What is 1 kg + 0.1 m? Handling dimensions and units of measure in a program

Arjen Markus, Brad Richardson

July 13, 2021

Abstract

Most, if not all, programs that perform numerical computations of some kind or other need to deal with the units of measure in which their variables are expressed. Many solutions for the general problem have been proposed, but the problem is more difficult than it looks at first sight. In this article we attempt to give insight in the various aspects and what the proposed solutions can and cannot do.

1 Introduction

In almost all programs that perform numerical computations the question arises whether the units of measure have been treated correctly. For instance: the sum of a variable that represents a mass and a variable that represents a length is undefined, yet for most computer languages, the variables would probably be simple floating-point numbers and it is the burden of the programmer to make sure that they are used in the right way, that is, two masses can be added and you can multiply or divide a mass by a length, but the sum or the difference has no meaning and so should not pop up in the source code.

Various solutions have been proposed for many different programming languages, but there does not seem to be a silver bullet (cf. Section 5). Examination of the publications on these solutions show several classes of solutions:

- Variables in the source code are decorated with static features that represent the units of measure [Pucci and Schonberg(s.a.), Moy et al.(s.a.)Moy, Becker and Regnath, Richardson(2020), Snyder(2019)]. The burden is on the programmer to provide these features. Depending on the language they are extensions of simple floating-point numbers or specific additions to the language itself, requiring support from the language tools (compilers and linkers).
- A preprocessor/static analyser is developed that helps identify violations of the arithmetic rules for units of measure [SimCon(2015)]. Typically, the programmer annotates the variables via special comments, so that the analysis can be refined.

- Rather than a static checker in the form of a compiler or a pre-processor, all (relevant) variables are defined to have a dynamic set of dimensions and the checks are made at run-time [Petty(2001)]. This clearly impacts the performance, but makes for a light-weight solution, because the developer of the package can use the standard facilities offered by the programming language.

Surprisingly, however, while the publications discuss in more or less detail the techniques used and how the user can use the solutions to their benefit, none seem to discuss the characteristics of units of measure and how these play a role in the average program. In fact, the distinction between *dimension* and *unit* is seldom made explicitly. This article is meant to explore precisely these characteristics and how they influence the design of a programming solution.

2 Dimensions and units

First of all, we need to be clear about what a dimension is and what a unit is. In science and engineering we typically rely on the *Système International* for defining dimensions and units, but in finance or economics other systems are of importance. But let us begin with the SI units – or dimensions.

A dimension is the general aspect of some measure. Here is an arbitrary definition found on the Internet: "a measurable extent of a particular kind, such as length, breadth, depth, or height." The SI distinguishes seven basic dimensions: mass (M), length (L), absolute temperature (Θ), time (T), amount of substance (N), electric current (I) and luminous intensity (J). Dimensions can be combined, for instance a surface area would be a squared length – L^2, or a density would be mass divided by length to the power three – M/L^3.

Many solutions that help verifying that the program code does the right thing as far as units in the expressions are concerned limit themselves to *dimensional analysis*. Small wonder, because the *units of measure* in which the actual value of a quantity is expressed, are legion: common units for length are meter, centimeter, inch, foot, kilometer, mile and so on. Each has the dimension of length, but each is numerically different. So, whether the solution allows you to work with expressions like 1 m + 10 inch very much depends on whehther it does *unit conversions* or not.

3 Use cases

Within a typical program, be it engineering or economical, you will find different usage patterns:

Input/output

The user should be able to provide the data in a convenient form, including a convenient unit. The program may work in meters, but to specify the

size of an atom in meters is not the most friendly possibility. Likewise for output.

Typical expressions

Expressions in which the variables and constants have dimensions and consequently units of measure come in various categories. Here are a few:

- *Mathematical formulae*:

  Consider the area of a circle:

  $$A = \pi R^2 \tag{1}$$

  In this formula the radius R has the dimension L (length) and the area A has the dimension L^2 (length squared). The constant π is essentially unitless. To deal with this formula, the system must either recognise that dimensions can be multiplied, forming new combinations, or it must be told that both L and L^2 are allowed.

