COMPUTER ENGINEERING LABORATORY

Luigi Rizzo

luigi.rizzo@unipd.it

October 2023-January 2024

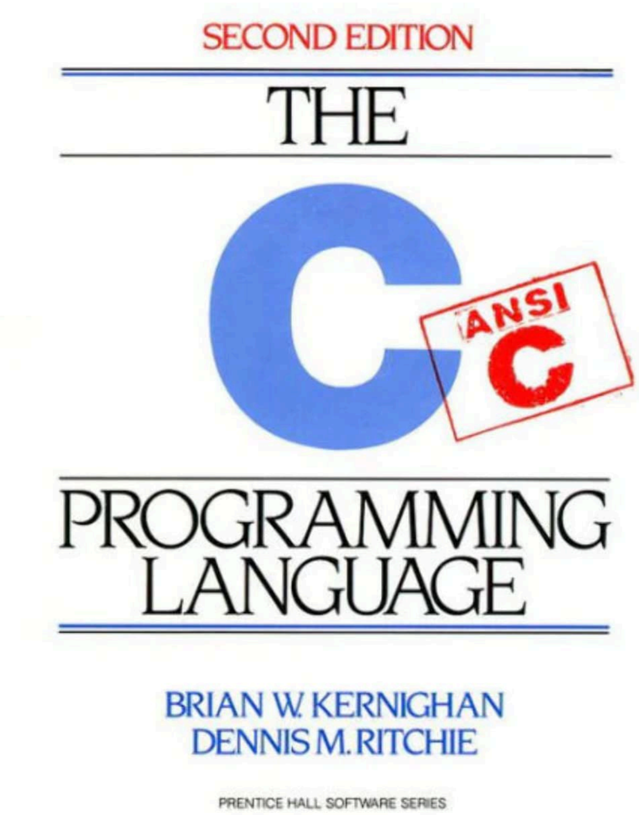
UNIVERSITÀ DEGLI STUDI DI PADOVA

Introduction

Resources

- Course Moodle:

  https://stem.elearning.unipd.it/course/.......



