What is 1 kg + 0.1 m?

Arjen Markus, Brad Richardson

What is 1 kg + 0.1 m?

- Using units and dimensions of measurement requires care
- The above question is nonsense, of course ...
- Another example:

```
volume = 4.0/3.0 * pi * radius ** 2
mass   = density * volume
write(*,*) 'Mass: ', mass
```

Mistakes are not always so easy to spot -> we need programming tools!

Units versus dimensions

- Dimensions: length, mass, ..., population size, amount of money
- Units:
  - meter versus foot, kilometer versus mile
  - second versus hour
  - amount of dollars for one euro (or vice versa)

Often (always?) there is a linear relationship, but you must get the numbers right!

Units versus dimensions – caveat emptor!

Consider temperature:

$27\,^{\circ}\mathrm{C} + 10\,^{\circ}\mathrm{C} = 37\,^{\circ}\mathrm{C}$

Convert – naïvely, automatically – all contributions to kelvin:

$300\ \mathrm{K} + 283\ \mathrm{K} = 583\ \mathrm{K}$

You can add a temperature and a temperature difference!

Programming tools

A wide variety is available – but what are the limitations?

Four categories:

- Strictly define the units for variables
- Define the dimensions (and provide conversions)
- Track dimensions during execution
- Static analysis and infer the units/dimensions

Define the units of variables:

Example (style: Snyder, 2016, 2019):

```
unit :: foot, second
unit :: fps = foot/second
real, unit(foot) :: distance
real, unit(second) :: time
real, unit(fps) :: velocity
velocity = distance / time
```

Advantage: Compile-time checking – location of the cause of problems
Disadvantage: How to make library routines generic?

Define the dimensions:

Example (style: Richardson, 2020, roughly):

```
type(length_dim)  :: radius, length
type(density_dim) :: density
type(mass_dim)    :: mass
real, parameter   :: pi = 3.1415926 ! Approximately
mass = pi * density * radius ** 2 * length
```

Advantage: Define quantities and their dimensions – limited set

Disadvantage: The programmer/user is responsible for the right unit

Track the dimensions at run-time:

Example (style: Petty, 2001):

```
type(preal) :: x(2), t, dt
x(1) = 1.0 * u_meter             ! Start position in [m]
x(2) = 0.0 * u_meter / u_second  ! Start velocity in [m/s]
t = 0.0 * u_second               ! Time in [s]
dt = 1.0 * u_second              ! Time step in [s]
do i = 0,times
   t = i * dt
   call solve( x, t, dt, func )
   write(*,*) real(t), real(x)
enddo
```

Advantage: One type fits all

Disadvantage: An error is not caught at its origin

Static analysis:

Example (style: Contrastin et al. 2016):

```
program energy
   != unit kg :: mass
   != unit m/s**2 :: gravity
   real, parameter :: mass = 3.00, gravity = 9.81, height = 4.20
   != unit kg m**2/s**2
   real :: potential_energy
   potential_energy = mass * gravity * height
end program energy
```

Advantage: Units can be inferred – like height in the example

Disadvantage: Annotations may get stale, no direct support for unit conversions

Observations and conclusions

- Tools differ in usages they allow
- Tools differ in the impact on the source code
- Tools differ in the required language support

Subjective conclusion:

Proposal for units of measure (N2113) for Fortran provides a robust and flexible solution

- even if not all “typical” uses are directly supported.