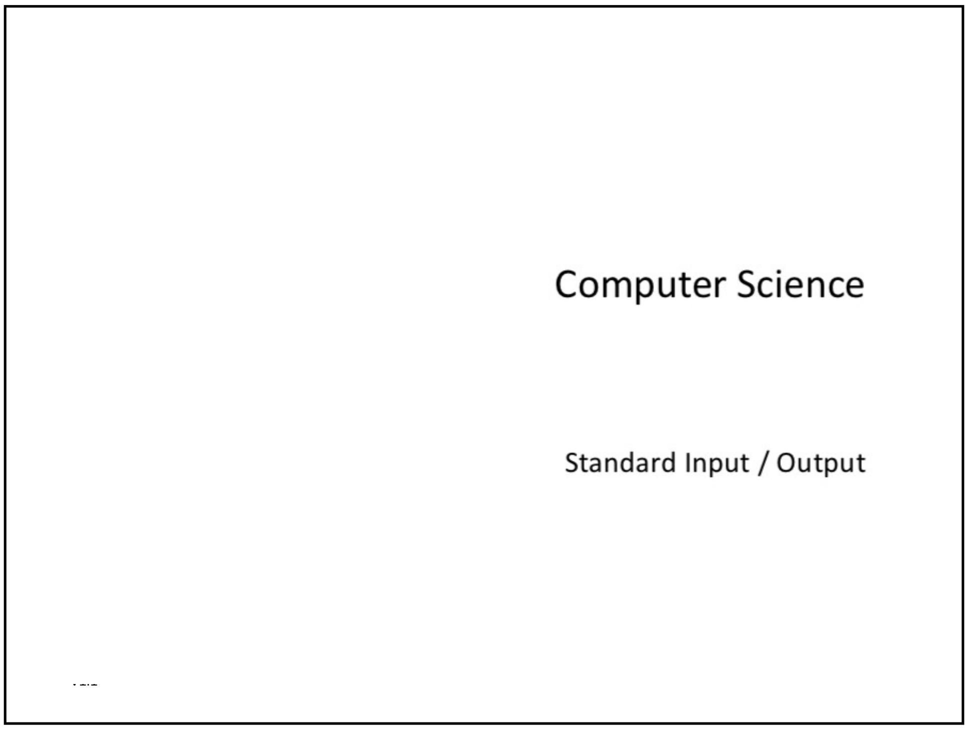
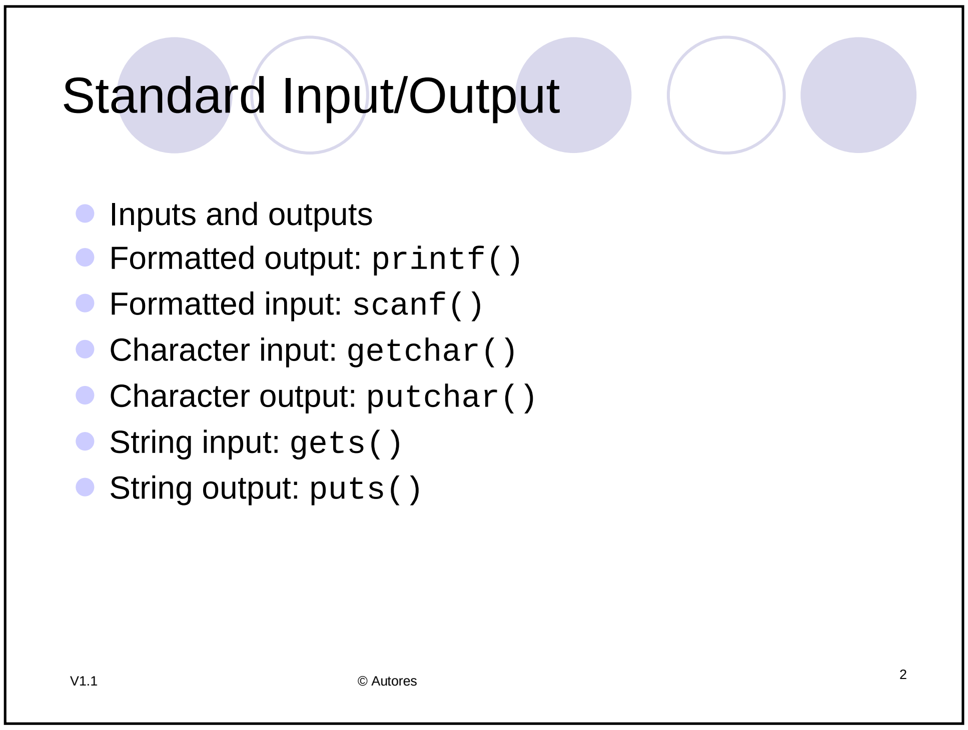