SECOND EDITION
THE
C
ANSI C
PROGRAMMING
LANGUAGE

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

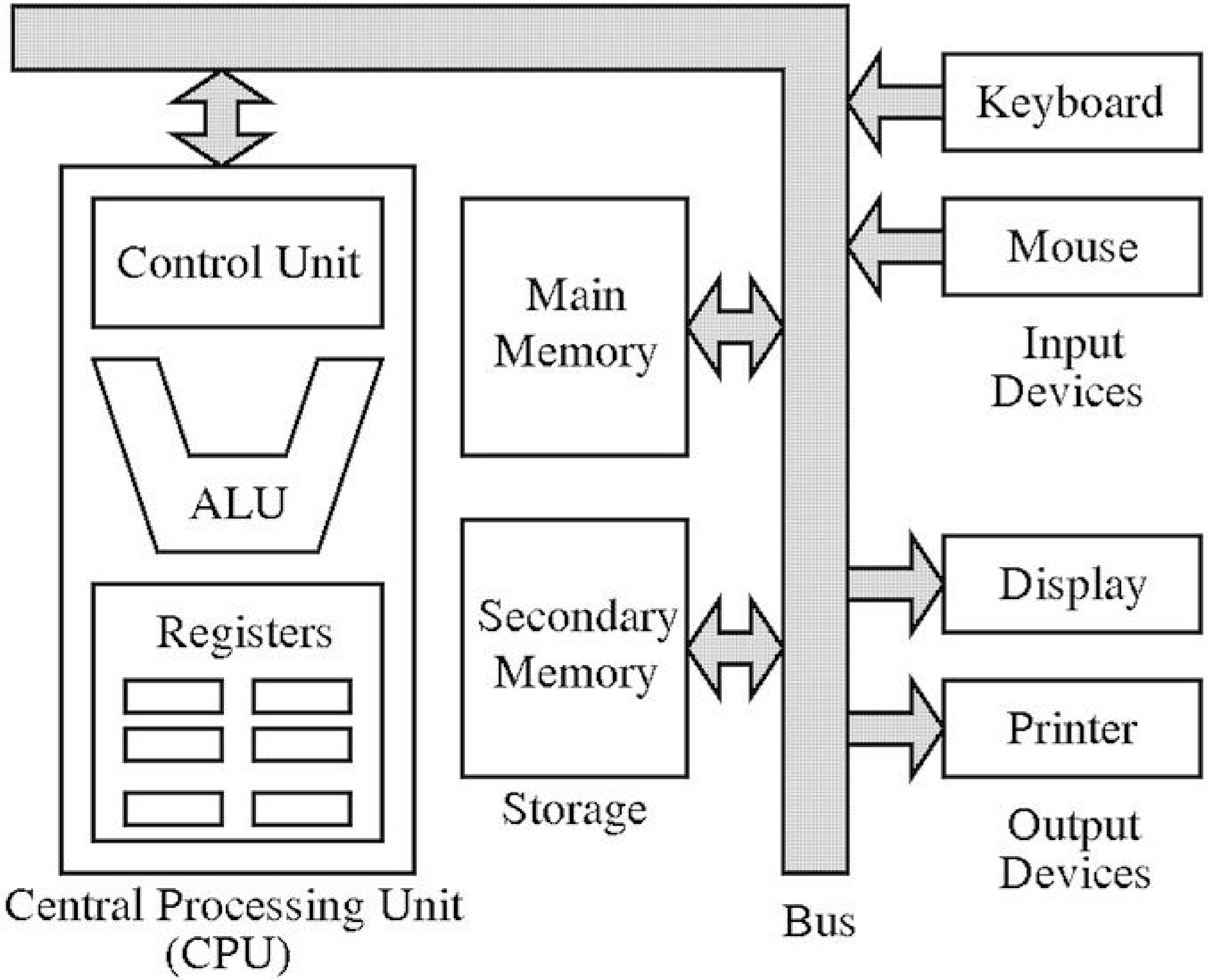
PRENTICE HALL SOFTWARE SERIES

The C Programming Language: ANSI C Version
Introduces the features of the C programming language, discusses data types, variables, operators, control flow, functions, pointers, arrays, and structures, and looks at the UNIX system interface

HW and SW computer architecture

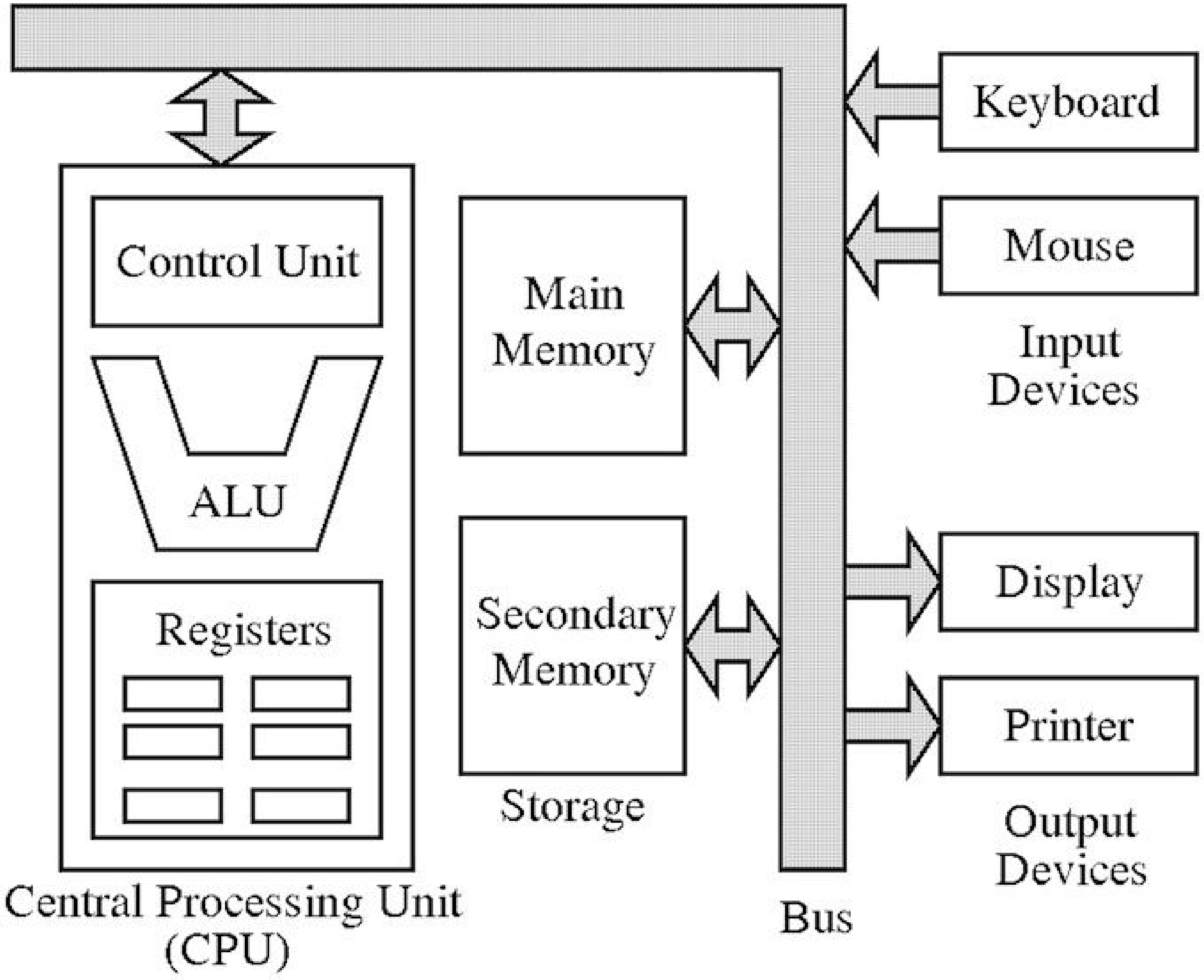
- First computers developed in the 1940s
- Large number of components increasingly smaller as the years pass
- All the information that reside there are encoded with sequences of 0s and 1s, in short with binary or base 2 numbers
- A program must reside in the computer's memory in order to run just like the data manipulated by the program itself
- Programs are also encoded by binary numbers
- Initially programs were written in binary (very difficult to write and correct them)

