Chapter 3
Introduction to Programming Languages

We have to go to another language in order to think clearly about the problem

Samuel R. Delany, Babel-17

Aims
The primary aim of this chapter is to provide a short history of program language development and give some idea as to the concepts that have had an impact on Fortran. It concentrates on some but not all of the major milestones of the last 40 years, in roughly chronological order. The secondary aim is to show the breadth of languages available. The chapter concludes with coverage of a small number of more specialised languages.

3.1 Introduction

It is important to realise that programming languages are a recent invention. They have been developed over a relatively short period — 60 years — and are still undergoing improvement. Time spent gaining some historical perspective will help you understand and evaluate future changes. This chapter starts right at the beginning and takes you through some, but not all, of the developments during this 55 year span. The bulk of the chapter describes languages that are reasonably widely available commercially, and therefore ones that you are likely to meet. The chapter concludes with a coverage of some more specialised and/or recent developments.

I. Chivers and J. Sleightholme, *Introduction to Programming with Fortran*,
https://doi.org/10.1007/978-3-319-75502-1_3

3.2 Some Early Theoretical Work

Some of the most important early theoretical work in computing was that of Turing and von Neumann. Turing's work provided the base from which it could be shown that it was possible to get a machine to solve problems. The work of von Neumann added the concept of storage and combined with Turing's work to provide the basis for most computers designed to this day.

3.3 What Is a Programming Language?

For a large number of people a programming language provides the means of getting a digital computer to solve a problem. There is a wide range of problems and an equally wide range of programming languages, with particular languages being suited to a particular class of problems, all of which often appears bewildering to the beginner.

3.4 Program Language Development and Engineering

There is much in common between the development of programming languages and the development of anything from the engineering world. Consider the car: old cars offer much of the same functionality as more modern ones, but most people prefer driving newer models. The same is true of programming languages, where you can achieve much with the older languages, but the newer ones are easier to use.

3.5 The Early Days

A concept that proves very useful when discussing programming languages is that of the level of a machine. By this is meant how close a language is to the underlying machine that the program runs on. In the early days of programming (up to 1954) there were only two broad categories: machine languages and assemblers. The language that digital machines use is that of 0 and 1, i.e., they are binary devices. Writing a program in terms of patterns of 0 and 1 was not particularly satisfactory and the capability of using more meaningful mnemonics was soon introduced. Thus it was realised quite quickly that one of the most important aspects of programming languages is that they have to be read and understood by both machines and humans.

3.5.1 Fortran's Origins

The next stage was the development of higher-level languages. The first of these was Fortran and it was developed over a 3 year period from 1954 to 1957 by an IBM team led by John Backus. This group achieved considerable success, and helped to prove that the way forward lay with high-level languages for computer-based problem solving. Fortran stands for formula translation and was used mainly by people with a scientific background for solving problems that had a significant arithmetic content. It was thus relatively easy, for the time, to express this kind of problem in Fortran.

By 1966 and the first standard Fortran:

- Was widely available.
- Was easy to teach.
- Had demonstrated the benefits of subroutines and independent compilation.
- Was relatively machine independent.
- Often had very efficient implementations.

Possibly the single most important fact about Fortran was, and still is, its widespread usage in the scientific community.

3.5.2 Fortran 77

The next standard in 1977 (actually 1978, and thus out by one — a very common programming error, more of this later!) added a number of major improvements including

- Block IF and END IF statements, with optional ELSE and ELSE IF clauses, to provide improved language support for structured programming
- DO loop extensions, including parameter expressions, negative increments, and zero trip counts
- OPEN, CLOSE, and INQUIRE statements for improved I/O capability
- IMPLICIT statement, to override implicit conventions that undeclared variables are INTEGER if their name begins with I, J, K, L, M, or N (and REAL otherwise)
- CHARACTER data type, replacing Hollerith strings with vastly expanded facilities for character input and output and processing of character-based data
- PARAMETER statement for specifying constants
- SAVE statement for persistent local variables
- Generic names for intrinsic functions
- A set of intrinsics (LGE, LGT, LLE, LLT) for lexical comparison of strings

One important feature sometimes overlooked was backwards compatibility. This meant that the standard did not invalidate any standard conformant Fortran 66 program. This protected investment in old code.

