PROGRAMACIÓN 1

Grado en Ingeniería Informática e I2ADE

Tema 2

Tipos de datos simples

Dept. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial

Universitat d'Alacant
Universidad de Alicante

Índice

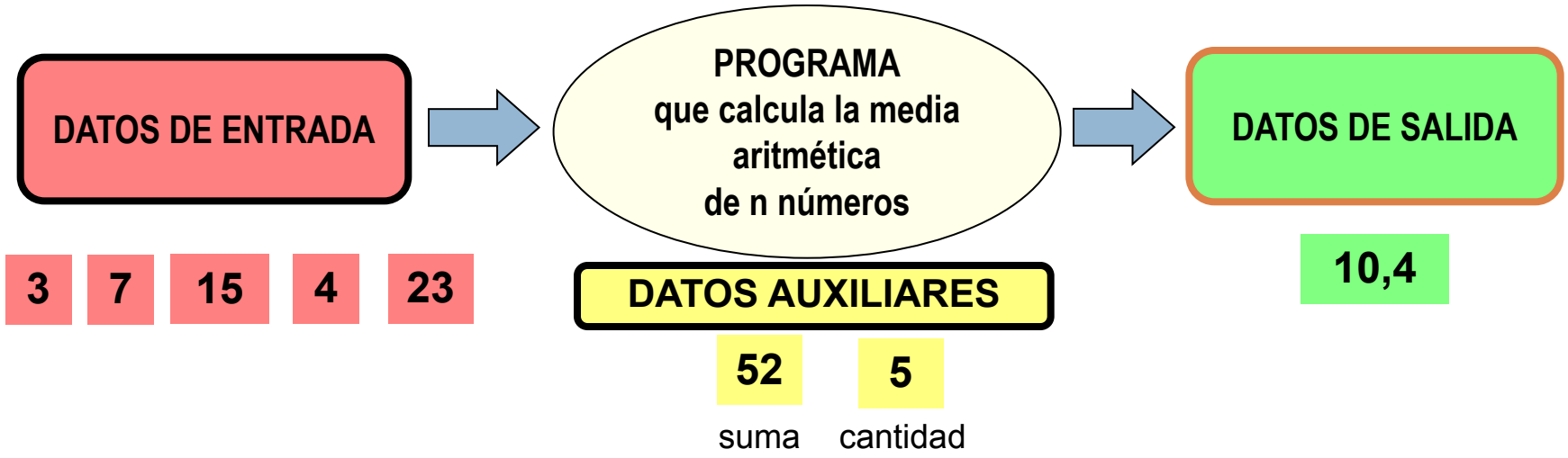
1. Tipos de datos

- **Dato**: Hecho o valor a partir del cual se puede inferir una conclusión (información).
- **Datos en un programa informático**: Datos con los que opera una computadora.
  - Los **datos de entrada** constituyen un punto de partida para obtener conocimiento y producir **datos de salida**.
  - El programa también puede necesitar **datos internos auxiliares** para producir los datos de salida.

1. Tipos de datos

Ejemplo de datos manejados en un programa

- Programa: Calcular la media aritmética de *n* números
- Datos de entrada: *n* números cualesquiera
- Datos de salida: La media aritmética de los *n* números
- Datos internos auxiliares: La suma de los números y la cantidad de ellos



1. Tipos de datos

Tipo de dato

Un tipo de dato viene definido por:

- El conjunto de posibles **valores** que puede tomar (dominio) en el programa. Si se le intenta dar un valor fuera de ese conjunto de valores puede producirse un error.
- El conjunto de **operaciones** que se pueden realizar sobre los posibles valores que puede tomar.

1. Tipos de datos

Ejemplo:

- Tipo de dato **booleano**
  - Valores = { `true`, `false` }
  - Operaciones = { `and`, `or`, `not` }

Valores		Resultado de las operaciones		
`a`	`b`	`not a`	`a and b`	`a or b`
`false`	`false`	`true`	`false`	`false`
`true`	`false`	`false`	`false`	`true`
`false`	`true`		`false`	`true`
`true`	`true`		`true`	`true`

