

holamundo.c — Kate

Archivo Editar Ver Proyectos Cliente LSP Marcadores Sesiones Herramientas

holamundo.c

```
1 #include<stdio.h>
2
3 int main(){
4     printf("Hola Mundo!!!\n");
5
6     return 0;
7 }
8
9 |
```

Línea 9, Columna 1 INSERTAR es\_ES Tabuladores débiles: 4 UTF-8 C

prog1@prog1-virtualbox : ~/Documentos \$

Salida Buscar y sustituir Proyecto actual Panel del terminal Cliente LSP

# Programming 1

## Lesson 2. Simple data types

**Degree in Computer Engineering**

# Syllabus

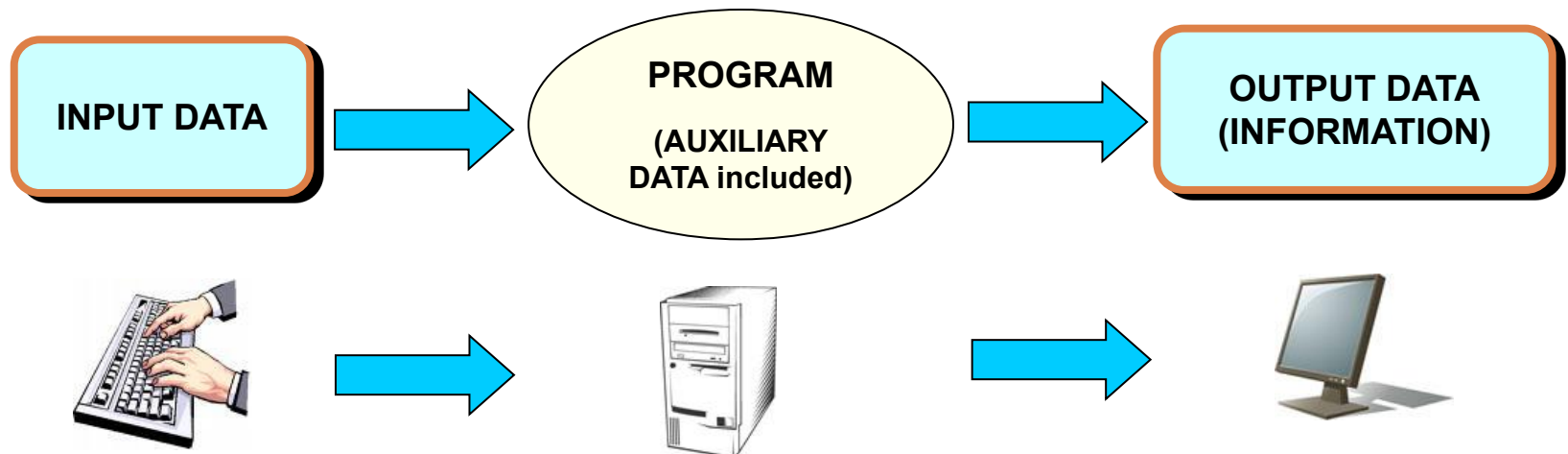
2

1. Data types
2. Variables and constants
3. Arithmetic and logical expressions
4. Operators in C
5. Data input and output sentences

# 1. Data types

3

- **Data:** Facts or values from which a conclusion (information) can be inferred.
- **Data in a computer program:** Data with which a computer operates.
  - **Input data** is a starting point for obtaining knowledge and producing **output data**.
  - The program may also need **auxiliary internal data** to produce the output data.