Computer Science

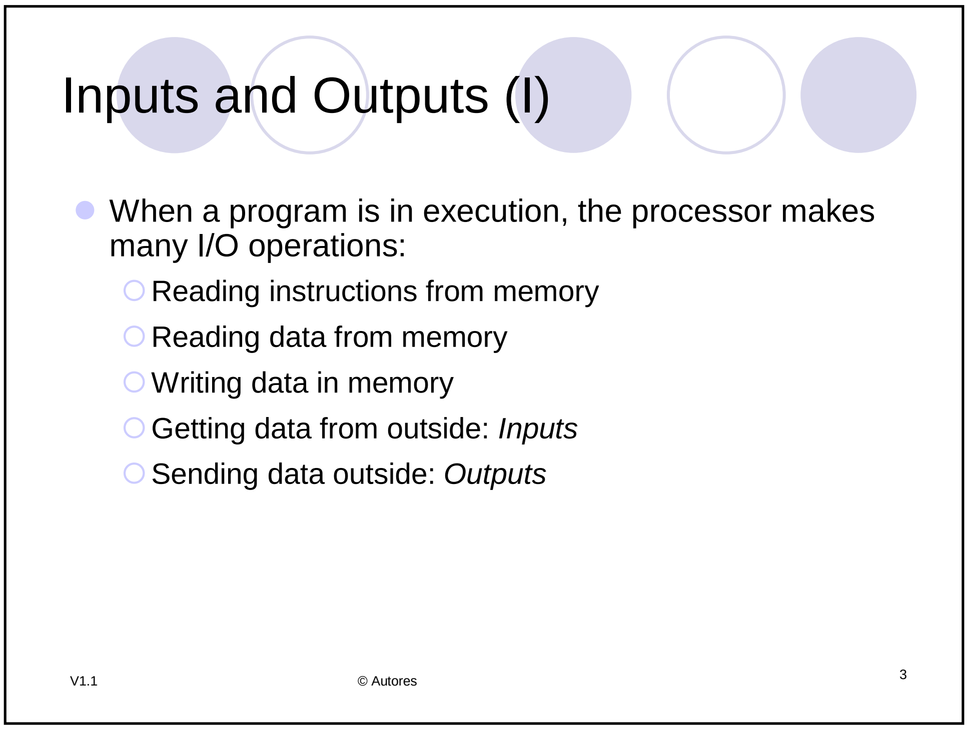
Standard Input / Output



Standard Input/Output

- Inputs and outputs
- Formatted output: `printf()`
- Formatted input: `scanf()`
- Character input: `getchar()`
- Character output: `putchar()`
- String input: `gets()`
- String output: `puts()`

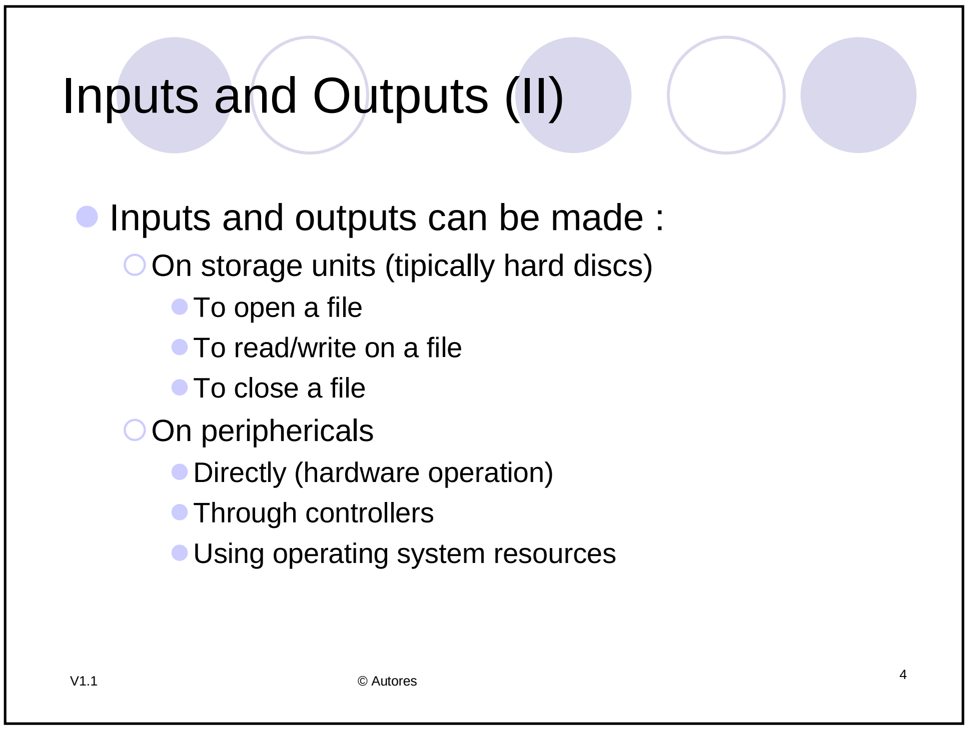
V1.1 © Autores 2



Inputs and Outputs (I)

- When a program is in execution, the processor makes many I/O operations:
  - Reading instructions from memory
  - Reading data from memory
  - Writing data in memory
  - Getting data from outside: *Inputs*
  - Sending data outside: *Outputs*

V1.1 © Autores 3



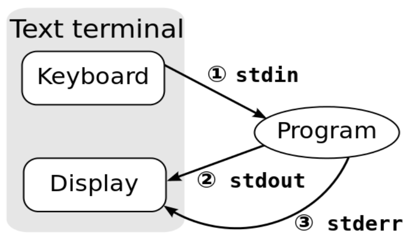
Inputs and Outputs (II)

- Inputs and outputs can be made :
  - On storage units (tipically hard discs)
    - To open a file
    - To read/write on a file
    - To close a file
  - On peripherals
    - Directly (hardware operation)
    - Through controllers
    - Using operating system resources

V1.1 © Autores 4

Inputs and Outputs (III)

- A program in execution becomes a process that can use:
  - An *input standard stream* associated to the keyboard: **`stdin`**
    - What the user writes goes to the stream `stdin`
  - An *output standard stream* associated to the display: **`stdout`**
    - What the process writes in the stream `stdout` goes to the display
  - An *output standard stream for errors* which is associated to the screen: **`stderr`**
    - What the process writes in the stream `stderr` goes to the display



From Wikipedia

5

Inputs and Outputs (IV)

