# LANGUAGE-INDEPENDENT DATA TYPES

David A. Joslin

Teesside Polytechnic
Middlesbrough TS1 3BA, England

JANET/EARN: DAJ @ UK.AC.TP.PA
Tel: +44-642-218121 ext.4118

February 9, 1988

## 1. Introduction

An ISO working group (ISO/IEC JTC1/SC22/WG11) has been charged with the production of international standards for language-independent data types and language-independent procedure-calling mechanisms, to facilitate interactions between routines (or programs, or processes) which may be written in different languages and which may be running on different machines. The procedure-calling mechanism effort has not yet progressed very far, but a considerable amount of work on the data types standard has been performed, and a reasonable idea of its contents can be given.

## 2. The Concept of Data Type

A data type may be regarded as a set of data values, together with the properties of, relationships between, and operations allowed on, these values. It should be emphasised that the standard here will specify abstract data types, not related to any particular hardware or software; it will not prescribe representations of data values. The mapping onto the standard abstract types must be defined by each implementation of programming languages etc.

### 2.1 Properties of Data Values

The important properties are countability (i.e whether or not the set of data values can be mapped onto the integers ..., -1, 0, 1, 2, 3, ... or not) and ordering.

#### 2.1.1 Countability

A countable type has either a finite or a denumerably infinite set of values; in the latter case it can be made finite by imposition of lower and/or upper bounds. The values of any finite type can be represented exactly in an implementation (e.g N distinct values can be mapped onto the subrange 0..(N-1) of integer). However, in a non-countable (i.e non-denumerably infinite) type, whatever finite subset is selected, any neighborhood (however small) of a selected value will contain an infinite number of values not so selected (and which can hence be represented only approximately). Non-countable types are often regarded as "approximate", the best-known example being REAL. (Of course the "approximate" refers to the representation, not to the type itself; the abstract real data type is as exact as any other abstract type.)

### 2.1.2 Ordering

A type may be unordered (e.g there is no conventional ordering of the Boolean values TRUE & FALSE, although particular languages or implementations may define one); partially ordered (not very useful - though partial orderings occur in e.g SET types); or (fully) ordered.

## 2.2 <u>Relationships between Data Values</u>

In any data type, a data value is either equal or unequal to another data value of the same type. In an ordered type, a data value is either less than, equal to, or greater than, another data value of the same type - though the meaning of "less" and "greater" may be a matter of definition. (Some types, e.g character strings, can be ordered in many different ways!)

## 2.3 <u>Operations on Data Values</u>

Operations on data values may be classified as: Monadic (Unary) - one operand - e.g NOT and "unary minus"; Dyadic (Binary) - two operands - e.g AND and OR; Polyadic - many operands - e.g MAX and MIN; and even "Niladic" - no operands - e.g RANDOM.

The dyadic operators = and <> (not equal), with Boolean results, are allowed for most data types; ordered types also have <, >, <= and >=.

All ordered finite types T, and INTEGER, have the following operations:

| | |
|---|---|
| SUCC(x) :<br>Result - T | The least value y in T such that y>x.<br>(Error if x is the maximum value in T.) |
| PRED(x) :<br>Result - T | The greatest value y in T such that y<x.<br>(Error if x is the minimum value in T.) |
| ORD(x) :<br>RESULT -<br>INTEGER | The integer corresponding to the "ordinal value" of x. If T is INTEGER, ORD(x) = x; otherwise conventionally<br>ORD(minimum value in T) = 0,<br>ORD(SUCC(x)) = ORD(x)+1.<br>ORD(PRED(x)) = ORD(x)-1. |

RELVALUE(x, i) : The value y in T such that ORD(y) = ORD(x)+i, where i is any integer.

RELVALUE is an inverse of ORD; if we know any <u>one</u> value t in T, and ORD(x) = i, then x = RELVALUE(t, i-ORD(t)).
RELVALUE provides "addition"/"subtraction" for "metric types"; e.g if T is "calendar year", then x+i doesn't make sense but RELVALUE(x, i) certainly does: "1988" = RELVALUE("1888", 100).

Other operations, e.g arithmetic, are specific to certain data types and may be defined differently for different data types.

## 3. <u>Types very likely to be included in the Standard</u>

The following types are felt to be well enough understood for inclusion in the standard (though definition of each by the working group may not yet be complete). For each type, the set of values and their properties are specified, as well as operations allowed (in addition to those specified in 2.3 above).

