WIKIPEDIA

Modula-2

Modula-2 is a structured, procedural programming language developed between 1977 and 1985 by Niklaus Wirth at ETH Zurich. It was created as the language for the operating system (OS) and application software of the Lilith personal workstation.[1] It was later used for programming outside the context of the Lilith.

Wirth viewed Modula-2 as a successor to his earlier programming languages Pascal and Modula.[2][3] The principal concepts are:

1. The module as a compiling unit for separate compiling
2. The coroutine as the basic building block for concurrent processes
3. Types and procedures that allow access to machine-specific data

The language design was influenced by the Mesa language and the Xerox Alto, both from Xerox PARC, that Wirth saw during his 1976 sabbatical year there.[4] The computer magazine *Byte* devoted the August 1984 issue to the language and its surrounding environment.[5]

Modula-2 was followed by Modula-3, and later by the Oberon series of languages.

Modula-2

Paradigms	imperative, structured, modular, data and procedure hiding, concurrent
Family	Wirth
Designed by	Niklaus Wirth
First appeared	1978
Typing discipline	Static, strong, safe
Scope	Lexical
OS	Cross-platform
Filename extensions	.mod, .m2, .def, .MOD, .DEF, .mi, .md
Website	www.modula2.org (http://www.modula2.org)
Major implementations	
ETH compiler (http://www.sysecol2.ethz.ch/RAMSES/MacMETH.html) written by Niklaus Wirth GNU Modula-2 (http://www.nongnu.org/gm2/) ADW Modula-2 (http://www.modula2.org/)	
Dialects	
PIM2, PIM3, PIM4, ISO	
Influenced by	
Modula, Mesa, Pascal, ALGOL W, Euclid	
Influenced	
Modula-3, Oberon, Ada, Fortran 90, Lua, Seed7, Zonnon, Modula-GM	

Contents

Description

Modula-2 is a general purpose procedural language, sufficiently flexible to do systems programming, but with much broader application. In particular, it was designed to support separate compiling and data abstracting in a straightforward way. Much of the syntax is based on Wirth's earlier and better-known language, Pascal. Modula-2 was designed to be broadly similar to Pascal, with some elements and syntactic ambiguities removed and the important addition of a *module* concept, and direct language support for multiprogramming.

The language allows use of one-pass compilers. Such a compiler by Gutknecht and Wirth was about four times faster than earlier multi-pass compilers.[6]

Here is an example of the source code for the "Hello world" program:

```
MODULE Hello;
FROM STextIO IMPORT WriteString;
BEGIN
  WriteString("Hello World!");
END Hello.
```

A Modula-2 *module* may be used to encapsulate a set of related subprograms and data structures, and restrict their visibility from other parts of the program. The module design implemented the data abstraction feature of Modula-2 in a very clean way. Modula-2 programs are composed of modules, each of which is made up of two parts: a *definition module*, the interface portion, which contains only those parts of the subsystem that are *exported* (visible to other modules), and an *implementation module*, which contains the working code that is internal to the module.

The language has strict scope control. The scope of a module can be considered as an impenetrable wall: Except for standard identifiers, no object from the outside is visible inside a module unless explicitly imported; no internal module object is visible from the outside unless explicitly exported.

Suppose module M1 exports objects a, b, c, and P by enumerating its identifiers in an explicit export list

```
  DEFINITION MODULE M1;
    EXPORT QUALIFIED a, b, c, P;
    ...
```

Then the objects a, b,c, and P from module M1 become now known outside module M1 as M1.a, M1.b, M1.c, and M1.P. They are exported in a *qualified* manner to the outside (assumed module M1 is global). The exporting module's name, i.e. M1, is used as a qualifier followed by the object's name.

Suppose module M2 contains the following IMPORT declaration

```
  MODULE M2;
    IMPORT M1;
    ...
```

Then this means that the objects exported by module M1 to the outside of its enclosing program can now be used inside module M2. They are referenced in a *qualified* manner, thusly: M1.a, M1.b, M1.c, and M1.P. Example:

```
    ...
    M1.a := 0;
    M1.c := M1.P(M1.a + M1.b);
    ...
```

Qualified export avoids name clashes: For example, if another module M3 would also export an object called P, then we can still distinguish the two objects, since M1.P differs from M3.P. Thanks to the qualified export it does not matter that both objects are called P inside their exporting modules M1 and M3.