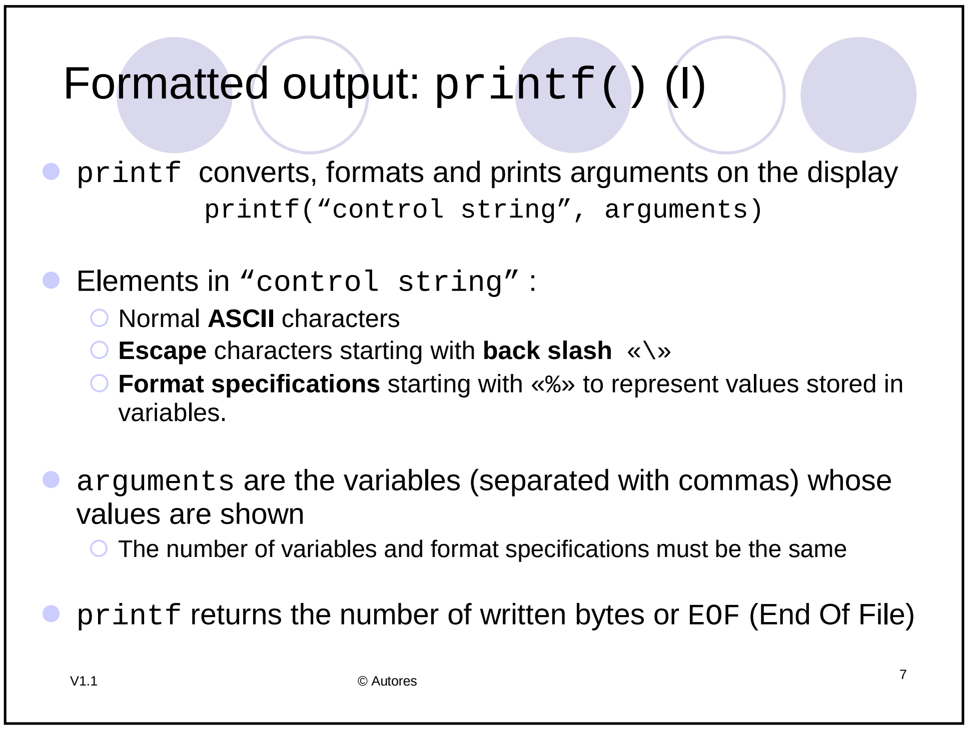
- ANSI C created a set of standard functions for I/O using streams **`stdin`** and **`stdout`**
  - Are defined in the file STDIO.H, so to use them in a program the following line must be included:

```
#include <stdio.h>
```

  - The most important are:

`printf()`	to write data with format
`scanf()`	to read data with format
`getchar()`	to read characters from the keyboard
`putchar()`	to write characters on the display

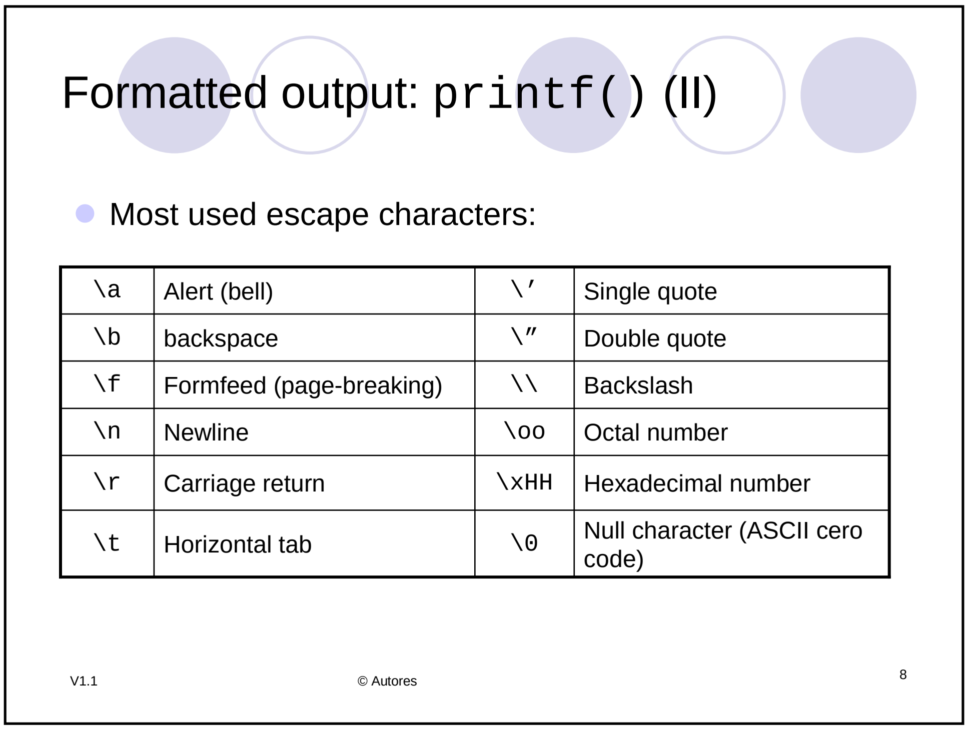
V1.1 © Autores 6



Formatted output: `printf()` (I)

- `printf` converts, formats and prints arguments on the display
  `printf("control string", arguments)`
- Elements in "`control string`":
  - Normal **ASCII** characters
  - **Escape** characters starting with **back slash** «\»
  - **Format specifications** starting with «%» to represent values stored in variables.
- `arguments` are the variables (separated with commas) whose values are shown
  - The number of variables and format specifications must be the same
- `printf` returns the number of written bytes or `EOF` (End Of File)

