Escuela de
Ciencia y Tecnología
ECyT_UNSAM

Programación (en C)

Segundo Cuatrimestre 2025

programacionbunsam@gmail.com

Tipos de datos

Datos

En este contexto, con **datos** nos referimos a la información que puede ser procesada por una computadora

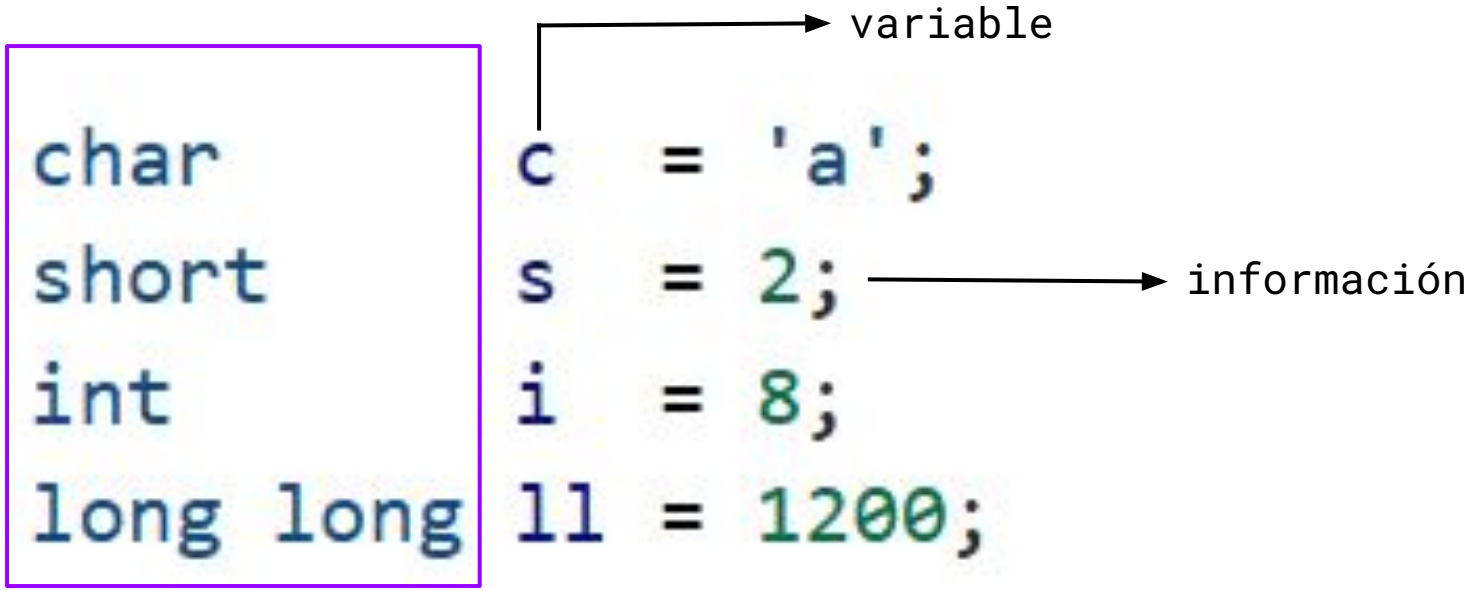
Datos

En este contexto, con **datos** nos referimos a la información que puede ser procesada por una computadora

```
char      c  = 'a';
short     s  = 2;
int       i  = 8;
long long ll = 1200;
```