1. Tipos de datos

Tipos de datos simples

- Son tipos de datos **elementales** que no se derivan de otros tipos de datos.
- Cada valor concreto del tipo de dato simple viene especificado por un literal. Por ejemplo, los literales enteros pueden expresarse de las siguientes maneras:
  - En decimal (base 10) : `255`
  - En binario (base 2) : `0b101` ($1*2^2+0*2^1+1*2^0=5$)
  - En octal (base 8) : `0377` ($3*8^2 + 7*8^1 + 7*8^0 = 255$)
  - En hexadecimal (base 16): `0xff` ($15*16^1 + 15*16^0 = 255$)

1. Tipos de datos

Tipos de datos simples en C

		Tipo	Significado	Bytes
carácter		`char`	carácter	1
		`unsigned char`	carácter sin signo	1
numérico	entero	`int`	entero	2-4
		`short`	entero corto	2
		`long`	entero largo	4
		`long long`	entero largo	8
		`unsigned`	entero sin signo	2-4
		`unsigned short`	entero corto sin signo	2
		`unsigned long`	entero largo sin signo	4
		`unsigned long long`	entero largo sin signo	8
	real	`float`	coma flotante (número real)	4
		`double`	coma flotante de doble precisión	8
		`long double`	coma flotante de doble precisión extendida	16
booleano		`bool`	booleano (hay que usar la librería `stdbool.h`)	1

En C, el tipo **`bool`** se puede emular con el tipo **`int`** (cero es valor false, y distinto de cero es valor true)

1. Tipos de datos

Valores de tipos de datos simples en C

Tipo	Bytes	Valores	Precisión
`char`	1	Alfabéticos: ‘a’, ‘b’,…’z’ ‘A’, ‘B’, …’Z’ Dígitos: ‘0’, ‘1’, ‘2’, .. ‘9’ Especiales: ‘+’, ‘-’, ‘/’, ‘=‘, ‘(‘, …	
`short`	2	`-32.767… 32.767`	
`int`	4	`-2.147.483.647… 2.147.483.647`	
`float`	4	Aprox. 10^{-38}… 10^{38}	7 dígitos
`double`	8	Aprox. 10^{-308}… 10^{308}	15 dígitos
`bool`	1	`true`, `false`	

1. Tipos de datos

Tipos de datos enumerados

- Generalmente los lenguajes de programación tienen tipos de datos predefinidos y además posibilitan al usuario definir sus propios tipos de datos.
- En el lenguaje C:
  - El usuario puede definir tipos de datos enumerados compuestos por un conjunto de identificadores que representan un valor entero.
  - No hay formato de impresión para estos tipos. El primer elemento tiene asociado el valor `0`, el segundo el valor `1` y así sucesivamente.

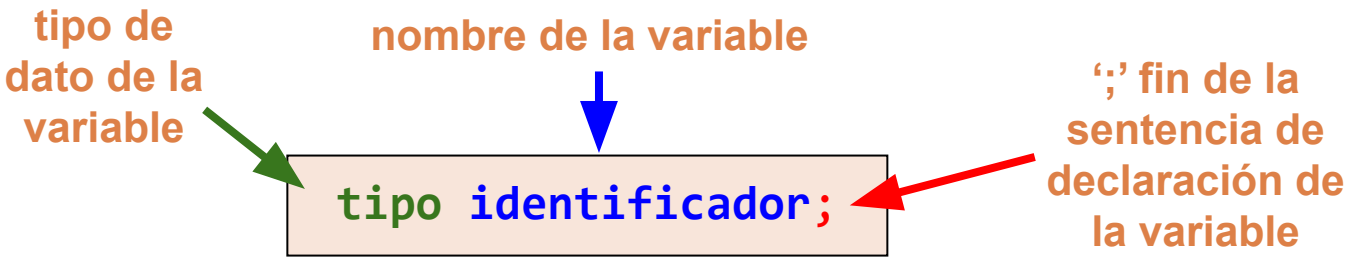
```
enum T_DiaSemana {lunes, martes, miercoles, jueves, viernes,
                  sabado, domingo};

enum T_ColorPrimario {rojo, verde, azul};
```

2. Variables y constantes

- Características comunes:
  - Permiten representar datos en un programa
  - Constituyen un espacio de memoria reservado para almacenar un valor de un tipo de dato
  - Se identifican con un nombre
- Se diferencian en:
  - El valor de una **variable puede cambiar** a lo largo del programa
  - El valor de una **constante nunca cambia** en el programa

Declaración de variables en C

Hay que asociar un tipo de dato a la variable para que en ésta se pueda almacenar cualquier valor de ese tipo de dato.