V1.1 © Autores 7

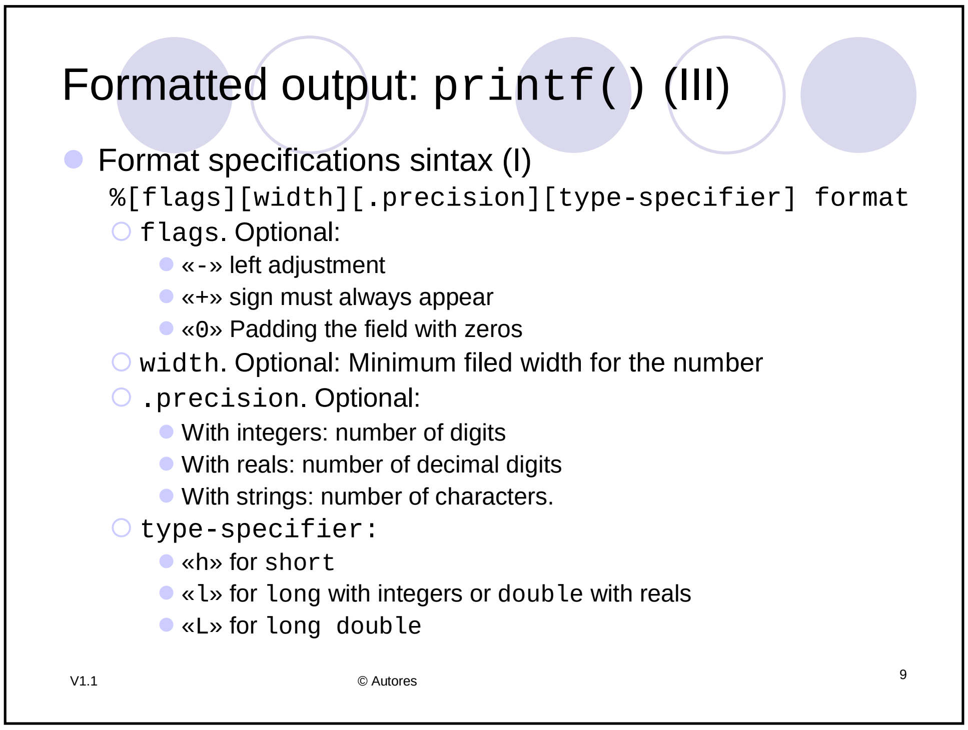


Formatted output: `printf()` (II)

- Most used escape characters:

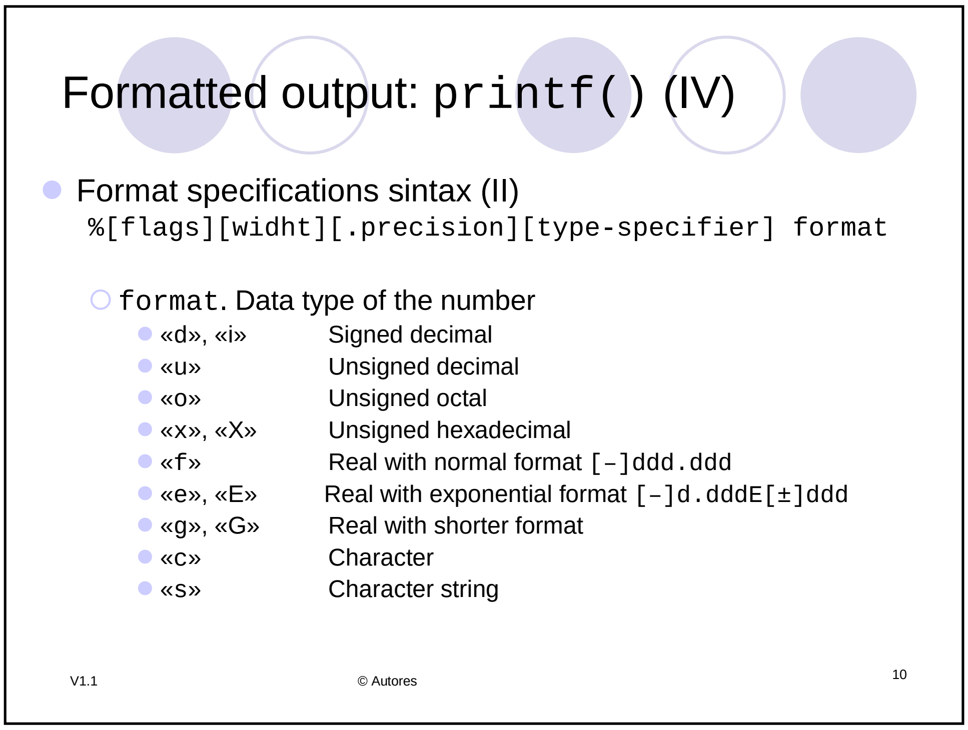
`\a`	Alert (bell)	`\'`	Single quote
`\b`	backspace	`\"`	Double quote
`\f`	Formfeed (page-breaking)	`\\`	Backslash
`\n`	Newline	`\oo`	Octal number
`\r`	Carriage return	`\xHH`	Hexadecimal number
`\t`	Horizontal tab	`\0`	Null character (ASCII cero code)

V1.1 © Autores 8



Formatted output: `printf()` (III)

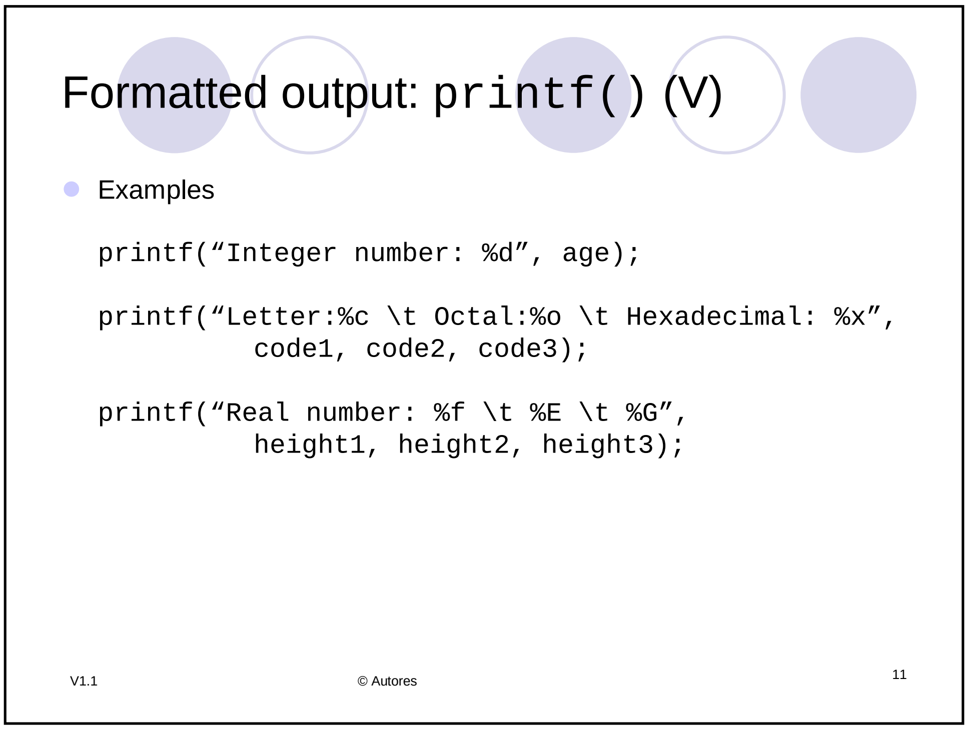
- Format specifications sintax (I)
  `%[flags][width][.precision][type-specifier] format`
  - `flags`. Optional:
    - «`-`» left adjustment
    - «`+`» sign must always appear
    - «`0`» Padding the field with zeros
  - `width`. Optional: Minimum filed width for the number
  - `.precision`. Optional:
    - With integers: number of digits
    - With reals: number of decimal digits
    - With strings: number of characters.
  - `type-specifier:`
    - «`h`» for `short`
    - «`l`» for `long` with integers or `double` with reals
    - «`L`» for `long double`