  But if you were to calculate the mass of a cylinder from its radius R, height H and the density of the material ρ via the slightly more complicated formula:

  $$M = \rho\pi R^2 L \tag{2}$$

  the system now has to deal with a dimension ML^{-3} for the density ρ as well as L^2 and L. Depending on the way the formula is programmed and the way the compiler (or the executable program) handles this expression, we may encounter the following intermediate results:

Density multiplied by length	$ML^{-3} \cdot L$
Density multiplied by length squared	$ML^{-3} \cdot L^2$
Density multiplied by length cubed	$ML^{-3} \cdot L^3$
Length squared multiplied by length	$L^2 \cdot L$
Density times length squared multiplied by length	$(ML^{-1}) \cdot L$

  and possibly others. In short: expressions that involve variables and constants with units of measure can generate new combinations of dimensions.

  An obvious seeming solution is to define a specific type or an attribute representing the desired unit or dimension:

```
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
```

  At least the operations *multiply* and *exponentiation* should be overloaded. For the above formula you would also need to represent

*density * length* because a *density* gets multiplied by a *length*, as well as any of the other combinations.

While in a given program there will be a finite number of combinations, some support from the compiler or analyser would help: the tool in question would then gather all the units/dimensions for each term, rather than for each combination separately and create the final unit/dimension. This is essentially what the package by Petty does [Petty(2001)]: for each variable of the type `type(preal)` the exponents of the basic SI dimensions are traced, so that the multiplication of a *density* [M L^{-3}] and a length [L] results in an intermediate variable with dimension [M L^{-2}]. There is only a single user-defined type.

It may seem that the dimensions are raised to integer powers, but rational powers can also arise, as in the period T for a harmonic pendulum:

$$T = 2\pi\sqrt{\frac{l}{g}} \tag{3}$$

- *Empirical formulae*:

  While mathematical formulae are usually rather "crisp", with clear definitions of exponents and constants, that is not so for empirical or semi-empirical formulae. Here is an example, a relation between the water level h and the flow rate Q in some unspecified river:

  $$Q = 5.3 \cdot h^{2.7} \tag{4}$$

  The water level must be expressed in meters relative to a reference level and the flow rate is in m^3/s. If you were to express the water level in fathoms instead, the result would be completely wrong. It would even be wrong if you expressed the water level relative to another reference level than implied in the formula.

  This type of formula can only be used with a lot of background information, information that may be found in the comments or the documentation or simply in the heads of the users of the program, because they have been working with it for years.

  A correct way to write this formula is:

  $$Q = Q_0 \cdot \left(\frac{h}{h_{norm}}\right)^{2.7} \tag{5}$$

  where $Q_0 = 5.3\ m^3/s$ and $h_{norm} = 1\ m$, a normalisation factor, rendering the value to exponentiated dimensionless. Still, this leaves implicit the reference level against which the water level is measured.

- *Chemical formulae*:

  According to the Système Internationale (*SI*) amounts of a substance are to be expressed in *moles*, the number of molecules or atoms or ions divided by Avogrado's number. This is a very convenient way to do calculations regarding chemical reactions. For instance, this equilibrium between CO_2 and HCO_3^- in water:

  $$CO_2 + H_2O \rightleftharpoons HCO_3^- + H^+ \tag{6}$$

The reaction constant is (since the amount of water normally exceeds the amounts of the other substances by many orders of magnitude, it is essentially constant and can be absorbed in the reaction constant):

$$K = \frac{[CO_2]}{[HCO_3^-][H^+]} \rightarrow \text{mol}^{-1} \tag{7}$$

but there are three molecules/ions involved, so actually three different molar units.