3.5.3 Cobol

The business world also realised that computers were useful and several languages were developed, including FLOWMATIC, AIMACO, Commercial Translator and FACT, leading eventually to Cobol — COmmon Business Orientated Language. There is a need in commercial programming to describe data in a much more complex fashion than for scientific programming, and Cobol had far greater capability in this area than Fortran. The language was unique at the time in that a group of competitors worked together with the objective of developing a language that would be useful on machines used by other manufacturers.

The contributions made by Cobol include:

- Firstly the separation among:
- The task to be undertaken.
- The description of the data involved.
- The working environment in which the task is carried out.
- Secondly a data description mechanism that was largely machine independent.
- Thirdly its effectiveness for handling large files.
- Fourthly the benefit to be gained from a programming language that was easy to read.

Modern developments in computing — of report generators, file-handling software, fourth-generation development tools, and especially the increasing availability of commercial relational database management systems — are gradually replacing the use of Cobol, except where high efficiency and/or tight control are required.

3.5.4 Algol

Another important development of the 1950s was Algol. It had a history of development from Algol 58, the original Algol language, through Algol 60 eventually to the Revised Algol 60 Report. Some of the design criteria for Algol 58 were:

- The language should be as close as possible to standard mathematical notation and should be readable with little further explanation.
- It should be possible to use it for the description of computing processes in publications.
- The new language should be mechanically translatable into machine programs.

A sad feature of Algol 58 was the lack of any input/output facilities, and this meant that different implementations often had incompatible features in this area.

The next important step for Algol occurred at a UNESCO-sponsored conference in June 1959. There was an open discussion on Algol and the outcome was Algol 60, and eventually the Revised Algol 60 Report.

It was at this conference that John Backus gave his now famous paper on a method for defining the syntax of a language, called Backus Normal Form, or BNF. The full

significance of the paper was not immediately recognised. However, BNF was to prove of enormous value in language definition, and helped provide an interface point with computational linguistics.

The contributions of Algol to program language development include:

- Block structure.
- Scope rules for variables because of block structure.
- The BNF definition by Backus — most languages now have a formal definition.
- The support of recursion.
- Its offspring.

Thus Algol was to prove to make a contribution to programming languages that was never reflected in the use of Algol 60 itself, in that it has been the parent of one of the main strands of program language development.

3.6 Chomsky and Program Language Development

Programming languages are of considerable linguistic interest, and the work of Chomsky in 1956 in this area was to prove of inestimable value. Chomsky's system of transformational grammar was developed in order to give a precise mathematical description to certain aspects of language. Simplistically, Chomsky describes grammars, and these grammars in turn can be used to define or generate corresponding kinds of languages. It can be shown that for each type of grammar and language there is a corresponding type of machine. It was quickly realised that there was a link with the earlier work of Turing.

This link helped provide a firm scientific base for programming language development, and modern compiler writing has come a long way from the early work of Backus and his team at IBM. It may seem unimportant when playing a video game at home or in an arcade, but for some it is very comforting that there is a firm theoretical basis behind all that fun.

3.7 Lisp

There were also developments in very specialized areas. List processing was proving to be of great interest in the 1950s and saw the development of IPLV between 1954 and 1958. This in turn led to the development of Lisp at the end of the 1950s. Lisp has proved to be of considerable use for programming in the areas of artificial intelligence, playing chess, automatic theorem proving and general problem solving. It was one of the first languages to be interpreted rather than compiled. Whilst interpreted languages are invariably slower and less efficient in their use of the underlying computer systems than compiled languages, they do provide great opportunities for

the user to explore and try out ideas whilst sitting at a terminal. The power that this gives to the computational problem solver is considerable.

Possibly the greatest contribution to program language development made by Lisp was its functional notation. One of the major problems for the Lisp user has been the large number of Lisp flavours, and this has reduced the impact that the language has had and deserved.

3.8 Snobol

Snobol was developed to aid in string processing, which was seen as an important part of many computing tasks, e.g., parsing of a program. Probably the most important thing that Snobol demonstrated was the power of pattern matching in a programming language, e.g., it is possible to define a pattern for a title that would include Mr, Mrs, Ms, Miss, Rev, etc., and search for this pattern in a text using Snobol. Like Lisp it is generally available as an interpreter rather than a compiler, but compiled versions do exist, and are often called Spitbol. Pattern-matching capabilities are now to be found in many editors and this makes them very powerful and useful tools. It is in the area of text manipulation that Snobol's greatest contribution to program language development lies.