Formatted output: `printf()` (IV)

- Format specifications sintax (II)
  `%[flags][widht][.precision][type-specifier] format`
  - `format`. Data type of the number

«d», «i»	Signed decimal
«u»	Unsigned decimal
«o»	Unsigned octal
«x», «X»	Unsigned hexadecimal
«f»	Real with normal format `[-]ddd.ddd`
«e», «E»	Real with exponential format `[-]d.dddE[±]ddd`
«g», «G»	Real with shorter format
«c»	Character
«s»	Character string



Formatted output: `printf()` (V)

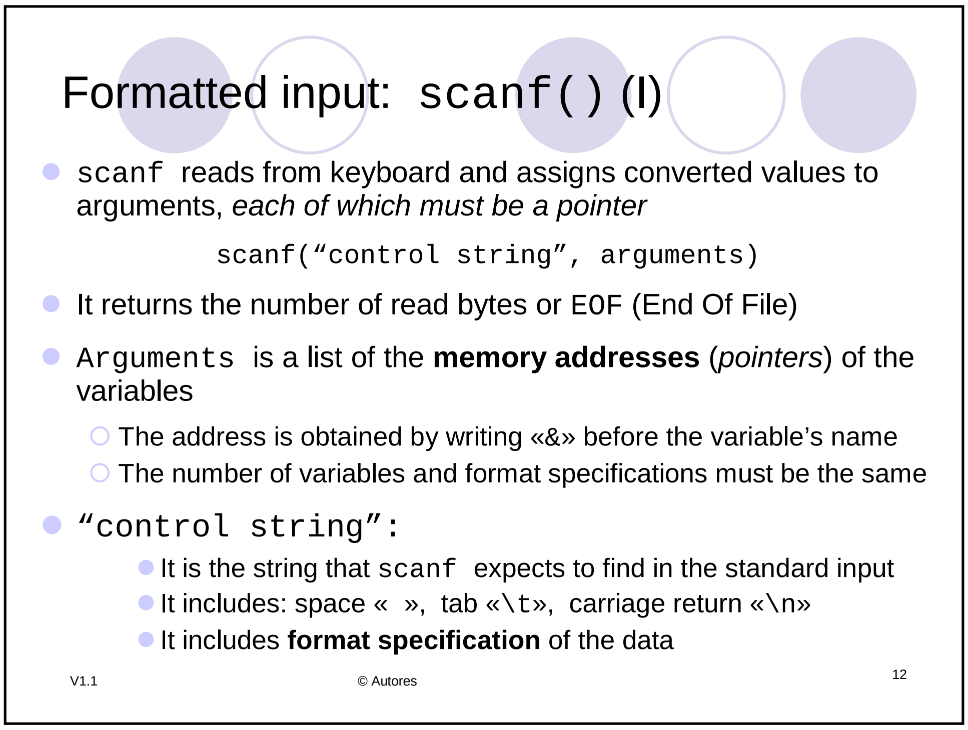
- Examples

```
printf("Integer number: %d", age);

printf("Letter:%c \t Octal:%o \t Hexadecimal: %x",
          code1, code2, code3);

printf("Real number: %f \t %E \t %G",
          height1, height2, height3);
```

V1.1 © Autores 11



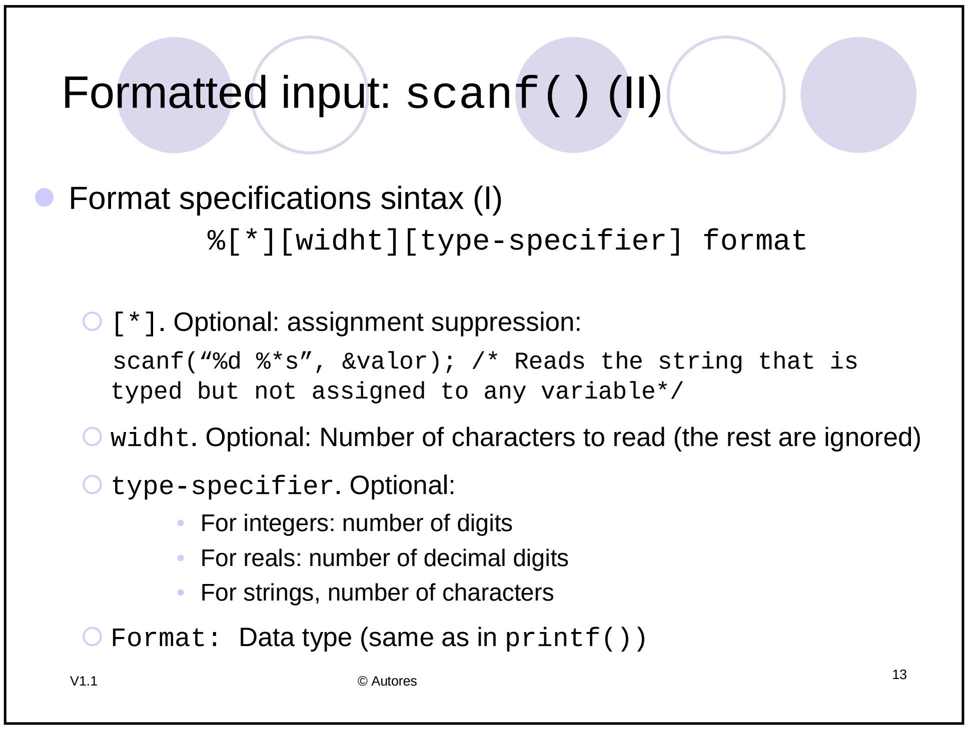
Formatted input: `scanf()` (I)

