# A Comparison of C++, FORTRAN 90 and Oberon-2 for Scientific Programming

Bernd Mösli
ARITHMETICA
CH-8307 Effretikon, Switzerland
moesli@arithmetica.ch

May 12, 1995

## Abstract

In the past decade, the programming languages C++, FORTRAN 90 and Oberon-2 allevolved from their ancestors. This invites to reflect upon the suitability ofthese modern programming languages for scientific and engineering computing. Inthe first part, we compare their primary language features, as needed byscientists and engineers. In the second part, we list some useful featuresmissing in Oberon-2. The report concludes by a personal assessment of the threelanguages with respect to the numerical context. The reader's experience inscientific programming in C or FORTRAN would be advantageous.

## 1 Introduction

Business computing holds the major share of the computer market. Here COBOL,PL/1 and C have been the languages of choice, but C slowly supersedes itscompetitors in recent software projects.

Scientific computing holds a minor market share. FORTRAN 77 dominatedprogramming in science and engineering in the past. The importance of Cincreases in all programming fields, especially in science and engineering.Data have been the primary valuables of business computing, while programs havebeen the primary valuables of scientific computing. Hence, switching fromFORTRAN environments to C or Oberon is generally more laborious than switchingfrom COBOL environments to C or Oberon.

The small remainder of the market is shared by dedicated applications, assystem software, for example. Oberon [6, 9] evolved from Modula-2[5]. Oberon-2 [7, 8, 19] has been and will be a valuablealternative to C++ [1, 2] in any context, where general-purposeprogramming languages are appropriate [10, 11]. We compare Oberon-2 onlyto C++ and FORTRAN 90 [3, 4], since these programming languages areextensions of their wide-spread ancestors C and FORTRAN 77. In the following, afamiliarity with these languages is expected.

## 2 Comparison

We compare only the main language features being of interest in a scientificand engineering context [12, 13, 14,15, 16, 17, 18]. Object-orientedfeatures are not discussed (see [19, 20, 21, 22]).

### 2.1 Identifiers and Reserved Words

Identifiers consist of a sequence of uppercase and lowercase letters,underscore characters and digits with leading letter.
**C++:** Reserved words consist of lowercase letters.
**FORTRAN 90:** Any identifiers up to 31 characters. Pitfall: reserved words maybe valid identifiers.
**Oberon-2:** No underscore characters allowed. Reserved words consist of uppercase letters.

### 2.2 Records

After more than three decades, FORTRAN 90 offers the record type. While somelanguage designers luckily leave out *packed records* (used in Pascal), C++and Modula-2 allow *variant records* that are unsafe or inefficient(run-time checking of the variant).
**C++:** Variant records, initialisation aggregates.
**FORTRAN 90:** Records and initialisation aggregates. The sequence statementforces record fields to be stored in order of definition.
**Oberon-2:** Exported record types may have *public* fields (visible outsidethe module) and *private* fields. Record types may be extensions of anotherrecord type. No initialisation aggregates available.

### 2.3 Multidimensional Arrays

It should be possible to create multidimensional arrays at run-time. The staticmemory management of FORTRAN 77 caused long parameter lists of procedures. Thelocal temporary arrays (*workspace* ) and array dimension had to be passedas parameters. The creation of global and local multidimensional arrays (withpossibly no elements) at run-time, allows safe array handling and readablesource code. The restriction to 7 dimensions in FORTRAN does not harm.
**C++:** Static multidimensional arrays, dynamic one-dimensional arrays. Lowerindex bounds are always zero.
**FORTRAN 90:** Static and dynamic multidimensional arrays up to 7 dimensions. Indexbounds may be declared. Lower index bounds of arrays in parameter lists ofprocedures may be declared.
**Oberon-2:** Dynamic multidimensional arrays. Lower index bounds are always zero.

### 2.4 Use of Uninitialised Variables

Using uninitialised non-pointer variables invalidates the program run (wrongdata, number overflow, array index error, etc.), but processing invalidpointers or procedure variables may crash the run-time system. The latter ismore harmful than the former.
**C++:** Possible.
**FORTRAN 90:** Possible.
**Oberon-2:** Possible. Local pointers may be initialised to NIL automatically,global pointers are always initialised to NIL.

