Precision (kind) constants for F95 on Pentium machines

Aleksandar Donev

June 25, 2002

1 Module *Precision*

The module *Precision* contains definitions of certain kind parameters for integer, real and boolean numbers. It is of course very platform dependent. I support each compiler separately, though something like the Portability Project *Standard_Types* module could also be used. Here I use the Fortran 77 extension INTEGER*4 (and alike) of specifying explicit byte-sizes to get the correct kind parameter for the types I commonly use.

Important kind parameters here are:

i_32 **and** *i_64* kinds for 32- and 64-bit IEEE 2's-complement integers (used in certain codes where bit-based operations are performed, such as random-number generation or Hilbert-curve generation)

i_word, *r_word* **and** *l_word* kinds of default INTEGER and REAL numbers.

i_wp, *r_wp* **and** *l_wp* kinds that represent the **working precisions** for integers (only for numbers that may grow too large), reals (single or double—very important for optimization) and boolean (default LOGICAL on Pentiums is a whole word, which is wasteful on memory, but usually faster). These are most important to this library as the user will most commonly declare variables of this kind.

```
"WEAVE.f90" 1.1 ≡
  @m _DeclareSampleVar(_type, _n_bytes)
      _type * _n_bytes :: _type@&_var_@&_n_bytes

  @m _DeclareSampleVar_1_to_8(_type)
          // Power of two can be used as an integer byte size on IA32, up to 64-bit integers
      _DeclareSampleVar(_type, 1)    // Pentiums are byte-addressable
      _DeclareSampleVar(_type, 2)
      _DeclareSampleVar(_type, 4)
      _DeclareSampleVar(_type, 8)

  @m _DeclareSampleVar_4_to_16(_type)
          // Power of two can be used as an float byte size on IA32, from single to quad precision
      _DeclareSampleVar(_type, 4)
```

```
   _DeclareSampleVar(_type, 8)
   _DeclareSampleVar(_type, 16)


 MODULE Precision
   IMPLICIT NONE
 PRIVATE

     /* In order to make this file as portable as possible, let's declare summy dummy (private) variables
        with a specified byte size (using the F77 extension). One might want to make these public too.
        */
   _DeclareSampleVar_1_to_8(INTEGER)
@#if (F90C ≡ nf95)
     // It seems that NAG on Linux does not have quadruple precision?
   _DeclareSampleVar(REAL, 4)
   _DeclareSampleVar(REAL, 8)
@#else
     // The other compilers do have quadruple precision as well
   _DeclareSampleVar_4_to_16(REAL)
@#endif
   _DeclareSampleVar_1_to_8(LOGICAL)
   _DeclareSampleVar(CHARACTER, 1)    // One-byte characters are all we need

     /* Now declare descriptive names for the integer, real, logical and character kinds commonly used.
        A value of −1 means that the kind is not supported: */
   INTEGER, PARAMETER, PUBLIC :: i_byte = KIND(INTEGER_var_1),
         i_short = KIND(INTEGER_var_2), i_sp = KIND(INTEGER_var_4),
         i_dp = KIND(INTEGER_var_8), i_word = KIND(0)    // Integers
   INTEGER, PARAMETER, PUBLIC :: r_sp = KIND(0.0 · 10^0), r_dp = KIND(0.0 · 10^0D),
         r_word = KIND(0.0)    // Real numbers
@#if (F90C ≡ nf95)
   INTEGER, PARAMETER, PUBLIC :: r_qp = −1    // There is no such kind
@#else
   INTEGER, PARAMETER, PUBLIC :: r_qp = KIND(REAL_var_8)
@#endif
   INTEGER, PARAMETER, PUBLIC :: l_byte = KIND(LOGICAL_var_1),
         l_short = KIND(LOGICAL_var_2), l_word = KIND(T)    // Boolean values—no
         need for long booleans, only short ones
   INTEGER, PARAMETER, PUBLIC :: c_byte = KIND(CHARACTER_var_1), c_ascii = KIND(' ')
         // ASCII characters

     // Sometimes it is neccessary to know the specific representation used (IEEE only here):
   INTEGER, PARAMETER, PUBLIC :: i_32 = KIND(INTEGER_var_4), i_64 = KIND(INTEGER_var_8)
     // 2's complement IEEE integers
   INTEGER, PARAMETER, PUBLIC :: r_32 = KIND(REAL_var_4), r_64 = KIND(REAL_var_8)
     // IEEE floating-point numbers

     // The working precision can be changed via FWEB macros at compile time:
   INTEGER, PARAMETER, PUBLIC :: r_wp = R_WP, i_wp = I_WP, l_wp = L_WP
         // Working precisions

 END MODULE Precision
```

§. `@I "Fortran.formatting.hweb"` *Section(s) skipped...*

§. `@I "Fortran.macros.hweb"` *Section(s) skipped...*

§. `@I "Fortran.F2x.hweb"` *Section(s) skipped...*