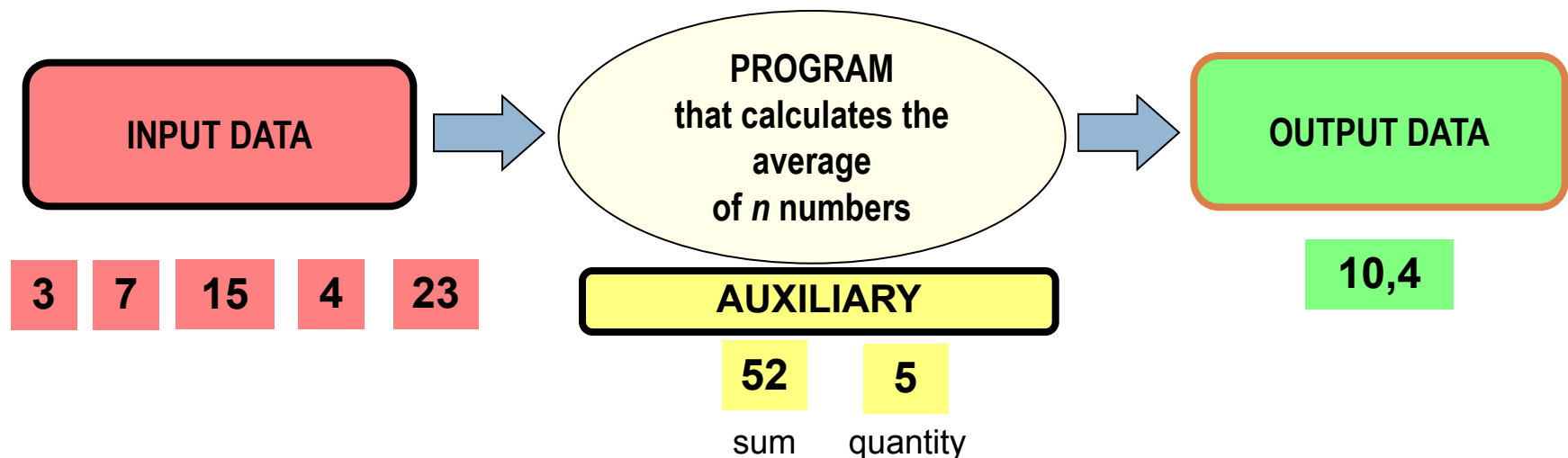


# 1. Data types

4

## Example of data handling in a program

- Program: Calculate the arithmetic mean of  $n$  numbers
- Input data: Any  $n$  numbers
- Output data: The arithmetic mean of the  $n$  numbers
- Internal auxiliary data: The sum of the numbers and the quantity of them



# 1. Data types

5

## ■ Data type

A data type is defined by:

- The set of possible **values** it can take (domain) in the program. Attempting to give the data type a value outside the set of possible values may result in an error.
- The set of **operations** that can be performed on the possible values it can take.

# 1. Data types

6

## Example:

- **Boolean** data type
  - Values = { true, false }
  - Operations = { and, or, not }

Values		Result of the operations		
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. Data types

7

## Simple data types

- These are **elementary** data types that do not derive from any other data type.
- Each particular value of the simple data type is specified by a literal. For example, integer literals can be expressed in the following ways:
  - Decimal (base 10) : 255
  - Binary (base 2) : 0101 ( $1*2^2+0*2^1+1*2^0=5$ )
  - Octal (base 8) : 0377 ( $3*8^2 + 7*8^1 + 7*8^0 = 255$ )
  - Hexadecimal (base 16): 0xff ( $15*16^1 + 15*16^0 = 255$ )

## 8

## character

# integer

real

# boolean

In C, the **bool** data type can be emulated with the **int** data type (zero is false, non-zero is true).



# 1. Data types

9

## Values of simple data types in C

Type	Bytes	Values	Precision
<b>char</b>	1	Letters: 'a', 'b',... 'z' 'A', 'B', ... 'Z' Digits: '0', '1', '2', .. '9' Special symbols: '+', '-', '/', '=', '(', ...	
<b>short</b>	2	-32.767... 32.767	
<b>int</b>	4	-2.147.483.647... 2.147.483.647	
<b>float</b>	4	Aprox. $10^{-38}$ ... $10^{38}$	7 digits
<b>double</b>	8	Aprox. $10^{-308}$ ... $10^{308}$	15 digits
<b>bool</b>	1	true, false	

