# 032CM - 2025

# PROGRAMMING FOR COMPUTATIONAL CHEMISTRY

## Introduction to Fortran

Gianluca Levi

gianluca.levi@units.it, giale@hi.is

Office: Building C11, 3rd floor, Room 329

Fall 2025

# What is a computer?

A computer is a machine that executes sequences of instructions (algorithms)

- **Store instructions and information** as data
- Execute instructions at **very high speed** (billions per second)

**Programs** tell the computer what sequence of instructions is required

- Written in **high-level language** (Fortran, Python)
- Translated into machine language (binary instructions) by a **compiler**/**interpreter**

**The computer only does what you tell it to do!**

# General Approach to Solving a Computational Problem

## 1. Problem Definition

- **Clearly** and **precisely** state the problem (makes development and implementation much easier)

# General Approach to Solving a Computational Problem

1. **Problem Definition**
   - **Clearly** and **precisely** state the problem (makes development and implementation much easier)
2. **Choose or design an algorithm**
   - How the problem will be solved **step-by-step**

# General Approach to Solving a Computational Problem

1. **Problem Definition**
   - **Clearly** and **precisely** state the problem (makes development and implementation much easier)
2. **Choose or design an algorithm**
   - How the problem will be solved **step-by-step**
3. **Implementation**
   - Translate the algorithm into code using a **programming language** of choice

# General Approach to Solving a Computational Problem

1. **Problem Definition**
   - **Clearly** and **precisely** state the problem (makes development and implementation much easier)
2. **Choose or design an algorithm**
   - How the problem will be solved **step-by-step**
3. **Implementation**
   - Translate the algorithm into code using a **programming language** of choice
4. **Verification/Testing**
   - Check that the problem is solved correctly
   - **Challenging** because the "correct" answer is typically not be known in advance (that's why we are writing a program in the first place!)

# Designing the Algorithm – Numerical example

**Goal:** Compute the square root of a number $S$ (find $x$ such that $x^2 = S$)

# Designing the Algorithm – Numerical example

**Goal:** Compute the square root of a number $S$ (find $x$ such that $x^2 = S$)

1. Start with an **initial guess** $g$
2. Check: Is $g \times g$ **close enough** to $S$ ?
   - If YES, stop. $g$ is a good approximation of the square root.
   - If NO, move on to step 3.
3. **Update guess** by averaging $g$ and $S/g$ : $g \leftarrow \frac{1}{2}\left(g + \frac{S}{g}\right)$

# Designing the Algorithm – Numerical example

**Goal:** Compute the square root of a number $S$ (find $x$ such that $x^2 = S$)

1. Start with an **initial guess** $g$
2. Check: Is $g \times g$ **close enough** to $S$ ?
   - If YES, stop. $g$ is a good approximation of the square root.
   - If NO, move on to step 3.
3. **Update guess** by averaging $g$ and $S/g$ : $g \leftarrow \frac{1}{2}\left(g + \frac{S}{g}\right)$

Try it for $S$ = 25 and initial guess 3

| $g$ | $g^2$ | $S/g$ | $(g + S/g)/2$ |
|---|---|---|---|
| 3.0000 | 9.0000 | 8.3333 | 5.6667 |
| 5.6667 | 32.1111 | 4.4118 | 5.0392 |
| 5.0392 | 25.3937 | 4.9611 | 5.0002 |
| 5.0002 | 25.0015 | 4.9998 | 5.0000 |

## Primitive constructs

- *English:* words
- *Programming:* literals (numbers, strings), variables, operators, statements

# Syntax and semantics

## Primitive constructs

- *English:* words
- *Programming:* literals (numbers, strings), variables, operators, statements

## Syntax (form, not meaning)

*English:*

- "Runs dog the fast" → **not syntactically valid**
- "The dog runs fast" → **syntactically valid**

*Programming:*

- 3+ * y → **invalid**
- 3 * y → **valid**

# Syntax and semantics

## Primitive constructs

- *English:* words
- *Programming:* literals (numbers, strings), variables, operators, statements

## Syntax (form, not meaning)

*English:*

- "Runs dog the fast" → **not syntactically valid**
- "The dog runs fast" → **syntactically valid**

*Programming:*

- 3* + y → **invalid**
- 3*(x + y) → **valid**

Syntactic errors (form mistakes) are **usually detected by the compiler/interpreter**

# Syntax and semantics

**Primitive constructs**

- *English:* words
- *Programming:* literals (numbers, strings), variables, operators, statement

**Semantics** (**meaning**)

*English* can be ambiguous:

- "I saw the man with the telescope"

*Programs* have **only one meaning**, but it may **not** match what we intended!

- average = number1 + number2 / 2

The program runs but gives an incorrect answer. Semantic errors the **most difficult type of error to find!**

