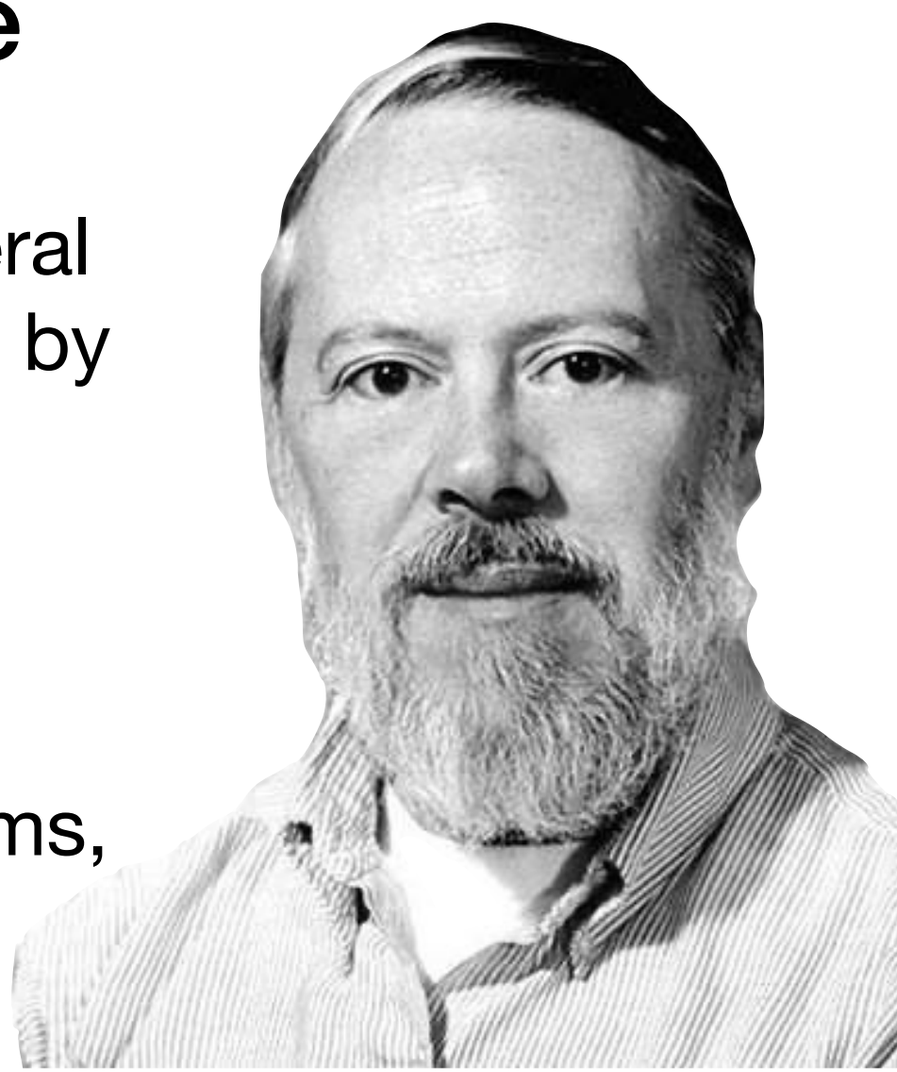
Princípios da Computação

The C programming language — a quick intro

The C programming language

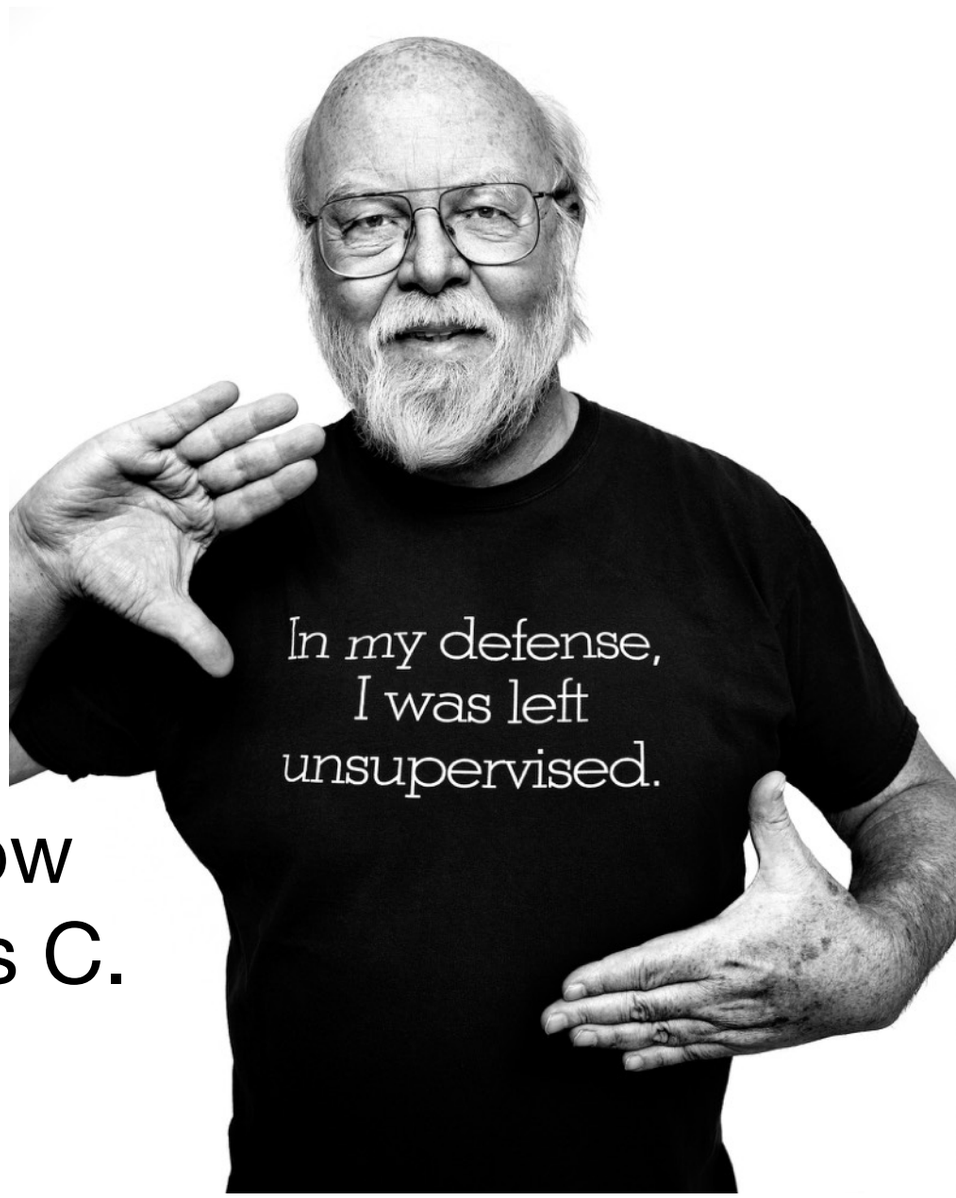
- The C programming language is a general programming language created in 1972 by Dennis Ritchie.



- Created to rewrite the Unix kernel.
- Currently used to write operating systems, device drivers, compilers, etc.
- Found in supercomputers to microcontrollers.

From Java to C

- Java is deeply influenced by the C programming language.
- Many of the syntax you already know from Java follows the same rules as C.



James Gosling
the creator of Java

Basic C program

- Source code saved in a file with **`.c`** extension.
- Structure:
  - Include dependencies.
  - Definition of constants and global variables.
  - Program specific functions.
  - Main function (where the program really starts).

```c
#include <stdio.h>
#include "my_header.h"

#define PI 3.1415926
int global_var;

int do_something()
{
   // Here is the function code.
}

int main()
{
   // The main program code is here.
   return 0;
}
```

Include header files, containing the declarations of external types and functions.
<file.h> : language or system standard file
"*file.h*" : project/user file

Text replacement definitions (improves code readability) and global variables.