3.9 Second-Generation Languages

3.9.1 PL/1 and Algol 68

It is probably true that Fortran, Algol 60 and Cobol are the three main first-generation high-level languages. The 1960s saw the emergence of PL/1 and Algol 68. PL/1 was a synthesis of features of Fortran, Algol 60 and Cobol. It was soon realised that whilst PL/1 had great richness and power of expression this was in some ways offset by the greater difficulties involved in language definition and use.

These latter problems were also true of Algol 68. The report introduced its own syntactic and semantic conventions and thus forced another stage in the learning process on the prospective user. However, it has a small but very committed user population who like the very rich facilities provided by the language.

3.9.2 Simula

Another strand that makes up program language development is provided by Simula, a general purpose programming language developed by Dahl, Myhrhaug and

Nygaard of the Norwegian Computing Centre. The most important contribution that Simula makes is the provision of language constructs that aid the programming of complex, highly interactive problems. It is thus heavily used in the areas of simulation and modelling. It was effectively the first language to offer the opportunity of object orientated programming, and we will come back to this very important development in programming languages later in this chapter.

3.9.3 Pascal

The designer of Pascal, Niklaus Wirth, had participated in the early stages of the design of Algol 68 but considered that the generality and complexity of Algol 68 was a move in the wrong direction. Pascal (like Algol 68) had its roots in Algol 60 but aimed at providing expressive power through a small set of straightforward concepts. This set is relatively easy to learn and helps in producing readable and hence more comprehensible programs.

It became the language of first choice within the field of computer science during the 1970s and 1980s, and the comment by Wirth sums up the language very well: "although Pascal had no support from industry, professional societies, or government agencies, it became widely used. The important reason for this success was that many people capable of recognising its potential actively engaged themselves in its promotion. As crucial as the existence of good implementations is the availability of documentation. The conciseness of the original report made it attractive for many teachers to expand it into valuable textbooks. Innumerable books appeared between 1977 and 1985, effectively promoting Pascal to become the most widespread language used in introductory programming courses. Good course material and implementations are the indispensable prerequisites for such an evolution."

3.9.4 APL

APL is another interesting language of the early 1960s. It was developed by Iverson early in the decade and was available by the mid to late 1960s. It is an interpretive vector and matrix based language with an extensive set of operators for the manipulation of vectors, arrays, etc., of whatever data type. As with Algol 68 it has a small but dedicated user population. A possibly unfair comment about APL programs is that you do not debug them, but rewrite them!

3.9.5 Basic

Basic stands for Beginners All Purpose Symbolic Instruction Code, and was developed by Kemeny and Kurtz at Dartmouth during the 1960s. Its name gives a clue to

its audience and it is very easy to learn. It is generally interpreted, though compiled versions do exist. It has proved to be well suited to the rapid development of small programs. It is much criticised because it lacks features that encourage or force the adoption of sound programming techniques.

3.9.6 C

There is a requirement in computing to be able to access the underlying machine directly or at least efficiently. It is therefore not surprising that computer professionals have developed high-level languages to do this. This may well seem a contradiction, but it can be done to quite a surprising degree. Some of the earliest published work was that of Martin Richards on the development of BCPL.

This language directly influenced the work of Ken Thompson and can be clearly seen in the programming languages B and C. The UNIX operating system is almost totally written in C and demonstrates very clearly the benefits of the use of high-level languages wherever possible.

With the widespread use of UNIX within the academic world C gained considerable ground during the 1970s and 1980s. UNIX systems also offered much to the professional software developer, and became widely used for large-scale software development and as Ritchie says: “C is quirky, flawed, and an enormous success. while accidents of history surely helped, it evidently satisfied a need for a system implementation language efficient enough to displace assembly language, yet sufficiently abstract and fluent to describe algorithms and interactions in a wide variety of environments.”

There have been several versions of C. Before the language was standardised most people relied on an informal specification contained in the book by Dennis Ritchie and Brian Kernighan, and this version is called K&R C. In 1989 the American National Standards Institute published the ANSI C or C89 standard. It became an ISO standard a year later. The second edition of the K&R book covers the ANSI C standard. ISO later released an extension to the internationalization support of the standard in 1995, and a revised standard (C99) in 1999.

C99 introduced several new features, including inline functions, several new data types (including long long int and a complex type to represent complex numbers), variable-length arrays, improved support for IEEE 754 floating point, support for variadic macros (macros of variable arity), and support for one-line comments beginning with // which are part of C++. This increased the compatibility of C and C++. Many of these had already been implemented as extensions in several C compilers.

The current version of the standard - C11 was approved in December 2011.

