WIKIPEDIA

Oberon (programming language)

Oberon is a general-purpose programming language first published in 1987 by Niklaus Wirth and the latest member of the Wirthian family of ALGOL-like languages (Euler, Algol-W, Pascal, Modula, and Modula-2).[1][2][3][4] Oberon was the result of a concentrated effort to increase the power of Modula-2, the direct successor of Pascal, and simultaneously to reduce its complexity. Its principal new feature is the concept of type extension of record types:[5] It permits the construction of new data types on the basis of existing ones and to relate them, deviating from the dogma of strictly static data typing. Type extension is Wirth's way of inheritance reflecting the viewpoint of the parent site. Oberon was developed as part of the implementation of the Oberon operating system at ETH Zurich in Switzerland. The name is from the moon of Uranus, Oberon.

Oberon is still maintained by Wirth and the latest Project Oberon compiler update is dated Mar 6, 2020.[6]

Oberon

Paradigm	Imperative, structured, modular, object-oriented
Designed by	Niklaus Wirth
First appeared	1987
Typing discipline	strong, hybrid (static and dynamic)
Website	www.projectoberon.com (http://www.projectoberon.com/)
Influenced by	
Modula-2	
Influenced	
Oberon-2, Oberon-07, Zonnon, Go, Nim, Active Oberon, Component Pascal	

Contents

Design

Oberon is designed with a motto attributed to Albert Einstein in mind: “Make things as simple as possible, but not simpler.” The principal guideline was to concentrate on features that are basic and essential and to omit ephemeral issues. Another factor was recognition of the growth of complexity in languages such as C++ and

Ada: in contrast to these, Oberon emphasizes the use of the library concept for extending the language. Enumeration and subrange types, which were present in Modula-2, have been removed; similarly, set types have been limited to small sets of integers, and the number of low-level facilities has been sharply reduced (most particularly, type transfer functions have been eliminated). Elimination of the remaining potentially-unsafe facilities concludes the most essential step toward obtaining a truly high-level language. Very close type-checking even across modules, strict index-checking at run time, null-pointer checking, and the safe type extension concept largely allow the programmer to rely on the language rules alone.

The intent of this strategy was to produce a language that is easier to learn, simpler to implement, and very efficient. Oberon compilers have been viewed as compact and fast, while providing adequate code quality compared to commercial compilers.[7]

Characteristics

The following features characterize the Oberon language:

- Case sensitive syntax with uppercase keywords
- Type-extension with type test
- Modules and separate compilation
- String operations
- Isolation of unsafe code
- Support for system programming

Object orientation

Oberon supports extension of record types for the construction of abstractions and heterogeneous structures. In contrast to the later dialects—Oberon-2 and Active Oberon—the original Oberon doesn't have a dispatch mechanism as a language feature but rather as programming technique or design pattern. This gives great flexibility in the OOP world. In the Oberon operating system two programming techniques have been used in conjunction for the dispatch call: Method suite and Message handler.

Method suite

In this technique a table of procedure variables is defined and a global variable of this type is declared in the extended module and assigned back in the generic module:

```
MODULE Figures; (* Abstract module *)

TYPE
   Figure*    = POINTER TO FigureDesc;
   Interface* = POINTER TO InterfaceDesc;

   InterfaceDesc* = RECORD
      draw*  : PROCEDURE (f : Figure);
      clear* : PROCEDURE (f : Figure);
      mark*  : PROCEDURE (f : Figure);
      move*  : PROCEDURE (f : Figure; dx, dy : INTEGER);
   END;

   FigureDesc* = RECORD
      if : Interface;
   END;

PROCEDURE Init* (f : Figure; if : Interface);
BEGIN
   f.if := if;
END Init;
```

```
PROCEDURE Draw* (f : Figure);
BEGIN
   f.if.draw(f);
END Draw;

(* Other procedures here *)

END Figures.
```

We extend the generic type Figure to a specific shape:

```
MODULE Rectangles;

IMPORT Figures;

TYPE
   Rectangle* = POINTER TO RectangleDesc;

   RectangleDesc* = RECORD
      (Figures.FigureDesc)
      x, y, w, h : INTEGER;
   END;

VAR
   if : Figures.Interface;

PROCEDURE New* (VAR r : Rectangle);
BEGIN
   NEW(r);
   Figures.Init(r, if);
END New;

PROCEDURE Draw* (f : Figure);
   VAR
      r : Rectangle;
BEGIN
   r := f(Rectangle); (* f AS Rectangle *)
   (* ... *)
END Draw;

(* Other procedures here *)

BEGIN (* Module initialisation *)
   NEW(if);
   if.draw  := Draw;
   if.clear := Clear;
   if.mark  := Mark;
   if.move  := Move;
END Rectangles.
```

Dynamic dispatch is only done via procedures in Figures module that is the generic module.