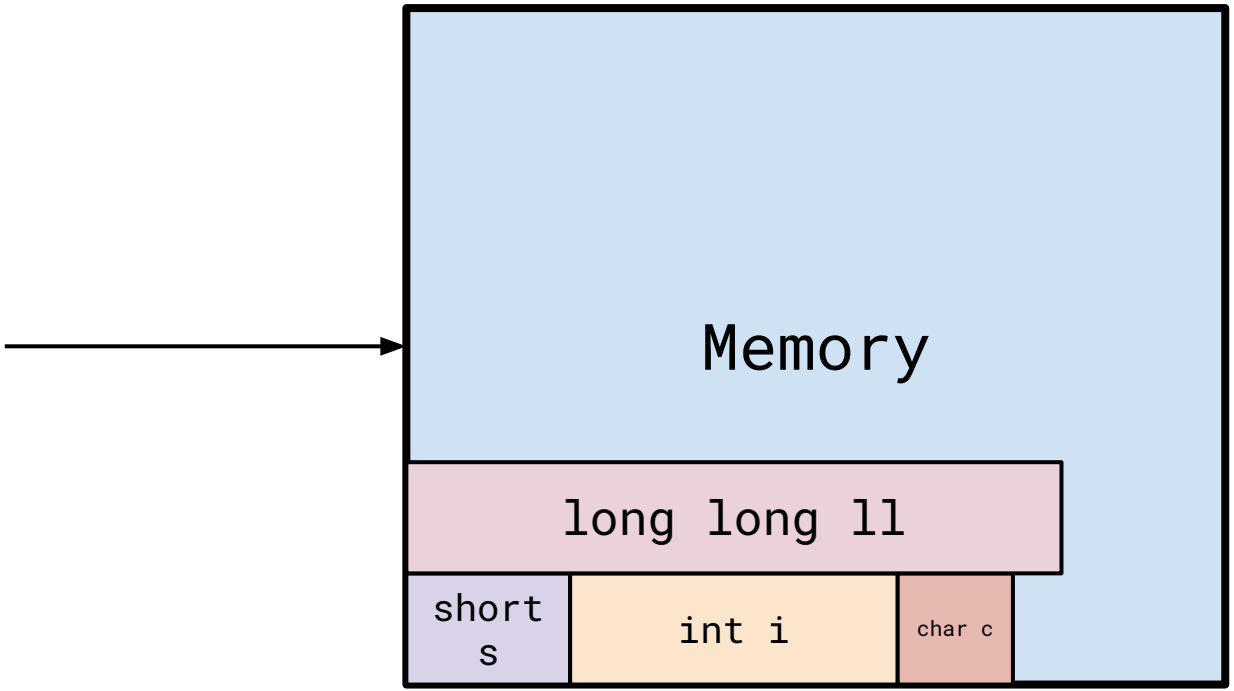
Datos

En este contexto, con **datos** nos referimos a la información que puede ser procesada por una computadora



Tipos de dato en la memoria

```c
int main(void)
{

  char c = 'a';
  short s = 2;
  int i = 8;
  long long ll = 1200;

  return 0;
}
```



Distintos tipos de dato

Tipo
char
short
int
long
long long
float *
double *

Distintos tipos de dato (C Standard)

Tipo	Tamaño mínimo
char	1 (8b)
short	2 (16b)
int	2 (16b)
long	4 (32b)
long long	8 (64b)
float *	4 (32b)
double *	8 (64b)

```
sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)
```

*Este tipo de variable está definido por IEEE-754 (Standard for Floating Point Arithmetic)

Distintos tipos de dato (C Standard)

Tipo	Tamaño mínimo	Tamaño típico
char	1 (8b)	1 (8b)
short	2 (16b)	2 (16b)
int	2 (16b)	4 (32b)
long	4 (32b)	4 (32b)
long long	8 (64b)	8 (64b)
float *	4 (32b)	4 (32b)
double *	8 (64b)	8 (64b)

```
sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)
```

*Este tipo de variable está definido por IEEE-754 (Standard for Floating Point Arithmetic)

Distintos tipos de dato (C Standard)

Tipo	Tamaño mínimo	Tamaño típico	Signed range	Unsigned range
char	1 (8b)	1 (8b)	-2^7 a 2^7-1	2^8
short	2 (16b)	2 (16b)	-2^{15} a $2^{15}-1$	2^{16}
int	2 (16b)	4 (32b)	-2^{31} a $2^{31}-1$	2^{32}
long	4 (32b)	4 (32b)	-2^{31} a $2^{31}-1$	2^{32}
long long	8 (64b)	8 (64b)	-2^{63} a $2^{63}-1$	2^{64}
float *	4 (32b)	4 (32b)	~(1.2E-38 a 3.4E+38)	-
double *	8 (64b)	8 (64b)	~(-1.7E+308 a 1.7E+308)	-

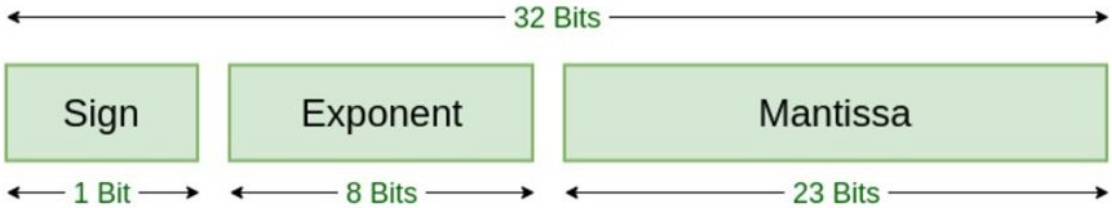
`sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)`

*Este tipo de variable está definido por IEEE-754 (Standard for Floating Point Arithmetic)

Floats

The size of the float is 32-bit, out of which:

- **The most significant bit (MSB)** is used to store the **sign** of the number.
- The next **8 bits** are used to store the **exponent.**
- The remaining **23 bits** are used to store the **mantissa.**