- `scanf` reads from keyboard and assigns converted values to arguments, *each of which must be a pointer*

  `scanf("control string", arguments)`

- It returns the number of read bytes or `EOF` (End Of File)
- `Arguments` is a list of the **memory addresses** (*pointers*) of the variables
  - The address is obtained by writing «&» before the variable's name
  - The number of variables and format specifications must be the same
- `"control string"`:
  - It is the string that `scanf` expects to find in the standard input
  - It includes: space « », tab «`\t`», carriage return «`\n`»
  - It includes **format specification** of the data

V1.1 © Autores 12



Formatted input: `scanf()` (II)

- Format specifications sintax (I)

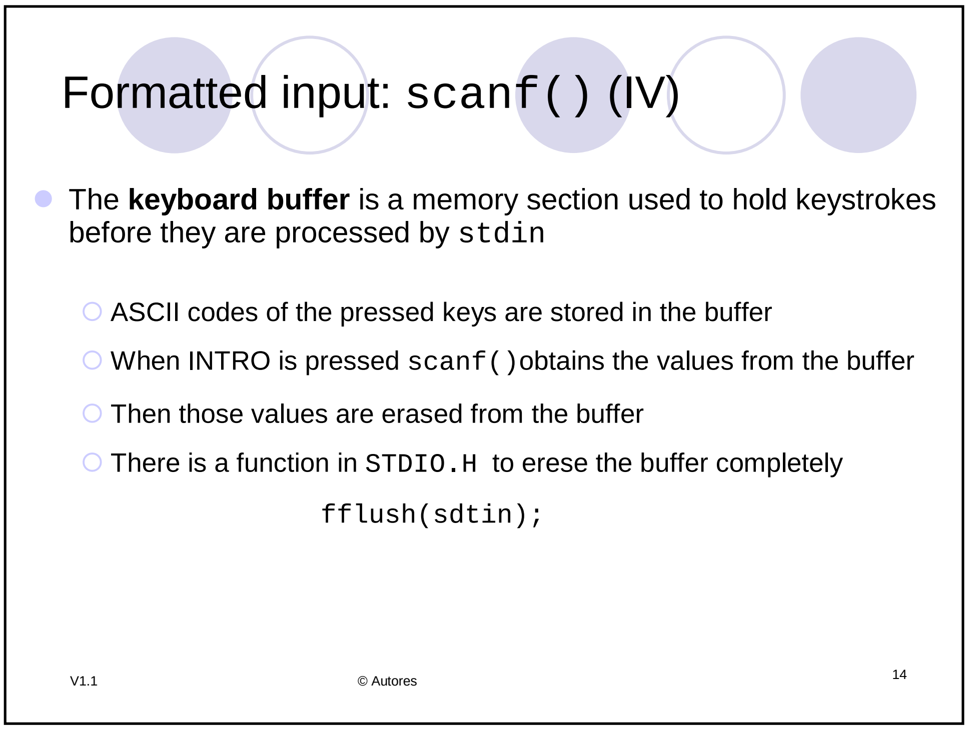
```
%[*][widht][type-specifier] format
```

  - `[*]`. Optional: assignment suppression:

```
scanf("%d %*s", &valor); /* Reads the string that is
typed but not assigned to any variable*/
```

  - `widht`. Optional: Number of characters to read (the rest are ignored)
  - `type-specifier`. Optional:
    - For integers: number of digits
    - For reals: number of decimal digits
    - For strings, number of characters
  - `Format:` Data type (same as in `printf()`)

V1.1 © Autores 13

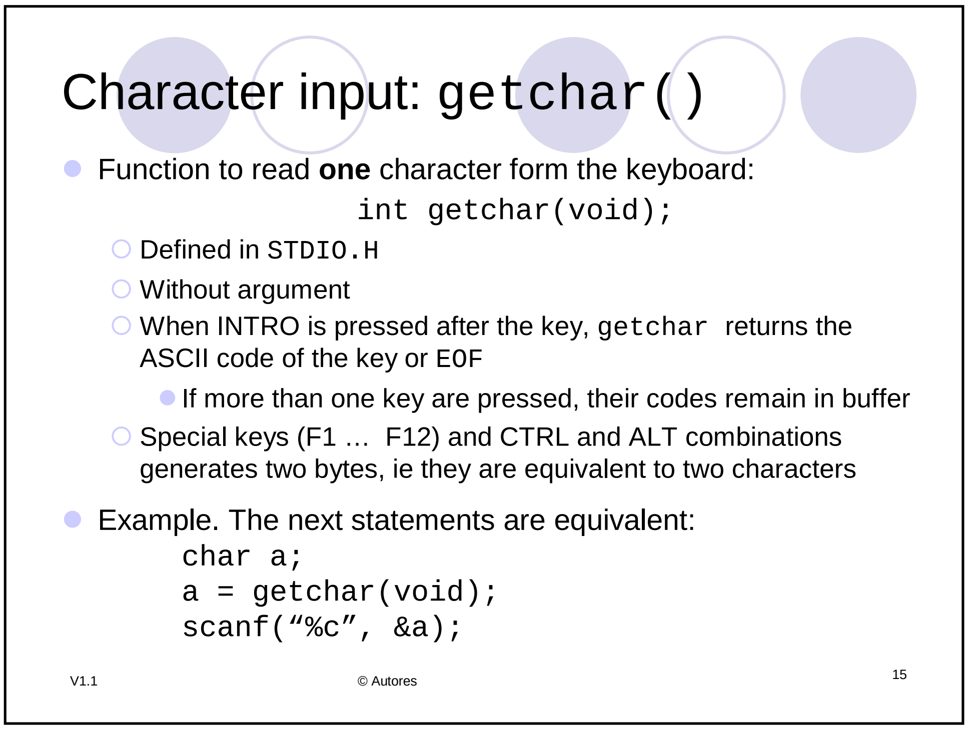


Formatted input: `scanf()` (IV)