An alternative method exists, which is in wide use by Modula-2 programmers. Suppose module M4 is formulated as this:

```
  MODULE M4;
    FROM M1 IMPORT a, b, c, P;
```

Then this means that objects exported by module M1 to the outside can again be used inside module M4, but now by mere references to the exported identifiers in an *unqualified* manner, thusly: a, b, c, and P. Example:

```
    ...
    a := 0;
    c := P(a + b);
    ...
```

This method of unqualifying import allows use of variables and other objects outside their exporting module in exactly the same simple, i.e. *unqualified*, manner as inside the exporting module. The walls surrounding all modules have now become irrelevant for all those objects for which this has been explicitly allowed. Of course unqualifying import is only usable if there are no name clashes.

These export and import rules may seem unnecessarily restrictive and verbose. But they do not only safeguard objects against unwanted access, but also have the pleasant side-effect of providing automatic cross-referencing of the definition of every identifier in a program: if the identifier is qualified by a module name, then the definition comes from that module. Otherwise if it occurs unqualified, simply search backwards, and you will either encounter a declaration of that identifier, or its occurrence in an IMPORT statement which names the module it comes from. This property becomes very useful when trying to understand large programs containing many modules.

The language provides for (limited) single-processor concurrency (monitors, coroutines and explicit transfer of control) and for hardware access (absolute addresses, bit manipulation, and interrupts). It uses a nominal type system.

Dialects

There are two major dialects of Modula-2. The first is *PIM*, named for the book *Programming in Modula-2* by Niklaus Wirth. There were three major editions of PIM, the second, third (corrected), and fourth editions, each describing slight variants of the language. The second major dialect is *ISO*, named for the standardization effort by the International Organization for Standardization. Here are a few of the differences among them.

- **PIM2** (1983)
  - Required explicit **EXPORT** clause in definition modules.
  - Function **SIZE** needs to be imported from module **SYSTEM**
- **PIM3** (1985)

    - Removed the **EXPORT** clause from definition modules following the observation that everything within a definition module defines the interface to that module, hence the **EXPORT** clause was redundant.
    - Function **SIZE** is pervasive (visible in any scope without import)
- **PIM4** (1988)
    - Specified the behaviour of the **MOD** operator when the operands are negative.
    - Required all **ARRAY OF CHAR** strings to be terminated by ASCII NUL, even if the string fits exactly into its array.
- **ISO** (1996 and 1998)
    - ISO Modula-2 resolved most of the ambiguities in PIM Modula-2. It added the data types **COMPLEX** and **LONGCOMPLEX**, exceptions, module termination (**FINALLY** clause) and a complete standard I/O library. There are numerous minor differences and clarifications.[7]

Supersets

There are several supersets of Modula-2 with language extensions for specific application domains:

- **PIM supersets**
    - Canterbury Modula-2 (https://web.archive.org/web/20010809233821/http://www.mhccorp.com/modula-2.html), extended with Oberon-like extensible records [This has been withdrawn and is no longer available anywhere]
    - Modula-2+, extended with preemptive threads and exceptions
    - Modula-2* (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.7784), parallel extension[8]
    - Modula-P (http://robotics.ee.uwa.edu.au/modula-p/), another parallel extension[9]
    - Modula-Prolog, adding a Prolog layer[10]
    - Modula/R, with relational database extensions
    - Modula-GM, extensions for embedded systems
- **ISO supersets**
    - ISO10514-2, adding an object-oriented programming layer[11]
    - ISO10514-3, adding a generics layer[12]
- **IEC supersets**
    - Mod51 (https://web.archive.org/web/20081021043956/http://www.designtools.co.nz/mod51.htm), extended with IEC1131 constructs for embedded development

Derivatives

There are several derivative languages that resemble Modula-2 very closely but are new languages in their own right. Most are different languages with different purposes and with strengths and weaknesses of their own:

- Modula-3, developed by a team of ex-Xerox employees who had moved to DEC and Olivetti[13]
- Oberon, developed at ETH Zürich for System Oberon[14] available online (https://inf.ethz.ch/personal/wirth/ProjectOberon1992.pdf).
- Oberon-2, Oberon with OO extensions[15]

- Active Oberon, yet another object-oriented Extension of Oberon, developed also at ETH with the main objective to support parallel programming on multiprocessor and multicore systems.
- Parallaxis, a language for machine-independent data-parallel programming[16]
- Umbriel, developed by Pat Terry as a teaching language[17]
- YAFL, a research language by Darius Blasband[18]

Many other current programming languages have adopted features of Modula-2.

Language elements

Reserved words

PIM [2,3,4] defines the following 40 reserved words:

```
AND         ELSIF           LOOP       REPEAT
ARRAY       END             MOD        RETURN
BEGIN       EXIT            MODULE     SET
BY          EXPORT          NOT        THEN
CASE        FOR             OF         TO
CONST       FROM            OR         TYPE
DEFINITION  IF              POINTER    UNTIL
DIV         IMPLEMENTATION  PROCEDURE  VAR
DO          IMPORT          QUALIFIED  WHILE
ELSE        IN              RECORD     WITH
```

Built-in identifiers

PIM [3,4] defines the following 29 built-in identifiers:

```
ABS         EXCL            LONGINT    REAL
BITSET      FALSE           LONGREAL   SIZE
BOOLEAN     FLOAT           MAX        TRUE
CAP         HALT            MIN        TRUNC
CARDINAL    HIGH            NIL        VAL
CHAR        INC             ODD
CHR         INCL            ORD
DEC         INTEGER         PROC
```

Use in embedded systems

Cambridge Modula-2

Cambridge Modula-2 by Cambridge Microprocessor Systems is based on a subset of PIM4 with language extensions for embedded development. The compiler runs on DOS and it generates code for Motorola 68000 series (M68k) based embedded microcontrollers running a MINOS operating system.

Mod51

Mod51 by Mandeno Granville Electronics is based on ISO Modula-2 with language extensions for embedded development following IEC1131, an industry standard for programmable logic controllers (PLC) closely related to Modula-2. The Mod51 compiler generates standalone code for 80C51 based microcontrollers.

Modula-GM

Delco Electronics, then a subsidiary of GM Hughes Electronics, developed a version of Modula-2 for embedded control systems starting in 1985. Delco named it Modula-GM. It was the first high-level programming language used to replace machine code (language) for embedded systems in Delco's *engine control units* (ECUs). This was significant because Delco was producing over 28,000 ECUs per day in 1988 for GM. This was then the world's largest producer of ECUs.[19] The first experimental use of Modula-GM in an embedded controller was in the 1985 Antilock Braking System Controller which was based on the Motorola 68xxx microprocessor, and in 1993 Gen-4 ECU used by the Champ Car World Series Championship Auto Racing Teams (CART) and Indy Racing League (IRL) teams.[20] The first production use of Modula-GM was its use in GM trucks starting with the 1990 model year *vehicle control module* (VCM) used to manage GM Powertrain's Vortec engines. Modula-GM was also used on all ECUs for GM's 90° Buick V6 engine family 3800 Series II used in the 1997-2005 model year Buick Park Avenue. The Modula-GM compilers and associated software management tools were sourced by Delco from Intermetrics.

Modula-2 was selected as the basis for Delco's high level language because of its many strengths over other alternative language choices in 1986. After Delco Electronics was spun off from GM (with other component divisions) to form Delphi Automotive Systems in 1995, global sourcing required that a non-proprietary high-level software language be used. ECU embedded software now developed at Delphi is compiled with commercial compilers for the language C.

Russian GPS satellites

According to the Modula-2 article in the Russian Wikipedia, all satellites of the Russian GPS framework GLONASS are programmed in Modula-2.

