

Programación 2

Repaso conceptos fundamentales del lenguaje C

- **Tipos de datos**

- Primitivos: char, int, float, double,...
- Arrays multidimensionales: int t[10]; int m[2][4];
- Punteros: int *p;
- Estructuras: struct
- Creación de nuevos tipos (typedef) y constantes enumeradas (enum)

- **Operaciones básicas**

- Aritméticas: +, -, *, /, %, +monario, -monario, ++, --, ...
- Relacionales: >, <, ==, !=
- Lógicas: &&, ||, !
- Control flujo ejecución: if, else if, else, for, while, do while, switch,...

- **Reglas de precedencia y asociatividad de operadores**

- **Funciones de E/S básicas**

- printf, scanf, gets, puts, sscanf, sprintf, strlen, strcpy, strcat, strcmp,getc, putc, ficheros (fopen, fclose, fscanf, fprintf, fgets, fputs...)

Tipos de Datos: Primitivos

- Almacenan un valor

Tipo de dato	Número de bits
char	8 (1 B)
unsigned char	8 (1 B)
short	16 (2 B)
unsigned short	16 (2 B)
int	32 (4 B)
unsigned	32 (4 B)
long	32 (4 B)
unsigned long	32 (4 B)
long long	64 (8 B)
unsigned long long	64 (8 B)
float	32 (4 B)
long float	64 (8 B)
double	64 (8 B)
long double	64 (8 B)

- **Definición del prototipo**

```
[tipo_ret] nombre ([tipo1 [arg1]] [, ____]);
```

- nombre nombre de la función
 - tipo1 tipo del primer argumento (si lo hay)
 - arg1 nombre del primer argumento (no es imprescindible)
 - tipo_ret tipo del dato que se devuelve
- Si no se quiere devolver nada, el tipo de retorno ha de ser **void**
 - Si no se pone nada, el tipo de retorno es **int**
 - No hacen falta nombres de argumentos en el prototipo (comparación sintáctica de nombre función, nº y tipo de argumentos y tipo retorno).

- **Implementación (cuerpo de la función**

```
[tipo_ret] nombre ([tipo1 [arg1]] [, ____]){  
  
    ...  
  
}
```

Ejemplo: función suma 2 enteros y devuelve el resultado

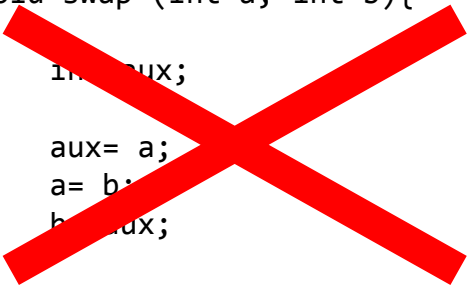
- **En C, paso de argumentos por valor**
- **Fases al llamar a una función en C:**
 - Se reserva memoria para los argumentos y variables locales (AdD de la función)
 - Se copian los valores de los argumentos a su espacio dentro del AdD de la función
 - Se ejecuta el código (usando el AdD de la función)
 - Retorno: se copian los valores devueltos en las variables receptoras
 - Se libera el AdD de la función

Funciones en C: llamadas por valor

- ¿Cómo se implementa una función swap en C?

prototipo: `void swap (int a, int b);`

```
void swap (int a, int b){  
    int aux;  
  
    aux= a;  
    a= b;  
    b= aux;  
}
```



```
void swap (int *a, int *b){  
  
    int aux;  
  
    aux= *a;  
    *a= *b;  
    *b= aux;  
}
```

Llamada desde el main?