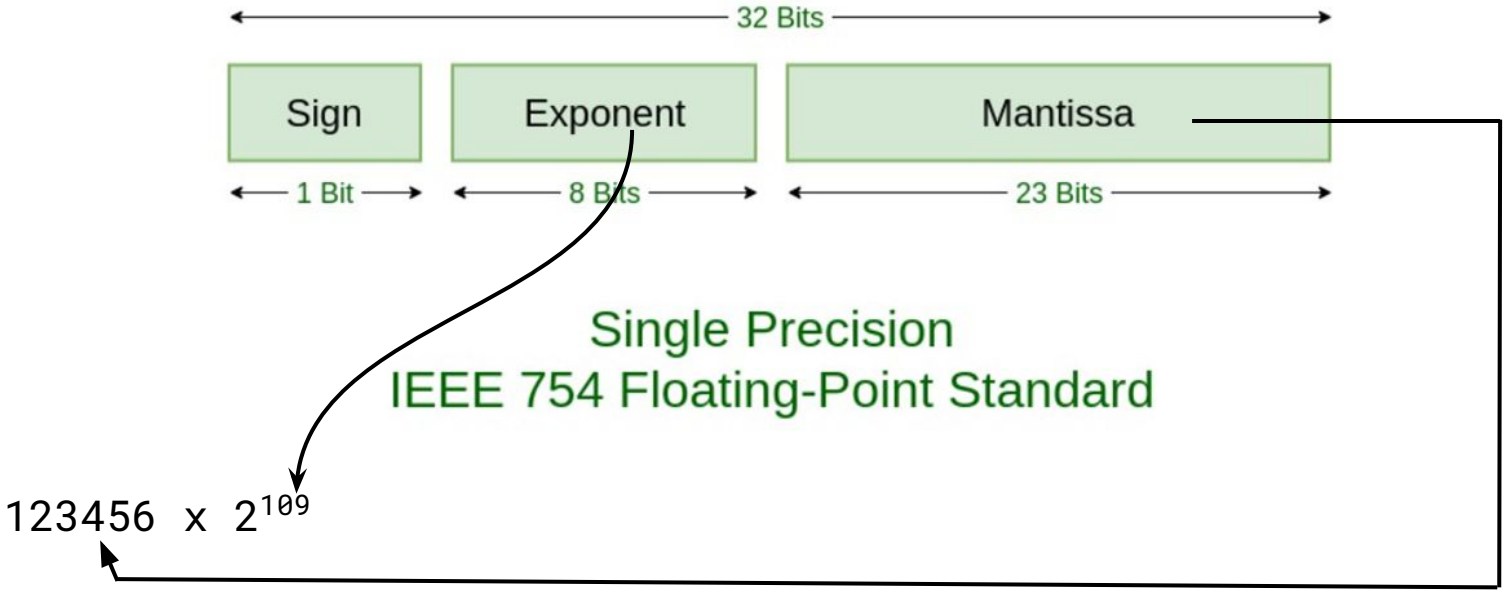


Single Precision
IEEE 754 Floating-Point Standard

Floats

The size of the float is 32-bit, out of which:

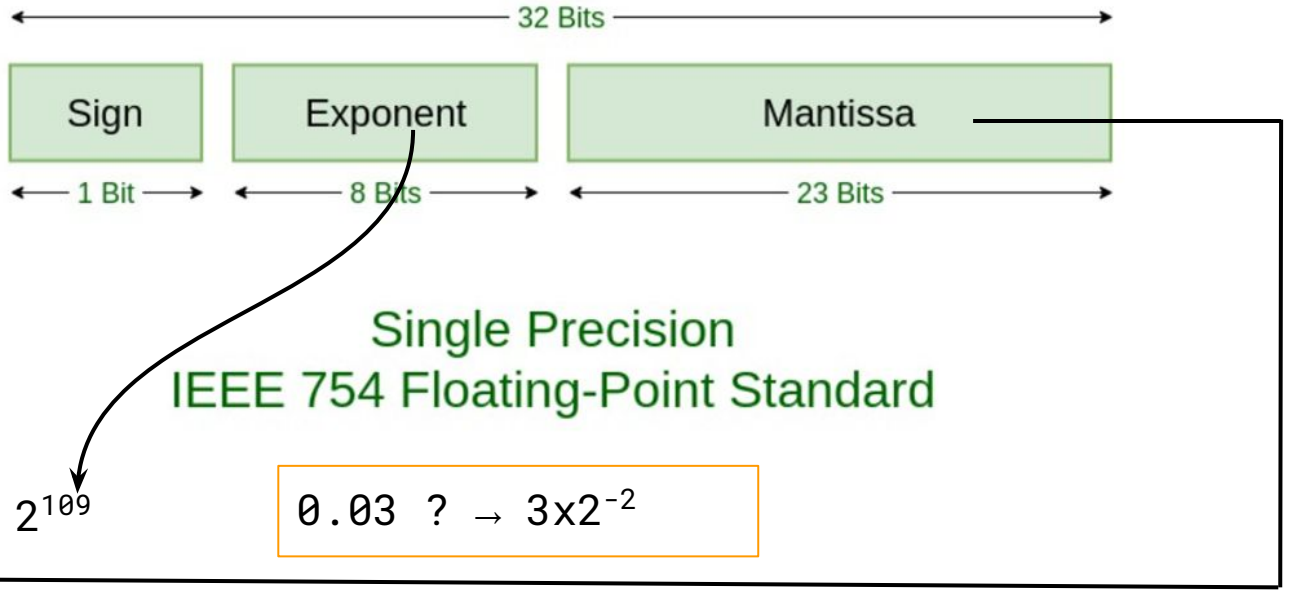
- **The most significant bit (MSB)** is used to store the **sign** of the number.
- The next **8 bits** are used to store the **exponent.**
- The remaining **23 bits** are used to store the **mantissa.**



Floats

The size of the float is 32-bit, out of which:

- **The most significant bit (MSB)** is used to store the **sign** of the number.
- The next **8 bits** are used to store the **exponent.**
- The remaining **23 bits** are used to store the **mantissa.**