- The **keyboard buffer** is a memory section used to hold keystrokes before they are processed by `stdin`
  - ASCII codes of the pressed keys are stored in the buffer
  - When INTRO is pressed `scanf()`obtains the values from the buffer
  - Then those values are erased from the buffer
  - There is a function in `STDIO.H` to erese the buffer completely

```
fflush(sdtin);
```

V1.1 © Autores 14



Character input: `getchar()`

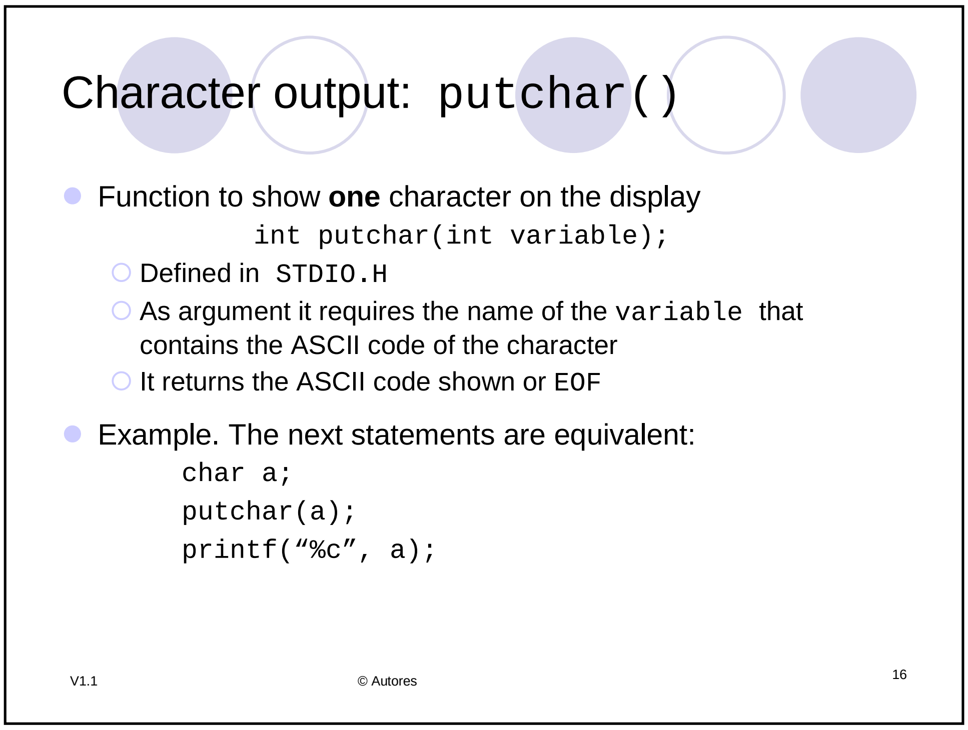
- Function to read **one** character form the keyboard:

  ```
  int getchar(void);
  ```

  - Defined in `STDIO.H`
  - Without argument
  - When INTRO is pressed after the key, `getchar` returns the ASCII code of the key or `EOF`
    - If more than one key are pressed, their codes remain in buffer
  - Special keys (F1 … F12) and CTRL and ALT combinations generates two bytes, ie they are equivalent to two characters
- Example. The next statements are equivalent:

  ```
  char a;
  a = getchar(void);
  scanf(“%c”, &a);
  ```

V1.1 © Autores 15



Character output: `putchar()`

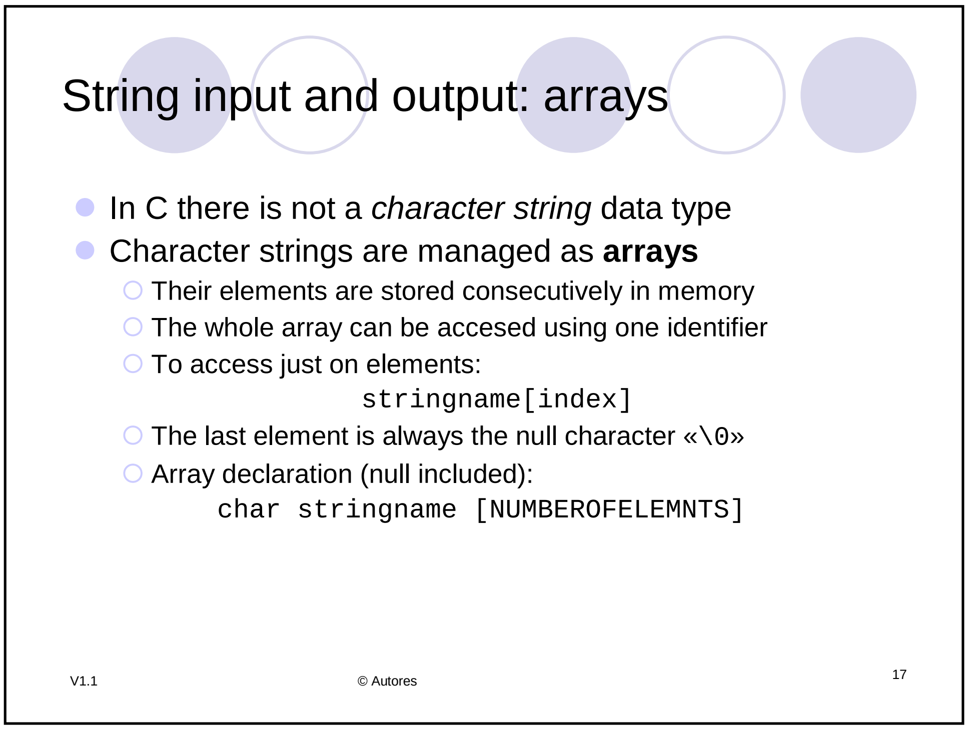
- Function to show **one** character on the display

  ```
  int putchar(int variable);
  ```

  - Defined in `STDIO.H`
  - As argument it requires the name of the `variable` that contains the ASCII code of the character
  - It returns the ASCII code shown or `EOF`
- Example. The next statements are equivalent:

  ```
  char a;
  putchar(a);
  printf(“%c”, a);
  ```

