032CM - 2026

PROGRAMMING FOR COMPUTATIONAL CHEMISTRY

Fortran: Constants and Variables, Data Types, Expressions and Assignments

Gianluca Levi

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

Office: Building C11, 3rd floor, Room 329

Fall 2026

Data objects

Programs manipulate **data objects**

Data objects have a **type**, which defined the type of operations we can perform with them

- 37 → number → can add, subtract, multiply, divide
- '37' → character (string of characters) → can concatenate, extract substring, but not divide by a number

Data objects

Programs manipulate **data objects**

Data objects have a **type**, which defined the type of operations we can perform with them

- 37 → number → can add, subtract, multiply, divide
- '37' → character (string of characters) → can concatenate, extract substring, but not divide by a number

In Fortran

Constants

Variables

Data objects

Programs manipulate **data objects**

Data objects have a **type**, which defined the type of operations we can perform with them

- 37 → number → can add, subtract, multiply, divide
- '37' → character (string of characters) → can concatenate, extract substring, but not divide by a number

In Fortran

Constants

- **Value does not change** during program execution
- Compiler stores the value in memory and uses it whenever referenced

Variables

Data types in Fortran

Built-in (intrinsic) **data types**

integer

- Whole numbers (…, -2, -1, 0, 1, 2, …)
- integer, parameter :: i = 42

real

- ***Approximate*** real numbers, stored in ***floating-point*** form
- real, parameter :: x = 3.141593
- real, parameter :: x = 2.998e8

complex

- Pair of real values (real part, imaginary part)
- complex , parameter :: z = (3.0, -2.0)

logical

- Values .true. or .false.
- logical , parameter :: flag = .true.

character

- Strings of characters, fixed length
- character(len=20), parameter :: name = 'Hello World!'

Data types in Fortran

Built-in (intrinsic) **data types**

Try it!

Declare the following constants and print their value:

- -17
- 5.9
- .false.
- 'Unknown error!'

Expressions

Expressions combine **data objects** and **operators**

- 4 + 3 ! sum

Execution gives a **value** to an expression, and that value has a **type**

- 4 + 3 → value 7, type integer

Fortran **evaluates** expressions and stores the value of the result (not the expression)

Expressions

Expressions combine **data objects** and **operators**

- 4 + 3 ! sum

Execution gives a **value** to an expression, and that value has a **type**

- 4 + 3 → value 7, type integer

Fortran **evaluates** expressions and stores the value of the result (not the expression)

Try it!

Define the constants below and carry out a division between them. Observe how the result changes depending on the operand types.

```
integer, parameter :: i1 = 7, i2 = 3
real,    parameter :: r1 = 7.0, r2 = 3.0

i1 / i2
r1 / r2
i1 / r2
r1 / i2
```

Expressions

Expressions combine **data objects** and **operators**

- 4 + 3 ! sum

Execution gives a **value** to an expression, and that value has a **type**

- 4 + 3 → value 7, type integer

Fortran **evaluates** expressions and stores the value of the result (not the expression)

Try it!

Define the constants below and carry out a division between them. Observe how the result changes depending on the operand types.

```
integer, parameter :: i1 = 7, i2 = 3
real,    parameter :: r1 = 7.0, r2 = 3.0

i1 / i2
r1 / r2
i1 / r2
r1 / i2
```

} **Automatic implicit type conversion**

Arithmetic operations

Basic operators

+ → addition

- → subtraction

* → multiplication

/ → division

** → exponentiation

Precedence: Exponentiation (**) > Multiplication (*) and division (/) come next, evaluated left to right > Addition (+) and subtraction (−)

Arithmetic operations

Basic operators

+ → addition

- → subtraction

* → multiplication

/ → division

** → exponentiation

Precedence: Exponentiation (**) > Multiplication (*) and division (/) come next, evaluated left to right > Addition (+) and subtraction (−)

Implied multiplication is not allowed → 7(3 + 4) must be 7 * (3 + 4)

Parentheses control order of operations → evaluated from inside out

Conversion functions

Explicitly convert between integers and real numbers

real(i) → converts integer i to real

int(x) → truncates real x to integer

nint(x) → rounds real x to nearest integer

ceiling(x) → smallest integer ≥ x

floor(x) → largest integer ≤ x

Try it!