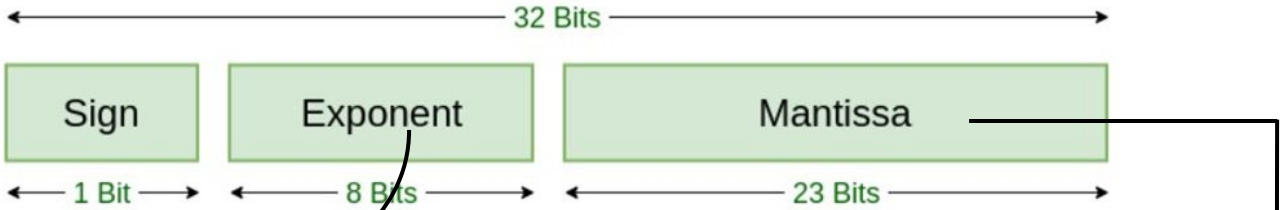
Floats

0ºC → 273ºK

0_{dec} → 0111 1111$_{bit}$ = 127_{dec} (para exponente)

The size of the float is 32-bit, out of which:

- **The most significant bit (MSB)** is used to store the **sign** of the number.
- The next **8 bits** are used to store the **exponent.**
- The remaining **23 bits** are used to store the **mantissa.**



Single Precision
IEEE 754 Floating-Point Standard

123456×2^{109}

0.03 ? → $3x2^{-2}$

Floats

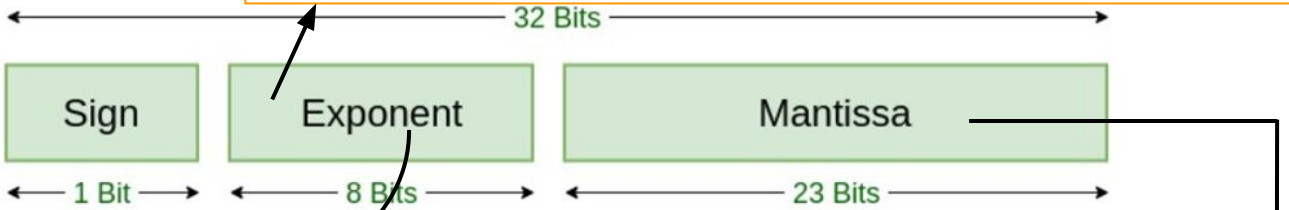
0ºC → 273ºK

0_{dec} → 0111 1111_{bit} = 127_{dec} (para exponente)

The size of the float is 32-bit, out of which:

- **The most significant bit (MSB)** is used to store the **sign** of the number.
- The next **8 bits** are used to store the **exponent.**
- The remaining **23 bits** are used to store the **mantissa.**

$0000\ 1111_{bin}$ → 15_{dec} → 15 - 127 = -112



Single Precision
IEEE 754 Floating-Point Standard

123456×2^{109}

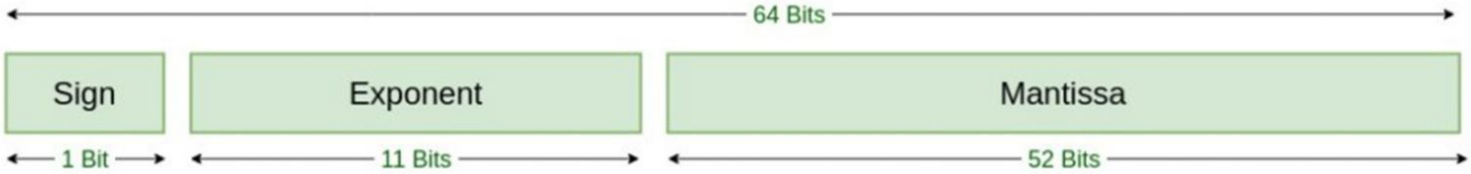
0.03 ? → $3x2^{-2}$

Double

C double Memory Representation

The size of the float is 32-bit, out of which:

- **The most significant bit (MSB)** is used to store the **sign** of the number.
- The next **11 bits** are used to store the **exponent.**
- The remaining **52 bits** are used to store the **mantissa.**



Double Precision
IEEE 754 Floating-Point Standard

<stdint.h> (1)

Este *header file* contiene definiciones de tipos de datos que nos dicen explícitamente su tamaño

<stdint.h> (1)

Este *header file* contiene definiciones de tipos de datos que nos dicen explícitamente su tamaño

```
char caracter = 8;
short corto   = 16;
int entero    = 32;
```

VS

```
int8_t ochoBits = 8;
int16_t unoSeisBits = 16;
uint32_t uTresDosBits = 32;
```

<stdint.h> (1)

Este *header file* contiene definiciones de tipos de datos que nos dicen explícitamente su tamaño

```
char caracter = 8;
short corto   = 16;
int entero    = 32;
```

VS

```
int8_t ochoBits = 8;
int16_t unoSeisBits = 16;
uint32_t uTresDosBits = 32;
```

Este approach a los tipos de datos mejoran la **legibilidad**,**robustez** y **portabilidad** del código

<stdint.h> - Familias

```
int8_t ochoBits;        int_least8_t almenosOcho;        int_fast8_t fastOcho;
```

Familia	Tamaño	Presencia
N_t	N bits garantizados	Solo si la implementación puede garantizarlo
_leastN_t	Al menos N bits, puede ser mayor	Siempre definido
_fastN_t	Al menos N bits, elegido para que dar la mayor velocidad en la arquitectura	Siempre definido