Existe la posibilidad de inicializar (definir su valor) la variable en la propia declaración:

```
tipo identificador = valor;
```

Ejemplos:

```
char letra_dni;    // variable para almacenar la letra
                   // del dni de cualquier persona.
int paginas = 20;  // variable para almacenar el nº de páginas
                   // de cualquier libro. Se inicializa a 20.
```

2. Variables y constantes

Declaración de constantes en C

Existen dos formas de declarar constantes en C:

- Como **macro** a nivel de preprocesador

```
#define identificador valor
```

palabra reservada de C

nombre de la constante

valor de la constante

Características:

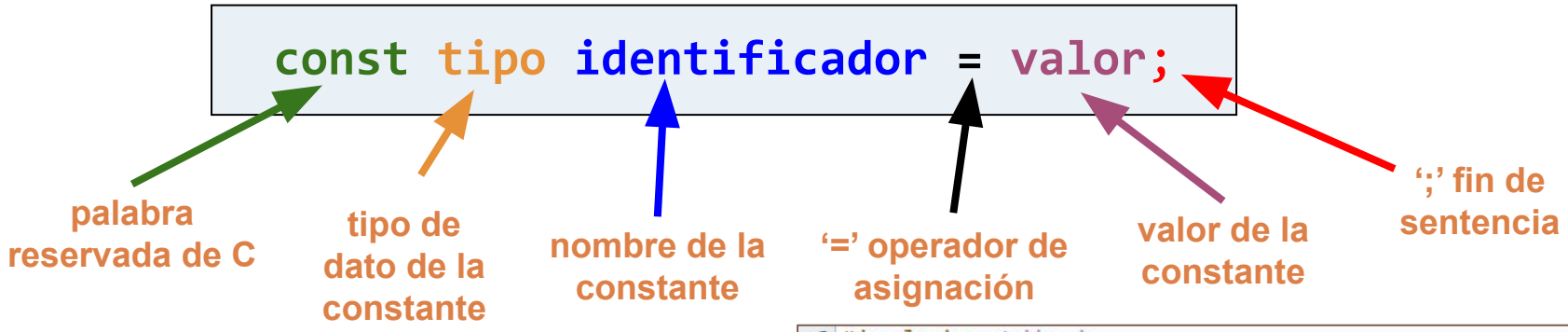
- No tiene tipo de dato.
- El preprocesador busca en el código fuente la cadena `identificador` y la sustituye por la cadena `valor`.
- Permite argumentos en la expresión **`identificador`**.

```
1  #include<stdio.h>
2
3  #define PI 3.141592
4
5  #define incrementar(x) x++
6
7  int main(){
8      int num;
9
10     num = 5;
11     printf("num: %d\n", num);
12
13     incrementar(num);
14     printf("num: %d\n", num);
15
16     return 0;
17 }
```

2. Variables y constantes

Declaración de constantes en C

- Como **variable constante** a nivel de compilador



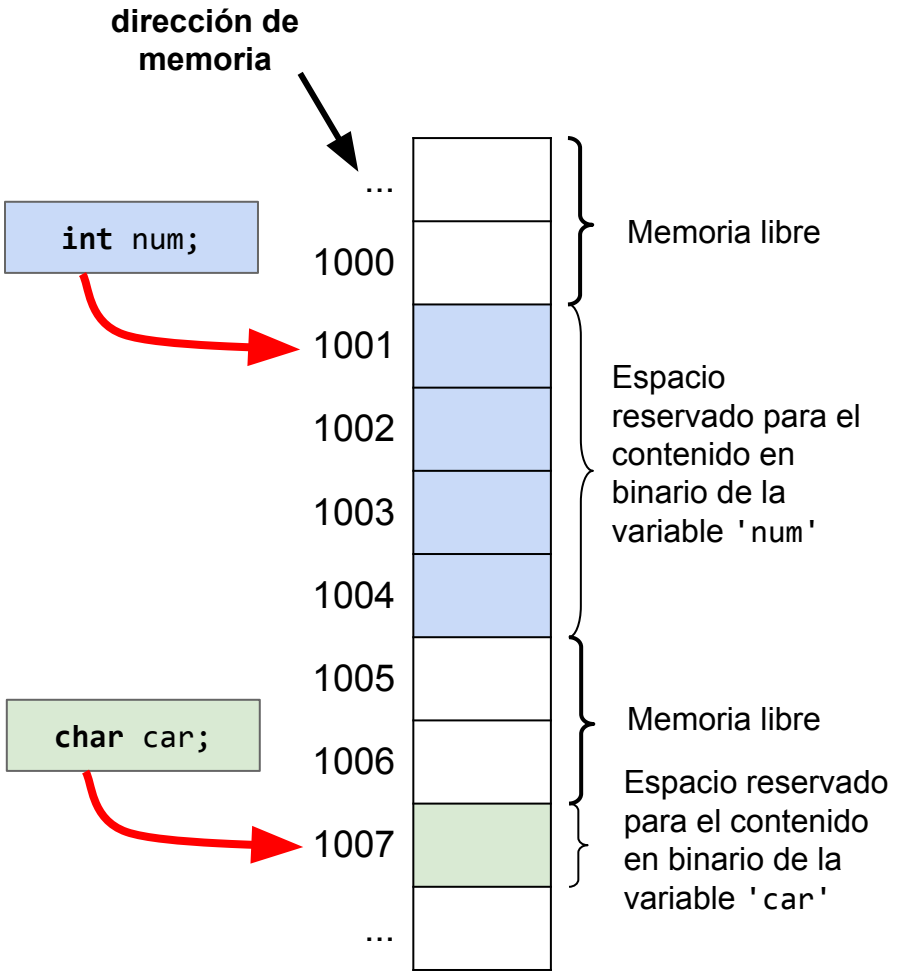
Características:

- Al declararse con tipo de dato se puede acceder a la zona de memoria en la que se encuentra.
- Se puede asignar su valor a variables con tipo de dato compatible.

```c
#include<stdio.h>

const float PI = 3.141592;

int main(){
    float area, radio;

    printf("Dime el radio del círculo: ");
    scanf("%f", &radio);

    area = PI * (radio * radio);

    printf("El área del círculo es: %f\n", area);

    return 0;
}
```

Representación en memoria de las variables

- La memoria consiste en una lista de posiciones numeradas (bytes)
- Una variable representa una porción de memoria compuesta por un grupo de bytes consecutivos
- Una variable en memoria viene determinada por:
  - La dirección en la memoria, que proporciona la ubicación del primer byte dedicado a esa variable
  - El tipo de dato, que determina cuántos bytes de memoria requiere la variable



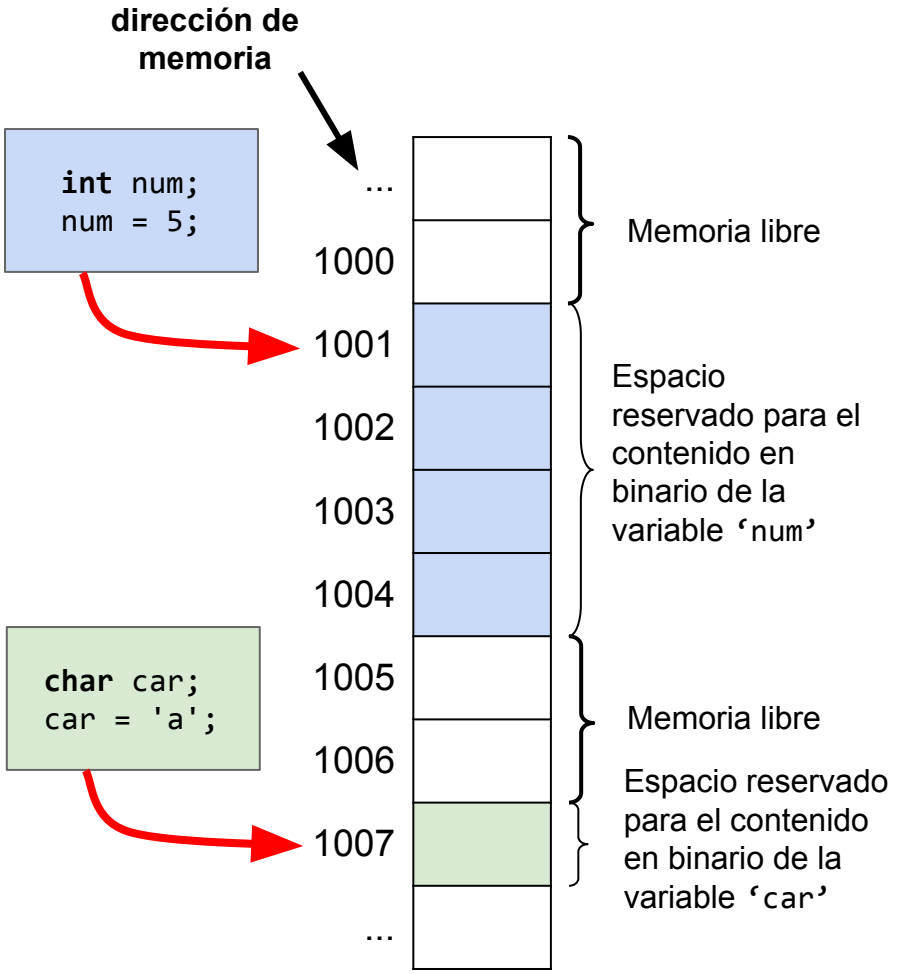
Representación en memoria de las variables