- Solución: punteros

- **¿Qué es un puntero?**
 - Coloquial: Un puntero es una dirección de memoria
 - Definición: Variable que contiene una dirección de memoria de un dato de cierto tipo determinado

```
int i= 1;
int *pi;

pi= &i;
(*pi)++;
printf ("%d",i);
printf ("%d", *pi);
```

- **Operadores sobre punteros:**
 - `&` dirección de memoria del dato al que apunta el puntero
 - `*` contenido del dato al que apunta el puntero
- `&var`: dirección 1er byte memoria donde se guarda la variable `var` como un dato de un cierto tipo
- Ojo: `*` tiene 2 usos con punteros:
 - Declaración de punteros. `int *pi`; declara un puntero a un integer
 - Operador de contenido. `*pi = 3`; asocia el valor 3 a la variable almacenada a partir de la dirección a que apunta `pi` (como entero)

• Ejemplos

```
int a, *pa;  
a= 3;  
pa= &a;  
*pa= 4;  
a= 5;  
printf("%d", *pa);
```

- a) 5
- b) 4
- c) 3
- d) No es posible saberlo

```
int a, *pa , **ppa;  
a= 3;  
ppa= &pa;  
*ppa= &a;  
*pa= 4;  
printf("%d", a);
```

- a) 5
- b) 4
- c) 3
- d) No es posible saberlo o hay error

```
int a, *pa , **ppa;  
a= 3;  
ppa= &pa;  
*ppa= a;  
*pa= 4;  
printf("%d", a);
```

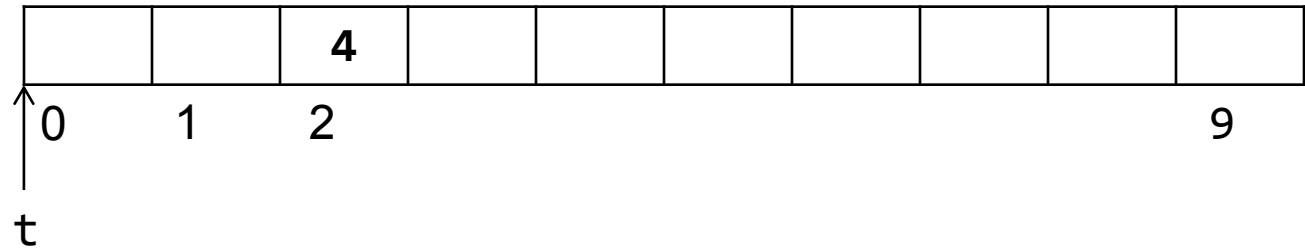
- a) 5
- b) 4
- c) 3
- d) No es posible saberlo o hay error

Tipos de Datos: Tablas (arrays)

9

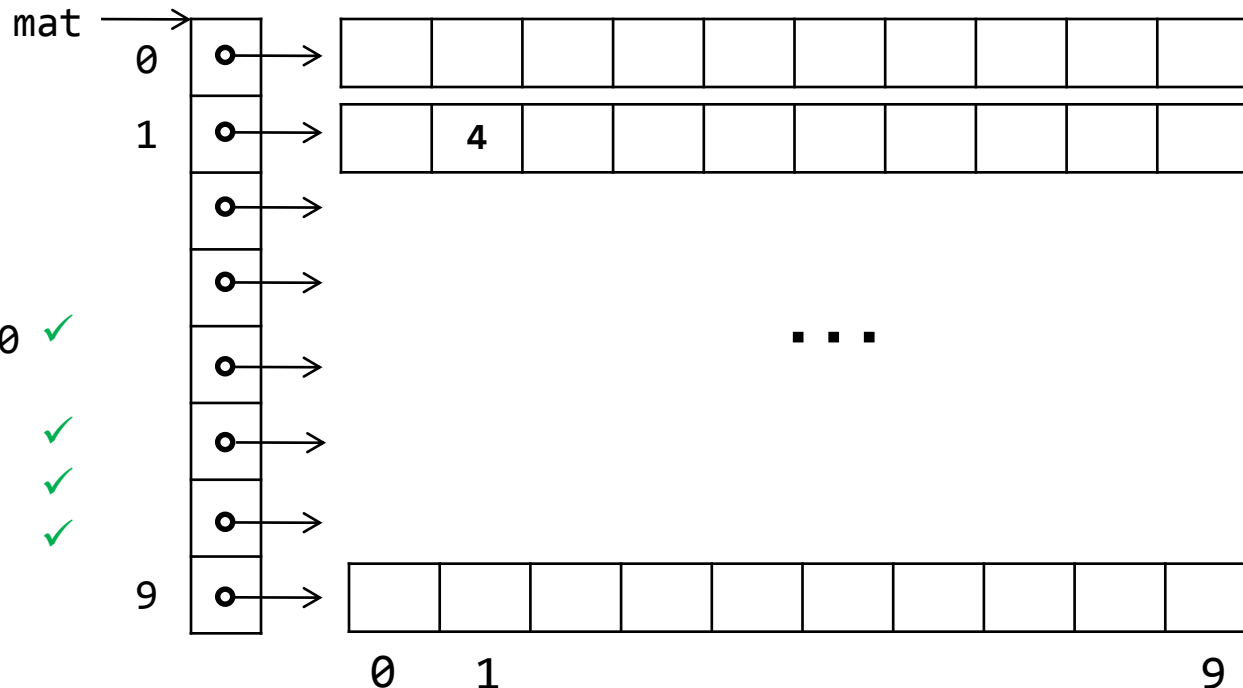
- Tabla de una dimensión

```
int t[10];  
t[2]= 4;
```