Function definition. Functions must be declared before the main function.

The main function. This is where the program starts executing. The main flow of the program is written here.
return 0; means program terminates without errors.

Compiling a C program

- To compile a C program, you need a compiler suite.
- **gcc**
  - Usually used in Linux (but also in other systems).
- **clang**
  - Usually used in macOS (but also in other systems).

Install gcc on Ubuntu (Debian way)

- Using the `apt` package manager.

```
$ sudo apt update
    ...
$ sudo apt install build-essential
    ...
$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR
A PARTICULAR PURPOSE.
$
```

Install clang on macOS

- Install `xcode` command line tools

```
$ xcode-select --install
    ...
$ clang --version
Apple clang version 15.0.0 (clang-1500.0.40.1)
Target: x86_64-apple-darwin23.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
$
```

Command line to compile program

- With gcc:

```
$ gcc source.c -o executable
$
```

- With clang

```
$ clang source.c -o executable
$
```

- No news is good news! :-)

Storing data: variables

isep Instituto Superior de Engenharia do Porto

Declaring variables

- To create a variable you have to *declare* it:
    - indicating the type of value, and
    - the name of the variable.

```
char a;
int b;
float c;
```

Main C types (signed, two's complement)

Type	Minimum size	LP64 (popular 64-bit archs)
`char`	**8 bits**	**8 bits**
`short`	16 bits	16 bits
`int`	**16 bits**	**32 bits**
`long`	**32 bits**	**64 bits**
`long long`	64 bits	64 bits
`float`	**32 bits**	**32 bits**
`double`	**64 bits**	**64 bits**
`long double`	128 bits	128 bits

Unsigned types

- Variables can also store non-negative values.
- Use the **`unsigned`** prefix in the variable declaration.
  - The same size of signed variables, but no two's complement codification.

```
unsigned char a;
unsigned int b;
unsigned float c;
```

Assigning a value to a variable

- Use the assignment operator **=**

```
int x;
unsigned int a;

x = -13;
a = 45000;
```

Basic input / output

isep Instituto Superior de Engenharia do Porto

Printing formatted text

- Use the **`printf()`** function.

- Requires including **`<stdio.h>`** to compile.

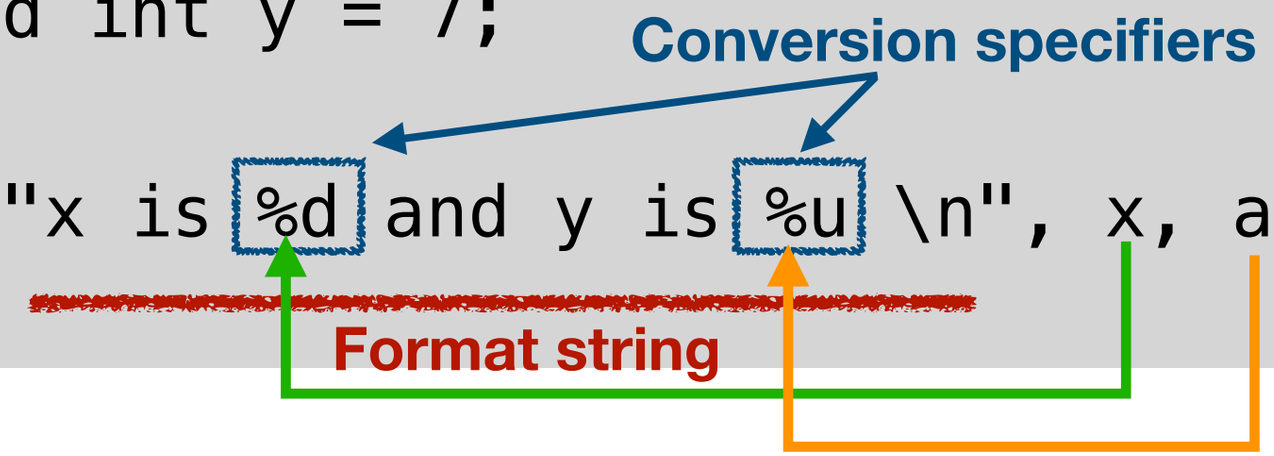
- First argument is the ***format string***, that specifies how subsequent arguments are converted for output.