Von Neumann architecture



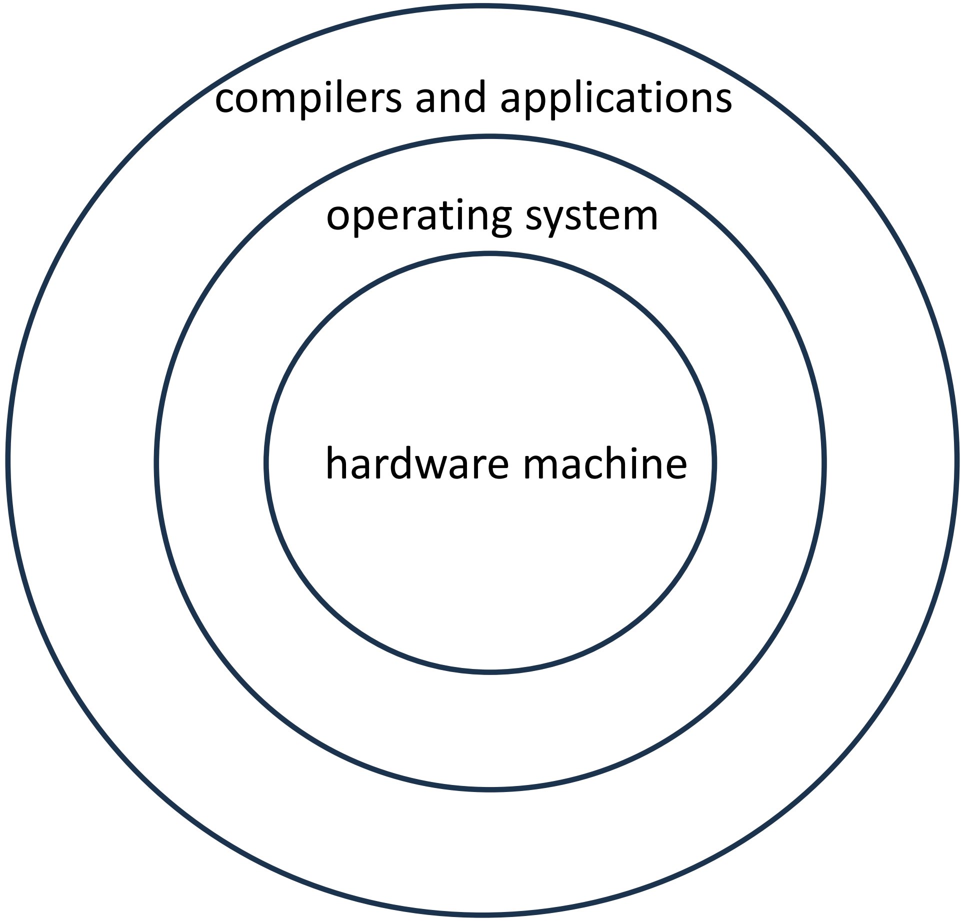
- RAM
  - sequence of bytes, 1 byte = 8 bits, 1 bit represents a 0 or a 1, each byte of memory has an address, from 0 to N-1, where N = total bytes of memory, 2^{10} bytes = kilobyte, 2^{20} bytes = megabyte…
  - RAM is used to hold information that can be written to or read from
- CPU
  - executes programs written in a very simple (machine) language repeating ADE cycle (access decoding execution)
    - accesses memory
    - decodes instruction
    - executes it

Von Neumann architecture



- Secondary memory
  - permanent memory where what is stored in the computer after it is turned off is preserved
- Input/output
  - devices through which data can be sent to the computer and vice versa, results can be output from the computer

Hardware and software machine



- Hardware machine:
  - pure electronic circuits made up of a very large number of components which perform the elementary operations of AND, OR, NOT

- Software machine:
  - operating system and other programs that allow simple use of the computer

high-level programming languages allow programmers to describe in a simpler way (than machine language) the operations that the CPU must perform, programs are translated into machine language by compilers

To be considered

- representation of integers, real numbers and characters
- elementary operations performed by machine languages
  - instructions that transfer contents from RAM (byte or word) to the CPU (register)
  - arithmetic operations (sum, subtraction, multiplication, division): operands in two CPU registers result in the first of the two
  - compare and jump instructions (comparison of the contents of two registers result in a third register)
  - instruction signaling the end of the program
- 3 different types of constructs
  - instructions that calculate results from initial values and assign them to some identifiers
  - test construct in which a condition is tested and depending on the result the calculation continues in a different way
  - the loop in which a sequence of instructions is repeated until a condition is satisfied

Data Types

Computer memory is made up of sequences of cells, bits, which can take one of the values {0,1}

The calculator cn process different types of information: numbers, characters, images, sounds, videos

- Information = Data + Interpretation

Since each type of information is represented by a sequence of bits in memory, we must always know the correct coding to read/write different types of data

For example, representation of positive integers

A sequence of digits forms a decimal number according to the following rule:

$528 = 8*10^0 + 2*10^1 + 5*10^2$

To determine the value of a positive binary number, we can use the same rule with base 2: $101011 = 1*2^0 + 1*2^1 + 0*2^2 + 1*2^3 + 0*2^4 + 1*2^5=43$

Signed integers representation

The number of different combinations of n bits is 2^n, a maximum of 2^n different numbers can be represented and the largest number that can be represented with n bits is $2^n - 1$ (because 0 is also counted)

