Un módulo que parametriza las clases de tipos de datos reales

Pablo Santamaría

v0.1 (Junio 2009)

Debido a la naturaleza finita de una computadora, no todos los números reales pueden ser representados exactamente, sino solamente un cierto conjunto finito de ellos, conocidos como *números de punto flotante*. Así todo número real es representado por el número de punto flotante más próximo de este conjunto, lo cual introduce un *error de redondeo* que hace inexacta por naturaleza toda operación con números reales en la computadora. Con el fin de evitar la proliferación de diversos sistemas de puntos flotantes incompatibles entre sí, a fines de la década de 1980 se desarrolló la norma o estandar IEEE754/IEC559[1] la cual es implementada en todas las computadoras actuales. Esta norma define dos formatos para la implementación de números de punto flotante en la computadora: uno de *precisión simple*, y otro de *precisión doble*. Mientras que el primero garantiza 6 dígitos decimales significativos, el segundo asegura unos 15 dígitos significativos.

Ahora bien, todos los compiladores Fortran admiten, al menos, dos *clases* para los tipos de datos reales: el primero, de simple precisión y, el segundo tipo, de doble precisión tal como se especifica en la norma IEEE754/IEC559. Para declarar el tipo de clase de una variable real se utiliza la sintaxis:

```
REAL(KIND=número de clase) :: nombre de variable
```

donde el *número de clase* es un número entero que identifica la clase de real a utilizar. Este número es dependiente del compilador utilizado. Por ejemplo, para el compilador `gfortran`, es 4 en el caso de simple precisión y 8 para doble precisión. Si no se especifica la clase, entonces se utiliza la clase por omisión, la cual es la simple precisión para el compilador `gfortran`.

Dado que el número de clase es dependiente del compilador es recomendable asignar los mismos a constantes con nombres y utilizar éstas en todas las declaraciones de tipo. Esto permite asegurar la portabilidad del programa entre distintas implementaciones cambiando simplemente el valor de las mismas.

```
INTEGER, PARAMETER :: SP = 4 ! Valor de la clase de simple precisión
INTEGER, PARAMETER :: DP = 8 ! Valor de la clase de doble precisión
...
REAL(KIND=SP) :: variables
REAL(KIND=DP) :: variables
```

Por otra parte, las constantes reales presentes en el código son declaradas de una dada clase agregando a las mismas el guión bajo seguida del número de clase. Por ejemplo:

```
34.0            ! Real de clase por omisión
34.0_SP         ! Real de clase SP
124.5678_DP     ! Real de clase DP
```

[1] IEEE = Institute of Electrical and Electronics Engineeers, IEC = International Electronical Commission

Es importante comprender que, por ejemplo, `0.1_SP` y `0.1_DP` son números de punto flotante *distintos*, siendo el último guardado internamente con una mayor precisión.

Una manera alternativa de especificar la clase de los tipos de datos reales y que resulta *independiente del compilador y procesador utilizado* consiste en seleccionar el número de clase vía la función intrínseca `SELECT_REAL_KIND`. Esta función selecciona automáticamente la clase del tipo real al especificar la precisión y rango de los números de punto flotante que se quiere utilizar. Para simple y doble precisión la asignación apropiada es como sigue:

```
INTEGER, PARAMETER :: SP = SELECTED_REAL_KIND(6,37)   ! Clase de simple precisión
INTEGER, PARAMETER :: DP = SELECTED_REAL_KIND(15,307) ! Clase de doble  precisión
```

Ahora bien, resulta de interés, poder escribir los programas de manera de que puedan ser compilado con variables ya sea de una clase u otra según se requiera, modificando el código mínimamente. Una manera de lograr ésto consiste en utilizar un módulo para definir la precisión de los tipos reales y luego, en el programa, invocarlo especificando el tipo de clase vía un *alias*, al que designaremos `WP` (por *working precisión*, precisión de trabajo), el cual es utilizado para declarar todos los tipos de datos reales (variables y constantes) en el programa. Específicamente, definimos el módulo `precision` como sigue:

```
MODULE precision
    ! -----------------------------------------------------
    ! SP : simple precision de la norma IEEE 754
    ! DP : doble  precision de la norma IEEE 754
    !
    ! Uso: USE precision, WP => SP ó USE precision, WP => DP
    ! -----------------------------------------------------
    INTEGER, PARAMETER :: SP = SELECTED_REAL_KIND(6,37)
    INTEGER, PARAMETER :: DP = SELECTED_REAL_KIND(15,307)
END MODULE precision
```

Entonces podemos escribir un programa que se compile ya sea con reales de simple o doble precisión escogiendo apropiadamente la sentencia que importa el módulo. Por ejemplo, en el siguiente programa los tipos de datos reales son definidos de precisión simple:

```
PROGRAM main
    USE precision, WP => SP
    IMPLICIT NONE
    REAL(KIND=WP) :: a
    a = 1.0_WP/3.0_WP
    WRITE (*,*) a
END PROGRAM main
```

La compilación y ejecución del mismo nos arroja el siguiente valor de la variable `a`:

```
0.3333333
```

Mientras que, cambiando simplemente la asignación de `WP` a `DP`:

```
PROGRAM main
    USE precision, WP => DP
    IMPLICIT NONE
    REAL(KIND=WP) :: a
    a = 1.0_WP/3.0_WP
    WRITE (*,*) a
END PROGRAM main
```

la compilación y ejecución conduce ahora un resultado de doble precisión:

```
0.333333333333333
```