Message handler

This technique consists of replacing the set of methods with a single procedure, which discriminates among the various methods:

```
MODULE Figures; (* Abstract module *)

TYPE
   Figure*    = POINTER TO FigureDesc;

   Message*   = RECORD END;
   DrawMsg*   = RECORD (Message) END;
   ClearMsg*  = RECORD (Message) END;
   MarkMsg*   = RECORD (Message) END;
   MoveMsg*   = RECORD (Message) dx*, dy* : INTEGER END;

   Handler*   = PROCEDURE (f : Figure; VAR msg : Message);

   FigureDesc* = RECORD
```

```
      (* Abstract *)
      handle : Handler;
   END;

PROCEDURE Handle* (f : Figure; VAR msg : Message);
BEGIN
   f.handle(f, msg);
END Handle;

PROCEDURE Init* (f : Figure; handle : Handler);
BEGIN
   f.handle := handle;
END Init;

END Figures.
```

We extend the generic type Figure to a specific shape:

```
MODULE Rectangles;

IMPORT Figures;

TYPE
   Rectangle* = POINTER TO RectangleDesc;

   RectangleDesc* = RECORD
      (Figures.FigureDesc)
      x, y, w, h : INTEGER;
   END;

PROCEDURE Draw* (r : Rectangle);
BEGIN
  (* ... *)
END Draw;

(* Other procedures here *)

PROCEDURE Handle* (f: Figure; VAR msg: Figures.Message);
   VAR
      r : Rectangle;
BEGIN
   r := f(Rectangle);
   IF    msg IS Figures.DrawMsg THEN Draw(r)
   ELSIF msg IS Figures.MarkMsg THEN Mark(r)
   ELSIF msg IS Figures.MoveMsg THEN Move(r, msg(Figures.MoveMsg).dx, msg(Figures.MoveMsg).dy)
   ELSE  (* ignore *)
   END
END Handle;

PROCEDURE New* (VAR r : Rectangle);
BEGIN
   NEW(r);
   Figures.Init(r, Handle);
END New;

END Rectangles.
```

In the Oberon operating system both of these techniques are used for dynamic dispatch. The first one is used for a known set of methods; the second is used for any new methods declared in the extension module. For example, if the extension module Rectangles were to implement a new Rotate() procedure, within the Figures module it could only be called via a message handler.

Implementations and variants

Oberon

No-cost implementations of Oberon (the language) and Oberon (the operating system) can be found on the Internet (several are from ETHZ itself).

Oberon-2

A few changes were made to the first released specification (object-oriented programming features were added, the 'FOR' loop was reinstated, for instance); the result was **Oberon-2**. There is a release called *Native Oberon* which includes an operating system, and can directly boot on PC class hardware. A .NET implementation of Oberon with the addition of some minor .NET-related extensions has also been developed at ETHZ. In 1993 an ETHZ spin off company brought a dialect of Oberon-2 to the market with the name Oberon-L, which was renamed to Component Pascal in 1997.

Oberon-2 compilers developed by ETH include versions for Microsoft Windows, Linux, Solaris, and classic Mac OS. Furthermore, there are implementations for various other operating systems, such as Atari-TOS or AmigaOS.

There is an Oberon-2 Lex scanner and Yacc parser by Stephen J Bevan of Manchester University, UK, based on the one in the Mössenböck and Wirth reference. It is at version 1.4.

There is also the Oxford Oberon-2 Compiler (http://spivey.oriel.ox.ac.uk/corner/Oxford_Oberon-2_compiler), which also understands Oberon-07 and Vishap Oberon (https://github.com/vishaps/voc). The latter is based on Josef Templ's Oberon to C transpiler called Ofront (https://github.com/jtempl/ofront/), which in turn is based on the OP2 Compiler developed by Regis Crelier at ETHZ .