Conclusion: the calculator can’t represent infinite numbers, therefore if the result of an expression gives a number beyond the maximum representable value, there will be an error (overflow)

The leftmost bit represents the sign: 0 = positive, 1 = negative

Positive numbers are represented in a “standard” way (with the rule just shown), using n bits

Negative numbers are represented “in 2's complement”, that is, you shall add 2^n to the number and then represent it in "standard“mode. Example for n = 5

-7 = 32 – 7 = 25 = 11001

Signed integers representation

Positive integers are represented inside the computer using a multiple of byte=8 bits, 4 or 8 bytes depending on the individual architecture

The sizeof(int) statement returns the number of bytes occupied by an integer

The limits.h file (#include <limits.h>) lists a series of useful numeric constants

E.g. INT_MAX: the maximum integer that can be represented in the computer or INT_MIN: the minimum representable integer

Knowing that the calculator uses 2's complement representation, if INT_MAX = 255 (1 byte) what is INT_MAX + 1 = ?

C language does not provide an automatic mechanism that checks whether the result of an addition is greater than INT_MAX, it is a programmer's job

Real numbers

Real numbers use floating point representation. The IEEE 754 standard provides various types of floating point numbers (single precision, 32 bit, and double precision, 64 bit)

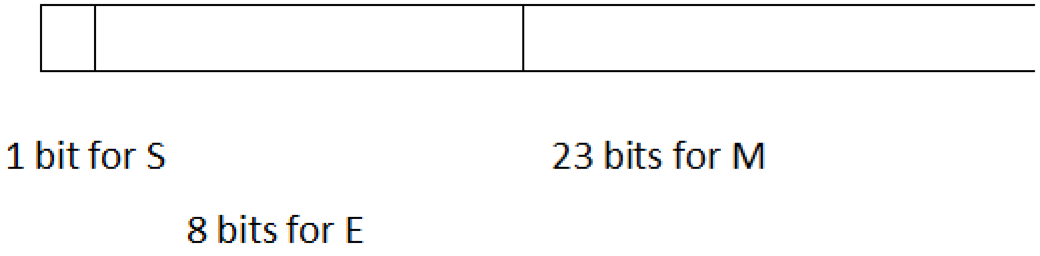
Single precision real type: float x

Double precision real type: double x, E.g. double x = 3.2;

Control the number of decimal places printed: printf(“%.3f”, 6.2781); ➔6.278.

But printf(“%.3f”, 6.2789);➔ 6.279 (the number is rounded when printed)

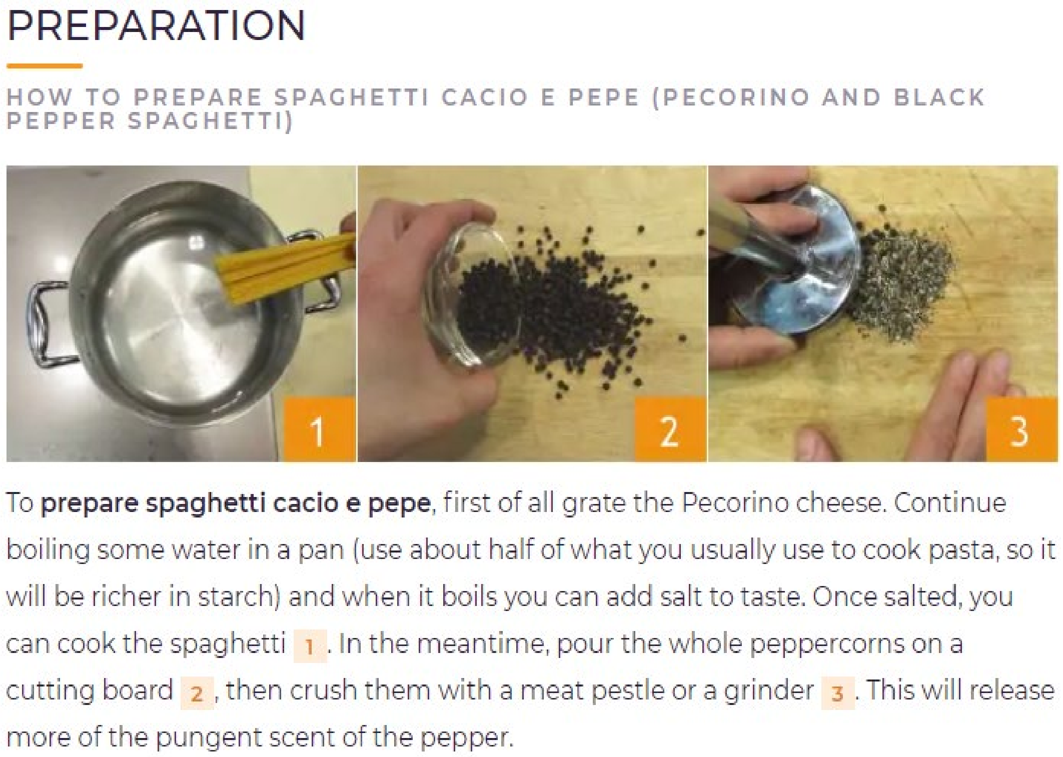
Since the reals do not have infinite precision, it may be that, comparing two equivalent real expressions, the operator of equality returns false due to approximations during calculations intermediates