The C11 standard adds several new features to C and the library, including type generic macros, anonymous structures, improved Unicode support, atomic operations, multithreading, and bounds-checked functions. It improved compatibility with C++.

3.10 Some Other Strands in Language Development

There are many strands that make up program language development and some of them are introduced here.

3.10.1 Abstraction, Stepwise Refinement and Modules

Abstraction has proved to be very important in programming. It enables a complex task to be broken down into smaller parts concentrating on what we want to happen rather than how we want it to happen. This leads almost automatically to the ideas of stepwise refinement and modules, with collections of modules to perform specific tasks or steps.

3.10.2 Structured Programming

Structured programming in its narrowest sense concerns itself with the development of programs using a small but sufficient set of statements and, in particular, control statements. It has had a great effect on program language design, and most languages now support the minimal set of control structures.

In a broader sense structured programming subsumes other objectives, including simplicity, comprehensibility, verifiability, modifiability and maintenance of programs.

3.10.3 Data Structuring and Procedural Programming

By the 1970's languages started to emerge that offered the ability to organise data logically - so called data structuring, and we will look at two of these in the coverage below - C and Pascal.

C provided this facility via structs and Pascal did it via records. These languages also offered two ways of processing the data - directly or via procedures. The terms concrete and abstract data type are sometimes also used in the literature.

An example may help here. Consider a date. This is typically made up of three components, a day, a month and a year. In C we can create a user defined type called a date using structs. We can then create variables of this type. This is done in Pascal in a similar way using records.

Access to the components of a date (day, month and year) can then either be direct - an example of a concrete data types, or indirect (via procedures) - an abstract data types.

Simplistically direct access (or concrete data types) offer the benefit of efficiency, and the possibility of lack of data integrity. In our date example we may set a day to the value 31 when the month is February.

Indirect access (or abstract data types) are slightly less efficient as we now have the overhead of a procedure call to access the data, but better opportunity for data integrity as we can provide hidden code within the procedures to ensure that the day, month and year combinations are valid.

Fortran did not provide this facility until the Fortran 90 standard.

3.10.4 Standardisation

The purposes of a standard are quite varied and include:

- Investment in people: by this we mean that the time spent in learning a standard language pays off in the long term, as what one learns is applicable on any hardware/software platform that has a standard conformant compiler.
- Portability: one can take the code one has written for one hardware/software platform and move it to any hardware/software platform that has a standard conformant compiler.
- Known reference point: when making comparisons one starts with reference to the standard first, and then between the additional functionality of the various implementations

These are some but not all of the reasons for the use of standards. Their importance is summed up beautifully by Ronald G. Ross in his introduction to the Cannan and Otten book on the SQL standard: "Anybody who has ever plugged in an electric cord into a wall outlet can readily appreciate the inestimable benefits of workable standards. Indeed, with respect to electrical power, the very fact that we seldom even think about such access (until something goes wrong) is a sure sign of just how fundamentally important a successful standard can be."

3.11 Ada

Ada represents the culmination of many years of work in program language development. It was a collective effort and the main aim was to produce a language suitable for programming large-scale and real-time systems. Work started in 1974 with the formulation of a series of documents by the American Department of Defence (DoD), which led to the Steelman documents. It is a modern algorithmic language with the usual control structures and facilities for the use of modules, and allows separate compilation with type checking across modules.

Ada is a powerful and well-engineered language. Its widespread use is certain as it has the backing of the DoD. However, it is a large and complex language and consequently requires some effort to learn.

The latest version of the language is Ada 2012. The following url

```
http://www.ada-europe.org/resources/online
```

provides a good starting point. Visit this site if you want up to date details about Ada.

Another good source is

```
http://www.adaic.org/ada-resources/standards/ada12
```

Both sites have free electronic versions of the various Ada standards.

3.12 Modula

Modula was designed by Wirth during the 1970s at ETH, for the programming of embedded real-time systems. It has many of the features of Pascal, and can be taken for Pascal at a glance. The key new features that Modula introduced were those of processes and monitors.

As with Pascal it is relatively easy to learn and this makes it much more attractive than Ada for most people, achieving much of the capability without the complexity.