Booleans (1)

→ Los booleanos son variables que pueden tomar 2 valores: ***True*** o ***False*** En C, los booleans no existen como un tipo de datos en sí mismo, pero los estamos usando constantemente, cada vez que empleamos operadores como < , > , == , etc.

```
if (a < b)
    printf("a<b");

1.  se evalua la expression a < b
    si a < b ⇒ True
    si a >= b ⇒ False
2. if (resultado de evaluar la expresion)
    if (True) ⇒ se entra en la condicion
    if (False) ⇒ no se entra en la condición
```

```
int a = 5, b = 3;

int bool = a < b;
    → bool = 0; //False

bool = a > b;
    → bool = 1; //True
```

Booleans (2)

```c
#include <stdio.h>

int main (void){
    int a = 1;
    float b = 0.0;
    char c = 'a', d = '0';

    if(a)
        printf("%d (int)\t\t is considered true\n", a);

    if(b)
        printf("%f (float)\t is considered true\n", b);

    if(c)
        printf("%c (char)\t is considered true\n", c);

    if(d)
        printf("%c (char)\t is considered true\n", d);
    return 0;
}
```

obs. En C todas las variables distintas de 0 son consideradas ***True***.

Booleans (3) - Coding Style Guides

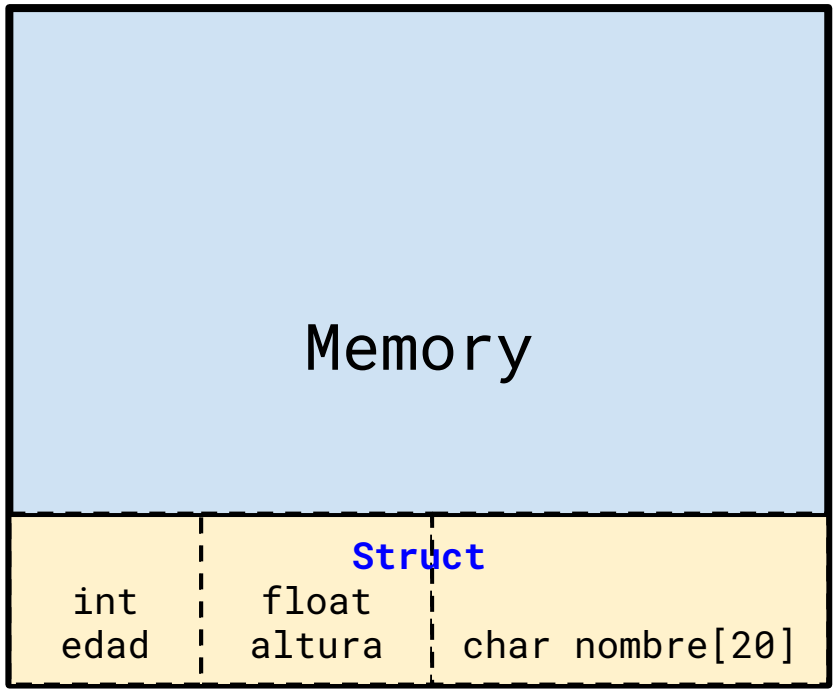
```
size_t length = 5;  /* Counter variable */
uint8_t is_ok = 0;  /* Boolean-treated variable */
if (length)         /* Wrong, length is not treated as boolean */
if (length > 0)     /* OK, length is treated as counter variable containing multi values, not only 0 or 1 */
if (length == 0)    /* OK, length is treated as counter variable containing multi values, not only 0 or 1 */

if (is_ok)          /* OK, variable is treated as boolean */
if (!is_ok)         /* OK, -||- */
if (is_ok == 1)     /* Wrong, never compare boolean variable against 1! */
if (is_ok == 0)     /* Wrong, use ! for negative check */
```

```
if (length != 0)
```

Structs y Unions

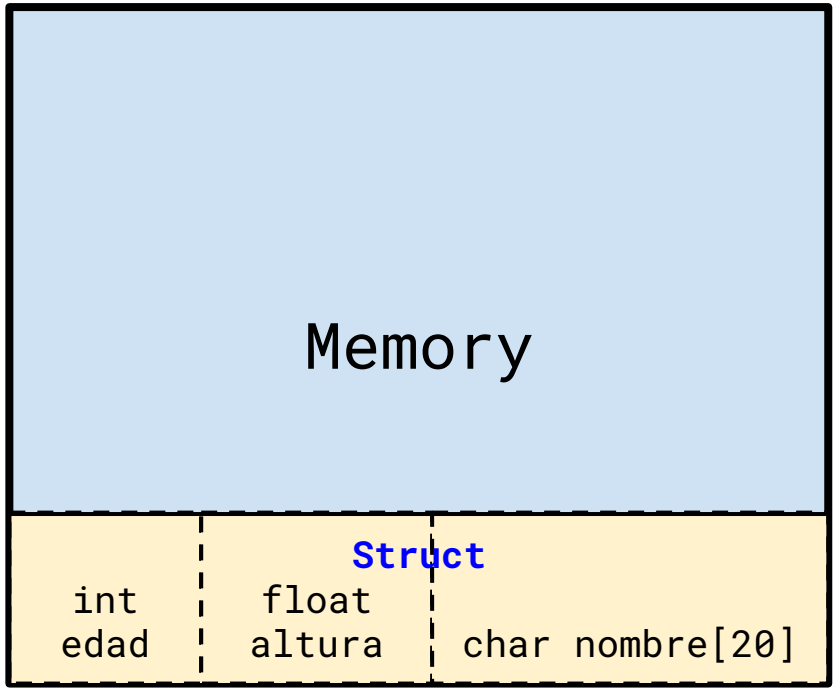
Structs (1)

