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

Number

TBD

Title

Named subranges of integers

Submitted By

J3

Status

For consideration.

Basic Functionality

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

Rationale

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

Estimated Impact

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

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

Detailed Specification

Provide a means to define named subranges of integers. Each subrange name defines an unique kind, or a quality that behaves like “kind,” even if it is defined by reference to the same subrange as another subrange name, or its subrange is the same as the range of a kind of integer defined by the processor. Subrange names can be used as kind type parameters in the declaration of integer entities, or in the intrinsic functions that need a kind parameter. Mixed-subrange arithmetic, assignment, and comparison are allowed, just like mixed-kind arithmetic, assignment, and comparison are allowed. Integer entities declared by reference to subrange names can also be used for mixed-type arithmetic, assignment and comparison. Mixed-subrange argument association is not permitted, just as mixed-kind argument association is not permitted. Integer entities of a subrange shall have values within the subrange. Array bounds may be specified by reference to a subrange — ideally by reference directly to the subrange name, rather than indirectly by using TINY and HUGE applied to an object of the subrange.

Here are some examples of possible syntax.

To define a subrange:

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

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

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

or

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

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

Some interesting subranges:

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

Using a subrange to get free bounds checking:

```
SUBRANGE :: MyRange ( -6 : 23 )
REAL :: Array ( myRange )
INTEGER ( myRange ) :: Sub
DO sub = tiny(sub), huge(sub)
    array(sub) = func(sub)
END DO
```

History