Subject: Named subranges of integers
From: Van Snyder
Reference: 03-258r1, section 1.3.1

1 Number

2 TBD

3 Title

4 Named subranges of integers

5 Submitted By

6 J3

7 Status

8 For consideration.

9 Basic Functionality

10 Provide named subranges of integers, with each one defining a different kind.

11 Rationale

12 There have been numerous requests for integers with explicit and not necessarily symmetrical range,
13 rather than symmetrical range specified by the number of base-ten digits. There have been numerous
14 requests for unsigned integers. There have been numerous requests for a bit data type. Subranges of
15 integers can provide the effect of all of these. In addition, if array bounds can be specified by reference
16 to subranges, and a subscript in a reference is of that kind, and the subscript gets its value in a DO
17 statement with *do-control* based on reference to the subrange, subscript bounds checking has no run-
18 time cost. Even if the subscript gets its value in an ordinary assignment, bounds checking is replaced
19 by checking the value's range during assignment. If the same subscript is used for several references,
20 perhaps to several arrays, the cost of array bounds checking is reduced.

21 Estimated Impact

22 If subranges are defined so as to create new kinds, this is a modest project.

23 If there is a problem with defining subranges to be kinds — perhaps because two largest-size integers
24 are needed to define the set of subranges, and the result of the KIND intrinsic might therefore be
25 problematical — then subranges should work as much like kinds as possible. This makes it a larger (but
26 not tremendously larger) project, because everywhere we say "type, kind and rank" we'll need to say
27 "type, kind, rank, and (if an integer) subrange." If we go in this direction, it would be a good excuse to
28 develop terms for "type and kind or subrange" and "type, kind or subrange, and rank."

29 Detailed Specification

30 Provide a means to define named subranges of integers. Each subrange name defines an unique kind,
31 or a quality that behaves like "kind," even if it is defined by reference to the same subrange as another
32 subrange name, or its subrange is the same as the range of a kind of integer defined by the processor.
33 Subrange names can be used as kind type parameters in the declaration of integer entities, or in the
34 intrinsic functions that need a kind parameter. Mixed-subrange arithmetic, assignment, and comparison
35 are allowed, just like mixed-kind arithmetic, assignment, and comparison are allowed. Integer entities
36 declared by reference to subrange names can also be used for mixed-type arithmetic, assignment and

1   comparison. Mixed-subrange argument association is not permitted, just as mixed-kind argument as-
2   sociation is not permitted. Integer entities of a subrange shall have values within the subrange. Array
3   bounds may be specified by reference to a subrange — ideally by reference directly to the subrange
4   name, rather than indirectly by using TINY and HUGE applied to an object of the subrange.

5   Here are some examples of possible syntax.

6   To define a subrange:

7   SUBRANGE :: *subrange-name* (*low-bound-expr* : *high-bound-expr*)

8   To declare an integer variable, named constant or function result:

9   INTEGER ([KIND =] *subrange-name*) :: *integer-entity-decl*

10 or

11 INTEGER (SUBRANGE = *subrange-name*) :: *integer-entity-decl*

12 In the latter case, even though a term different from KIND is used, integers having different subrange
13 names are considered to have different kinds, including kinds different from integers declared using the
14 KIND keyword.

15 Some interesting subranges:

16 SUBRANGE :: BIT(0:1), BYTE(0:255), UCHAR(0:255), UNSIGNED_INT(0:2**16-1)
17 ! BYTE and UCHAR are different subranges.

18 Using a subrange to get free bounds checking:

19 SUBRANGE :: MyRange (-6 : 23)
20 REAL :: Array (myRange)
21 INTEGER (myRange) :: Sub
22 DO sub = tiny(sub), huge(sub)
23   array(sub) = func(sub)
24 END DO

History