- Matrices: tablas de varias dimensiones

```
int mat[10][10];  
mat[1][1]= 4;
```



¿Correcto?

```
#define DIM 10 ✓
```

...

```
int t1 [10]; ✓
```

```
int t2 [DIM]; ✓
```

```
int N = 3; ✓
```

```
int t3 [N];
```

Tipos de Datos: Tablas

- En C, el nombre de una tabla se corresponde con la dirección del primer elemento de la tabla.

- **Punteros y tablas**

- Esencial que un puntero se defina asociado a un tipo de dato!

- Asignación del dato
- Operaciones con tablas

```
int n[10];
double d[10];
int *p;
double *pd;

//Asignaciones correctas??
p= n;
n= p;

//Avanzar y acceso a datos
*(p + 1)= 2;
p[1]= 3;

pd= d;
*(pd + 3) = 2.5;
```

Obsevaciones:

- &n

- Aritmética de punteros: sumar es avanzar el puntero

* (p+1) == n[1] → * (p+i) == n[i]

*p == n[0] → 1er elemento tabla (contenido)

- d = 3;

- *p = 3;

- d[2] == pd [2]?

- Operador sizeof: sizeof(char) --> nº bytes necesarios para almacenar un carácter.

Tipos de Datos: Estructuras (I)

- **Ejemplo: Definición de una variable que sigue la estructura para un número complejo**
 - Tres formas diferentes

1) “Implícita” (v1)
si solo 1 variable
de ese tipo

```
struct{    //sin etiqueta
    float re, im;
} c1;    // nombre de variable
```

2) “Implícita” (v2)

```
struct _COMPLEJO{
    float re, im;
};

struct _COMPLEJO c1; //variable
```

3) “Explícita”

```
struct _COMPLEJO{
    float re, im;
};
typedef struct _COMPLEJO complejo;

complejo c1; //variable
```



```
typedef struct {
    float re, im;
} complejo;

complejo c1; //variable
```

Tipos de Datos: Estructuras(II)

• Asignación en estructuras. Quiero esto:

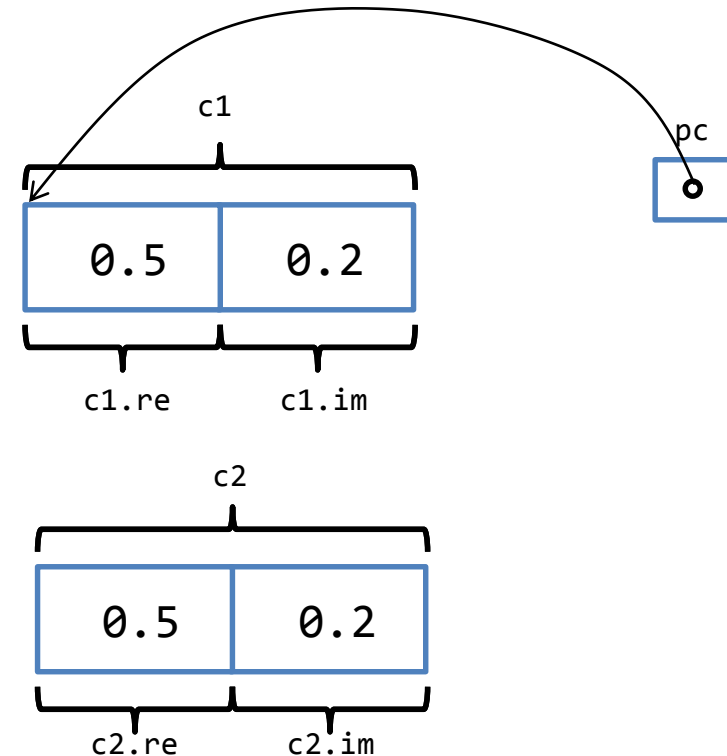
*SI TUVIERA TODO EN 1 MISMO FICHERO
(SIN .H NI VARIOS .C):*

```
typedef struct {
    float re, im;
} complejo;
```

```
complejo c1, c2;
complejo *pc;
pc = &c1;
```

```
c1.re = 0.5; // pc->re = 0.5;
c1.im = 0.2; // pc->im = 0.2;
c2 = c1;
```

```
// Obs: (*pc).re == pc->re
```



• Arrays

- Al igual que un tipo de dato primitivo

```
complejo vector[10];
```

• Entrada/Salida

- Debe ser miembro a miembro

```
scanf("%f %f", &(c1.re), &(c1.im));
```

Tipos de Datos: Enumeraciones

- **Definición por omisión**

- `typedef enum {TRUE, FALSE} BOOL;`

- **Definición con valor inicial**

- `typedef enum {TERCETO=3, CUARTETO, QUINTETO} GRUPO;`

- **Definición con asignación de valores**

- `typedef enum {TRUE=1, FALSE=0 } BOOL;`

- `//typedef enum {FALSE, TRUE} BOOL;`

- **Uso**

```

BOOL b;
GRUPO g;
b= FALSE;
g= CUARTETO; // g=4;
...
if (b == FALSE){
    ...
}
if (g == TERCETO) {
    ...
}


```

```

GRUPO grupos [10];
grupos[0] =QUINTETO;
grupos[1]=TERCETO;
...
for (i=0; i<10; i++)
    if (grupos[i] == TERCETO) {
        ...
    }
}

```

precedencia

	Operador	Asociatividad
	() [] . ->	izda-dcha
	* / %	izda-dcha
	+ -	
	&&	
	=	dcha-izda

- **Asociatividad**

- A igualdad de precedencia se aplica la asociatividad

- **Ejemplos**

```
int a;  
a= 5 + 3 * 4 % 5 - 1;  
a= (5 + ((3 * 4) % 5)) - 1 => 6
```

```
int a, b;  
b= a= (5 + 3) * 4 / 5 || 1;  
b= a= (((5 + 3) * 4) / 5) || 1 => 1
```

6

Organización del código en C

- **Opción monoarchivo: todo en un fichero .c**

1. **Órdenes para el preprocesador: #include , #define**

Incluir bibliotecas y definir constantes, tipos, macros,...

2. **Definición de nuevos tipos de datos: struct, typedef**

3. **Prototipos de las funciones (declaración)**

4. **Variables globales**

5. **Main. Opciones:**

```
[void] main ( ){  
    } // sin argumentos
```

```
[void] main (int argc, char * argv []) {  
    } //programa recibe argumentos en línea comandos
```

6. **Cuerpo de las funciones (definición)**

Organización del código en C

- **Opción multiarchivo: varios ficheros**

- Biblioteca:

- .h : definición de **nuevos tipos de datos (TADs)** y **prototipos** de sus primitivas y funciones específicas.
- .c: definición de las **estructuras de datos (EdD)** y **cuerpo** de las primitivas y funciones basándose en esas EdD.