### 2.5 Pointer Manipulation

Low-level pointer manipulation should be restricted to modules close tohardware as device drivers, for example. Pointer manipulation is oftenresponsible for software flaws and prevents portability.
**C++:** Abuse is easy, even constants or stack objects could be defected.

**FORTRAN 90:** None.
**Oberon-2:** Only by explicit type cast.

### 2.6 Dangling References

Dynamic data structures and abstract data types are usually implemented throughpointers. When explicit deallocation of objects is possible, unintentionaldereferencing of *dangling pointers* leads to system corruption, which ishard to debug.
**C++:** Possible.
**FORTRAN 90:** Possible.
**Oberon-2:** Safe, because no explicit freeing of objects (garbage collection frees objects).

### 2.7 Logical Objects and Operators

Logical objects and operators are part of conditional statements anyway. Theintegers zero and one are no adequate substitutes for logical values *false* and *true* , they rather confuse the programmer. The rather simpleC++ statement "if(a=b && c+d<e)f" is legal.
**C++:** No logical objects but bitwise operations on characters,enumerations, integers and bit fields. Conditions return integer values.
**FORTRAN 90:** Type LOGICAL with operators.
**Oberon-2:** Type BOOLEAN with operators.

### 2.8 Set Objects and Operators

Bit sets need less storage and operations are faster, compared to arrays oflogical values. In some application fields, restricting to bit sets is notflexible enough.
**C++:** Set operations on integers, bit access by masking.
**FORTRAN 90:** Set and bit operations on integers.
**Oberon-2:** Set and bit operations on bit sets.

### 2.9 Relations

Beneath simple comparison operators, relations of structured types should bepossible. Relation overloading is necessary for abstract data types.
**C++:** Relations apply to characters, enumerations, numeric and pointer types.Warning: "a<b<c" means "(a<b)<c" and not "(a<b) and (b<c)".
**FORTRAN 90:** Relations apply to strings and numeric types except complex types,equality applies to complex types.
**Oberon-2:** Relations apply to numeric types, characters and strings, equalityalso applies to logical, set, pointer and procedure types.

### 2.10 Arithmetic Operators

The numerous numeric types are rather hindering than a benefit. The unsignedintegers in C++ (and Modula-2) have been primarily used for address arithmetic.The type COMPLEX of FORTRAN could be implemented by structured functionresults and operator overloading. The subtypes in FORTRAN 90 causes even moreproblems. Mixing different numeric types in expressions leads to conversionproblems. It is tricky to manage operator overloading and automatic typecoercion simultaneously. It is quite stupendous that none of these programminglanguages supports fixed point numbers for business computing. The number

ofdecimal digits in 32-bit integers and 32-bit reals are by far too small. Ingeneral, the definition of numeric types (no minimum range!) and operatorsshould be more precise and portable.
**C++:** +,- apply to characters, enumerations, numeric and pointer types.*,/ apply to characters, enumerations and numeric types. % (*remainder* )applies to characters, enumerations and types where x = (x/y)*y + (x%y)holds, but for xϴ or yϴ the sign of the remainder is implementationdependent [1] ! No complex type.
**FORTRAN 90:** +,-,*,/,** (exponentiation, a**b**c = a**(b**c) but2**(-3) truncates to zero) apply to integer, real and complex types.Fractions of integers are truncated towards zero.
**Oberon-2:** +,-,*,/ (/ with real result), DIV (integers only, round to- infinity), MOD (integers only, *modulus* ) where x = (x DIV y)*y + (xMOD y), 0<=(x MOD y)<y holds. Numeric types are coerced automaticallyaccording to the so-called *type inclusion* : SHORTINT <= INTEGER<= LONGINT <= REAL <= LONGREAL. No complex type.

### 2.11 Go-to Statement

The lack of well-structured control statements in older programming languageslead to excessive use of the harmful go-to statement. Maintaining so-called*spaghetti code* is expensive.
**C++:** Unconditional go-to, return statement in procedures, break statement initerations and switch (case) statements.
**FORTRAN 90:** Unconditional go-to, arithmetic if (a conditional jump), computedgo-to (a case statement), assigned go-to (a jump to label variable). Returnstatement in procedures.
**Oberon-2:** No explicit go-to, but return and exit statements in procedures and loops.

