Differences between Revised Oberon and Oberon

Niklaus Wirth, 22.03.2008 / 15.7.2011

Revised Oberon (Oberon-07) is a revision of the original language Oberon as defined in 1988/1990. It is accepted by the compiler recently completed for the ARM processor. Most changes in the language might easily be called features of a dialect. However, there are a few that merit a stronger distinction, because they should be considered as permanent, and as corrections of unsatisfactory properties of the original Oberon. These are the elimination of the loop statement, function result specification, array assignments, constant parameters, and read-only import of variables. All changes were made in the interest of regularity, simplicity, completeness, and well-structuredness.

1. The Loop and the While statements

It had been thought that the while statement with the termination condition at its entry, and the repeat statement with the termination condition at its end must be amended by a general and flexible construct with termination conditions anywhere. The loop statement with its exit statements represents, however, a break with the idea of a structured language, where properties of a statement can be derived from those of its components. The loop statement with its syntactically unconnected exit statements does not allow this. It has therefore been deleted from the language together with the associated exit statement.

As a sort of compensation the flexibility of the while statement has been enhanced. Its extended syntax is

WhileStatement = "WHILE" expression "DO" StatementSequence
{“ELSIF” expression "DO" StatementSequence} "END".

As long as any of the Boolean expressions yields TRUE, the corresponding statement sequence is executed. Repetition terminates, when all conditions are false. This is Dijkstra’s form. His favourite example was a simple form of the Euclidean algorithm to compute the greatest common divisor of *m* and *n*:

```
WHILE m > n DO m := m – n
ELSIF n > m DO n := n – m
END
```

2. The Case statement

The case statement performs the same function as the if statement. However, it is intended to use a different technique of implementation, namely a single, indexed branch instead of a cascade of conditional branches. This is sensible only if the cases are distinguished by a (mostly) contiguous range of label values.

1. Labels are integers or character constants.
2. Labels form a range of values, starting with 0, like array indices.

3. The With statement

The With statement has been eliminated.

4. Specification of function procedures

The result of a function procedure was specified by a return statement. This form has the unpleasant property that the return statement is syntactically disconnected from the function procedure declaration, similar to the exit from the loop statements. It is therefore difficult to check, whether or not a function procedure declaration specifies a result, or perhaps even several of

them. Now the result specification becomes syntactically a part of the procedure declaration, and vanishes as an independent statement form.

Old form:

```
PROCEDURE F(x, y: INTEGER): INTEGER;
BEGIN
   IF x < y THEN RETURN y – x
   ELSE RETURN x – y
   END
END F
```

New form:

```
PROCEDURE F(x, y: INTEGER): INTEGER;
BEGIN
   IF x < y THEN x := y – x
   ELSE x := x – y
   END ;
RETURN x END F
```

4. Assignment of arrays and records

Because assignments of arrays are typically more complex operations than simple assignments, it was believed that this should be visible to a programmer. The standard procedure *COPY(src, dst)* was therefore introduced. But now we handle array and record assignments just like other assignments, writing *dst := src,* and discard the COPY procedure. The destination array must not be shorter than the source array. This is in accordance with the assignment rule for strings and with the compatibility of record types and their extensions.

5. Parameters

In Oberon, there are two kinds of parameters: Value and variable parameters. In the latter case the formal parameter is considered as a local variable, and the actual parameter's value is assigned to it. This implies making a copy of that value, which is undesirable in the case of structured types. In order to avoid the necessity of copying, we postulate that if a value parameter is if a structured type, no assignments are possible. Hence, the parameter can be passed by a reference like in the case of VAR parameters.

6. Read-only import of variables

According to the guidelines of modular programming and information hiding, it is recommended in general to export and import constant objects only. These are constants, data types, and procedures. Unfortunately, this sometimes makes it necessary to introduce a (exported) procedure merely for the simple purpose of reading the value of a variable.

In Oberon, variables can be exported freely. In the spirit of exemplary modular programming, this export is now – not abolished – but subject to read-only mode. The imported variable appears as a constant.

7. Miscellaneous

- The types SHORTINT and LONGINT are removed, as well as the concept of type inclusion.

- The standard procedures and functions CAP, HALT, COPY, ASH, ENTIER, MIN, and MAX are removed. (ASH is replaced by LSL, and ASR, ENTIER is renamed FLOOR, and HALT is replaced by ASSERT(FALSE).

- New are the standard procedures ASSERT, PACK, and UNPK, and the functions LSL, ASR for shifting, and FLOOR, and FLT for type transfers. ROT has been renamed ROR (right rotation only).

- Applied to values of type SET, the unary minus denotes the set complement. The relations <= and >= denote set inclusion.

- Pointers must point to records.