# 1. Data types

10

## Enumerated data types

- Generally, programming languages have predefined data types and also allow the user to define his own data types.
- In the C language:
  - The user can define enumerated data types composed of a set of identifiers representing an integer value.
  - There is no print format for these types. The first element has a value of 0 associated with it, the second element has a value of 1 and so on.

```
enum T_WeekDay {Monday, Tuesday, Wednesday, Thursday, Friday,  
                Saturday, Sunday};  
  
enum T_Primary_Color {red, green, blue};
```

## 2. Variables and constants

11

### ■ Common features:

- They allow data to be represented in a program
- They provide a memory space reserved for storing a value of a data type.
- They are identified by a name

### ■ They differ in:

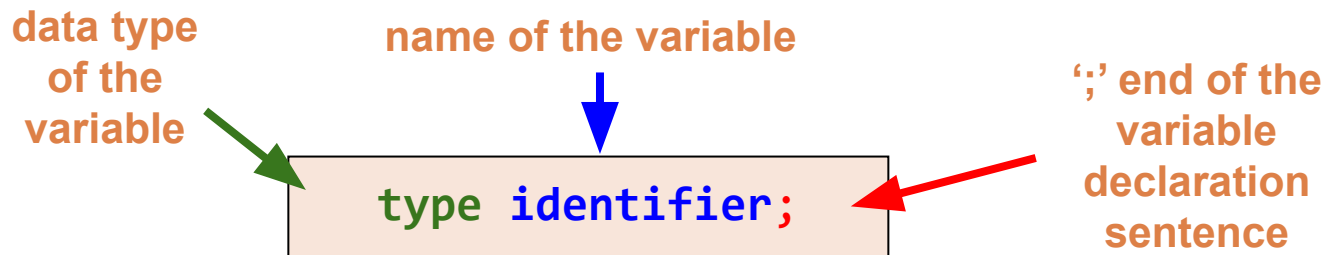
- **The value of a variable can change** throughout the program.
- **The value of a constant never changes** in the program.

## 2. Variables and constants

12

### Variable declaration in C

A data type must be associated with the variable so that any value of that data type can be stored in the variable.



It is possible to initialize (define its value) the variable in the declaration itself:

```
type identifier = value;
```

Examples:

- `char dni_letter;` // variable to store the letter of the dni of any person.
- `int pages = 20;` // variable to store the number of pages of any book. It is initialized to 20.

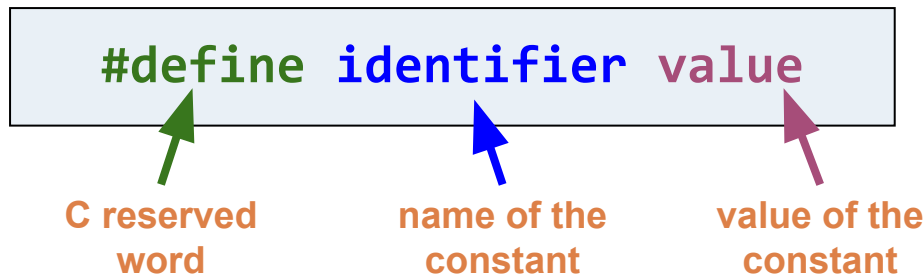
## 2. Variables and constants

13

### Declaration of constants in C

There are two ways to declare constants in C:

- As a **macro** at preprocessor level



#### Features:

- It has no data type.
- The preprocessor searches the source code for the `identifier` string and replaces it with the value string.
- It allows arguments in the `identifier` expression.