3.13 Modula 2

Wirth carried on developing his ideas about programming languages and the culmination of this can be seen in Modula 2. In his words: "In 1977, a research project with the goal to design a computer system (hardware and software) in an integrated approach, was launched at the Institut fur Informatik of ETH Zurich. This system (later to be called Lilith) was to be programmed in a single high level language, which therefore had to satisfy requirements of high level system design as well as those of low level programming of parts that closely interact with the given hardware. Modula 2 emerged from careful design deliberations as a language that includes all aspects of Pascal and extends them with the important module concept and those of multi-programming. Since its syntax was more in line with Modula than Pascal's the chosen name was Modula 2."

The language's main additions with regard to Pascal are:

- The module concept, and in particular the facility to split a module into a definition part and an implementation part.

- A more systematic syntax which facilitates the learning process. In particular, every structure starting with a keyword also ends with a keyword, i.e., is properly bracketed.
- The concept of process as the key to multiprogramming facilities.
- So-called low-level facilities, which make it possible to breach the rigid type consistency rules and allow one to map data with Modula 2 structure onto a store without inherent structure.
- The procedure type, which allows procedures to be dynamically assigned to variables.

A sad feature of Modula 2 was the long time taken to arrive at a standard for the language.

3.14 Other Language Developments

The following is a small selection of language developments that the authors find interesting — they may well not be included in other people's coverage.

3.14.1 Logo

Logo is a language that was developed by Papert and colleagues at the Artificial Intelligence Laboratory at MIT. Papert is a professor of both mathematics and education, and has been much influenced by the psychologist Piaget. The language is used to create learning environments in which children can communicate with a computer. The language is primarily used to demonstrate and help children develop fundamental concepts of mathematics. Probably the turtle and turtle geometry are known by educationalists outside of the context of Logo. Turtles have been incorporated into the Smalltalk computer system developed at Xerox Palo Alto Research Centre — Xerox PARC.

3.14.2 Postscript, TEX and LATEX

The 1980s saw a rapid spread in the use of computers for the production of printed material. The 3 languages are each used quite extensively in this area.

Postscript is a low-level interpretive programming language with good graphics capabilities. Its primary purpose is to enable the easy production of pages containing text, graphical shapes and images. It is rarely seen by most end users of modern desktop publishing systems, but underlies many of these systems. It is supported by an increasing number of laser printers and typesetters.

TEX is a language designed for the production of mathematical texts, and was developed by Donald Knuth. It linearises the production of mathematics using a standard computer keyboard. It is widely used in the scientific community for the production of documents involving mathematical equations.

LATEX is Leslie Lamport's version of TEX, and is regarded by many as more friendly. It is basically a set of macros that hide raw TEX from the end user. The TEX ratio is probably 1–9 (or so I'm reliably informed by a TEXie).

3.14.3 Prolog

Prolog was originally developed at Marseille by a group led by Colmerauer in 1972/73. It has since been extended and developed by several people, including Pereira (L.M.), Pereira (F), Warren and Kowalski. Prolog is unusual in that it is a vehicle for logic programming. Most of the languages described here are basically algorithmic languages and require a specification of how you want something done. Logic programming concentrates on the what rather than the how. The language appears strange at first, but has been taught by Kowalski and others to 10-year-old children at schools in London.

3.14.4 SQL

SQL stands for Structured Query Language, and was originally developed by people mainly working for IBM in the San Jose Research Laboratory. It is a relational database language, and enables programmers to define, manipulate and control data in a relational database. Simplistically, a relational database is seen by a user as a collection of tables, comprising rows and columns. It has become the most important language in the whole database field.

3.14.5 ICON

ICON is in the same family as Snobol, and is a high-level general purpose programming language that has most of the features necessary for efficient processing of nonnumeric data. Griswold (one of the original design team for Snobol) has learnt much since the design and implementation of Snobol, and the language is a joy to use in most areas of text manipulation.

It is available for most systems via anonymous FTP from a number of sites on the Internet.

3.15 Object Oriented Programming

Object oriented represents a major advance in program language development. The concepts that this introduces include:

- Classes.
- Objects.
- Messages.
- Methods.

These in turn draw on the ideas found in more conventional programming languages and correspond to

- Extensible data types.
- Instances of a class.
- Dynamically bound procedure calls.
- Procedures of a class.

Inheritance is a very powerful high-level concept introduced with object oriented programming. It enables an existing data type with its range of valid operations to form the basis for a new class, with more data types added with corresponding operations, and the new type is compatible with the original.

Fortran 2003 offered support for object oriented programming. This is achieved via the module facility rather than the class facility found in other languages like C++, Java and C#.

