From: SC22/WG2 - Pascal
To: SC22/WG11
Subject: WG2 comments on LIDT (DIS 11404) Annex E
Date: April 22, 1994

At its April 1994 meeting, WG2 produced the following comments on Annex E of LIDT (DIS 11404). If WG11 accepts them in principle, perhaps Ed & I could work by email to produce the final text, which could be incorporated when you process the comments arising from the JTC1 DIS ballot.

daj

WG2 COMMENTS ON LIDT (DIS 11404) ANNEX E

(1) Intro: It is not clear which "Pascal" is meant. Many of the examples are invalid "classic" (ISO 7185) Pascal, but the default does not seem to be Extended Pascal (ISO 10206) as this is referred to explicitly in one section.

(2) E.1.1 (& other sections): The word-symbol Boolean is conventionally spelled with an initial capital in Pascal. (Occurs twice in bold and once in ordinary type in this section.)

(3) E.1.4: Replace "Pascal character datatype" by "Pascal type char", with "char" in bold type.

(4) E.1.5: Replace "integer type (1..maxint)" by "integer type 1..maxint" (or perhaps by "subrange type 1..maxint").

(5) E.1.7: Replace "Pascal integer type" by "Pascal type integer", with "integer" in bold type. Pascal does not require "each implementation to define the values of minint ..." - neither classic nor Extended Pascal has a predefined constant minint. Type integer is essentially the range -maxint..maxint .

(6) E.1.8: Replace "1:2" by "1..2" in definition of type rational.

(7) E.1.8 (similar errors throughout): In the first line of Reduce replace "procedure" by "function", and add a semi-colon to the end of the line.

(8) E.1.8 (similar errors throughout): Type rational is invalid as a function result type in classic Pascal. FUNCTION Reduce would have to be replaced by a PROCEDURE with an extra parameter for the result, e.g:

```
procedure Reduce(x:rational; var result:rational);
```

and calls such as "Add:=Reduce(t)" would have to be replaced by e.g:

```
Reduce(t,t2);        Add:=t2
```

(where t2 is a variable of type rational).

(9) E.1.8: Reduce does not appear to perform any actions!

(10) E.1.8: Add, Multiply, Negate, Reciprocal - see (7) & (8) above.

Multiply, Negate, Reciprocal - delete first "end;" line from each.

Replace definitions of NonNegative & Equal by e.g:

```
function NonNegative(x:rational): Boolean;
  begin
  NonNegative:=(x[1]>=0)
  end;

function Equal(x,y:rational): Boolean;
  begin
  Equal:=((x[1]=y[1]) and (x[2]=y[2]))
  end;
```

Is the above definition of Equal correct? If x is (1,2) and y is (3,6), Equal(x,y) would be false. Should Reduce come into it?

The simple algorithms used for Add and Multiply are good for example purposes but could cause overflow unnecessarily in some circumstances; add note that more complicated algorithms would be needed in practice?

(11) E.1.9: Replace ":=" by "=" in definition of type scaled (and in comment about non-local constant rtothef).

scaledMultiply, scaledDivide: see (2) & (7) above.

"mod" is misused in both functions - it is an arithmetic operator in Pascal, and the name "mod" could not be used for a function. Replace "mod(t,rtothef)" by "(t mod rtothef)", and "mod(x*rtothef,y)" by "(x*rtothef mod y)".

Note that WG2 did not the algorithms used in the functions.

In the NOTE: for classic Pascal, see (8) above.
for Extended Pascal, replace "value" by e.g "val"
("value" being now a word-symbol).

(12) E.1.9 (similar errors throughout): In the NOTE, replace the definition of scaled by e.g:

```
type scaled = record                { no "(" following }
                   val:    integer;
                   radix:  0..maxint;     { not (0..maxint) }
                   factor: integer
              end;                  { not ")" }
```

(13) E.1.10: Replace "Pascal real type" by "Pascal type real", with "real" in bold type.

(14) E.1.11: Extended Pascal is mentioned explicitly here. The NOTE also refers to "basic Pascal", a term not otherwise used in the standards world - formally just "Pascal", informally "classic Pascal". In the NOTE, replace the definition of type complex by e.g:

```
type complex = record realpart,imagpart:real end;
```

- and note the restriction on its use, as in (8) bove.

(15) E.2.1: See (12) above.

(16) E.2.2: Replace "pointer to mapped-type" by "^mapped-type".

(17) E.2.3: Note restrictions on function result types in classic Pascal.

(18) E.2.4: Replace (the Pascal, not LI) "record(field-list)" by "record field-list end". The operator = cannot be applied to records (even in Extended Pascal).

(19) E.2.7: See (12) above. Also replace "pointer to sequenceoftype" by "^sequenceoftype" (twice), and ""null"" by "nil" (twice). It may be that sequence datatypes can be mapped to Pascal file types.

(20) E.2.8: The operator = cannot be applied to arrays (even in Extended Pascal).

(21) E.2.9: See (12) above.

(22) E.3.2: Period missing at end of second sentence.

(23) E.3.3: Period missing at end of second sentence.

(24) E.4.1: Replace "Pascal Integer type" by "Pascal integer type" (or perhaps by "Pascal subrange type"). Replace definition of type bit by "type bit = 0..1;". Add: see (7) above.

(25) E.4.2: Parameter and function result types must be identifiers, so that Head & Tail should be defined as e.g

```
type osk = packed array [1..k] of octet;
     osm = packed array [1..k-1] of octet;
function Head(x:osk): octet;
   ...
function Tail(x:osk): osm;
   var i: integer;
         y: osm;
   ...
```

Classic Pascal: see (8) above.

The Pascal operator = cannot be used for packed arrays of octet.

(26) E.4.3: Contains errors similar to those in E.4.2 (except that the Pascal = operator can exceptionally, be used in this case).

(27) E.4.4: Replace "integer-type" by "integer type" (or "subrange type").

(28) E.4.5: Contains errors similar to those in E.4.2. Also replace "octetstring" by "octet-string" (twice).

(29) E.4.6: Delete one (either will do!) period at end of section.

(30) E.4.7: The section appears to have two "lead-in"s to the definition of type object_identifier_component. Underscores are not permitted in identifiers in classic Pascal.

(31) E.5.1: This is simply incorrect. The definition "type x=y" does NOT create a new type x; it simply provides another name for type y (in the case where y is an identifier).

```
Pascal syntax:  type-denoter = type-identifier | new-type
                  new-type = new-ordinal-type | new-...
                  new-ordinal-type = enumerated-type | subrange-type
```

(32) E.5.3: Many generators could be mapped to schemata in Extended Pascal.