Punteros en C

- Permiten acceder directamente a posiciones de memoria
- Para una variable permiten diferenciar entre su posición en memoria (**dirección**) y su valor (**contenido**)
- Operador **dirección**: **&**
- Operador **contenido**: *

```
num       ⇒  5
&num      ⇒  1001
*(&num)   ⇒  5
```

```
car       ⇒  'a'
&car      ⇒  1007
*(&car)   ⇒  'a'
```



Representación en memoria de las variables

Punteros en C

Ejemplo:

```
#include<stdio.h>
int main(){
  int num; // variable num
  int *pNum; // puntero a dirección de memoria
  num = 5; // se asigna el valor 5 a num
  pNum = &num; // dirección de memoria de num
  // Las dos sentencias siguientes
  // imprimen el valor de num
  printf("num: %d\n", num);
  printf("*pNum: %d\n", *pNum);
  // Las dos sentencias siguientes
  // imprimen la dirección de memoria de num
  printf("pNum: %p\n", pNum);
  printf("&num: %p\n", &num);
  return 0;
}
```

Declaración de una variable de tipo puntero. Hay que utilizar el *.

Resultado:

```
$ ./ejemplo

num: 5
*pNum: 5
pNum: 0x7fff458af05c
&num: 0x7fff458af05c
```

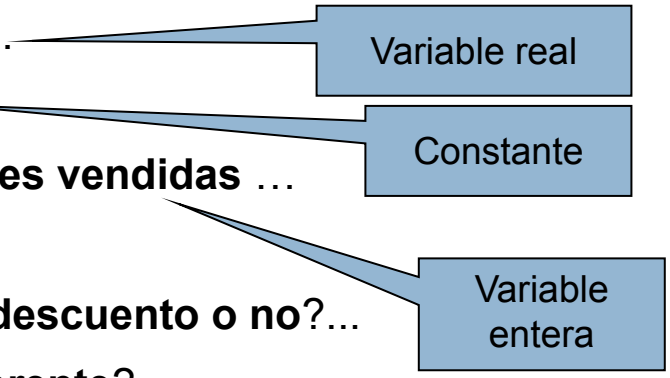
2. Variables y constantes

Ejemplo de uso de variables y constantes

Un club de futbol X propietario de un estadio Y, necesita calcular la recaudación de cada uno de los partidos disputados en su estadio, teniendo en cuenta que pone a la venta tres tipos de entrada dependiendo del lugar en la grada del asiento seleccionado por el aficionado: entrada de fondo (gradas de detrás de las porterías), entrada general (gradas laterales sin techo) y entrada preferente (gradas laterales con techo). Durante toda la temporada, el precio de una entrada de fondo es la mitad que el de una entrada general y el precio de una entrada preferente es el doble de una entrada general. En cada partido, el club de fútbol fija un precio para la entrada general y además establece unos descuentos para toda la temporada para los niños (el 80%) y para los pensionistas (el 50%).

Para calcular la recaudación de cada partido de la temporada …

- Para almacenar el **precio de la entrada general** …
- Para almacenar el **descuento para los niños** …
- Para almacenar el **número de entradas preferentes vendidas** …
- ¿y para almacenar el **tipo de entrada** vendida?...
- ¿y para almacenar si a un aficionado **se le aplica descuento o no**?...
- ¿y para almacenar el **precio de una entrada preferente**?...