Compilers

- ACK (https://github.com/davidgiven/ack) Modula-2 for MINIX (freeware)
- ADW Modula-2 (http://www.modula2.org/adwm2/) ADW Modula-2 for Windows, ISO compliant, ISO/IEC 10514-1, ISO/IEC 10514-2 (OO extension), ISO/IEC 10514-3 (Generic extension) (freeware)
- Aglet Modula-2 (http://aglet.web.runbox.net) for Amiga OS 4.0/PPC (freeware)
- Cambridge Modula-2 (http://www.cms.uk.com) for various micro-controllers and embedded MINOS operating system (commercial + proprietary software)
- FST (ftp://ftp.psg.com/pub/modula-2/fst/fst-40s.lzh) Fitted Software Tools Modula-2 for DOS (freeware)
- Gardens Point Modula-2 (https://web.archive.org/web/20130323002046/http://plas.fit.qut.edu.au/gpm/) for BSD, Linux, OS/2 and Solaris - ISO compliant (freeware), as of July 30, 2014, original website (http://plas.fit.qut.edu.au/gpm/) is down
- Gardens Point Modula-2 (GPM/CLR) (http://gpmclr.codeplex.com) for .NET (freeware)
- GNU Modula-2 (http://www.nongnu.org/gm2/) for GCC platforms, version 1.0 released December 11, 2010; PIM2, PIM3, PIM4, and ISO compliant (free software, GPLed)
- M2Amiga (https://web.archive.org/web/20190120022439/https://m2amiga.claudio.ch/) for Amiga (free software)

- M2M (http://www.cfbsoftware.com/modula2) by N. Wirth and collaborators from ETH Zurich, platform independent, generates M-code for virtual machine (freeware)
- MacMETH (http://www.sysecol2.ethz.ch/RAMSES/MacMeth.html) by N. Wirth and collaborators from ETH Zurich for Macintosh, but Classic only (freeware)
- Mod51 (https://web.archive.org/web/20081021043956/http://www.designtools.co.nz/mod51.htm) for the Intel 80x51 micro-controller family, ISO compliant, IEC1132 extensions (commercial + proprietary)
- Megamax Modula-2 (http://www.tempel.org/files-e.html) for Atari ST computers by T. Tempelmann. Documentation only in German language.
- Modula-2 R10 (http://www.ohloh.net/p/m2r10) Reference compiler for Modula-2 R10 (open-source/peer-review)
- ModulaWare (http://www.modulaware.com) for OpenVMS, both VAX and Alpha, ISO compliant (commercial + proprietary)
- Native XDS-x86 (https://web.archive.org/web/20090511080825/http://www.excelsior-usa.com/xdsx86.html) for Windows and Linux (x86), ISO compliant, TopSpeed compatible library (freeware)
- ORCA/Modula-2, a compiler available for the Apple Programmer's Workshop, for the Apple IIgs
- p1 Modula-2 (http://modula2.awiedemann.de) for Macintosh, both Classic and Mac OS X (PPC and Carbon API only), ISO compliant (commercial + proprietary)
- The Karlsruhe Modula-2 Compiler MOCKA (https://web.archive.org/web/20090904204227/http://www.info.uni-karlsruhe.de/~modula/index.php) for various platforms, PIM compliant (commercial, freeware Linux/BSD versions)
- TDI Modula-2 for Atari ST from TDI Software, Inc.
- TERRA M2VMS (http://www.terraterra.ch/modula-2/spdm2vms.html) for OpenVMS, both VAX and Alpha, PIM compliant (commercial + proprietary)
- The Ulm Modula-2 System (http://www.mathematik.uni-ulm.de/modula) for Solaris, both SPARC and MC68K (free software, GPLed)
- XDS-C (https://web.archive.org/web/20090529005403/http://www.excelsior-usa.com/xdsc.html) for Windows and Linux, 16- and 32-bit platforms, targeting C (K&R & ANSI), ISO compliant, TopSpeed compatible library (freeware)

Turbo Modula-2

Turbo Modula-2 was both a compiler and an integrated development environment (IDE) for Modula-2 running on DOS, developed by Borland, but unreleased by them. Instead, a group including Borland cofounder Niels Jensen, acting as Jensen and Partners, bought the unreleased codebase and redeveloped and released it as TopSpeed Modula-2. TopSpeed was eventually sold to Clarion, now owned by SoftVelocity, which still offers the Modula-2 compiler as part of its Clarion product line.