Other units that need to be considered are:

- Using *ml/l* as the unit for gas dissolved in water. In particular, the concentration of dissolved oxygen is sometimes expressed in this unit. It should be interpreted as "milliliter of dissolved oxygen per liter of water".
- The concentration of salt in sea water, salinity is expressed in units like *ppt* (parts per thousand), *psu* (practical salinity units) or *1e-3*. This latter is the official unit prescribed by the "CF conventions" organisation [Anonymous(2021)]. Since it looks very much like a plain number, supporting something like that requires extra care, also on the part of the programmer, one would think.

- *Financial/economic expressions*:

  As engineers with physical or chemical background we are used to thinking in terms of the SI dimensions and units, or perhaps imperial units like inches and ounces. In other areas of computation very different dimensions are involved. Consider the gross domestic product, GDP. It is expressed as the amount of currency per year. And the exchange rate of dollars to euros or yens involve the ratio of two currencies. Formally, that may be a dimensionless value, but you should not use the wrong way around. A unit checking mechanism therefore might be required to distinguish the exchange rate dollar/euro from the rate euro/dollar.

  But even seemingly dimensionless numbers require some care: the number of inhabitants of a city is a mere count without a dimension, but it does not make sense to add a bare number like 100 to it, unless that number also represents some number of people.

  And what to think of percentages? Combining two percentages α and β should be done by adding them, but by a formula like:

$$\gamma = ((1 + \frac{\alpha}{100}) \cdot (1 + \frac{\beta}{100}) - 1) \cdot 100\% \tag{8}$$

Dimensional analysis

The very least we may require from a solution for the problem we consider here is that it checks the dimensional correctness of expressions. This means that the arithmetic for dimensions is checked and verified [Wikipedia(2021)]:

- Multiplication and division is possible with any combination of dimensions, but leads to a new combination.

- Addition and subtraction is possible only when the dimensions of the terms are identical.
- Exponentiation is – in general – limited to rational powers. One may argue whether, like in the case of empirical formulae, exponentiation to some arbitrary power implicitly means the unit is ignored.
- Elementary functions as *sin* and *log* do not take arguments which have a dimension.

Things become much more complicated if we also require conversion of units be taken into account. Quite apart from the vast number of units that is in use for the various basic dimensions or the combined dimensions that are derived from them, some units of measure do not adhere to the above general rules.

The first example is units of temperature other than *kelvin*. Suppose you were to add a temperature of $10°C$ and $20°C$ – or the same values in degrees Fahrenheit. The naïve result is $30°C$, or about $303K$, but if this was an allowable operation, we could also convert the two terms first to kelvin and then the result would be quite different: $283K + 293K = 776K$. You can, however, subtract two temperatures and you can add a temperature and a temperature difference.

A second example is the position on the globe: A position of 100° longitude differs 10° from a position 110° (same latitude), but the distance in kilometers depends on the latitude.

Ideally, a programming solution for checking the dimensions (and units) used in a program would take all such considerations into account.

General calculator

As an extension of the requirement on flexible input and output, we may also want the program to support interactive use, as in a general calculator. Feed it data with different units of length and it prints the sum in the unit of your choice.

State vector

Some of the solutions that have been suggested define a variable to have a fixed unit or dimension. But that presents a problem if you have an array of which each element can have a different dimension. Here is a simple example, a (forced) harmonic oscillator:

$$\frac{d^2x}{dt^2} - kx = F(t) \tag{9}$$

The system has two state variables, the position (L) and the velocity (LT^{-1}). It is natural to put these two state variables into an array and solve the system of equations via a standard ODE solver:

```
real :: x(2)

x(1) = 1.0
x(2) = 0.0
```

```
    do i = 1,times
        call solve( x, t, dt, func )
        write(*,*) t, x
    enddo
```