2. Variables y constantes

Identificadores para variables y constantes

- Notaciones más utilizadas por la mayoría de programadores:
  - Las variables en minúsculas y las constantes en mayúsculas
  - Con identificadores compuestos por varias palabras:
    - ✓ Todo en minúsculas excepto las iniciales de cada palabra adicional (camelCase)

      `nombreAlumno`
    - ✓ Todo en minúsculas, separando las palabras con el carácter subrayado

      `nombre_alumno`
    - ✓ Todo en mayúsculas, separando las palabras con el carácter subrayado

      `NOMBRE_ALUMNO`
    - ✓ Utilización de abreviaturas con la misma longitud

      `nom_alu`

Nota: Es muy importante **no cambiar arbitrariamente de notación** y seguir sólo una de ellas para mantener una coherencia en nuestros programas y facilitar la legibilidad y la comprensión de los mismos

2. Variables y constantes

Identificadores en un programa

- Un identificador es un nombre que utiliza el programador para referenciar los datos y otros elementos del programa
- Reglas generales de construcción de identificadores:
  1. Debe resultar significativo
  2. No puede coincidir con palabras reservadas propias del lenguaje de programación
  3. La longitud no debe ser excesivamente larga
  4. Deben comenzar por una letra o el símbolo de subrayado y pueden contener letras, dígitos y el símbolo de subrayado
  5. No se acentúan
  6. Según el lenguaje de programación, el identificador se podrá utilizar, o no, indistintamente en mayúsculas o en minúsculas

Nota: El lenguaje C es **sensible** a mayúsculas y minúsculas

2. Variables y constantes

Identificadores en un programa

- Identificadores **correctos**:
  - `distancia`
  - `distancia_euclidea`
  - `_fecha`
  - `fechaNacimiento`
  - `NUMERO_PI`
  - `numero1`
  - `numero_2`

- Identificadores **incorrectos**:
  - `distancia-euclidea`
  - `3libros`
  - `Numero$1`
  - `Mas_preguntas?`
  - `número`

Nota: Los siguientes **identificadores** son distintos en lenguaje C:

`Color_coche` `color_coche` `COLOR_COCHE` `color_Coche` `Color_Coche`

3. Expresiones aritméticas y lógicas

- Una expresión en un programa es una combinación de variables, constantes, operadores, paréntesis e identificadores de funciones, de cuya evaluación se obtiene un valor.
  - Las expresiones se pueden escribir en cualquier lugar del programa en donde pueda utilizarse el valor que devuelven
- Una expresión aritmética …
  - se construye con operadores aritméticos
  - devuelve un valor numérico

```
(x_rad * 360) / (2 * PI)
```

calcula los grados correspondientes al valor en radianes almacenado en la variable x_rad, utilizando la constante PI

- Una expresión lógica …
  - se construye con operadores relacionales y lógicos
  - pueden aparecer operadores aritméticos
  - devuelve un valor booleano

```
(año modulo 4 == 0) AND (NOT (año modulo 100 == 0) OR (año modulo 400 == 0) )
```

4. Operadores en C

operadores aritméticos	significado	tipos de operandos	tipo de resultado
+ - * /	suma, resta, multiplicación, división	numéricos enteros o reales	numérico entero o real
%	resto de división	enteros	entero
operadores relacionales			
< > <= >=	menor que, mayor que, menor o igual que, mayor o igual que	tipos simples	booleano
== !=	igual que, distinto de	tipos simples	booleano
operadores lógicos			
&&	AND lógico	booleano	booleano
||	OR lógico	booleano	booleano
!	NOT lógico	booleano	booleano

Notas:

- En el lenguaje C el **operador de asignación** '=' y el operador relacional de **igualdad** '==' son distintos. Es habitual confundirlos dando lugar a errores difíciles de detectar.
- Con el **operador de división** '**/**', cuando los operandos son de tipo numérico entero, el resultado es la parte entera del cociente. Para obtener un resultado con decimales, alguno de los operandos debe ser de tipo numérico real.