# Fortran

**First high-level programming language** (1950s)

- FORmula TRANslator → designed for **science and engeneering**

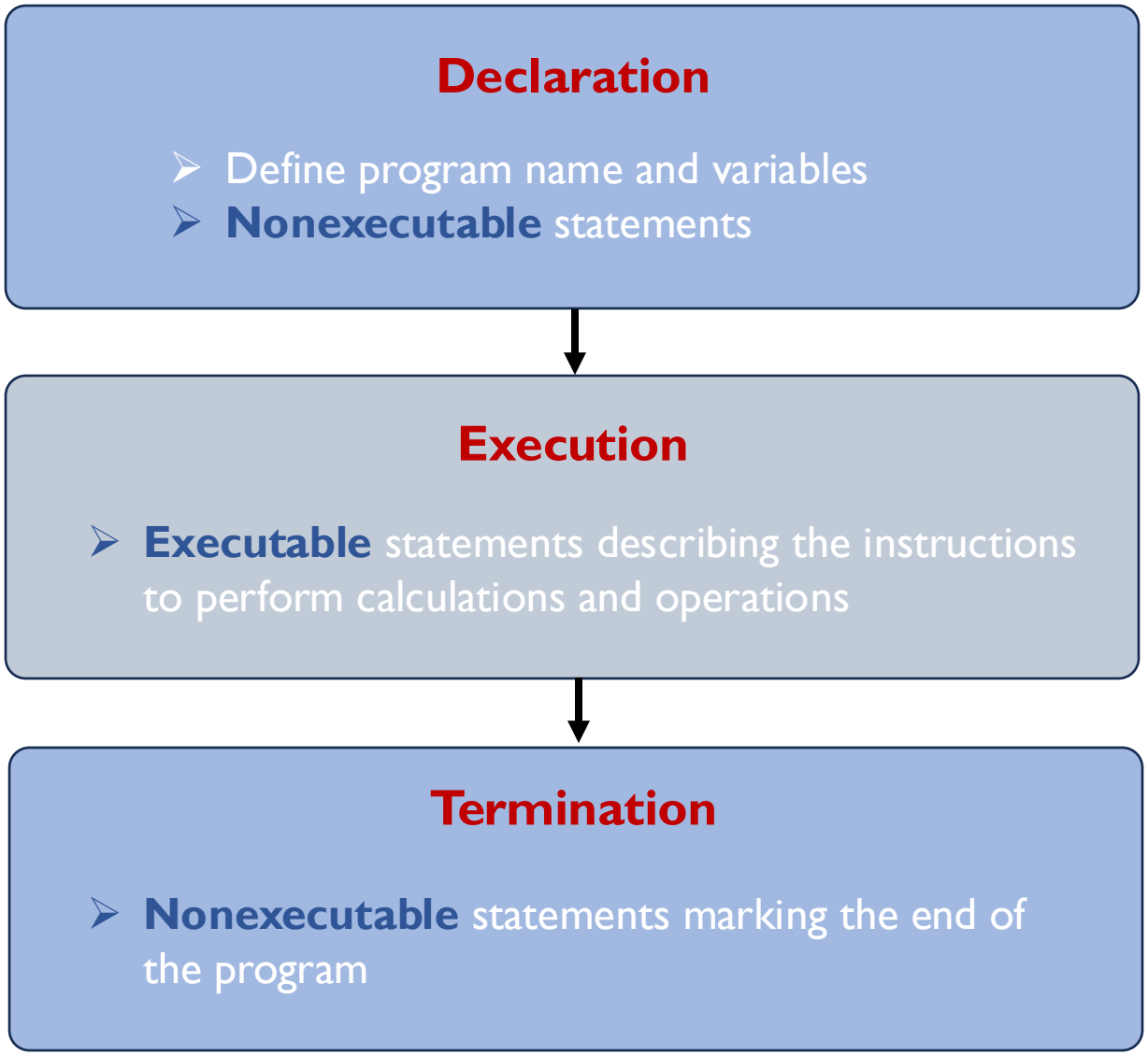
Used in HPC (**High Performance Computing**)

Used in major **quantum chemistry programs** (e.g., Gaussian, CP2K, ADF, Crystal, etc.)

**Fortran 90**

- Still compatible with decades of legacy scientific software

# General structure of a Fortran program



# Our first code

1. Log in to the Linux machine and open a terminal
2. Navigate to your directory
3. Create a new file with vim:

   >> vim hello_world.f90

4. Let's write our first Fortran code together!

# Compiling and linking the code

## Compiling

- Translates source code (.f90) into **machine instructions** (low-level language)
- Produces an object file (.o)

## Linking

- Links one or more object files to **system libraries**
- Connects code to **OS routines** for, e.g., reading from keyboard, printing to screen
- Produces the **final executable program**

Typically, the **final executable** can be run on any machine with the same architecture

# Compiling and linking the code

First **compiling**

>> ifort -c hello_world.f90 → produces hello_world.o

and then **linking**

>> ifort -o hello_world.x hello_world.o → produces the executable hello_world.x

or **compiling and linking at the same time**

>> ifort -o hello_world.x hello_world.f90 → produces the executable hello_world.x

# Assignment 1

## Problem 2

(a) For each code snippet below, identify whether the error is a syntax or semantic error (or there is no error). Briefly justify your choice.

```
program p
    integer :: x
    if (x > ) then
        write(*,*) 'ok'
    end if
end program p
```

```
program p
    integer :: x, y, z
    z = 3* + y
    write(*,*) z
end program p
```

```
program p
    implicit none
    real :: g, S, q
    g = 0.0
    S = 25.0
    q = S/g
    write(*,*) q
end program p
```

# Assignment 1

(b) Write a Fortran program that declares a character variable for your name, assigns your name to the variable, prints the message `"Hello World, my name is [your name]!"`.

*Hint:* Use the following statements:

```
character(len=20) :: name       ! variable declaration
name = "Student"                ! variable assignment
write(*,*) 'My name is ', name  ! printing
```

(c) Compile and runs the code from the previous question.

(d) Identify the different sections of a Fortran program in the code from the previous question: Declaration section, Execution section, Termination section.