def. Las structs son un tipo de dato compuesto en C que agrupa, en sí misma, un grupo de variables bajo un mismo nombre. Estas variables pueden ser de distinto tipo, lo que hace que las structs sean muy útiles para crear conjuntos de datos complejos que representen objetos o registros.
Una ventaja importante de las structs es que permiten a una función retornar varios datos agrupados en una variable.



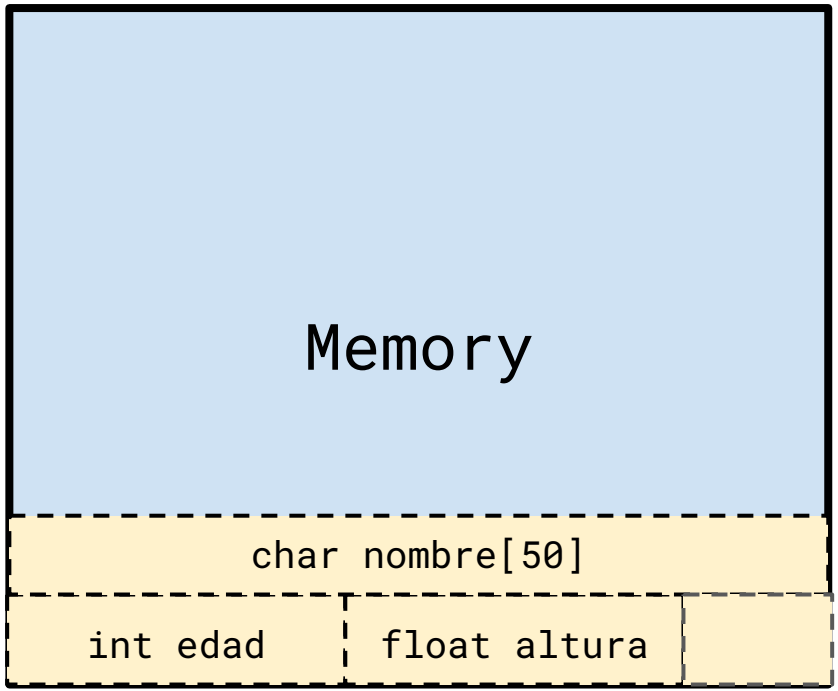
Structs (2) - example

```c
struct persona{
    char nombre[20];
    int edad;
    float altura;
};

struct persona p1 = {
    .nombre = "Felipe",
    .edad = 25,
    .altura = 1.8
};

/* Una forma de equivalente de inicializar un struct es: */
struct persona p2 = {"Marco", 24, 1.75};
```



Structs (3) - con padding

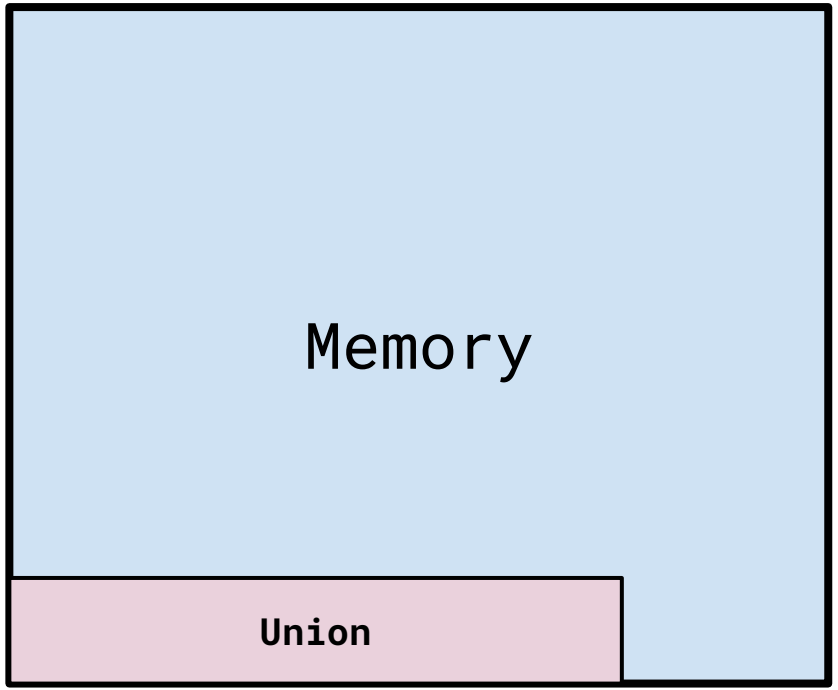
```c
struct persona{
    char nombre[50];
    int edad;
    float altura;
};

struct persona p1 = {
    .nombre = "Felipe",
    .edad = 25,
    .altura = 1.8
};

/* Una forma de equivalente de inicializar un struct es: */
struct persona p2 = {"Marco", 24, 1.75};
```