4. Operadores en C

Precedencia y asociatividad de operadores

- La **precedencia** o **prioridad** de un operador indica el orden en que se ejecutan las operaciones en una expresión que contiene distintos operadores.
- La **asociatividad** de un operador indica el orden en que se ejecutan las operaciones en una expresión que contiene operadores con la misma prioridad.

operadores	significado	asociatividad
- !	signo negativo de un número, NOT lógico	de derecha a izquierda
* / %	multiplicación, división, resto	de izquierda a derecha
+ -	suma, resta	de izquierda a derecha
< > <= >=	comparación	de izquierda a derecha
== !=	igualdad/desigualdad	de izquierda a derecha
&&	AND lógico	de izquierda a derecha
||	OR lógico	de izquierda a derecha

Se recomienda el uso de **paréntesis**
- cuando tengamos alguna duda en el orden de evaluación
- para hacer más legible la expresión
- para modificar el orden de evaluación

5. Sentencias de Entrada y Salida de datos

- Las variables pueden utilizarse en sentencias de entrada
- Las variables, constantes y, en general, las expresiones pueden utilizarse en sentencias de salida
- Las sentencias de entrada permiten almacenar en variables datos que el usuario introduce por teclado
- Las sentencias de salida permiten visualizar datos en la pantalla

Nota: La entrada y salida puede estar asociada a distintas **fuentes y dispositivos**, tales como ficheros, impresoras, pantallas táctiles, ratón, etc. En esta asignatura, nosotros sólo usaremos en nuestros programas el **teclado y la pantalla** que suelen ser los dispositivos de entrada y salida por defecto.

Sentencia de salida en C: `printf`

Permite escribir en pantalla cualquier combinación de valores de variables, constantes, expresiones y cadenas de texto

```
printf(const char *formato [,argumentos]);
```

- ***formato***. Cadena de texto a escribir. Opcionalmente puede contener marcas o *flags*, que comienzan por **%** y permiten interpolar valores que se indican en *argumentos*.
- ***argumentos***. Lista de valores a interpolar en la cadena de texto *formato* en el lugar en el que se encuentren las marcas correspondientes y en el orden en que se indican.

Ejemplo:

```
int radio;
float area;
...
printf("El área del círculo de radio %d es %f\n", radio, area);
```

Como se puede ver, **el orden de las marcas** en la cadena de formato **se corresponde con el orden de los datos en la lista de argumentos** a imprimir

5. Sentencias de Entrada y Salida de datos

Sentencia de salida en C: `printf`

Nota: Los valores entre corchetes son optativos

Formato de las marcas:

```
%[parámetro][flags][ancho][.precisión][longitud]tipo
```

parámetro	Descripción
n$	Permite cambiar el orden en que se procesan los argumentos. Por ejemplo, la marca `%3$d` haría referencia al tercer valor de la lista de argumentos, independientemente del orden en el que aparezca la marca en la cadena *`formato`*.

flags	Descripción
número	Rellena con espacios a la izquierda hasta el valor del número
`0`	Rellena con ceros a la izquierda hasta el valor indicado por **`[ancho]`**
+	Imprime el signo del número
-	Alinea el número a la izquierda (por defecto está alineado a la derecha)
#	Para números reales se imprime la coma y se añaden ceros al final. Para números que no están en base 10, se añade el prefijo de la base.

Sentencia de salida en C: `printf`

Formato de las marcas:

```
%[parámetro][flags][ancho][.precisión][longitud]tipo
```

ancho	Descripción
número	Ancho a utilizar para imprimir el valor
*	Permite especificar el ancho en la lista de argumentos, justo antes del valor. Por ejemplo, `printf("%*d",5,10)` imprime el número 10 en 5 caracteres de ancho, rellenando con 3 espacios en blanco a la izquierda del 10.

precisión	Descripción
número	Número de cifras a mostrar en la parte decimal del número, o número de caracteres a imprimir para cadenas de texto.
*	Permite especificar el número de cifras decimales o el número de caracteres en la lista de argumentos, justo antes del valor. Por ejemplo, `printf("%.*s",2,"hola")` imprime “he”.

Sentencia de salida en C: `printf`

Formato de las marcas:

```
%[parámetro][flags][ancho][.precisión][longitud]tipo
```

longitud	Descripción
hh	Convierte una variable de tipo `char` a `int`
h	Convierte una variable de tipo `short` a `int`
l	Para enteros, se espera una variable de tipo `long`
ll	Para enteros, se espera una variable de tipo `long long`
L	Para reales, se espera una variable de tipo `long double`
z	Para enteros, se espera una variable de tipo `size_t`