Algorithm

- An orderly and finite set of elementary and unambiguous instructions for solving a problem.
- The concept of algorithm is general (there are no references to the calculator)

- 3 items
  1. The problem to be solved
  2. The sequence of instructions
  3. The solution to the problem
- 2 actors
  - who creates the instructions
  - who carries them out



PREPARATION

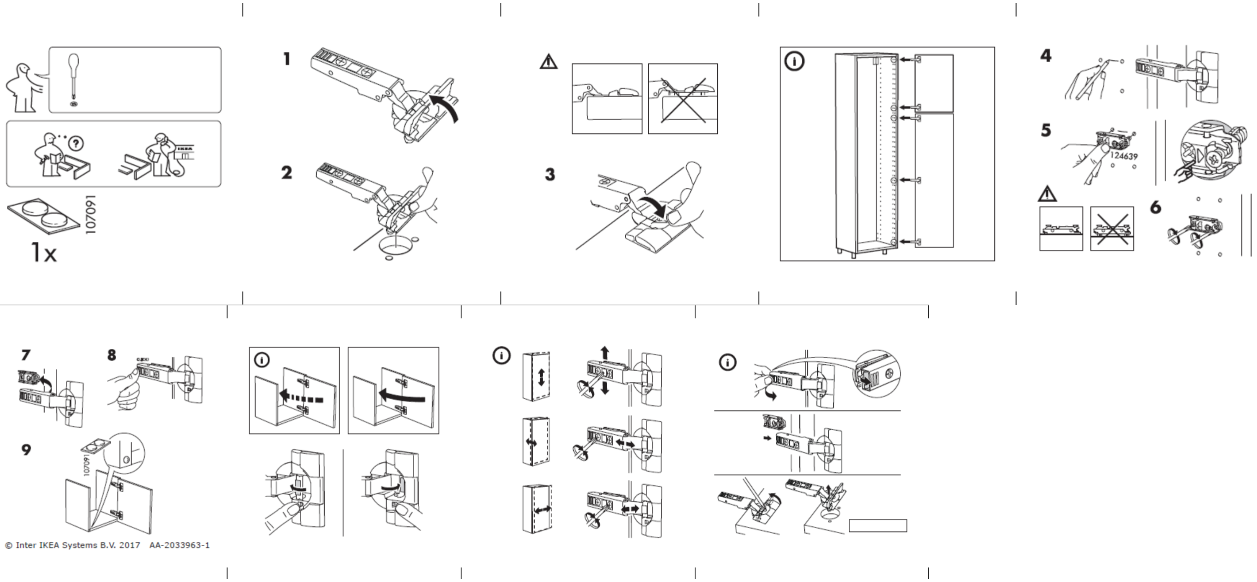
HOW TO PREPARE SPAGHETTI CACIO E PEPE (PECORINO AND BLACK PEPPER SPAGHETTI)

To **prepare spaghetti cacio e pepe**, first of all grate the Pecorino cheese. Continue boiling some water in a pan (use about half of what you usually use to cook pasta, so it will be richer in starch) and when it boils you can add salt to taste. Once salted, you can cook the spaghetti 1. In the meantime, pour the whole peppercorns on a cutting board 2, then crush them with a meat pestle or a grinder 3. This will release more of the pungent scent of the pepper.

Algorithm

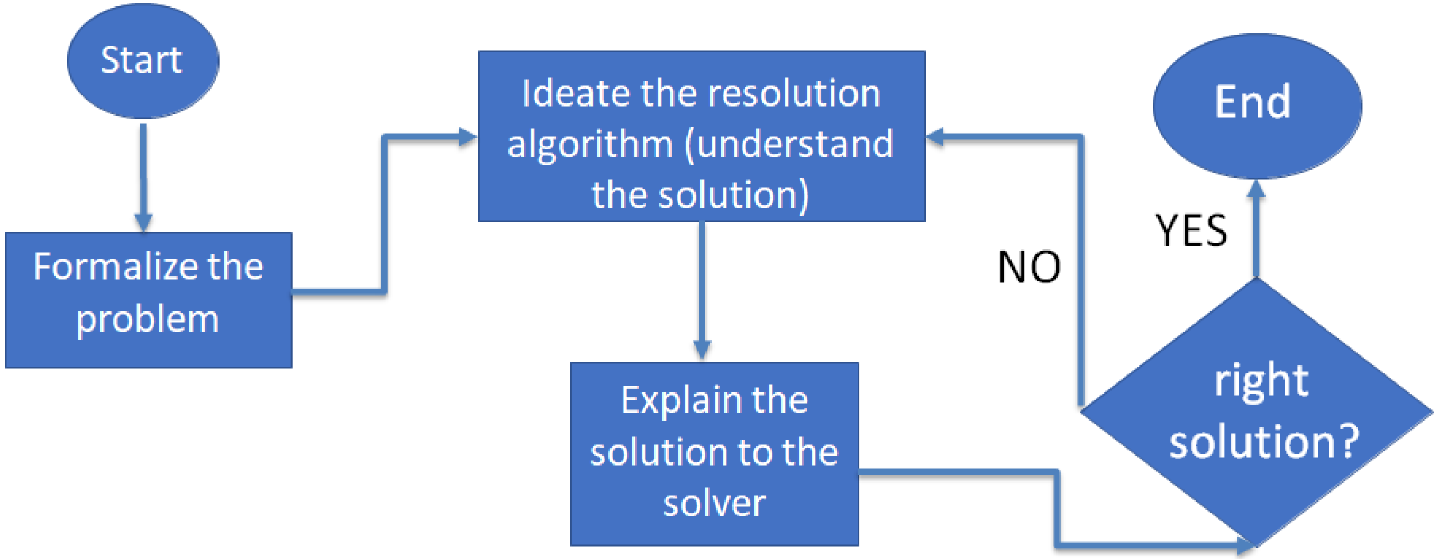
- The resolution of a problem often involves the resolution of a series of subproblems
  - problem and sub-problem are interchangeable concepts
  - we will call the problems/sub-problems functions