Unions (1)

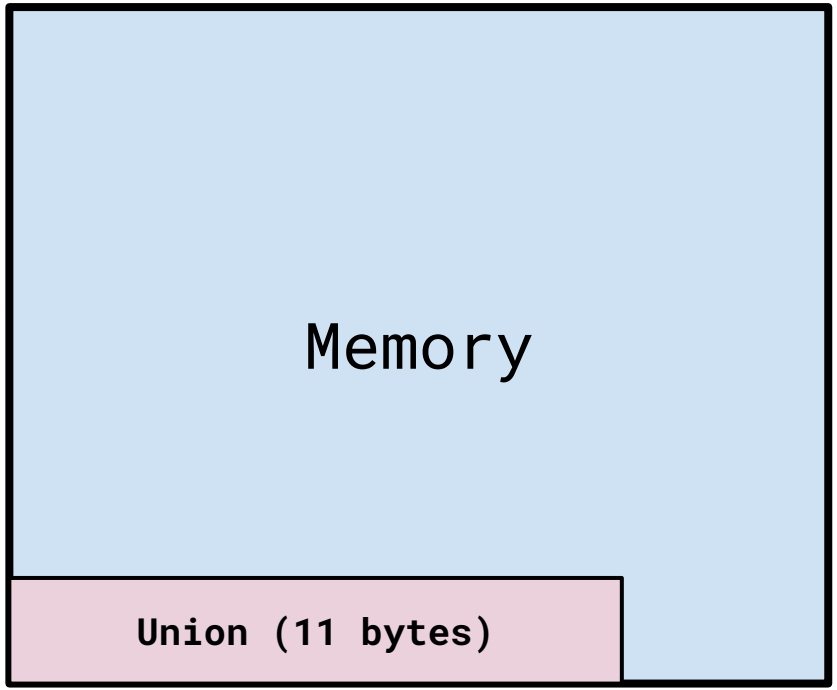
def. Las unions son un tipo de datos similares a las structs en el sentido en que agrupan varios tipos de variables dentro de sí mismas, sin embargo, hay una diferencia clave: el espacio que ocupan en memoria.

Las unions, a diferencia de las stucts, ocupan un único lugar en memoria, de tamaño igual al de la variable más grande que la misma contenga.



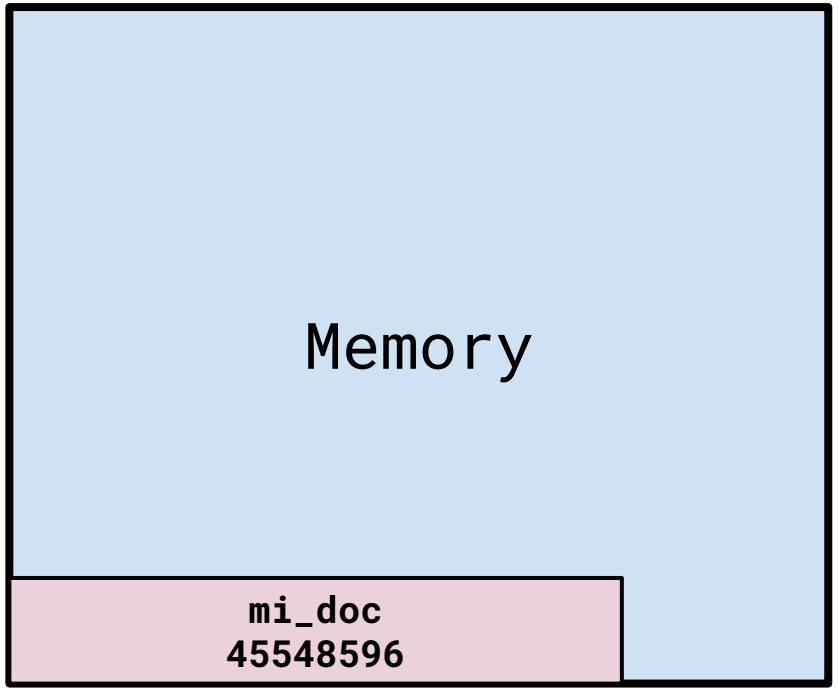
Unions (2) - example

```c
union documento{
    int dni;
    char pasaporte[11];
};

union documento mi_doc = {.dni = 45548596};
printf("%d\n", mi_doc.dni);

strcpy(mi_doc.pasaporte, "C4313AAK10");
printf("%s\n", mi_doc.pasaporte);
```