Oberon-07

Oberon-07, defined by Niklaus Wirth in 2007 and revised in 2011, 2013, 2014, 2015 and 2016 is based on the original version of Oberon rather than Oberon-2. The main changes are: explicit numeric conversion functions (e.g. FLOOR and FLT) must be used, the LOOP and EXIT statements have been eliminated, WHILE statements have been extended, CASE statements can be used for type extension tests, RETURN statements can only be connected to the end of a function, imported variables and structured value parameters are read-only and arrays can be assigned without using COPY. For full details, see The Programming Language Oberon-07 (http://www.inf.ethz.ch/personal/wirth/Oberon/Oberon07.Report.pdf).

Oberon-07 compilers have been developed for use with several different computer systems. Wirth's compiler targets a RISC processor of his own design that was used to implement the 2013 version of the Project Oberon operating system on a Xilinx FPGA Spartan-3 board. Ports of the RISC processor to FPGA Spartan-6, Spartan-7, Artix-7 and a RISC emulator for Windows (compilable on Linux and OS X, as well as binaries available for Windows) also exist. OBNC (https://miasap.se/obnc/) compiles via C and can be used on any POSIX compatible operating system. The commercial Astrobe (http://www.astrobe.com) implementation targets 32-bit ARM Cortex-M3, M4 and M7 microcontrollers. The Patchouli (https://github.com/congdm/Patchouli-Compiler) compiler produces 64-bit Windows binaries. Oberon-07M (http://www.exaprog.com/) produces 32-bit Windows binaries and implements revision 2008 of the language. Akron's (https://sites.google.com/site/oberon07compiler/versii) produces binaries for both Windows and Linux. OberonJS (http://oberspace.org/oberonjs.html) translates Oberon to JavaScript. There is online IDE for Oberon (https://visual.sfu-kras.ru). oberonc (https://github.com/lboasso/oberonc) is an implementation for the Java virtual machine.

Active Oberon

Active Oberon is yet another variant of Oberon, which adds objects (with object-centered access protection and local activity control), system-guarded assertions, preemptive priority scheduling and a changed syntax for methods (- type-bound procedures in the Oberon world). Objects may be active, which means that they may be threads or processes. Additionally, Active Oberon has a way to implement operators (including overloading), an advanced syntax for using arrays (see OberonX language extensions (http://www.ethoberon.e thz.ch/native/compiler/x.index.html) and Proceedings[8] of the 7th Joint Modular Languages Conference 2006 Oxford, UK), and knows about namespaces (see Proposal for Module Contexts (http://www.ocp.inf.ethz.ch/wiki/Documentation/Language?action=download&upname=contexts.pdf)). The operating system A2 - Bluebottle, especially the kernel, synchronizes and coordinates different active objects.

ETHZ has released Active Oberon which supports active objects, and the Bluebottle operating system and environment (JDK, HTTP, FTP, etc.) for the language. As with many prior designs from ETHZ, versions of both are available for download on the Internet. As this is written, both single and dual x86 CPUs and the StrongARM family are supported.

Related languages

Development has continued on languages in this family. A further extension of Oberon-2, originally named Oberon/L but later renamed to Component Pascal, was developed for Windows and classic Mac OS by Oberon microsystems, a commercial company spin-off from ETHZ, and for .NET by Queensland University of Technology. In addition, the Lagoona and Obliq languages carry the Oberon spirit into specialized areas.

Recent .NET development efforts at ETHZ have been focused on a new language called Zonnon. This includes the features of Oberon and restores some from Pascal (enumerated types, built-in IO) but has some syntactic differences. Additional features include support for active objects, operator overloading and exception handling. Zonnon is available as a plug-in language for the Microsoft Visual Studio for .NET development environment.

Oberon-V (originally called Seneca, after Seneca the Younger) is a descendant of Oberon designed for numerical applications on supercomputers, especially vector or pipelined architectures. It includes array constructors and an ALL statement. (See "Seneca - A Language for Numerical Applications on Vectorcomputers", Proc CONPAR 90 - VAPP IV Conf. R. Griesemer, Diss Nr. 10277, ETH Zurich.)

See also

- Oberon (operating system)
- Bluebottle OS
- Oberon on Wikibooks

References

1. Wirth, Niklaus: From Modula to Oberon and the programming language Oberon, ETH Technical Reports D-INFK, Band 82, https://doi.org/10.3929/ethz-a-005363226
2. Wirth, Niklaus: The Programming Language Oberon. Software - Practice and Experience, 18:7, 661-670, Jul. 1988
3. Wirth, Niklaus: From Modula to Oberon. Software - Practice and Experience, 18:7, 671-690, Jul. 1988
4. Wirth, Niklaus: Type Extensions. ACM Transactions on Programming Languages, 10:2, 204-214, Apr. 1988
5. D. Pountain, Modula's Children, Part II: Oberon - BYTE 16(3), 135-142, Mar. 1991. (https://archive.org/stream/byte-magazine-1991-03/1991_03_BYTE_16-03_Network_Management#page/n187/)
6. Wirth, Niklaus. "Oberon Change Log" (https://www.inf.ethz.ch/personal/wirth/news.txt). ETH Zurich. Retrieved 10 March 2020.