- Ordered and finished set of instructions, not ambiguous, to resolve a problem (not ambiguous for who or which executes)



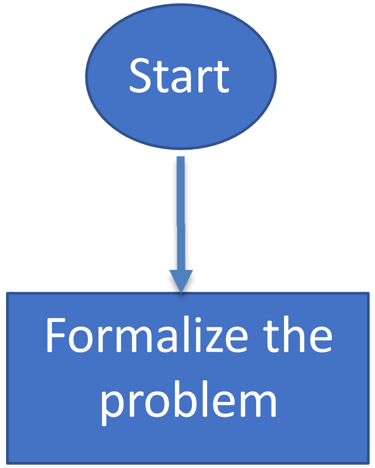
How create algorithms

1. What is the specific problem you want to solve or the task you want it to accomplish?
2. Finding starting and ending point are crucial to listing the steps of the process
   1. Decide on a starting point
   2. Find the ending point of the algorithm
3. List the steps from start to finish
4. Determine how you will accomplish each step
5. Review the algorithm


Start
Formalize the problem
Ideate the resolution algorithm (understand the solution)
Explain the solution to the solver
right solution?
NO
YES
End

Problem formalization

- Describing Inputs and Outputs
- Input: what is the input data, what is assumed about them
  - For which inputs do we guarantee that we can calculate the solution
  - We will use the term Preconditions for data assumptions
- Output: what our algorithm calculates
  - must be described unambiguously for whom will use our algorithm
  - in general we associate a Postcondition
    - an assertion (a formula that can be true or false) that expresses what does a code snippet calculate



Example problem
Calculate the square root of a number x
PRE: x>=0
POST: return y such that x=y*y

Ideate the resolution algorithm

- Find the solution and communicate it to myself in a language familiar to myself – for example Italian or English
  - focusing on the ideation part of the solution, there is more freedom about elementary operations (as long as they are understandable and as little ambiguous as possible)

Ideate the resolution algorithm (understand the solution)

Example problem

Sort a list of numbers in ascending order

10, 7, 14, 3, 18, 1

- List of "unordered" numbers and list of "ordered" numbers
- POST: the “sorted” list contains the numbers sorted in ascending order
- as long as the “unsorted numbers” list is not empty
  1. select the minimum number in the “unordered numbers” list
  2. move it to the end of the "sorted numbers" list

1, 3, 7, 10, 12, 18

Communicate the algorithm to the solver

- Programming: communicating algorithms to the computer
- Programming Language: set of (elementary) instructions that can be performed by the computer (and rules for their composition)
- How to implement an algorithm:
  1. Know the basic instructions made available by programming language
  2. Express the solution to the previous step in programming language
     - if the distance between the 2 languages is large, it may have elements of difficulty, you will learn implementation patterns
     - important to perform the two steps separately, to keep under control the difficulty

Explain the solution to the solver

Programming languages

Low-level languages (machine language):

- depend on the architecture
- instructions are really basic, very far from spoken language, so it is more difficult to think of complex algorithms directly in machine language

High-level languages:

- some new instructions are implemented in machine language
  - the user expresses a program through these new instructions
- new instructions are automatically translated in the machine language, via a program called a translator(compiler)