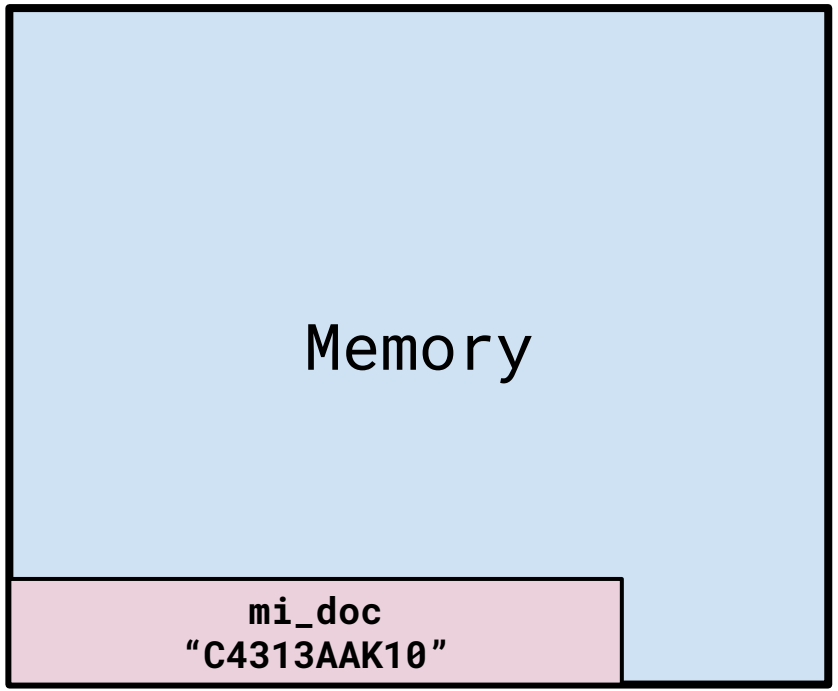
Unions (2) - example

```c
union documento{
    int dni;
    char pasaporte[11];
};

union documento mi_doc = {.dni = 45548596};
printf("%d\n", mi_doc.dni);

strcpy(mi_doc.pasaporte, "C4313AAK10");
printf("%s\n", mi_doc.pasaporte);
```



Unions (2) - example

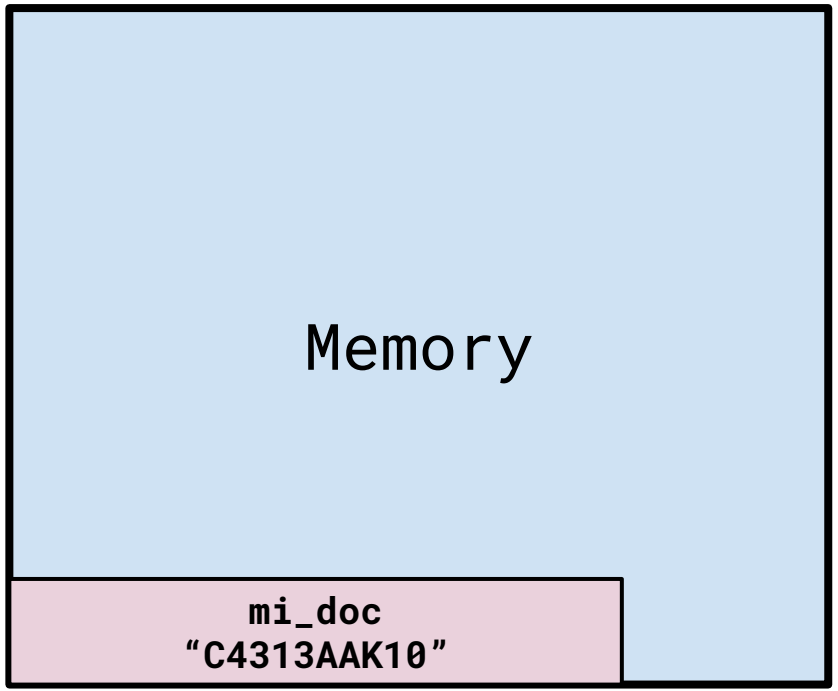
```c
union documento{
    int dni;
    char pasaporte[11];
};

union documento mi_doc = {.dni = 45548596};
printf("%d\n", mi_doc.dni);

strcpy(mi_doc.pasaporte, "C4313AAK10");
printf("%s\n", mi_doc.pasaporte);
```



Unions (3) - example

```c
union documento{
    int dni;
    char pasaporte[11];
};

union documento mi_doc = {.dni = 45548596};

strcpy(mi_doc.pasaporte, "C4313AAK10");

printf("%d\n", mi_doc.dni);
printf("%s\n", mi_doc.pasaporte);
```

???



Uso de typedef

```c
/* Uso de typedef */
typedef struct{
    char nombre[20];
    int edad;
    float altura;
}persona_t;

typedef union{
    int dni;
    char pasaporte[11];
}documento_t;

int main(void){
  documento_t mi_doc = {.dni = 45548596};
  persona_t p1;
  strcpy(p1.nombre, "p1pe");
  p1.edad = 25;
  p1.altura = 1.8;

  printf("%d - dni\n", mi_doc.dni);
  printf("%s - nombre", p1.nombre);
}
```

Utilizando *typedef* puedo crear mi propio tipo de dato customizado.

Ejemplo de uso - Manejo de Registros

```c
typedef union {
    struct {
      uint32_t ISR;
      uint32_t IFCR;
      struct {
        uint32_t CCR;
        uint32_t CNDTR;
        uint32_t CPAR;
        uint32_t CMAR;
        uint32_t RESERVED;
      } CHN[8];
    } REGs;
    page reserved;
  } ejemplo_registro_t;
```

```c
ejemplo_registro_t ejemplo;
uint32_t valor = ejemplo.REGs.CHN[6].CNDTR;
```