

# Re: increasing width

---

*Source:* <http://coding.derkeiler.com/Archive/Fortran/comp.lang.fortran/2007-03/msg00697.html>

---

- *From:* Dick Hendrickson <[dick.hendrickson@xxxxxxx](mailto:dick.hendrickson@xxxxxxx)>
  - *Date:* Sat, 10 Mar 2007 22:43:20 GMT
- 

Lane Straatman wrote:

I've been writing a fortran95 prog and was getting nine significant figures for a real-valued calculation.

Some fellow over in sci.math.num-analysis tells me I can get at least 6 more sig figs, if instead of declaring as type real, I declare as real(8). My compiler complains:

```
numsci1.F95(3) : error 62 – Invalid KIND specifier  
warning : comment 981 – Specifying the kind of the type REAL with the constant '8' is  
non-portable – 'SELECTED_REAL_KIND(6,37)' would be better.
```

The error is from the line:

```
real(8) :: total, term
```

, which I get every time I want to declare real(8). The warning that it issues would seem to point to the way out. How do I make a real variable with 'SELECTED\_REAL\_KIND(6,37)'?

--

LS

The problem is that the standard doesn't specify the notation used to declare single or double precision variables. Many, if not most, compilers use a byte oriented notation. REAL(4) for single precision, REAL(8) for DOUBLE. Some use 1 and 2 in place of 4 and 8. The 1, 2, 4, and 8 are called "kind values", because they denote the kind of the variable :). The standard doesn't hardwire in specific numeric values because different processors might easily support more than one "kind" of single precision variable. Maybe full IEEE floating point and a faster proprietary version of floating point values.

So, there are a bunch of intrinsic functions that let you control the kind in a processor dependent way.

SELECTED\_REAL\_KIND(precision, range) is an intrinsic that will return a kind value that is appropriate for the precision and range you chose. Normally, people declare the kind values they need in a module and then use those values everywhere else. Something like:

## Re: increasing width

```
module precision_names
integer, parameter :: sp = selected_real_kind(6,37)
integer, parameter :: dp = selected_real_kind(13)
end module precision_names
```

```
program whatever
use precision_names
real(sp) :: x,y,z
real(dp) :: a,b,c
```

Then x, y, and z will be "single precision" with at least 6 digits of precision and an exponent range of at least 37. a, b, and c will have at least 13 digits of precision, and an unspecified (default) exponent range. These values pretty much correspond to single and double on a 32 machine.

If you want all of your values to be double precision, it's easy to change the definition of "sp". Later on, if you move your code to a 64 bit word machine, you can change the definition of sp back and probably get faster execution.

Two warnings, it's unlikely you were getting nine digits for your real valued calculations. Your PRINT statements might display that many digits, but 32 bit single precision only calculates to 7 or so digits. The rest is noise.

The other thing you need to remember is that Fortran constants are determined from the FORM of the constant, not the CONTEXT and that most interesting numbers are not exactly representable as real numbers. So, with the definitions above something like

```
x = .1
a = .1
```

won't do what you want. In both cases, the ".1" is a single precision value, accurate to 6 or 7 digits. But, in the second case, you want a double precision value accurate to about 13 digits. you will need to affix the kind value to all of the constants.

```
x = .1_sp
a = .1_dp
```

alerts the compiler to use the correct representation for the constants.

Sorry, this is a long answer to a short question.

Dick Hendrickson

.