7. Mössenböck, Hanspeter. "Compiler Construction: The Art of Niklaus Wirth" (ftp://ftp.ssw.uni-linz.ac.at/pub/Papers/Moe00b.pdf) (PDF). Ftp.ssw.uni-linz.ac.at.
8. Friedrich, Felix; Gutknecht, Jürg (2006). "Array-Structured Object Types for Mathematical Programming". In Lightfoot, David E.; Szyperski, Clemens (eds.). *Modular Programming Languages*. *"Modular Programming Languages"*. Lecture Notes in Computer Science. **4228**. Springer, Berlin Heidelberg. pp. 195–210. doi:10.1007/11860990_13 (https://doi.org/10.1007%2F11860990_13). ISBN 978-3-540-40927-4.

External links

General

- *Official website (latest available copy at archive org) (https://web.archive.org/web/20191219125640/http://www.ethoberon.ethz.ch/)* at ETH-Zürich
- *Niklaus Wirth's Oberon Page (http://people.inf.ethz.ch/wirth/Oberon/)* at ETH-Zürich
- *Oberon Page (http://www.ssw.uni-linz.ac.at/Research/Projects/Oberon.html)* at SSW, Linz
- *Oberon: The Programming Language (http://www.mathematik.uni-ulm.de/oberon/reports/)* at Ulm
- *Project Oberon, The Design of an Operating System and a Compiler (http://people.inf.ethz.ch/wirth/ProjectOberon1992.pdf)*, book in PDF by Niklaus Wirth and Jürg Gutknecht, 2005 Edition
- *Oberon Language Genealogy (https://web.archive.org/web/20130529020132/http://www.ethoberon.ethz.ch/genealogy.html)*
- *Astrobe (http://www.astrobe.com)* ARM Oberon-07 Development System
- *Oberon System V4 for HP OpenVMS Alpha (http://modulaware.com/mwovms.htm)* with source code upward-compatible 64 bit addressing
- *64 bit Oberon-2 compiler (http://modulaware.com/mwcvms.htm)* for HP OpenVMS Alpha
- *Oxford Oberon-2 Compiler (http://spivey.oriel.ox.ac.uk/corner/Oxford_Oberon-2_compiler)* and its User Manual (https://bitbucket.org/Spivey/obc-3/downloads/obcman.pdf)
- *Free Oberon-07 IDE (https://github.com/rochus-keller/Oberon)* Free Oberon-07 IDE for Windows, Mac and Linux with syntax colouring, semantic navigation and source-level debugger
- *Oberon article by Joseph Templ (https://www.drdobbs.com/architecture-and-design/the-oberon-programming-language/184409405)* in the January 1994 issue of Dr.Dobbs

Evolution of Oberon

- *Modula-2 and Oberon (http://people.inf.ethz.ch/wirth/Articles/Modula-Oberon-June.pdf)* Wirth (2005)
- *The Programming Language Oberon (http://people.inf.ethz.ch/wirth/Oberon/Oberon.Report.pdf)* Wirth, (1988/90)
- *The Programming Language Oberon (Oberon-7, Revised Oberon) (http://people.inf.ethz.ch/wirth/Oberon/Oberon07.Report.pdf)* Wirth, (2016, most current language report)
- *Differences between Oberon-07 and Oberon (http://people.inf.ethz.ch/wirth/Oberon/Oberon07.pdf)* Wirth (2011)
- *The Programming Language Oberon-2 (ftp://ftp.ethoberon.ethz.ch/Oberon/OberonV4/Docu/Oberon2.Report.ps)* H. Mössenböck, N. Wirth, Institut für Computersysteme, ETH Zürich, January 1992
- *Differences between Oberon and Oberon-2 (ftp://ftp.ethoberon.ethz.ch/Oberon/OberonV4/Docu/Oberon2.Differences.ps)* Mössenböck and Wirth (1991)

- *What's New in Component Pascal (https://web.archive.org/web/20110515111149/http://www.oberon.ch/pdf/CP-New.pdf)* (Changes from Oberon-2 to CP), Pfister (2001)