- The following arguments are inserted in order in the output string, according to the ***conversion specifiers*** in the format string.

Printing formatted text — example

```
int x = -13;
unsigned int y = 7;

printf("x is %d and y is %u \n", x, a);
```



Output:

```
x is -13 and y is 7
```

Conversion specifiers (integers)

Type	character	signed decimal	unsigned decimal	unsigned octal	unsigned hexadecimal
char	**%c**	**%hhd / %hhi**	**%hhu**	**%hho**	**%hhx / %hhX**
short	%hd / %hi	%hu	%ho	%hx / %hX
int	**%d / %i**	**%u**	**%o**	**%x / %X**
long	**%ld / %li**	**%lu**	**%lo**	**%lx / %lX**
long long	%lld / %lli	%llu	%llo	%llx / %llX

Conversion specifiers (integers) — example

```
char c = 'J';

printf("c : %c (char)\n", c);
printf("hhd : %hhd (2's compl.)\n", c);
printf("hhu : %hhu (unsigned)\n", c);
printf("hho : %hho (octal)\n", c);
printf("hhx : %hhx (hex)\n", c);
printf("hhX : %hhX (HEX)\n", c);
```

Output:

```
c : J (char)
hhd : 74 (2's compl.)
hhu : 74 (unsigned)
hho : 112 (octal)
hhx : 4a (hex)
hhX : 4A (HEX)
```

Other conversion specifiers

Conversion specifier	Description
`%e`	**`float`** and **`double`** Prints the number in decimal scientific notation: `[-]d.ddde±dd`
`%f`	**`float`** and **`double`** Prints the number in decimal: `[-]ddd.ddd`
`%p`	**`void *`** pointer (i.e. a memory address) Prints a memory address in hexadecimal.

Size of a type / variable

- Use the **`sizeof()`** keyword.
    - Returns the size in bytes of the type / variable.

```
char a;

printf("char: %lu bytes\n", sizeof(a));
printf("int: %lu bytes\n", sizeof(int));
printf("pointer: %lu bytes\n", sizeof(void *));
```

Reading formatted text

- Use the **`scanf()`** function to read from stdin.
- Also requires including **`<stdio.h>`** to compile.
- First argument is the ***format string***, that specifies how to convert the input values.
    - Uses the same conversion specifiers as **`printf()`**.
- The following arguments are the ***addresses*** where the input values are stored.

Address of a variable

- Use the reference operator **&** :"the address of…"
  - (Example) "The address of variable x": **&x**

- In a machine, the addresses are all the same size, regardless of the type of variable.

Reading formatted text — example

```
int i;
float x;

printf("i = ");
scanf("%d", &i);
printf("x = ");
scanf("%f", &x)
printf("i is %d and x is %f\n", i, x);
```

Output:

```
i = -20
x = 3.78
i is -20 and x is 3.780000
```

Conditional branch

Conditional statement

- ***`if-else`*** construct
    - **`else`** is optional: it may not be necessary!

```
if(condition)
{
   // TRUE, do this section.
}
else
{
   // FALSE, do this section
}
```

Conditional statement

```c
int x;
// ...

if ((x % 2) == 0) {
   printf("%d is an even number.\n", x);
} else {
   printf("%d is an odd number.\n", x);
}
```

Loops

isep Instituto Superior de Engenharia do Porto

Count-controlled loops

- ***for*** construct.

- Three arguments:
  - the *initialisation*; the *condition*; the *advancement*.

```
for (int i = 0; i < limit; i++)
{
   // Do loop stuff.
}
```

Condition-controlled loops

- ***`while`*** construct.
- One argument: the *condition*.
- Minimum number of iterations: zero.

```
while (condition)
{
   // Do loop stuff.
}
```

Condition-controlled loops

- ***do-while*** construct.
- One argument: the *condition*.
- Minimum number of iterations: one.