3.15.1 Simula

As was mentioned earlier, the first language to offer functionality in this area was Simula, and thus the ideas originated in the 1960s. The book Simula Begin by Birtwistle, Dahl, Myhrhaug and Nygaard is well worth a read as it represents one of the first books to introduce the concepts of object oriented programming.

3.15.2 Smalltalk

Language plus use of a computer system.

Smalltalk has been under development by the Xerox PARC Learning Research Group since the 1970s. In their words: “Smalltalk is a graphical, interactive programming environment. As suggested by the personal computer vision, Smalltalk is designed so that every component in the system is accessible to the user and can be presented in a meaningful way for observation and manipulation. The user interface issues in Smalltalk revolve around the attempt to create a visual language for

each object. The preferred hardware system for Smalltalk includes a high resolution graphical display screen and a pointing device such as a graphics pen or mouse. With these devices the user can select information viewed on the screen and invoke messages in order to interact with the information." Thus Smalltalk represents a very different strand in program language development. The ease of use of a system like this has long been appreciated and was first demonstrated commercially in the Macintosh microcomputers.

Wirth has spent some time at Xerox PARC and has been influenced by their work. In his own words "the most elating sensation was that after sixteen years of working for computers the computer now seemed to work for me." This influence can be seen in the design of the Lilith machine, the original Modula 2 engine, and in the development of Oberon as both a language and an operating system.

3.15.3 Oberon and Oberon 2

As Wirth says: "The programming language Oberon is the result of a concentrated effort to increase the power of Modula-2 and simultaneously to reduce its complexity. Several features were eliminated, and a few were added in order to increase the expressive power and flexibility of the language."

Oberon and Oberon 2 are thus developments beyond Modula 2. The main new concept added to Oberon was that of type extension. This enables the construction of new data types based on existing types and allows one to take advantage of what has already been done for that existing type.

Language constructs removed included:

- Variant records.
- Opaque types.
- Enumeration types.
- Subrange types.
- Local modules.
- With statement.
- Type transfer functions.
- Concurrency.

The short paper by Wirth provides a fuller coverage. It is available at ETH via anonymous FTP.

3.15.4 Eiffel

Eiffel was originally developed by Interactive Software Engineering Inc. (ISE) founded by Bertrand Meyer. Meyer's book Object-Oriented Software Construction

contains a detailed treatment of the concepts and theory of the object technology that led to Eiffel's design.

The language first became available in 1986, and the first edition of Meyer's book was published in 1988. The following is a quote from the Wikipedia entry.

- The design goal behind the Eiffel language, libraries, and programming methods is to enable programmers to create reliable, reusable software modules. Eiffel supports multiple inheritance, genericity, polymorphism, encapsulation, type-safe conversions, and parameter covariance. Eiffel's most important contribution to software engineering is design by contract (DbC), in which assertions, preconditions, postconditions, and class invariants are employed to help ensure program correctness without sacrificing efficiency.

3.15.5 C++

Stroustrup did his PhD thesis at the Computing Laboratory, Cambridge University, England, and worked with Simula. He had previously worked with Simula at the University of Aarhus in Denmark. His comments are illuminating: "but was pleasantly surprised by the way the mechanisms of the Simula language became increasingly helpful as the size of the program increased. The class and co-routine mechanisms of Simula and the comprehensive type checking mechanisms ensured that problems and errors did not (as I - and I guess most people - would have expected) grow linearly with the size of the program. Instead, the total program acted like a collection of very small (and therefore easy to write, comprehend and debug) programs rather than a single large program."

He designed C++ to provide Simula's functionality within the framework of C's efficiency, and he succeeded in this goal as C++ is one of the most widely used object oriented programming language.

The language began as enhancements to C, adding classes, virtual functions, operator overloading, multiple inheritance, templates and exception handling by the time of the first standard.

Its influence in the area of programming language design can be seen in Java and C#.

Table 3.1 summarises the C++ standardisation history.

The following are some of the guidelines used by the standards committee in the development of C++11.

Table 3.1 C++ standardisation history

Year	C++ standard	Informal name
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2007	ISO/IEC TR 19768:2007	C++TR1
2011	ISO/IEC 14882:2011	C++11

- Maintain stability and compatibility with C++98 and possibly with C;
- Prefer introduction of new features through the standard library, rather than extending the core language;
- Prefer changes that can evolve programming technique;
- Improve C++ to facilitate systems and library design, rather than to introduce new features useful only to specific applications;
- Increase type safety by providing safer alternatives to earlier unsafe techniques;
- Increase performance and the ability to work directly with hardware;
- Provide proper solutions for real-world problems;
- Implement zero-overhead principle (additional support required by some utilities must be used only if the utility is used);
- Make C++ easy to teach and to learn without removing any utility needed by expert programmers.

