11 Arithmetic Range Checking

The Component Pascal Language Report does not specify the out-of-range behaviour of the built in arithmetic functions, but deliberately allows the compiler writer freedom to make implementation decisions; see the quotation below:

> This report is not intended as a programmer's tutorial. It is intentionally kept concise. Its function is to serve as a reference for programmers. What remains unsaid is mostly left so intentionally, either because it can be derived from stated rules of the language, or because it would require to commit the definition when a general commitment appears as unwise

Some of the *current* decisions with *this* compiler are documented below, primarily because they can lead to unexpected behaviour, which in turn can lead to wasted time on the part of the developer.

These behaviours are not considered to be bugs, and are considered to be within the freedom permitted by the Language Report.

Developers are advised to exploit these behaviours with care as:
- other compilers may behave differently
- this compiler may behave differently in the future
- code that uses them may be hard for another reader to understand.

11.1 INTEGER Arithmetic

INTEGER arithmetic (+, -, *) is *not* range checked, and the low 32 bits of the *correct* answer are returned. This is equivalent to reducing the answer MODULO 2↑32, to a number in the range [-2↑31 … 2↑31).

11.2 LONGINT Arithmetic

LONGINT arithmetic (+, -, *) *is* range checked, and overflows lead to a run-time "undefined real result" exception.

The explanation for this behaviour is that INTEGER arithmetic uses integer instructions on the main CPU, whereas LONGINT arithmetic uses floating point instructions on the co-processor.

11.3 SHORT () procedure

Converts from LONGINT to INTEGER by taking the lower 32 bits *without* range checking.

11.4 ENTIER () procedure

Converts from REAL to LONGINT *with* range checking. Overflows lead to a run-time "undefined real result" exception.
The permitted range is [-2↑63 … 2↑63)

11.5 The compound SHORT (ENTIER ()) procedure

Converts from REAL to INTEGER *with* range checking. Overflows lead to a run-time "undefined real result" exception.

Note carefully that the range checking here is to the range [-2↑31 … 2↑31), so that using this compound

procedure is *not* functionally equivalent to assigning the intermediate LONGINT value to a variable.

Explanation: Using the floating point unit to perform the compound type transfer in one operation is more efficient that performing the two operations sequentially. The SHORT procedure is defined as the *identity* operation, which is clearly only possible for in-range values. It is not defined for out-of-range values.

11.6 The IN operator

The expression "el IN set" is only defined for values of *el* that could be valid elements of *set*, thus for *el* in the range [0 … 31].
Currently, if *el* is outside that range it is reduced MODULO 32.

This leads to the behaviour "33 IN {1} = TRUE".

11.5 REAL arithmetic

Intermediate values in REAL expressions are usually calculated using the extended 80 precision format of modern floating point units, then converted to 64 bit precision prior to assignment to a REAL variable. This design choice generally reduces arithmetic rounding errors.

It does mean that breaking a long expression into parts may not lead to exactly equivalent results.

Example:
```
   x  :=  Math.Pi (); ASSERT(x=Math.Pi(), 30)
```
leads to a TRAP,

and so does
```
   ASSERT(Math.Pi()=Math.Pi(), 30).
```
(In this case the first call is rounded to 64 bits, the second is left at 80!).