Carry out a division and convert the result in real/int.

```
integer, parameter :: i1 = 7, i2 = 3
real,    parameter :: r1 = 7.0, r2 = 3.0

real(i1 / i2)
nint(r1 / r2)
ceiling(r1 / r2)
```

Many data objects, what to do with them?

300

'September'

False

5.2

True

1e-3

'Gianluca'

2

Many data objects, what to do with them?

Variables assignment

temp = 300

month = 'September'

stop = False

a = 5.2

flag = True

eps = 1e-3

name = 'Gianluca'

const = 2

Data objects

Programs manipulate **data objects**

Data objects have a **type**, which defined the type of operations we can perform with them

- 37 → number → can add, subtract, multiply, divide
- 'Time' → character (string of characters) → can concatenate, extract substring, but not divide by a number

In Fortran

Constants

- **Value does not change** during program execution
- Compiler stores the value in memory and uses it whenever referenced

Variables

- **Can change value** during program execution
- Compiler reserves a location in memory and references that memory location whenever the variable is used

Variable assignment

Variable assignment in Fortran

Equal sign (=) is an **assignment operator**, not a test for equality

- The expression on the **right-hand side** is evaluated
- The resulting value is **stored** in the variable on the **left-hand side**

```
variable_name = expression
```

i = i + 1 → take the current value of i, add 1, and store the result back in i

Variables in programming vs. mathematics

Mathematics

- a variable can represent many possible values (abstract, symbolic)
- i = i + 1 has no solution

```
a + 2 = b – 5
```

```
x * x = y
```

Programming

- a variable holds **one specific value at a time**
- i = i + 1 updates variable i

```
a = b – 5
```

```
m = 7
F = m * 9.81
```

Good coding practices

```
real :: circle_area, circle_circumference
real :: radius
real, parameter :: pi = 3.1415926535
radius = 2.5

! compute area
circle_area = pi * radius**2

! compute circumference
circle_circumference = 2.0 * pi * radius
```

```
! compute
A = 3.14159*2.5*2.5
C = 2*3.14159*2.5
```

Good coding practices

Choose **meaningful names**

Follow **naming conventions**

Make code **self-documenting**

Use **constants**

Easier to **read and understand**

Simpler to **debug and modify**

Collaboration-friendly for teams

Future-you will thank present-you!

```
real :: circle_area, circle_circumference
real :: radius
real, parameter :: pi = 3.1415926535
radius = 2.5

! compute area
circle_area = pi * radius**2

! compute circumference
circle_circumference = 2.0 * pi * radius
```



```
! compute
A = 3.14159*2.5*2.5
C = 2*3.14159*2.5
```



Assignment 2

Problem 1

(a) The following list contains valid and invalid Fortran constants. State whether or not each constant is valid. If the constant is valid, specify its type. If it is invalid, briefly state why it is invalid.

- `'Distance =`
- `0.5e-3`
- `0.5d-3`
- `.true.`
- `.maybe.`
- `2^3`
- `'Don't panic'`
- `"Don't panic"`
- `19,5`
- `"3.14159"`
- `3.14159`
- `10.`
- `"Pass / Fail"`

(b) For each expression below, determine whether it is valid or invalid in Fortran. If valid, predict the value and type of the result. Verify by writing, compiling, and running a Fortran program that executes the expression.

- `5 ** 2 / 3 + 1`
- `3.(-4.  / 2)`
- `(2.)  ** real(nint(-2.2))`
- `int(1.5) ** 2.  / floor(2.2)`

Assignment

(c) Assume a Fortran program with implicit none and these declarations:

```
integer  :: x, y
real     :: r
```

For each variable assignment below, determine if it is valid or invalid. If valid, say what it evaluates to. If invalid, provide a brief explanation.

- `x*y = 2 + 5`
- `xy = 2 + 5`
- `int(2.)  = x`
- `r = 7 / 2`
- `r = 7.  / 2`

(d) Given the atomic masses m_1 and m_2, the reduced mass μ of a diatomic molecule is

$$\mu = \frac{m_1 m_2}{m_1 + m_2}.$$

Write a Fortran program that reads the masses of the two atoms in atomic mass units, computes the reduced mass μ, and writes out μ in grams. Use meaningful variable names and include comments. What is the reduced mass of the CO molecule?