A Z80 CP/M version of Turbo Modula-2 was briefly marketed by Echelon, Inc., under license from Borland. A companion release for Hitachi HD64180 was marketed by Micromint, Inc., as a development tool for their SB-180 single-board computer.

Books

- Wirth, Niklaus (1988). *Programming in Modula-2* (https://link.springer.com/book/10.1007/978-3-642-83565-0) (4th ed.). Berlin Heidelberg: Springer-Verlag. ISBN 978-0-387-96051-7.
- King, K. N. (1 January 1988). *Modula-2: A Complete Guide* (http://knking.com/books/modula2/). Burlington, Massachusetts: Jones and Bartlett Publishers. ISBN 978-0669110913.

- Sutcliffe, Richard J. (2004–2005). *Modula-2: Abstractions for Data and Programming Structures* (http://www.arjay.bc.ca/Modula-2/Text/). Arjay Books. Uses ISO-Standard Modula-2.
- Gleaves, Richard (1984). *Modula-2 for Pascal Programmers* (https://link.springer.com/book/10.1007%2F978-1-4613-8531-8) (1st ed.). Switzerland: Springer Nature. ISBN 978-0-387-96051-7.
- Cooper, Doug (1 September 1990). *Oh My! Modula-2: An Introduction to Programming*. New York City, New York: W. W. Norton & Company. ISBN 978-0393960099.

References

1. Summary of Projects by N. Wirth, 1962–1999 (http://www.inf.ethz.ch/personal/wirth/projects.html)
2. N. Wirth, "Pascal and its Successors". *Software Pioneers*, M. Broy and E. Denert, Eds. Springer-Verlag, 2002, ISBN 3-540-43081-4
3. Niklaus Wirth, "History and Goals of Modula-2". (http://www.drdobbs.com/open-source/history-and-goals-of-modula-2/223000094)*Dr. Dobb's Journal*, 2005
4. N. Wirth, Programming in Modula-2, fourth Edition, page 4.
5. *Byte – The Small Systems Journal*, 1984 (8), pp. 143–232. Available at archive.org (https://archive.org/details/byte-magazine-1984-08)
6. Wirth, N. (1 May 1984). "A Single-pass Modula-2 Compiler for Lilith" (http://www.cfbsoftware.com/modula2/M2SinglePass.pdf) (PDF). *CFB Software*. Retrieved 28 January 2019.
7. ISO/IEC 10514-1:1996
8. Tichy et al., Modula-2*: An Extension for Modula-2 for highly parallel portable programs, University of Karlsruhe [1990]
9. Bräunl, Parallel Programming, Prentice-Hall [1993]
10. Muller, C. (1986). "Modula--Prolog: A Software Development". *IEEE Software*. **3** (6): 39–45. doi:10.1109/MS.1986.229475 (https://doi.org/10.1109%2FMS.1986.229475).
11. modula2.org, 5. Where can I get information about ISO Modula-2?, [1] (http://freepages.modula2.org/m2faq.html)
12. modula2.org, 5. Where can I get information about ISO Modula-2?, [2] (http://freepages.modula2.org/m2faq.html)
13. Cardelli et al., *Modula-3 Report*, Research Report 31, Systems Research Center, Digital Equipment Corporation, [1988]
14. N. Wirth & J. Gutknecht, Project Oberon: the design of an operating system and compiler, ISBN 0-201-54428-8, Addison-Wesley [1992]
15. Moessenboeck & Wirth, *The Programming Language Oberon-2*, ETH Zurich [1995]
16. Thomas Bräunl, Parallaxis, a Language for Structured Data-parallel Programming, University of Stuttgart [1996]
17. Pat D. Terry, Another Minimal Programming Language, ACM SIGPLAN Notices No. 30 [1995]
18. D. Blasband, The YAFL Programming Language, Second Edition, [1993]
19. Delco Electronics Electron Magazine, The Atwood Legacy, Spring '89, page 25
20. Development of electronics for GM auto racing

External links

- Official website (http://www.modula2.org)