```
00 int main(){  
01     char *buffer;  
02     int *numeros;  
03  
04     buffer = (char*)malloc(10);  
05  
06     numeros = (int*)malloc(5);  
07     ⇒ //Hacer algo con los buffer  
08     free((void*)buffer);  
09     free((void*)numeros);  
10     return 0;  
11 }
```



```
00 int main(){
01     char *buffer;
02     int *numeros;
03
04     buffer = (char*)malloc(10);
05
06     numeros = (int*)malloc(5);
07     ⇒ //Hacer algo con los buffer
08     free((void*)buffer);
09     free((void*)numeros);
10     return 0;
11 }
```

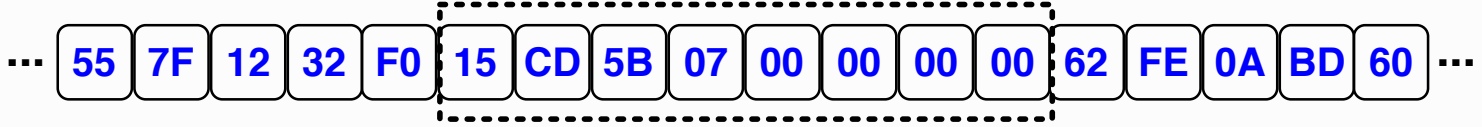


```
00 int main(){
01     char *buffer;
02     int *numeros;
03
04     buffer = (char*)malloc(10);
05
06     numeros = (int*)malloc(5);
07     ⇒ //Hacer algo con los buffer
08     free((void*)buffer);
09     free((void*)numeros);
10     return 0;
11 }
```



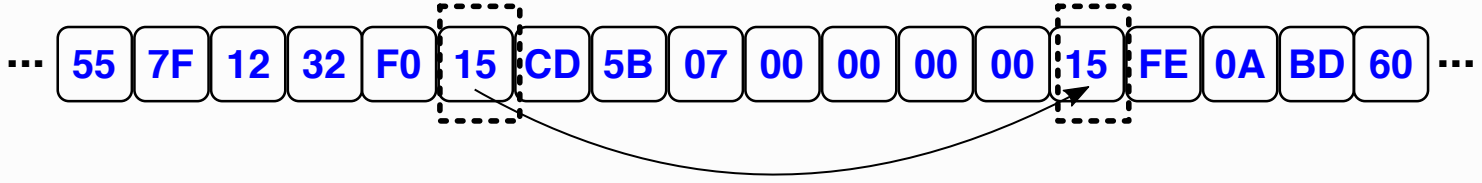
```
long i = 123456789;  
char un_byte;
```

i = 123456789 = 0x75BCD15



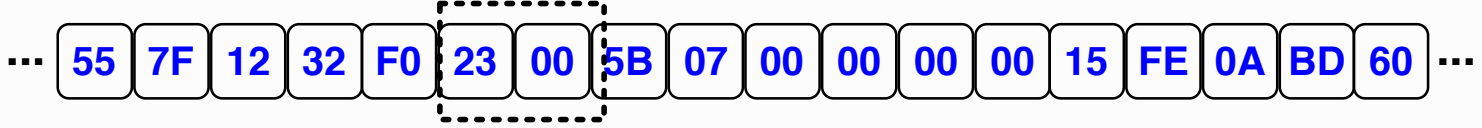
```
un_byte = (char)i;
```

(char)i un_byte



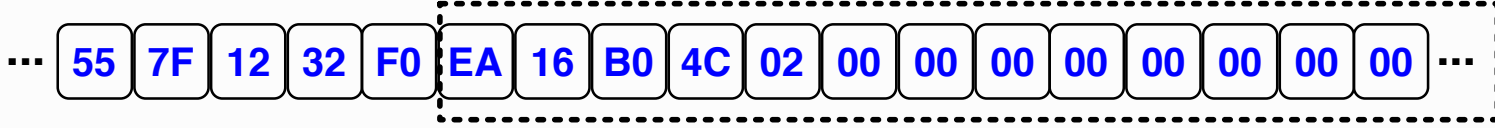
```
(short)i = 35;
```

(short)i = 35 = 0x23 (i = 0x075B0023 = 123404323)



```
(long double)i = 987643210;
```

(long double)i = 9876543210 = 0x24CB016EA

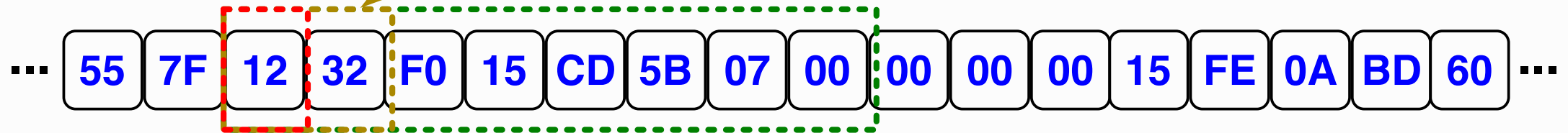


void* buffer;