### 2.12 Procedures and Parameters

**C++:** Types and number of parameters are checked except for special procedures(as *printf* ), where the number of parameters is unspecified. Optionalparameters are possible. Parameter passing: call by value (modify local copy),call by reference (modify original), and call by reference with prefix *const* (read-only). Arrays can not be passed by value. The number of array elements must be passed separately.Procedure variables are pointers to functions.
**FORTRAN 90:** Types and number of parameters are checked. Optional parametersare possible. Parameter passing: call by value or call by reference (compilerdecision)! The programmer may assign attributes to parameters: *in* forread-only parameters, *out* for returned, and *inout* for modifiedparameters. The number of array elements may be passed separately (*assumed-shape array, automatic array* ). Procedure variables must be external ormodule procedures.
**Oberon-2:** Types and number of parameters are checked. Parameter passing: call byvalue and call by reference. Procedure variables can not be predefined procedures.

### 2.13 Recursive Procedure Calls

Calling procedures recursively allows adequate programming of numerousalgorithms based on divide and conquer.
**C++:** Possible.
**FORTRAN 90:** Possible when procedure marked as recursive.
**Oberon-2:** Possible.

### 2.14 Overloading of Procedures and Operators

Procedure and operator overloading supports implementation of libraries.Operator precedence and associativity is important. Overloading of predefinedprocedures and operators should be possible.
**C++:** Overloading of procedures and operators, definition of new operators.
**FORTRAN 90:** Overloading of procedures and operators, definition of newoperators.
**Oberon-2:** None.

### 2.15 Exception and Error Handling

An error handling by the programmer enables handling of exceptions or errorswithin libraries. Preventing some errors is often less efficient and morecomplicated than handling exceptions, as numeric overflow, for example.
**C++:** By exception handler.
**FORTRAN 90:** None.
**Oberon-2:** None.

### 2.16 Language Support for Parallelism

Parallelism is needed to solve large and time-consuming problems in science,engineering and business. Actually, the oldest languages are used to programsupercomputers. Parallelism should be supported by modern general-purposeprogramming languages.
**C++:** None.
**FORTRAN 90:** No explicit parallelism, but implicit parallelism in arrayoperations.
**Oberon-2:** None.

### 2.17 Programs and Compilation Units

A *program* consists of several *compilation units* , each encapsulatingdata declarations and code. Compilation units are compiled and storedseparately. The compilation unit *interface* controls access to dedicatedlocal objects. Interface consistency means that the interface is consistent withclient and server.
**C++:** Files are compilation units. Interface objects are declared in so-called*header files* . Interface consistency is not checked.
**FORTRAN 90:** A program consists of one compilation unit (*main program* ) andoptional compilation units (modules, external functions and procedures). Theobject attributes *private* and *public* in modules control the accessof clients. Interface consistency is not checked, but user may copy theinterface into compilation unit as so-called *interface block* , which ischecked locally.
**Oberon-2:** Modules are compilation units. Interface objects are marked in thesource. Read-or-modify control for exported variables is available. Interfaceconsistency is checked.

## 3 Features Missing in Oberon-2

Oberon-2 needs additional features to enlarge the field of application. Some ofthe features described subsequently are new, while some fit well in the presentlanguage definition [8, 19]. Features of minor importance, as exception handling, are not listed. The first four features primarily improve readabilityand flexibility. Library programmers will benefit most. The last feature,parallelism, is mandatory for future languages, since parallel computing willsoon be available on workstations and personal computers. At present, parallelsystems are mainly

programmed in FORTRAN and C (with hardware-dependentlanguage extensions). If Oberon will not offer parallelism soon, it will hardlybe possible to compete with future C++ and FORTRAN 90 environments.

### 3.1 Arrays

The array features are important for porting the FORTRAN libraries to Oberon-2.In Oberon-2, constant arrays are missing. For initialisation of arrays,see aggregation. FORTRAN 90 has a different storage representation of arrays(column-by-column) than C++ and Oberon-2 (row-by-row). This impedes calling ofFORTRAN libraries by C++ or Oberon-2 programs. The different lower arraybounds are a severe obstacle for porting FORTRAN software to C++ and Oberon-2.

### 3.2 Arbitrary Function Result Types

Actually, the result type of a procedure can be neither a record nor an array.Arbitrary types as function results allows more compact and readable sourcecode. Combined with overloaded operators, the language permits orthogonalextensions of expressions. The type COMPLEX should be offered by a portablelibrary module, and not by the compiler (as FORTRAN does). The compilercomplexity will be reduced. The flexibility will be enhanced. See alsoaggregation and operators.