Explain the solution to the solver

Compilers

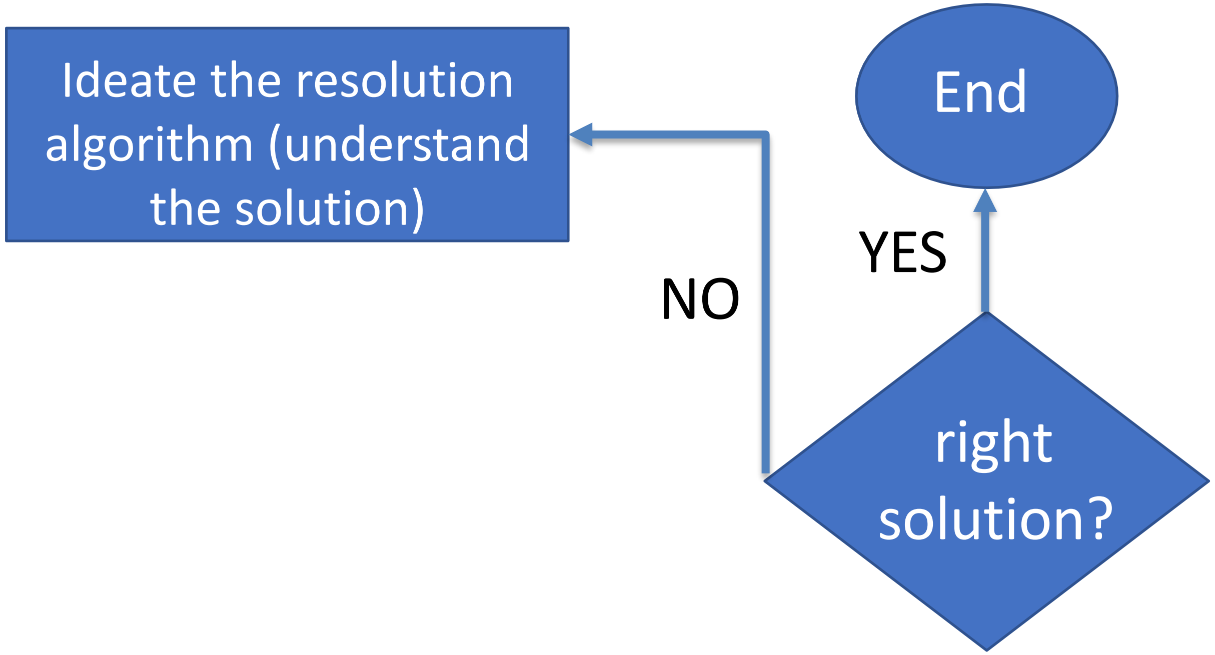
- If the architecture changes, we just need to provide a translator for the new one (and retranslate our programs)
- Desired characteristics for a language:
- Low complexity of the translator ← the number of new instructions implemented in the low-level language is reduced
- Power of language ← you can create additional “instructions” (functions) directly in the new language
  - these instructions are collected in libraries
  - knowing them allows you to save time and not reinvent something already available
- Examples of high-level languages: C, C++, Java, Python

C language

- High-level languages are a balance between two conflicting goals:
  - have the programming language do more checks to avoid user errors (never trust the programmer)
  - remain efficient
- C language: developed in 1970 by Ken Thompson and Dennis Ritchie
  - little memory available in computers, in the order of Kb with efficiency as the principal objective
  - it is close to low-level languages (some principles, for example the object-oriented programming, will be developed only later)
- Language specifications published in 1978
  - very popular, many compilers are created, even with different behaviors
  - a standard, ANSI C, is created and is up to date.

Correctness

- After you have implemented the algorithm, you shall provide evidence that the program is correct, that is, it achieves the postcondition
- The type of evidence depends on the context
  - unit tests (check that, for certain inputs we obtain the desired output)
  - correctness tests
- If the program is incorrect, you shall analyze the operation via the debugger



Programs are different

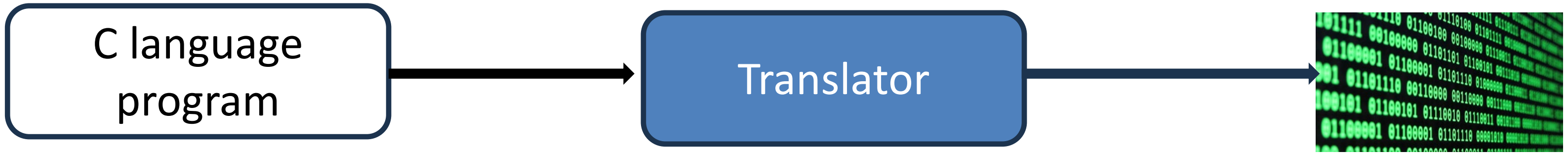
How different algorithms and different implementations can be evaluated?

- The most important criterion is the correctness
- How can a program be improved with equal correctness?
  - Efficiency (time and space) both at an algorithmic and implementation level
  - Organization: you create code (well-done) once, then you reuse such code
    - The problem is divided into sub-problems and an attempt is made to reuse the existing solutions implemented in the past
  - Style: the code shall be understandable to your colleagues and yourself after some months

Interpretation vs compilation

The computer can only execute programs in machine language that is a low level language (depending on architecture)

Together with the language specification, a tool that translates our programs into the language of the host machine is provided: the translator



Interpreter: translates a high-level instruction and carries it out immediately

Compiler: translates all the instructions together which are then executed together directly into machine language (C, C++)

There are intermediate solutions: compilation into bytecode and interpretation (Java)

Interpretation vs compilation

When using the interpreter:

- the program execution is slower
- requires translator to run the program
- having the translator and the source code, it can be run on any computer

When using the compiler:

- the program execution is faster (usually the code is optimized too)
- there is no need of a translator, but every time the program is changed it has to be recompiled
- the code must be compiled for each different architecture

In the case of C language: compiled language; restricted set of basic commands (many function libraries to be used), the compiler is "easy" to write, therefore quite portable