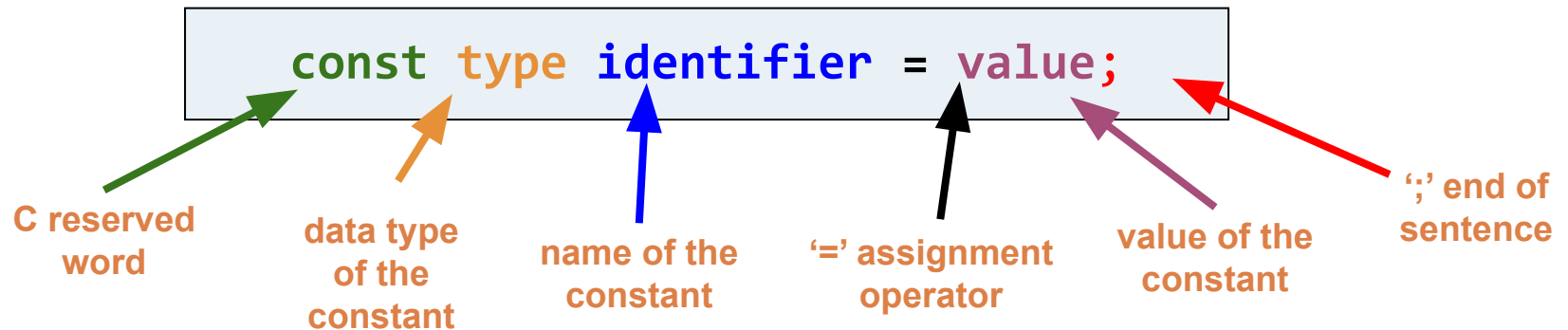
```
1 #include<stdio.h>
2
3 #define PI 3.141592
4
5 #define incrementar(x) x++
6
7 int main(){
8     int num;
9
10    num = 5;
11    printf("num: %d\n", num);
12
13    incrementar(num);
14    printf("num: %d\n", num);
15
16    return 0;
17 }
```

## 2. Variables and constants

14

### Declaration of constants in C

- As a **constant variable** at compiler level



#### Features:

- When declared with a data type, the memory area in which it is located can be accessed.
- Its value can be assigned to variables with compatible data type.

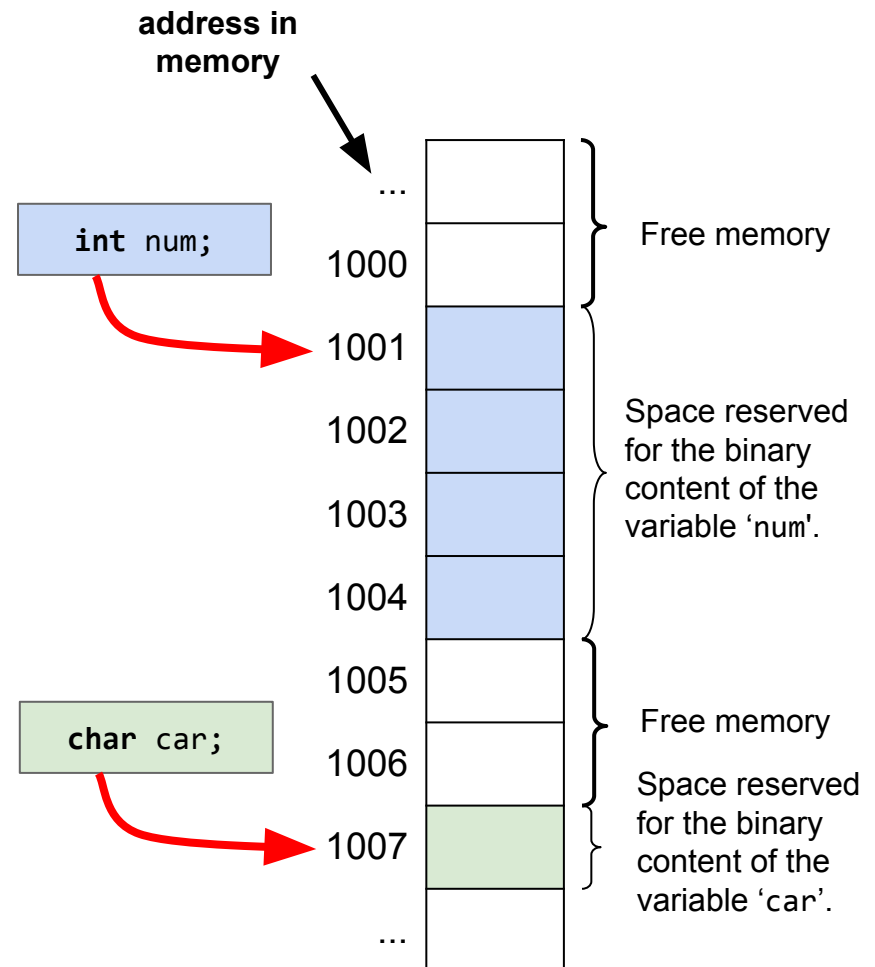
```
1 #include<stdio.h>
2
3 const float PI = 3.141592;
4
5 int main(){
6     float area, radio;
7
8     printf("Dime el radio del círculo: ");
9     scanf("%f", &radio);
10
11     area = PI * (radio * radio);
12
13     printf("El área del círculo es: %f\n", area);
14
15     return 0;
16 }
```