### 3.1 <u>Unordered, finite</u>

#### 3.1.1 BOOLEAN

| | |
|---|---|
| Values: | TRUE and FALSE |
| Properties: | finite, unordered |
| Operations: | NOT, AND, OR |
| | All Boolean operations can be defined from any two of NOT, AND, OR; some of the more useful are: |

| | |
|---|---|
| XOR: | (a AND NOT b) OR (NOT a AND b) |
| implies (a -> b): | b OR NOT a |
| equivalence: | (a AND b) OR (NOT a AND NOT b). |

#### 3.1.2 "MARK" (temporary name)

| | |
|---|---|
| Values: | N distinct user-defined values<br>- thus there can be many such types |
| Properties: | finite, unordered |
| Operations: | (only those specified in 2.3). |

### 3.2 <u>Ordered, finite</u>

#### 3.2.1 CHARACTER

| | |
|---|---|
| Values: | characters from some set (e.g ISO 646) |
| Properties: | finite, ordered |
| Operations: | (only those specified in 2.3). |

### 3.2.2 ENUMERATED

|  |  |
|---|---|
| Values: | N distinct user-defined values, with order also user-defined - again there can be many such types |
| Properties: | finite, ordered |
| Operations: | (only those specified in 2.3). |

## 3.3 <u>Ordered, denumerably infinite</u>

### 3.3.1 INTEGER

```
Values:        all positive, zero and negative integers
Properties:    denumerably infinite, ordered
Operations:    addition, subtraction, multiplication,
               division:  truncated towards zero, so that
                  5 DIV  3 =  1     -5 DIV  3 = -1
                  5 DIV -3 = -1     -5 DIV -3 =  1,
               remainder:  x REM y = x - (x DIV y) * y , so that
                  5 REM  3 =  2     -5 REM  3 = -2
                  5 REM -3 =  2     -5 REM -3 = -2,
               modulo:  defined only if divisor positive, result
                        always non-negative, so that
                  5 MOD  3 =  2     -5 MOD  3 =  1,
               exponentiation:  equal to repeated multiplication  -
                  x POW y   =  Error  if (x=0 and y<=0),
                               1  if y=0,
                               (1 DIV x) POW -y  if y<0,
                               x * (x POW y-1)  if y>0.
```

## 3.4 <u>Ordered, non-denumerably infinite</u>

### 3.4.1 REAL

```
Values:        all real numbers
Properties:    non-denumerably infinite, ordered
Operations:    addition, subtraction, multiplication, division,
               exponentiation:  to integer power  -
                  x POW y   =  Error  if (x=0 and y<=0),
                               1  if y=0,
                               (1/x) POW -y  if y<0,
                               x * (x POW y-1)  if y>0;
                  to real power  -
                  x**y   =  Error  if x<0 or (x=0 and y<=0),
                            1  if y=0,
                            else  exp(y*ln(x)),
               mathematical functions:  SQRT, EXP, LN, SIN, COS,
                  ARCTAN etc (all definable as power-series).
```

## 3.5 Unordered, non-denumerably infinite

### 3.5.1 COMPLEX

```
Values:        all complex numbers
Properties:    non-denumerably infinite, unordered
Operations:    addition, subtraction, multiplication, division,
               exponentiation:  to integer power  -
                   x POW y  =  Error  if (x=0 and y<=0),
                               1  if y=0,
                               (1/x) POW -y  if y<0,
                               x * (x POW y-1)  if y>0;
                   to real power  -
                   x**y  =  Error  if (x=0 and y<=0),
                            1  if y=0,
                            else  exp(y*ln(x)),
               real part:          RE(z),
               imaginary part:     IM(z),
               modulus:            ABS(z),
               argument:           ARG(z)  -  principal value,
                                              i.e value in (-PI,+PI],
               constructors:       if x and y are real,
                                   CMPLX(x,y)  =  x + iy    &
                                   POLAR(x,y)  =  x * exp(iy),
               mathematical functions:  as for REAL  -  SQRT, LN,
                  ARCTAN to be defined as principal value.
```

## 3.6 Character Strings

```
Values:        all strings of zero or more CHARACTERs (see 3.2.1),
               including the null string ''
Properties:    denumerably infinite, ordered
Operations:    Two sets of relational  -
               (a) shorter string padded at right with spaces,
               (b) true lexicographic ordering,
               trim:              TRIM('ABC   ') = 'ABC',
               concatenation:     'ABC' + 'XYZ' = 'ABCXYZ',
               substring:         SUBSTR('ABCXYZ',2,3) = 'BCX',
               selection:         if s = 'pqrst',
                                  s[2] = 'q' & s[2..4] = 'qrs',
               length:            LENGTH('XYZ') = 3,
               matching:          INDEX('pqrst','rs') = 3,   etc.
```

## 3.7 "Structuring Methods"

The types in 3.1-3.5 above are "simple" types, i.e not structures of any simpler types. (Yes, type COMPLEX is indeed a simple type, even though one of its representations may look like a pair of REALs.) Various "structuring methods" allow restriction of the range of another type, or construction of aggregate types from other types.

### 3.7.1 Subranges (of 3.2 & 3.3)

Subranges of any ordered type may be formed by specifying lower and upper bounds; they possess the properties, relationships and operations of the base type (although certain operations may produce an error if the bounds are exceeded). Subranges of countable types (3.2 & 3.3) are the most useful; subranges of REAL may also be included in the standard.