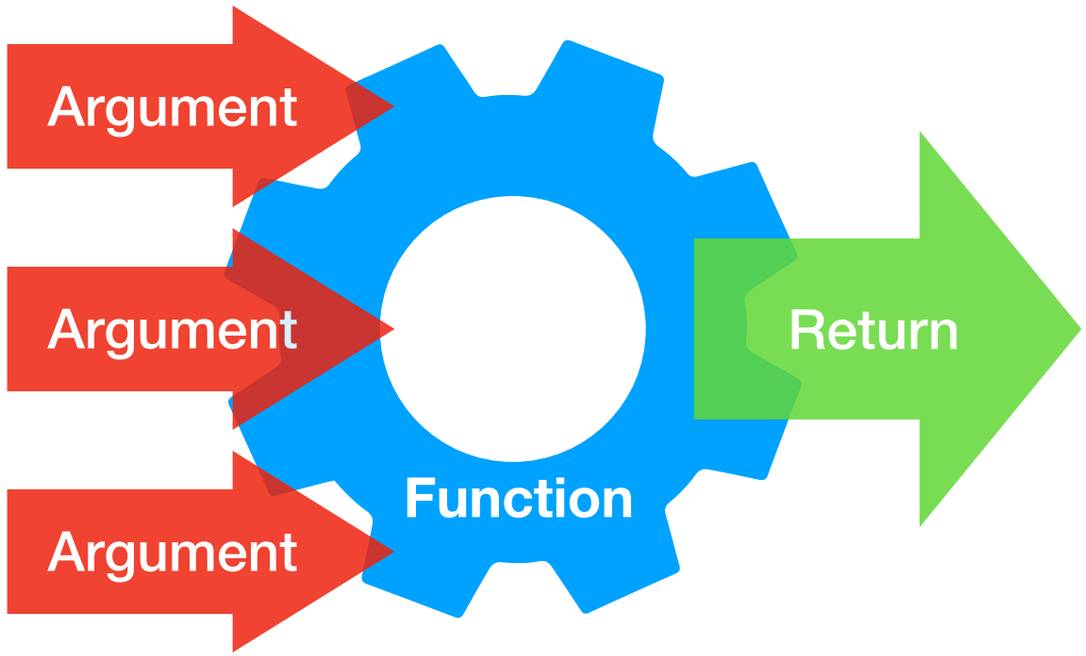
```
do
{
   // Do loop stuff.
} while (condition);
```

Functions

isep Instituto Superior de Engenharia do Porto

Functions

- Sequence of code that performs a specific task.
- Packaged as a unit, called by a unique name.
- Input values: ***arguments***.
- Output value: ***returned value***.



Function syntax

- The Java language uses the same syntax as the C language.

```
return_type function_name(list of arguments)
{
   // Function code.
}
```

Function (example)

```
int maximum_int(int value1, int value2)
{
   if (value1 > value2) {
      return value1;
   }
   else {
      return value2;
   }
}
```

Functions

- A function is called by its name, passing it the list of arguments.

```
max = maximum_int(a, b);
```

- The function name (without arguments) returns the base address of the function.

```
printf("Base address: %p\n", maximum_int);
```

Arrays

isep Instituto Superior de Engenharia do Porto

Arrays

- Arrays are used to store multiple values of the same type, in a single variable.

- Each value is accessed by its index.

- The first index of an array is the **`[0]`**.

- In an array with *n* elements, the last index is **`[n-1]`**.

- The C language does not check if the array is being accessed out f bounds: **buffer overflow**!!!

Arrays (example)

```
int array[10];

for(int i = 0; i < 10; i++) {
  array[i] = i * 5 - 3;
}
```

Arrays

- An array occupies a contiguous memory block with a size strictly necessary to store its elements.

```
int array[10];
/* sizeof(array) == sizeof(int) * 10 */
```

- The name of the array (without index) returns the base address of the array.

```
int array[10];
/* array == &array[0] */
```