## 2. Variables and constants

15

### Representation of variables in memory

- Memory consists of a list of numbered locations (bytes).
- A variable represents a portion of memory consisting of a group of consecutive bytes.
- A variable in memory is determined by:
  - the address in memory, which provides the location of the first byte dedicated to that variable
  - the data type, which determines the amount of bytes of memory required by the variable



## 2. Variables and constants

16

### Representation of variables in memory

#### Pointers in C

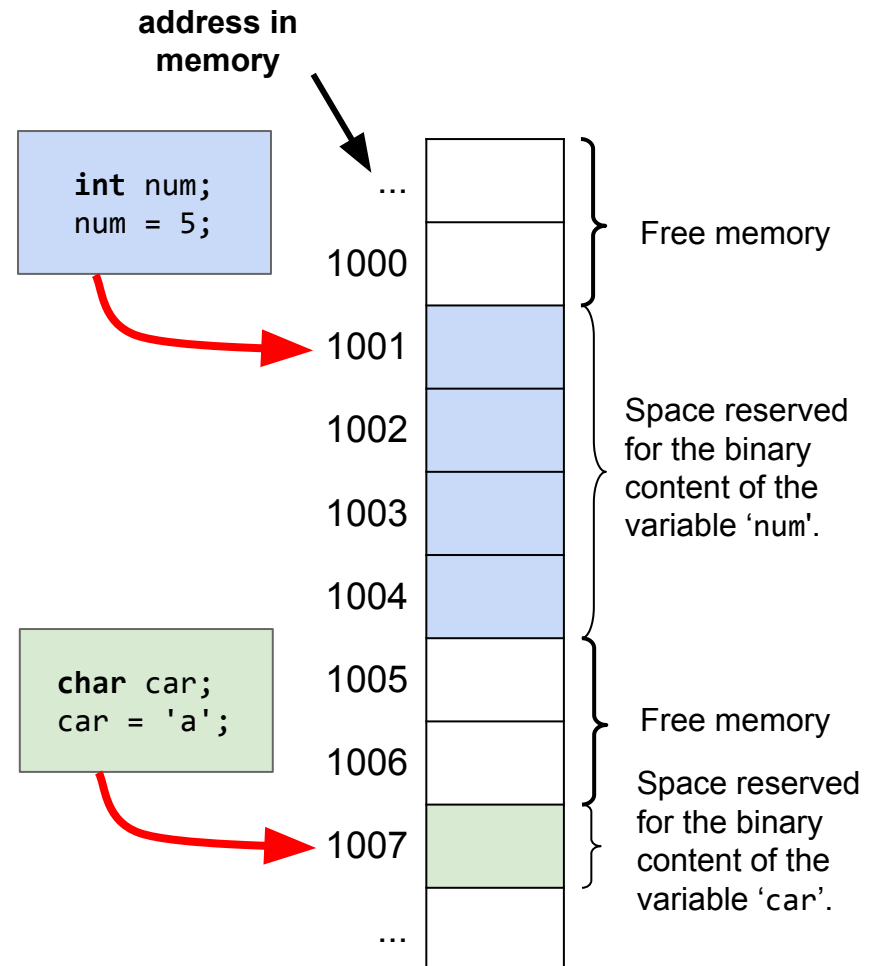
- They allow direct access to memory locations
- For a variable, they allow to differentiate between its position in memory (**address**) and its value (**content**).