- Fichero principal (main)

- .c: función **main**
(se debe incluir arriba el .h de la biblioteca y utilizar únicamente lo que se encuentre en ese .h – solo TAD, no EdD)

- Obs:

- en cada fichero .c se ponen los #includes necesarios

- Compilación y construcción del ejecutable:

- compilar cada .c por separado, dentro de un mismo proyecto
- enlazarlos (build) para obtener el ejecutable

Variables y su alcance

- **Alcance de una variable:** parte del código donde la variable está accesible (puede ser utilizada, es “visible”).
- Variables **globales**:
 - Visibles desde su declaración hasta el final del archivo en que se definen
 - Se declaran fuera de cualquier bloque o función (fuera de toda { })
 - Son “permanentes”: se reserva espacio para ellas en el segmento de datos, no en la pila de ejecución
- Variables **locales** (= automáticas):
 - Visibles dentro del bloque o función donde se declaran
 - Se reserva espacio para ellas en la pila (AdD de la función)
 - Al salir del bloque o función desaparecen
- Variables **estáticas**:
 - Variables locales “permanentes”, se almacenan en el segmento de datos
 - Se definen dentro de un bloque o función (añadiendo static)
 - Son capaces de guardar su valor de una llamada a la siguiente

Reserva de memoria dinámica

- **Heap**

- **Reservar memoria: malloc**

- Prototipo: `void * malloc (size_t n);`
 - `size_t`: tipo devuelto por `sizeof` (`unsigned int`)
 - `void *` => puntero “genérico”, dirección comienzo memoria (sin tipo específico). Devuelve `NULL` si error.

- Ej uso:

```
int * p;  
p = (int *) malloc (10 * sizeof (int));
```

- **Liberar memoria con free**

- Prototipo: `free (void *);`
- Uso: `free (p);` // `free ((void*) p);`
casting

Reserva de Memoria

- **Realojar memoria**

- `void * realloc (void *p, size_t n)`
 - Reserva memoria dinámica para n bytes
 - Copia a la nueva zona min (n, nº de bytes anteriormente reservados)
 - Devuelve el puntero a la nueva zona o NULL si error

- Ej uso:

```
int * p;  
p = (int *) malloc (10 * sizeof(int));  
(if p == NULL) {                // if (!p)    // error  
    ...  
}  
p = (int *) realloc (p, 20 * sizeof(int));  
... //idem con el error  
free (p);    //solo 1 vez!
```

- Array de punteros a enteros.
- Matriz.
 - Reservar memoria dinámicamente para matriz de $x * y$
 - Liberar memoria

- Escribir el código necesario para pedir al usuario que introduzca un n° x y reservar memoria para una tabla de x enteros.
 - Escribir una versión 2 con control de errores.
- Escribir el código necesario para pedir al usuario que introduzca dos números *filas* y *columnas* y reservar memoria para una matriz de dimensión *filas* x *columnas*.
 - Escribir una versión 2 con control de errores.
- Escribir el código necesario para liberar la memoria reservada en los 2 puntos anteriores.

- **Escribir una función main en la que:**
 - Se declare una tabla de NMAX=10 números de precisión doble inicializados a 0.
 - Se llame a una función auxiliar que lea los números de teclado.
 - Se llame a otra función que calcule el máximo, el mínimo y la media de los números leídos con la función anterior.
 - Imprima los resultados de la función anterior por pantalla
- **Escribir el cuerpo de las funciones, cuyos prototipos son:**

```
int leerDatos (double *array); //devuelve entero, para tratar errores  
void calcularEstadisticas (const double *tabla, double *minimo, double *maximo, double *media);
```