### 3.3 Operators

The language is more orthogonal, readable and flexible when some operators maybe overloaded. Abstract mathematical data types (complex numbers, matrices,polynomials etc.) will be easier to implement. This supports sophisticatedprogramming, as used in mathematical expert systems or software libraries.There may be some pitfalls when using numeric type hierarchy and automatic typeextension in mixed mode. It is advantageous to merge the numeric type hierarchyin the operator concept. Arbitrary operator definitions, as in PROLOG, shouldbe omitted for simplicity.

### 3.4 Aggregation

Aggregates lightens the initialisation of structured types (records, arrays,etc.) in definitions and assignments. This is cumbersome and error-prone whenprogrammed explicitly. The code will be more efficient and more readable.

### 3.5 Parallelism

This feature is most ambitious [24]. Some challenging problems (weatherforecast, fluid dynamics, molecular design etc.) are very time-consuming. Theyrequire the computing power of parallel systems, since the hardware developmentof sequential computers (temporarily) reaches physical or commercial limits.Hence, Oberon-2 should offer parallelism to tackle this performance problem. Inaddition to efficiency, parallelism is a valuable structuring tool. An Oberonextension for vector computers is described in [23]. The programming ofsequential computers would benefit too.

## 4 Summary

We compile a comparative overview of the language features. The marks mean:-- no support, - poor, + average, ++ good support. Most motives for theranking are discussed in the first part of this paper.

| **Feature** | **C++** | **F 90** | **O-2** |
|---|---|---|---|
| the language is easy to learn, easy to use | -- | -- | ++ |
| one language construct per features only | - | -- | + |
| are numeric libraries for the language available | - | ++ | -- |
| object-oriented features | + | - | + |
| exception and error handling | - | -- | -- |
| support for parallelism | -- | - | -- |
| block-structured features | - | -- | + |
| go-to like statements avoided | - | -- | + |
| | | | |
| variable initialisation in definition | ++ | + | -- |
| logical and set types | - | - | ++ |
| type complex | -- | ++ | -- |
| record extension/variants | + | - | ++ |
| initialisation aggregates for records | + | + | -- |
| safe multidimensional arrays | -- | + | ++ |
| user defined lower index bound | -- | ++ | -- |
| | | | |
| safe pointer handling | -- | ++ | + |
| forced pointer initialisation | -- | -- | + |
| omit dangling references | -- | -- | ++ |
| | | | |
| all types as procedure parameters | - | - | + |
| call by value of procedure parameters | - | + | ++ |
| call by reference of procedure parameters | + | + | ++ |
| arbitrary function result types | + | + | + |
| procedure and operator overloading | ++ | ++ | -- |
| procedure recursion available | + | ++ | + |
| safe module and interface | -- | - | ++ |

After three decades, the FORTRAN language became a partly useful patchwork ofnumerous features, reflecting the history of software techniques. Actually, toomany constructs cover the same features with different side-effects andrestrictions. This aggravates software development, and is of no benefit atall. The dusty decks will keep dusty.

In the chapter "design notes" [1], the author writes, "Simplicity was animportant design criterion for C++ ..."! But the language reference chapter forC++ covers about 150 pages, versus less than 30 pages for Oberon [9] orOberon-2 [19]. A tool should support the solving of problems, not createproblems. The originally hardware-oriented design lacks a sound

programmer-oriented model. C++offers too many concepts. Their interaction is error-prone.
We listed some features to be included in Oberon-2. Applications written inthis value-added Oberon-2 will be safer and more readable. At present,Oberon-2 implementations are not less efficient than C++ implementations atall. The language is easier to learn than FORTRAN 90 and C++; compare the sizeof their reports; [1] 680 pages, [3] 740 pages, and [8] 16pages. The small number of language constructs facilitates a correct compilerimplementation. The programmer easily understands and memorises theinterference of Oberon-2 constructs, which is hardly possible for FORTRAN 90 and C++.

## Conclusions

At present, the old-fashioned but value-added FORTRAN 90 seems to be inevitablefor scientific and engineering work, when one of the numerous FORTRAN librariesis required.
When starting from scratch, or when the necessary numerical libraries areavailable, Oberon-2 competes with FORTRAN 90 and C++ at ease. We prefer usingan improved Oberon-2 to master complex systems, for not being mastered bycomplex systems written in FORTRAN or C.