■ Address operator: &

■ Content operator: \*

<code>&amp;num</code>	$\Rightarrow$	1001
<code>*(&amp;num)</code>	$\Rightarrow$	5

<code>&amp;car</code>	$\Rightarrow$	1007
<code>*(&amp;car)</code>	$\Rightarrow$	'a'





## 2. Variables and constants

17

### Representation of variables in memory

#### Pointers in C

##### Example:

```
#include<stdio.h>
int main(){
    int num; // num variable
    int *pNum; // pointer to a memory address
    num = 5; // value of 5 is assigned to num
    pNum = &num; // memory address of num
    // The following two sentences
    // print the value of num
    printf("num: %d\n", num);
    printf("*pNum: %d\n", *pNum);
    // The following two sentences
    // print the memory address of num
    printf("pNum: %p\n", pNum);
    printf("&num: %p\n", &num);
    return 0;
}
```

Declaration of a pointer type variable. We have to use \*.

##### Result:

\$ ./example

num: 5

\*pNum: 5

pNum: 0x7fff458af05c

&num: 0x7fff458af05c





## 2. Variables and constants

18

### Example of the use of variables and constants

A football team X, owner of a stadium Y, needs to calculate the collection of money in each one of home matches. You have to keep in mind that there three kind of tickets are sold: **ticket A** (Behind the Goal), **ticket B** (Alongside) and **ticket C** (Covered Alongside). For the whole season, the price of ticket A is half the price of ticket B. The price of ticket C is twice the price of ticket B. For each match, the price of ticket B is set, along with **discounts** for **kids** (80% off) and for **retirees** (50% off).

To calculate the revenue for each match of the season ...

- To store the price of **ticket B** ...  Real variable
- To store the **discount for kids** ...  Constant
- To store the **number of ticket C sold** ...  Integer variable
- To store the **kind of ticket sold** ...  Integer variable
- To store **whether a discount is applied or not** to a ticket ...
- To store the price of ticket A...

## 2. Variables and constants

19

### Identifiers for variables and constants

- The most common programming notations
  - Variables are written in lowercase and constants are written in uppercase
  - With identifiers composed of several words:
    - ✓ Everything in lowercase except for the first letter of each additional word (camelCase)  
`nombreAlumno`
    - ✓ Everything in lowercase, separating the words with the underlined character  
`nombre_alumno`
    - ✓ Everything in uppercase, separating the words with the underlined character  
`NOMBRE_ALUMNO`
    - ✓ Use of abbreviations of the same length  
`nom_alu`

**Note:** It is very important **not to arbitrarily change notation** and to follow only one notation in order to maintain coherence in our programmes and to facilitate readability and comprehension.

## 2. Variables and constants

20

### Identifiers in a program

- An identifier is a name used by the programmer to reference data and other elements of the program.
- General rules for building identifiers:
  1. Must be meaningful
  2. It must not be the same as the programming language's own reserved words.
  3. The length must not be excessively long
  4. Must begin with a letter or the underscore symbol and may contain letters, digits and the underscore symbol
  5. They are not accented
  6. Depending on the programming language, the identifier may or may not be used in either upper or lower case.