### 3.7.2 SET OF base-type

| | |
|---|---|
| Values: | all collections of zero or more distinct values from the base type (which must be countable) |
| Properties: | denumerably infinite, partially ordered |
| Operations: | union (+), intersection (*), difference (−), symmetric difference ( x><y = (x−y)+(y−x) ), member of (IN), subset ( x<=y if every member of x is also a member of y; but the 'strict inequality' x<y is not useful, as it is possible for x<y and y<x to both be false − this is a partial ordering). |

### 3.7.3 LIST

| | |
|---|---|
| Values: | all collections of zero or more values from the base type (which can be almost anything) or (recursively) LISTs of such values |
| Properties: | denumerably infinite (if base type is countable), unordered |
| Operations: | head, tail, construct (add new head), attach (one list to another), appears in. |

### 3.7.4 ARRAY [index-type] OF base-type

| | |
|---|---|
| Values: | in any value of the type, there is one element from the base type (which can be almost anything) for each value of the index type (which must be ordered and countable) |
| Properties: | denumerably infinite (if base type is countable), unordered |
| Operations: | element selection (subscripting) |
| Note: | one-dimensional array; higher dimensionality achieved by ARRAY [...] OF ARRAY [...] OF ... |

### 3.7.5 RECORD

Similar to ARRAY, except that the elements may be of different types and are referred to by name ("field name") rather than subscript. Elements may themselves be of RECORD, ARRAY etc types.

### 3.7.6 "CHOICE" (variants)

In some contexts alternative types may be acceptable, and this is specified by setting out the allowable choices. Some sort of "tag field" may be necessary so that the actual type of a value can be determined.

## 3.8 References to external entities

A method of reference to external data, routines, files etc is required; but standardisation work has yet to start in this area.

## 4. Types requiring further study

The following types require further study to determine whether there is sufficient benefit from their inclusion in the standard. They are described in the same format as the types in section 3 above, though generally in less detail.

## 4.1 Unordered, finite

### 4.1.1 BIT

| | |
|---|---|
| Values: | 0 and 1 (or other pairs of opposites, e.g + and −, North and South, flip and flop) |
| Properties: | finite, unordered |
| Operations: | + (OR), * (AND), "invert" (NOT) |
| Note: | Similar to BOOLEAN, without the specific connotation of "truth" or otherwise. |

## 4.2 Ordered, finite

### 4.2.1 Generic ordinal

| | |
|---|---|
| Values: | (zeroth,) first, second, ... |
| Properties: | denumerably infinite, ordered |
| Operations: | "addition" and "subtraction" (via RELVALUE). |

### 4.2.2 INTEGER modulo N

| | |
|---|---|
| Values: | all integers in the subrange 0..(N−1) |
| Properties: | finite, ordered |
| Operations: | +, −, *, DIV, REM/MOD, POW (all performed modulo N). |

## 4.3 Ordered, denumerably infinite

### 4.3.1 RATIONAL

| | |
|---|---|
| Values: | all pairs of integers (i, j) with j<>0; the arithmetic value of (i, j) is the fraction i/j |
| Properties: | denumerably infinite, ordered |
| Operations: | +, -, *, DIV, REM, MOD, POW. |

### 4.3.2 SCALED

Equivalent to RATIONAL with a specified value of j, the scale factor. Operations *, DIV, REM, MOD, POW require investigation.

## 4.4 Bit Strings

| | |
|---|---|
| Values: | all strings of zero or more BITs (see 4.1.1) |
| Properties: | denumerably infinite, unordered |
| Operations: | arithmetic & Boolean operations could be defined by regarding the string as a binary number; the type would then become ordered. |

## 4.5 "Structuring Methods"

### 4.5.1 Subranges (of 4.2 & 4.3)

As in 3.7.1, subranges of countable types (4.2 & 4.3) could be included.

### 4.5.2 "Multiset"

Similar to SET, but with duplication of members permitted.

### 4.5.3 TABLE [index-type] OF base-type

Similar to ARRAY, but with no requirement for the index type to be ordered. Subscripting (e.g A[i]) permitted, but operations on subscripts (e.g A[i+2]) not generally defined.

### 4.5.4 MATRIX [index-type-1, index-type-2] OF base-type

| | |
|---|---|
| Values: | in any value of the type, there is one element from the base type (INTEGER, REAL or COMPLEX) for each pair of values of the index types (which must be ordered and countable) - a mathematical matrix |
| Properties: | denumerably infinite (if base type is countable), unordered |
| Operations: | subscripting, slicing, transposition, addition & subtraction (if same shape), multiplication (if conformable), multiplication by a value of the base type, inversion (if not singular), etc |
| Note: | NOT the same thing as a two-dimensional array (different mathematical properties & operations); generalisation to higher-order tensors possible. |

## 5. Acknowledgement

ISO/IEC JTC1/SC22/WG11 is indebted for many of the above ideas to Brian Meek, of Kings College London, who is chairman of the BSI programming languages committee IST/5.