V1.1 © Autores 16



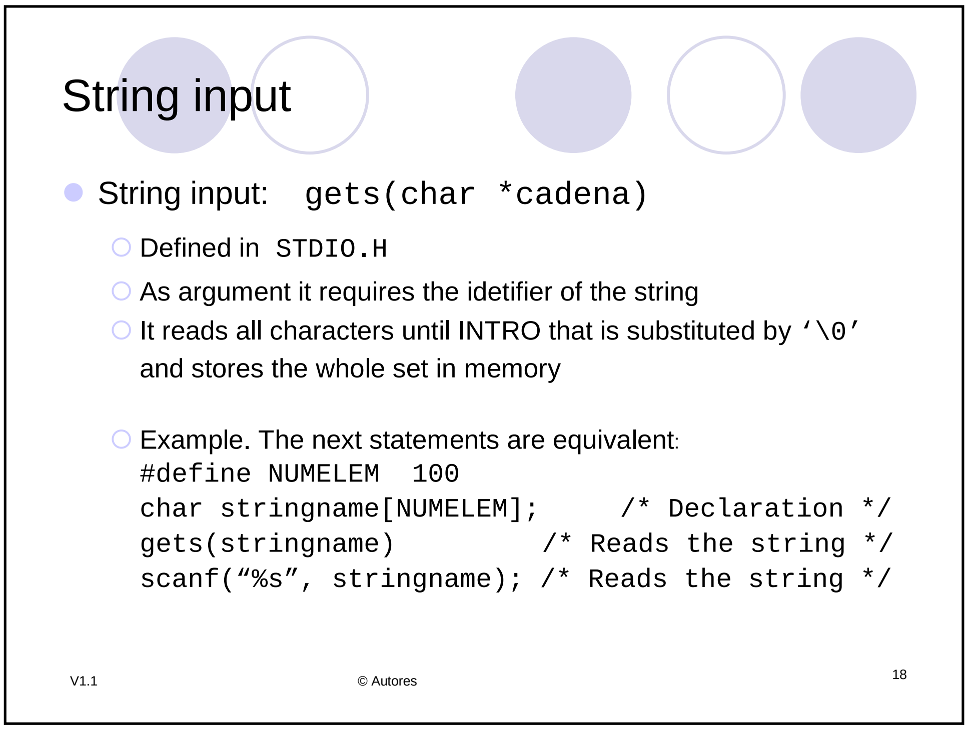
String input and output: arrays

- In C there is not a *character string* data type
- Character strings are managed as **arrays**
  - Their elements are stored consecutively in memory
  - The whole array can be accesed using one identifier
  - To access just on elements:

    ```
    stringname[index]
    ```
  - The last element is always the null character «`\0`»
  - Array declaration (null included):

    ```
    char stringname [NUMBEROFELEMNTS]
    ```

V1.1 © Autores 17

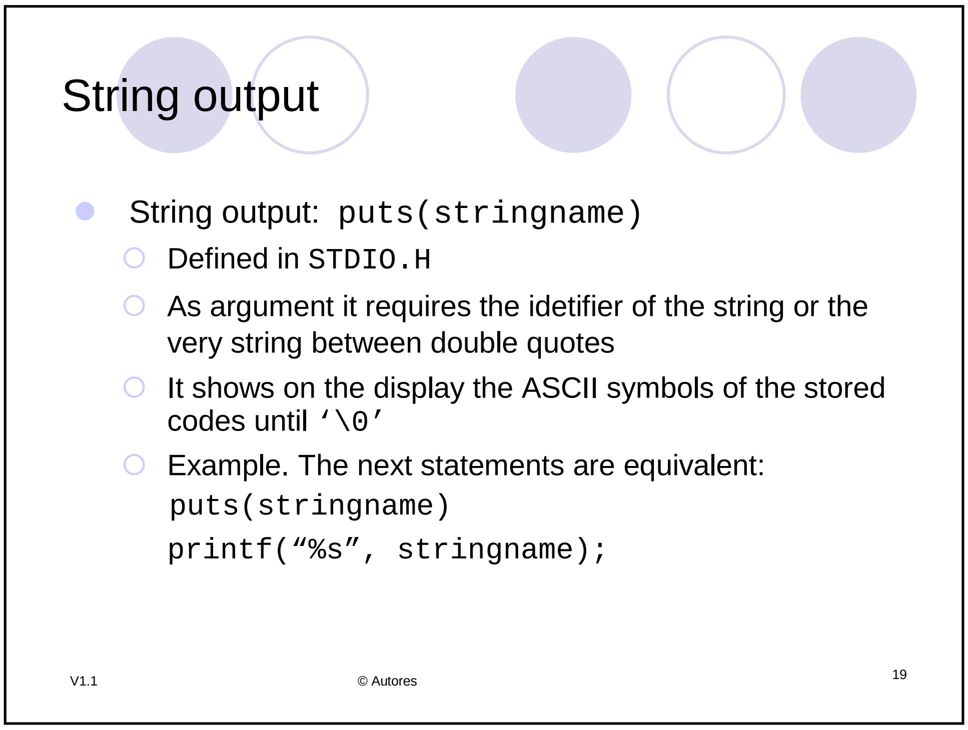


String input

- String input: `gets(char *cadena)`
  - Defined in `STDIO.H`
  - As argument it requires the idetifier of the string
  - It reads all characters until INTRO that is substituted by `‘\0’` and stores the whole set in memory
  - Example. The next statements are equivalent:

    ```
    #define NUMELEM  100
    char stringname[NUMELEM];     /* Declaration */
    gets(stringname)         /* Reads the string */
    scanf(“%s”, stringname); /* Reads the string */
    ```

V1.1 © Autores 18



String output

- String output: `puts(stringname)`
  - Defined in `STDIO.H`
  - As argument it requires the idetifier of the string or the very string between double quotes
  - It shows on the display the ASCII symbols of the stored codes until `‘\0’`
  - Example. The next statements are equivalent:

```
puts(stringname)
printf(“%s”, stringname);
```

V1.1 © Autores 19