**Note:** The C language is **case sensitive**.

## 2. Variables and constants

21

### Identifiers in a program

#### ■ Good identifiers:

- distance
- euclidean\_distance
- \_date
- birthDate
- PI\_NUMBER
- number1
- number\_2



#### ■ Bad identifiers:

- euclidean-distance
- 3books
- Number\$1
- Any\_question?
- número



**Note:** The following **identifiers** are different in C language:

Car\_color   car\_color   CAR\_COLOR   car\_Color   Car\_Color

# 3. Arithmetic and logical expressions

22

- An expression in a program is a combination of variables, constants, operators, parentheses and function identifiers, the evaluation of which yields a value.
  - Expressions can be written anywhere in the program where the value they return can be used
- An arithmetic expression ...
  - is built with arithmetic operators
  - returns a numeric value

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

*calculates the degrees corresponding to the radian value stored in the variable x\_rad, using the constant PI*

- A logical expression ...
  - it is built with relational and logical operators
  - arithmetic operators may be present
  - returns a boolean value

```
(year modulus 4 == 0) AND (NOT (year modulus 100 == 0) OR (year modulus 400 == 0))
```

# 4. Operators in C

23

arithmetic operators	meaning	types of operand	types of result
+      -      *      /	addition, subtraction, multiplication, division	numeric integer or real	numeric integer or real
%	remainder of the division	integer	integer
<b>relational operators</b>			
<      >      <=      >=	less than, greater than, less than or equal to, greater than or equal to	simple data type	boolean
==      !=	equal to, not equal to	simple data type	
<b>logical operators</b>			
&&	logical AND	boolean	boolean
	logical OR	boolean	boolean
!	logical NOT	boolean	boolean

## Notes:

- In the C language, the **assignment operator** '=' and the **relational equality operator** '==' are different. It is common to confuse them, giving rise to errors that are difficult to detect.
- With the **division operator** '/', when the operands are of integer numeric data type, the result is the integer part of the quotient. To obtain a result with decimals, one of the operands must be of real numeric data type.

# 4. Operators in C

24

## Precedence and associativity of operators

- The **precedence** or **priority** of an operator indicates the order in which operations are executed in an expression containing different operators.
- The **associativity** of an operator indicates the order in which operations are executed in an expression containing operators with the same priority.

operators	meaning	asociativity
-            !	negative sign of a number, logical NOT	from right to left
*            /            %	multiplication, division, remainder	from left to right
+            -	addition, subtraction	from left to right
<    >    <=    >=	comparison	from left to right
==            !=	equality/inequality	from left to right
&&	logical AND	from left to right
	logical OR	from left to right

The use of **parentheses** is recommended

- when in doubt about the order of evaluation
- to make the expression more readable
- to modify the order of evaluation



# 5. Data input and output sentences

25

- Variables can be used in input statements.
- Variables, constants and, in general, expressions can be used in output statements.
- **Input sentences** allow storing data from keyboard in variables.
- **Output sentences** allow displaying data on the screen.

**Note:** Input and output can be associated with different **sources** and **devices**, such as files, printers, touch screens, mouse, etc. In this subject, we will only use the **keyboard** and the **screen** in our programs, which are usually the default input and output devices.

# 5. Data input and output sentences

26

## Output sentence in C: `printf`

It allows to display on the screen any combination of variable values, constants, expressions and text strings.

```
printf(const char *format [,arguments]);
```

- *format*. String of text. It may optionally contain *flags*, which start with % and allow interpolation of values specified in *arguments*.
- *arguments*. List of values to be interpolated into the *format* string where the corresponding flags are located and in the order in which they are indicated.

### Example:

```
int radius;  
float area;  
...  
printf("The area of the circle of radius %d is %f\n", radius, area);
```

As it can be seen, the order of the flags in the format string corresponds to the order of the data in the list of arguments to be printed.

# 5. Data input and output sentences

27

## Output sentence in C: printf

### Format tags:

Note: The values in square brackets are optional.

`%[parameter][flags][width][.precision][length]type`

parameter	Description
n\$	It allows changing the order in which arguments are processed. For example, the %3\$d flag would refer to the third value in the argument list, regardless of the order in which the flag appears in the <i>format</i> string.

flags	Description
<i>number</i>	Fills with spaces to the left up to the value of the number.
<i>number</i> 0	Fills with zeros to the left up to the value of the number.
+	Prints the number sign
-	Aligns the number to the left (by default it is aligned to the right).
#	For real numbers, the decimal point is printed and zeros are added at the end. For numbers that are not in base 10, the base prefix is added.