Sentencia de salida en C: `printf`

Formato de las marcas:

```
%[parámetro][flags][ancho][.precisión][longitud]tipo
```

tipo	Descripción
c	Imprime el carácter ASCII correspondiente
d, i	Imprime como entero decimal con signo
p	Imprime direcciones de memoria (punteros)
e, E	Imprime números en coma flotante con signo, en notación científica
f, F	Imprime números en coma flotante con signo, usando punto decimal
g, G	Imprime números en coma flotante, usando la notación más corta
x, o	Imprime números enteros en formato hexadecimal (x) u octal (o) sin signo
u	Imprime números enteros en formato decimal sin signo
s	Imprime cadenas de caracteres acabadas en el carácter fin de cadena (`'\0'`)

Sentencia de entrada en C: `scanf`

Permite leer desde teclado uno o más valores del mismo o distinto tipo de dato de una sola vez.

```
scanf(const char *formato, argumentos);
```

- ***formato***. Cadena de texto con las marcas o *flags* (las marcas `tipo` explicadas para `printf`), que comienzan por **%** y permiten especificar el orden de las variables que se indican en *`argumentos`* y en las que se guardarán los valores leídos. Nota: Es conveniente dejar un espacio en blanco al inicio de la cadena de texto *`formato`* para asegurar una lectura correcta ignorando posible caracteres que queden en el *buffer* de entrada.
- ***argumentos***. Lista de variables que tomarán los valores leídos en el orden indicado en la cadena de texto *`formato`*. Las variables cuyo tipo de dato es simple requieren ir precedidas del símbolo de dirección, **&**, ya que `scanf` necesita acceder a la dirección de memoria de la variable para escribir el valor leído.

5. Sentencias de Entrada y Salida de datos

Sentencia de entrada en C: `scanf`

Formato de las marcas:

tipo	Descripción
`c`	Lee un carácter simple
`d`	Lee un entero decimal
`i`	Lee un entero decimal, octal o hexadecimal
`e, f`	Lee un número en coma flotante
`g`	Lee un número en coma flotante, usando la más corta de las dos: `%e` o `%f`
`o`	Lee un número octal
`s`	Lee una cadena de caracteres sin espacios en blanco. Para leerla con espacios en blanco, se utiliza el modificador [...]: `%[^\n]s`
`u`	Lee un número entero decimal sin signo
`x`	Lee un entero hexadecimal
`p`	Lee un puntero

Sentencia de entrada en C: scanf

Lectura de cadenas de caracteres

- Para leer cadenas de caracteres se utiliza el modificador **s**:

```
scanf("%s", &cad); // lee cadenas de caracteres sin espacios
```

- Se puede especificar el número máximo de caracteres a leer especificándolo antes del modificador s:

```
scanf("%50s", &cad); // lee un máximo de 50 caracteres
```

- Se puede especificar el conjunto de caracteres permitidos utilizando **[ ]** antes del modificador s:

```
scanf("%[A-Z,a-z]s", &cad); // lee caracteres mientras sean letras
                            // mayúsculas o minúsculas
```

El modificador **^** permite indicar qué caracteres no están permitidos. Hay que ponerlo como primer carácter dentro de **[ ]**:

```
scanf("%[^\n]s", &cad); // lee cualquier carácter hasta encontrar un
                        // carácter \n (tecla Enter). Este uso de ^
                        // permite leer texto con espacios.
```

5. Sentencias de Entrada y Salida de datos

Ejemplo de uso de sentencias de entrada y salida

```
#include<stdio.h>
#define MAX_CAD 30
int main(){
  int base, altura;
  float area;
  char nombre[MAX_CAD]; // cadena (array) de caracteres. El array no es un tipo
                        // de dato simple.
  printf("Cálculo del área de un triángulo\n"); // el \n es para terminar la
                                                // línea y bajar a una nueva
  printf("Dime tu nombre: ");
  scanf(" %20s", nombre); // se lee una cadena de máximo 20 caracteres sin espacios.
                          // scanf(" %20[^\n]s", nombre); permitiría espacios.
                          // Se pone un espacio en blanco al principio de la cadena
                          // formato de scanf para eliminar caracteres residuales en
                          // el buffer de entrada y que la lectura sea correcta
  printf("Indica base y altura separados por un espacio: ");
  scanf("%d %d", &base, &altura); // se leen dos enteros separados por un espacio.
  area = (base * altura) / 2.0; // se pone 2.0 para que el resultado sea float
  // se imprimen todos los datos en una única instrucción printf
  printf("Hola %s, el área del triángulo de base %d y altura %d, es: %.2f\n",
          nombre, base, altura, area);
  return 0;
}
```