To start

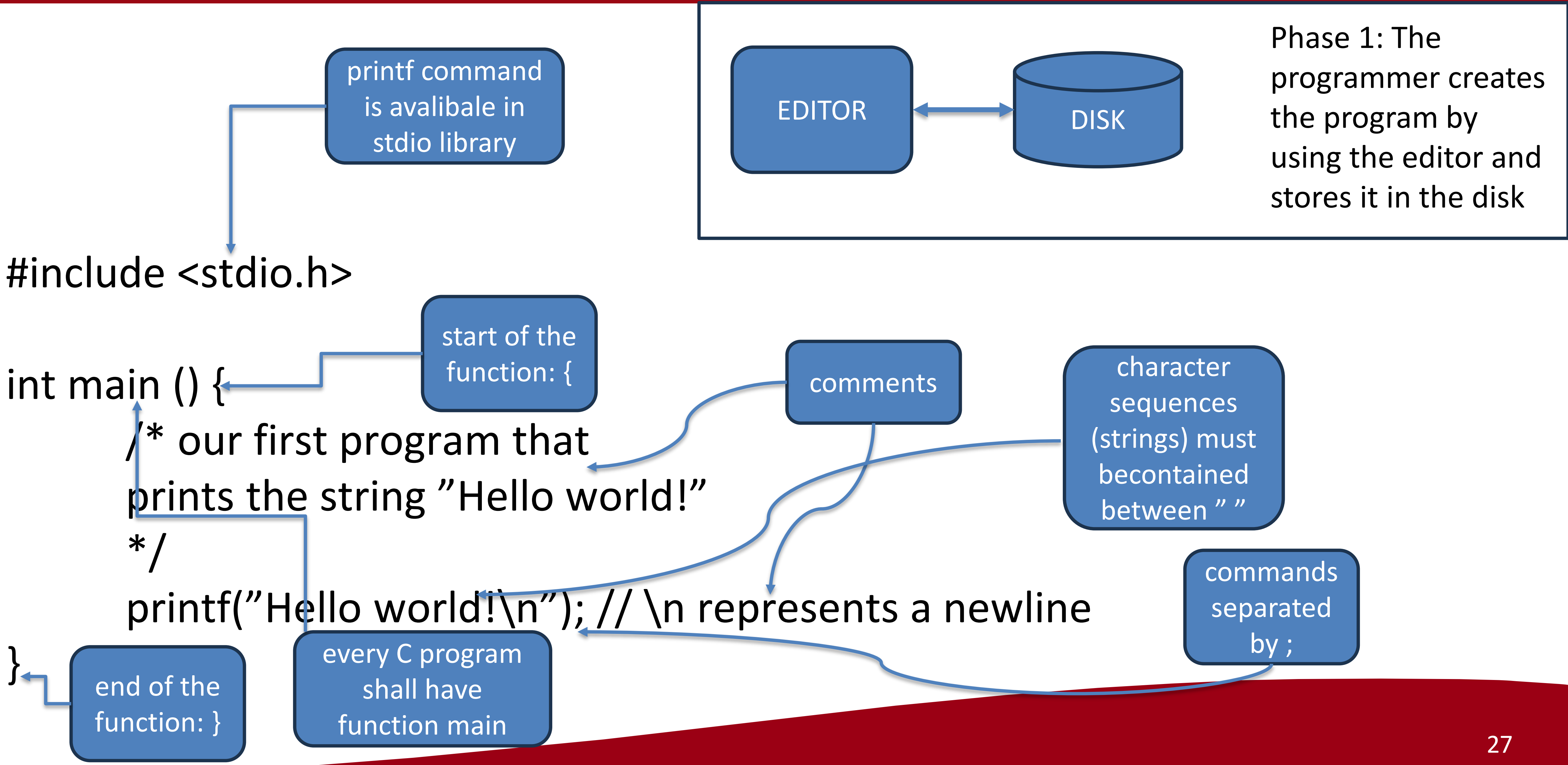
A quick introduction to C language illustrating the essential elements of the language in real programs, focusing on the fundamentals: variables and constants, arithmetic, control structures, functions, input/output elementary principles.

The first program to write is the same for every language: print the words “Hello world!”

What shall be done:

- create the program text
- compile it correctly
- run it
- know where the output is sent

Hello world



Phase 1: The programmer creates the program by using the editor and stores it in the disk

printf command is avalibale in stdio library

```
#include <stdio.h>

int main () {
        /* our first program that
        prints the string ”Hello world!”
        */
        printf(”Hello world!\n”); // \n represents a newline
}
```

start of the function: {

comments

character sequences (strings) must becontained between ” ”

commands separated by ;

end of the function: }

every C program shall have function main

Hello world

The way to run the program depends on the system you are using. As a specific example on a Linux system you shall create the source program in a file with .c extension like helloWorld.c and compile it with the command gcc

For example the command

```
gcc helloWorld.c –o helloWorld
```

if no errors have been made in writing the program, it will compile, creating an executable file with the name helloWorld

Running the program via helloWorld command will produce the output

```
Hello world!
```

In other system the rules may be a bit different but more or less very similar

Hello world: C language main characteristics

- A C program consists of functions and variables. A function contains statements that specify the computing operations to be done, and variables store values used during the computation
- Programmers may give functions whatever names they like, but ''main'' is special - every program begins executing at the beginning of main
- #include <stdio.h>
  - tells the compiler to include information about the standard input/output library; the line appears at the beginning of many C source files
- One method of communicating data between functions is for the calling function to provide a list of values, called arguments, to the function it calls. The parentheses after the function name surround the argument list. In this example, main is defined to be a function that expects no arguments, which is indicated by the empty list ( )
- The statements of a function are enclosed in braces { }