# 5. Data input and output sentences

28

## Output sentence in C: printf

### Format tags:

width	Description
<i>number</i>	Width to be used to print the value
*	It allows to specify the width in the argument list, just before the value. For example, <code>printf("%*d", 5, 10)</code> prints the number 10 in 5 character widths, padded with 3 blanks to the left of the 10.

precision	Description
<i>number</i>	Number of digits to display in the decimal part of the number, or number of characters to print for text strings.
*	It allows to specify the number of decimal places or the number of characters in the argument list, just before the value. For example, <code>printf("%.s", 2, "hello")</code> prints "he".

# 5. Data input and output sentences

29

## Output sentence in C: printf

### Format tags:

length	Description
hh	Converts a variable of type char to int
h	Converts a variable of type short to int
l	For integers, a variable of type long is expected
ll	For integers, a variable of type long long is expected
L	For reals, a variable of type long double is expected
z	For integers, a variable of type size_t is expected

# 5. Data input and output sentences

30

## Output sentence in C: printf

### Format tags:

type	Description
c	Prints the corresponding ASCII character
d, i	Prints as signed decimal integer
x, X	To print whole numbers in unsigned hexadecimal format
p	To print memory addresses (pointers)
e, E	To print signed floating point numbers in scientific notation
f, F	To print signed floating point numbers, using decimal point
g, G	To print floating point numbers, using the shortest notation
o	To print integers in unsigned octal format
u	To print whole numbers in unsigned decimal format
s	To print character strings ending in the end-of-string character ('\0')

# 5. Data input and output sentences

31

## Input sentence in C: `scanf`

It allows reading from the keyboard one or more values of the same or different data type at once.

```
scanf(const char *format, arguments);
```

- ***format***. Text string with the *flags* already explained for `printf`, which start with % and set the order of the variables specified in *arguments*. The data read from keyboard will be store in these variables. Note: It is convenient to leave a blank space at the beginning of the format string to ensure a correct reading by ignoring possible characters left in the input buffer.
- ***arguments***. List of variables that will take the values read in the order indicated in the *format* string. Variables whose data type is simple need to be preceded by the address symbol, &, since `scanf` needs to access the variable's memory address to write the value read.

# 5. Data input and output sentences

32

## Input sentence in C: scanf

### Format flags:

type	Description
c	Reads a single character
d	Reads a decimal integer
i	Reads a decimal, octal or hexadecimal integer
e, f	Reads a floating point number
g	Reads a floating point number, using the shortest of %e or %f.
o	Reads an octal number
s	Reads a string of characters without blank spaces. To read it with blank spaces, the modifier [...] used is: %[^\n]s
u	Reads an unsigned decimal integer
x	Reads a hexadecimal integer
p	Reads a pointer



# 5. Data input and output sentences

33

## Input sentence in C: scanf

### Example:

```
#include<stdio.h>
#define SIZE 30
int main(){
    int base, height;
    float area;
    char name[SIZE]; // string (array) of characters. The array data type
                     // is not a simple data type.

    printf("Calculate the area of a triangle\n"); // \n ends the line and goes down to a new one
    printf("Type your name: ");
    scanf(" %s", name); // a string of characters with no blank space is read.
                       // By using scanf(" %[^\n]%s", name); strings of characters are
                       // allowed to have blank spaces.

    printf("Type base and height separated by a space: ");
    scanf(" %d %d", &base, &height); // Two integers, separated by a space, are read. A blank
                                     // is added at the beginnig of the string format
                                     // of scanf so that the reading operation is correct

    area = (base * height) / 2.0; // we use 2.0 in order to get a float result
    // all data is printed in a single printf instruction
    printf("Hi %s, the area of the triangle of base %d and height %d, is: %.2f\n",
           name, base, height, area);

    return 0;
}
```