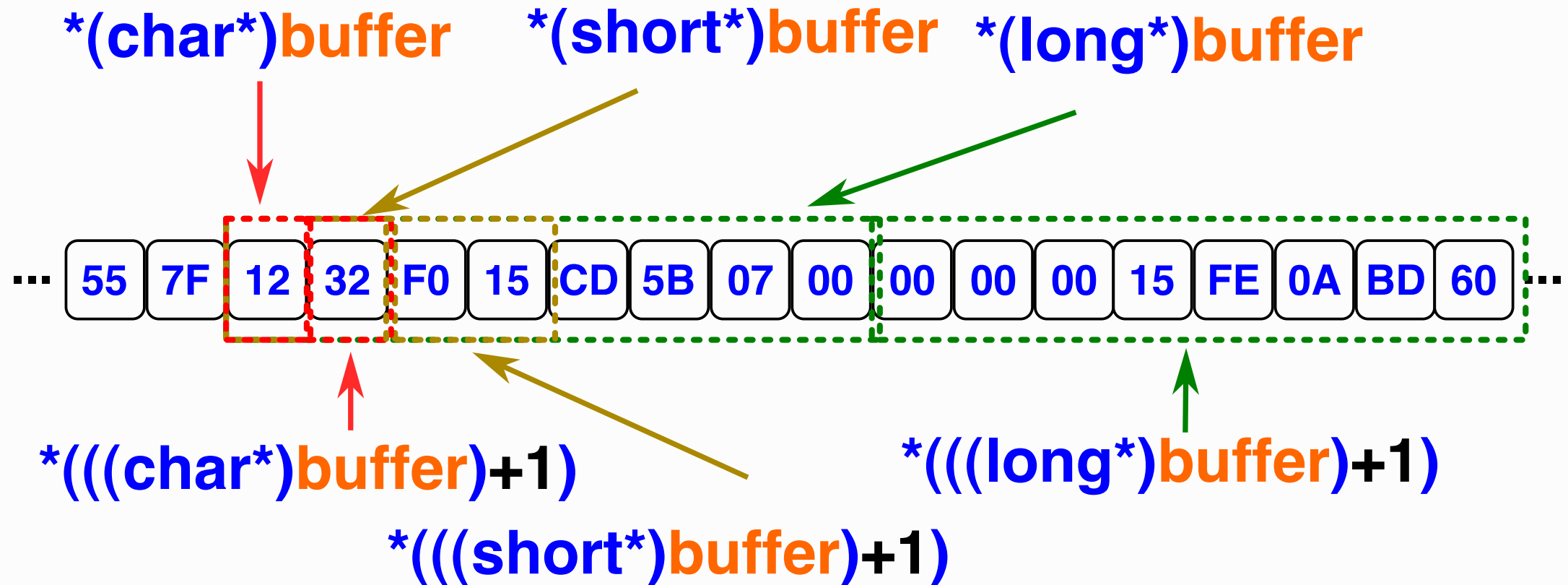
***(short*)buffer**

***(char*)buffer**

***(long*)buffer**



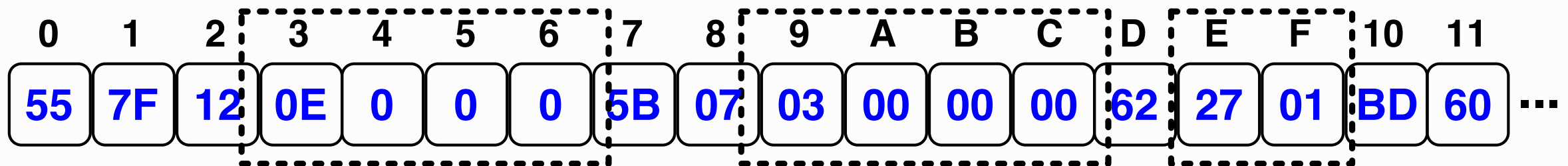
void* buffer;



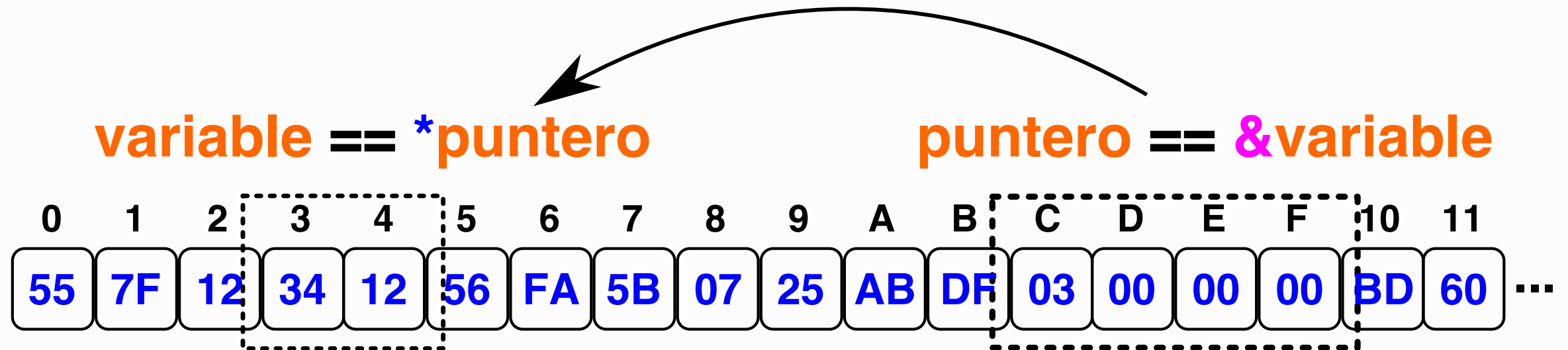
```
void** p1 = 0x09;  
void* p2 = 0x03;  
short* p3 = 0x0E
```

```
*p1 == (void*)0x03;  
*(char*)p2 == 0x0E;  
*p3 == 0x0127;
```

```
**(short**)p1 == *((short*)(*p1)) == 0x000E
```




```
short variable = 0x1234;  
short* puntero = &variable;
```



¿PREGUNTAS?

FIN



.UBAfiuba



FACULTAD DE INGENIERÍA