3.1 Dynamic units of measure

Another question that arises in every day programming practice is whether dimensions (or units) are static properties of a variable or not. [**?**] explicitly consider this possibility, as illustrated by the following code fragment:

```
! Sort the tables by weight
change_f = .TRUE.
DO WHILE (change_f)
    DO i = 2, n
        IF (weight(i) < weight(i-1)) THEN
            temp = weight( i )
            weight( i ) = weight( i-1 )
            weight( i-1 ) = temp
            temp = height( i )
            height( i ) = height( i-1 )
            height( i-1 ) = temp
            change_f = .TRUE.
        ENDIF
    ENDDO
ENDDO
```

The variable `temp` is used first as a weight (M) and then as a height (L). If the dimension of each variable were a static property – which allows checking at compile-time – then this usage pattern would be prohibited. In the proposal by [Snyder(2019)] a similar situation is described: here it is suggested to allow dimensionally polymorphic functions that inherit the dimension of the result from their arguments. One example, mentioned in [SimCon(2015)], is the *maximum* function – all of its arguments should have the same dimension (unit) and the result will be that dimension (or, more precisely, unit).

4 Design questions

The requirements we can deduce from the above use cases and from general considerations can be divided into two categories. On the one hand we have requirements that are necessary for the feature to be at all useful, and on the other hand we have requirements that are nice to have but are not actually essential or might hinder some usages.

Here is a list of both types:

- Requirement 1:
  It is essential that violations of the rules that gouvern units and dimensions be clearly identified in the source code. Suppose the program calls a routine that expects an acceleration as argument

but gets a velocity. When this mistake can be detected *statically*, the reported error can pinpoint the exact location. If a variable receives its dimension/unit as a result of some calculation, it can be very hard to identify that location.

- Requirement 2:
  The feature must be able to deal with different units for the same dimension. A classical example is the use of input in feet as the unit of length, where the program itself (or the library it uses) prefers meters.
- Requirement 3:
  The programmer should be able to define their own dimensions, not only units. This is important, because the *SI* dimensions are but a subset of the dimensions that are really in use. This could be within the context of physical and chemical applications – think of "ml oxygen per liter water" but also of financial dimensions.
- Requirement 4:
  If the feature does not support arrays whose elements have different dimensions/units of measure, then certain use patterns are not possible. That may or may not be a breaking requirement for the feature.

Notes – to be worked out:
Ideally: compile-time checks – but completely doable?
Input and output – respect the units
Distinction between units and dimensions
Should variables have a fixed dimension or a fixed unit?
What about performance if we need to do run-time checks?
Burden on the programmer
Error reporting

5 Existing solutions

TODO

References

[Anonymous(2021)] Anonymous, 2021. Cf standard name table. URL: `http://cfconventions.org/Data/cf-standard-names/77/build/cf-standard-name-table.html`.

[Moy et al.(s.a.)Moy, Becker and Regnath] Moy, Y., Becker, M., Regnath, E., s.a. Physical units pass the generic test. URL: `https://blog.adacore.com/physical-units-pass-the-generic-test`.

[Petty(2001)] Petty, G.W., 2001. Automated computation and consistency checking of physical dimensions and units in scientific programs. Software practice and experience 31, 1067–076. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.401`, doi:10.1002/spe.401.

[Pucci and Schonberg(s.a.)] Pucci, V., Schonberg, E., s.a. Implementation of a simple dimensionality checkng system in ada 2012. URL: https://blog.adacore.com/uploads/dc.pdf.

[Richardson(2020)] Richardson, B., 2020. quaff - quantities for fortran. make math with units more convenient. URL: https://gitlab.com/everythingfunctional/quaff.

[SimCon(2015)] SimCon, 2015. Research - physical units and dimensions. URL: http://www.simconglobal.com/units_and_dimensions.html.

[Snyder(2019)] Snyder, V., 2019. Physical and engineering units of measure. URL: https://github.com/j3-fortran/fortran_proposals/issues/50.

[Wikipedia(2021)] Wikipedia, 2021. Dimensional analysis. URL: https://en.wikipedia.org/wiki/Dimensional_analysis.