## Acknowledgements

The author would like to thank the referees for helpful suggestions, and JörgWaldvogel, Seminar for Applied Mathematics ETH Zürich, for carefully reading this paper.

## Bibliography

[1] B. Stroustrup: *The C++ Programming Language,* Addison-Wesley, 1993.

[2] B. Stroustrup: *The Design and Evolution of C++,* Addison-Wesley, 1994.

[3] J.C. Adams, W.S. Brainerd, J.T. Martin, B.T. Smith, J.L. Wagener:*Fortran 90 Handbook. Complete ANSI/ISO Reference,* McGraw-Hill, 1992.

[4] M. Metcalf, J. Reid: *Fortran 90 Explained,* Oxford University Press, 1990.

[5] N. Wirth: *From Modula to Oberon,* Dept. Informatik, Report 143, ETH Zürich, 1990.

[6] N. Wirth: *The Programming Language Oberon,* Dept. Informatik, Report 143, ETH Zürich, 1990.

[7] H. Mössenböck, N. Wirth: *Differences between Oberon andOberon-2,* Structured Programming, Vol. 12 Iss. 4 p. 175-177, 1991.

[8] H. Mössenböck, N. Wirth: *The Programming LanguageOberon-2,* Structured Programming, Vol. 12 Iss. 4 p. 179-195, 1991.

[9] M. Reiser, N. Wirth: *Programming in Oberon. Steps beyond Pascal and Modula,* Addison-Wesley, 1992.

[10] M. Reiser: *The Oberon System. User Guide and Programmer's Manual,* Addison-Wesley, 1991.

[11] J. Gutknecht, N. Wirth: *Project Oberon. The Design of anOperating System and Compiler,* Addison-Wesley, 1992.

[12] T.D. Brown: *C for FORTRAN Programmers,* Silicon Press, 1990.

[13] R. Schäfer, F. Bomarius: *Modula2C. Ein Übersetzer vonModula-2 nach C,* Report 178, Universität Kaiserslautern, 1988.

[14] C.A. Wiatrowski, R.S. Wiener: *From C to Modula-2 and Back.Bridging the Language Gap,* Wiley & Sons, 1987.

[15] J.T. Smith: *C++ for Scientists and Engineers,* McGraw-Hill, 1991.

[16] *Fortran and C in Scientific Computing,* Brunel Conference Center, London, 1993.

[17] C. Überhuber, P. Meditz: *Software-Entwicklung in Fortran 90,* Springer, 1993.

[18] A.J.E. van Delft: *Comments on Oberon,* SIGPLAN Notices, Vol. 24 Iss. 3 p. 23-30, 1989.

[19] H. Mössenböck: *Object-oriented Programming in Oberon-2,* Springer, 1993.

[20] G. Blaschek, G. Pomberger, A. Stritzinger: *A Comparison ofObject-Oriented Programming Languages,* Structured Programming, Vol. 10 Iss. 4 p. 187-197, 1989.

[21] R. Henderson, B. Zorn: *A Comparison of Object-orientedProgramming in Four Modern Languages,* Software - Practice and Experience, Vol. 24 Iss. 11 p. 1077-1095, 1994.

[22] J. Templ: *Vergleich der Programmiersprachen Oberon und C++,* iX. Multiuser. Multitasking, Iss. 9 p. 138-143, 1994.

[23] R. Griesemer: *A Programming Language for Vector Computers,* dissertation, ETH Zürich, 1993.

[24] P.D. Stotts: *A Comparative Survey of Concurrent ProgrammingLanguages,* ACM SIGPLAN Notices, Vol. 17 Iss. 10 p. 50-61, 1982.

---

**Report Reference**

Bernd Mösli
A Comparison of C++, FORTRAN 90 and Oberon-2 for Scientific Programming,
GISI 95,
editors: Friedbert Huber-Wäschle, Helmut Schauer, Peter Widmayer
Berlin, Springer, p. 740-748, 1995.

Full german title:
GISI 95: Herausforderungen eines globalen Informationsverbundes für die Informatik.

25. GI-Jahrestagung und 13. Schweizer Informatikertag,Zürich, 18-20. September 1995,
ISBN 3-540-60213-5

---