032CM - 2026

PROGRAMMING FOR COMPUTATIONAL CHEMISTRY

Fortran: Data representation

Gianluca Levi

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

Office: Building C11, 3rd floor, Room 329

Fall 2026

Relational and logical operators

Relational operators

General form: a1 op a2 ⟶ logical

== (.eq.)

/= (.ne.)

> (.gt.)

>= (.ge.)

< (.lt.)

<= (.le.)

Relational and logical operators

Relational operators

== (.eq.)

/= (.ne.)

> (.gt.)

>= (.ge.)

< (.lt.)

<= (.le.)

General form: a1 op a2 ⟶ logical

- Between two numbers or two characters
- If between integer and real, the **integer is converted to real**

Valid

7 < 2 + 8 ⟶ .true.

Invalid

100 >= 'Hello!'

'A' == 4

Relational and logical operators

Logical (combinatorial) operators General form: l1 .op. l2 ⟶ logical

.and.

.or.

.eqv. (equivalence)

.neqv. (nonequivalence)

.not.

Relational and logical operators

Logical (combinatorial) operators General form: l1 .op. l2 ⟶ logical

.and.

.or.

.eqv. (equivalence)

.neqv. (nonequivalence)

.not.

Valid	Invalid
(a > 0) .or. (b <= 10)	(a > 0) .and. 3

Precedence: **arithmetic** → **relational** → **logical (combinatorial)**

Relational and logical operators

Try it!

Write a Fortran program that computes x = 10 * 0.1 and y = 0.1 + 0.1 + ... (**10 times**). Compare the logical tests:

```
write(*,*) x == 1
write(*,*) y == 1
```

Relational and logical operators

Try it!

Write a Fortran program that computes x = 10 * 0.1 and y = 0.1 + 0.1 + ... (**10 times**). Compare the logical tests:

```
write(*,*) x == 1
write(*,*) y == 1
```

- 0.1 cannot be represented exactly in binary (only an **approximation** is stored)
- Repeated application of an operation can lead to **accumulation of rounding errors**

Never use == to compare floating-point numbers.
Instead, check if they are **within a small tolerance**.

Binary representation

The **memory** of a computer is made of billions of tiny **switches**

Each switch represents a **binary digit (bit)**:

- `0` = OFF
- `1` = ON

Numbers and all other types of data are represented in groups of bits:

- **Byte = 8 bits**, the smallest unit
- Larger units: KB, MB, GB, TB …

Binary representation

The **memory** of a computer is made of billions of tiny **switches**

Each switch represents a **binary digit (bit)**:

- `0` = OFF
- `1` = ON

Numbers and all other types of data are represented in groups of bits:

- **Byte = 8 bits**, the smallest unit
- Larger units: KB, MB, GB, TB …

Everything in a computer is ultimately a **sequence of bits (0 and 1)**

Binary representation

$$45_{10} = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 101101_2$$

Binary representation

$$45_{10} = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 101101_2$$

n bits can represent 2^n different values

- Half reserved for **negative numbers**
- Half for **zero and positive numbers**

} From -2^{n-1} to $2^{n-1} - 1$

Example: **4-byte integer (32 bits)**

- Smallest value: $-2^{31} = -2{,}147{,}483{,}648$
- Largest value: $2^{31} - 1 = 2{,}147{,}483{,}647$

Integers can be represented **exactly** within the allowed range

Binary representation

Fractions in general cannot be represented exactly in binary

Example:

$$0.1_{10} = 0.0001100110011001100110011\ldots_2$$

→ must be **approximated** in memory

Binary representation

Fractions in general cannot be represented exactly in binary

Example:

$$0.1_{10} = 0.0001100110011001 \ldots_2$$

→ must be **approximated** in memory

Real data are represented as **floating-point numbers:**

$$x = f \times 2^e$$

- f: mantissa (fraction)
- e: exponent (integer, scales the value)

Binary representation

Real data are represented as **floating-point numbers:** $x = f \times 2^e$

- f: mantissa (fraction)
- e: exponent (scales the value)

Precision = number of significant digits preserved.

Range = smallest to largest representable values

Binary representation

Real data are represented as **floating-point numbers:** $x = f \times 2^e$

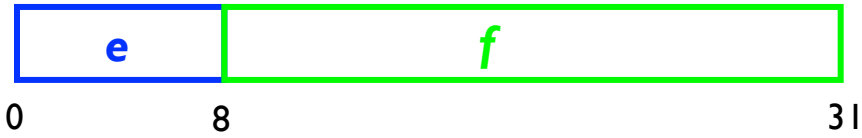
- f: mantissa (fraction)
- e: exponent (scales the value)

Precision = number of significant digits preserved.

Range = smallest to largest representable values

Single precision (32 bits) real*4

- 1 sign, 8 exponent, 23 mantissa
- ≈ **7 decimal digits** precision
- $\sim 10^{-38}$ to $\sim 10^{38}$ range



Binary representation

Real data are represented as **floating-point numbers:** $x = f \times 2^e$

- f: mantissa (fraction)
- e: exponent (scales the value)

Precision = number of significant digits preserved.

Range = smallest to largest representable values

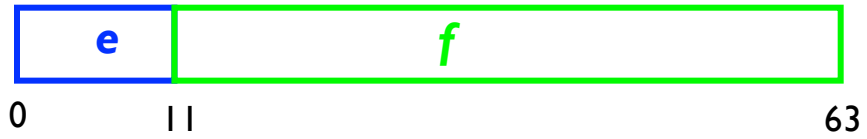
Single precision (32 bits) real*4

- 1 sign, 8 exponent, 23 mantissa
- ≈ **7 decimal digits** precision
- $\sim 10^{-38}$ to $\sim 10^{38}$ range



Double precision (64 bits) real*8

- 1 sign, 11 exponent, 52 mantissa
- ≈ **16 decimal digits** precision
- Range: $\sim 10^{-308}$ to $\sim 10^{308}$



Round-off error: extra digits are lost

- 0.1 stored in 32-bit is an approximation

Binary representation

Try it!

Write a Fortran program that computes x = 10 * 0.1 and y = 0.1 + 0.1 + ... (**10 times**) with ***double precision*** (real*8, 0.1d0). Compare the results to the number 1.