C++14 was a small extension over C++11 and was published in December 2014. C++17 was a major update and was published in December 2017.

3.15.6 Java

Bill Joy (of Sun fame) had by the late 1980s decided that C++ was too complicated and that an object oriented environment based upon C++ would be of use. At around about the same time James Gosling (mister emacs) was starting to get frustrated with the implementation of an SGML editor in C++. Oak was the outcome of Gosling's frustration.

Sun over the next few years ended up developing Oak for a variety of projects. It wasn't until Sun developed their own web browser, Hotjava, that Java as a language hit the streets. And as the saying goes the rest is history.

Java is a relatively simple object oriented language. Whilst it has its origins in C++ it has dispensed with most of the dangerous features. It is OO throughout. Everything is a class.

It is interpreted and the intermediate byte code will run on any machine that has a Java virtual machine for it. This is portability at the object code level, unlike portability at the source code level — which is what we expect with most conventional languages. Some of the safe features of the language include:

- Built in garbage collection.
- Array subscript checking.
- No pointers — everything is passed by reference.

It is multithreaded, which makes it a delight for many applications. It has an extensive windows toolkit, the so called AWT that was in the original release of the language and Swing that came in later.

It is under continual development and at the time of writing was in its eighth major release.

Sun was acquired by Oracle in 2010.

3.15.7 C#

C# is a recent language from Microsoft and is a key part of their .NET framework. It is a modern, well-engineered language in the same family of programming languages in terms of syntax as C, C++ and Java. If you have a knowledge of one of these languages it will look very familiar.

One of the design goals was to produce a component oriented language, and to build on the work that Microsoft had done with OLE, ActiveX and COM:

- ActiveX is a set of technologies that enables software components to interact with one another in a networked environment, regardless of the language in which they were created. ActiveX was built on the Component Object Model (COM).
- COM is the object model on which ActiveX Controls and OLE are built. COM allows an object to expose its functionality to other components and to host applications. It defines both how the object exposes itself and how this exposure works across processes and networks. COM also defines the object's life cycle.
- OLE is a mechanism that allows users to create and edit documents containing items or objects created by multiple applications. OLE was originally an acronym for Object Linking and Embedding. However, it is now referred to simply as OLE. Parts of OLE not related to linking and embedding are now part of Active technology.

Other design goals included creating a language:

- Where everything is an object — C# also has a mechanism for going between objects and fundamental types (integers, reals, etc.).
- Which would enable the construction of robust and reliable software — it has garbage collection, exception handling and type safety.
- Which would use a C/C++/Java syntax which is already widely known and thus help programmers converting from one of these languages to C#.

It has been updated three times since its original release. Some of the more important features added in C# 2 were Generics, Iterators, Partial Classes, Nullable Types and Static Classes. The major feature that C# 3 added for most people was LINQ, a mechanism for data querying. C# 4 was released in 2010 and added a number of additional features.

3.15.8 Python

Python is an object-oriented, interpreted, and interactive programming language. Python was conceived in the late 1980s, and its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the Amoeba operating system. Van Rossum is Python's principal

author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, (benevolent dictator for life - BDFL).

Heres a very brief summary of what started it all, written by Guido van Rossum:

> I had extensive experience with implementing an interpreted language in the ABC group at CWI, and from working with this group I had learned a lot about language design. This is the origin of many Python features, including the use of indentation for statement grouping and the inclusion of very-high-level data types (although the details are all different in Python). I had a number of gripes about the ABC language, but also liked many of its features. It was impossible to extend the ABC language (or its implementation) to remedy my complaints in fact its lack of extensibility was one of its biggest problems. I had some experience with using Modula−2+ and talked with the designers of Modula-3 and read the Modula−3 report. Modula−3 is the origin of the syntax and semantics used for exceptions, and some other Python features. I was working in the Amoeba distributed operating system group at CWI. We needed a better way to do system administration than by writing either C programs or Bourne shell scripts, since Amoeba had its own system call interface which wasnt easily accessible from the Bourne shell. My experience with error handling in Amoeba made me acutely aware of the importance of exceptions as a programming language feature. It occurred to me that a scripting language with a syntax like ABC but with access to the Amoeba system calls would fill the need. I realized that it would be foolish to write an Amoeba-specific language, so I decided that I needed a language that was generally extensible. During the 1989 Christmas holidays, I had a lot of time on my hand, so I decided to give it a try. During the next year, while still mostly working on it in my own time, Python was used in the Amoeba project with increasing success, and the feedback from colleagues made me add many early improvements. In February 1991, after just over a year of development, I decided to post to USENET. The rest is in the Misc/HISTORY file.

Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed.

Python 3.0 (also called Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008 after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6 and 2.7.

Here is the main Python web site.

```
https://www.python.org/
```

It is quite widely used. Large organizations that make use of Python include Google, Yahoo!, CERN, and NASA.

Our involvement with Python started when we were asked about Python training by people working at the Atomic Weapons Establishment in Aldermaston. We put together a short 3 day intensive course for them.

Quite a fun language!

3.16 Back to Fortran!

We finish off with a coverage of the developments since the Fortran 77 standard. Practically all of the Fortran compilers available today fully support the Fortran 90 and 95 standards. Support for features from the Fortran 2003 and 2008 standards is improving on a regular basis. See the following document

```
https://www.fortranplus.co.uk/
fortran-information/
```

for up to date information on what each compiler offers in terms of standard support.

3.16.1 Fortran 90

Almost as soon as the Fortran 77 standard was complete and published, work began on the next version. The language drew on many of the ideas covered in this chapter and these help to make Fortran 90 a very promising language. Some of the new features included:

- New source form, with blanks being significant and names being up to 31 characters.
- Implicit none.
- Better control structures.
- Control of the precision of numerical computation.
- Array processing.
- Pointers.
- User defined data types and operators.
- Procedures.
- Modules.
- Recursion.
- Dynamic storage allocation.

This was the major update that the Fortran community had been waiting a long time for. Backwards compatibility was again a key aim. This standard did not invalidate any standard conformant Fortran 77 program.

3.16.2 Fortran 95

Fortran was next standardised in 1996 — yet again out by one! Firstly we have a clear up of some of the areas in the standard that had emerged as requiring clarification. Secondly Fortran 95 added the following major concepts:

- The `forall` construct.
- Pure and elemental procedures.
- Implicit initialisation of derived-type objects.
- Initial association status for pointers.

The first two help considerably in parallelization of code.
Minor features include amongst others:

- Automatic deallocation of allocatable arrays.
- Intrinsic `sign` function distinguishes between –0 and +0.
- Intrinsic function `null` returns disconnected pointer.
- Intrinsic function `cpu_time` returns the processor time.
- References to some pure functions are allowed in specification statements.
- Nested `where` constructs.
- Masked `elsewhere` construct.
- Small changes to the `ceiling`, `floor`, `maxloc` and `minloc` intrinsic functions

Some of these were added to keep Fortran in line with High Performance Fortran (HPF). More details are given later.

Part 2 of the standard (ISO/IEC 1539-2:1994) adds the functional specification for varying length character data type, and this extends the usefulness of Fortran for character applications very considerably.

3.16.3 ISO Technical Reports TR15580 and TR15581

There are two additional reports that have been published on Fortran. TR 15580 specifies three modules that provide access to IEEE floating point arithmetic and TR15581 allows the use of the allocatable attribute on dummy arguments, function results and structure components.

3.16.4 Fortran 2003

The language is known as Fortran 2003 even though the language did not make it through the standardisation process until 2004. It was a major revision.

- Derived type enhancements
  - parameterised derived types (allows the kind, length, or shape of a derived type's components to be chosen when the derived type is used)
  - mixed component accessibility (allows different components to have different accessibility)
  - public entities of private type
  - improved structure constructors
  - finalisers

- Object oriented programming support
  - enhanced data abstraction (allows one type to extend the definition of another type)
  - polymorphism (allows the type of a variable to vary at run time)
  - dynamic type allocation
  - select type construct (allows a choice of execution flow depending upon the type a polymorphic object currently has)
  - type-bound procedures
- The associate construct (allows a complex expression or object to be denoted by a simple symbol)
- Data manipulation enhancements
  - allocatable components
  - deferred-type parameters
  - volatile attribute
  - explicit type specification in array constructors
  - intent specification of pointer arguments
  - specified lower bounds of pointer assignment, and pointer rank remapping
  - extended initialisation expressions
  - `max` and `min` intrinsics for character type
  - enhanced complex constants
- Input/output enhancements
  - asynchronous transfer operations (allow a program to continue to process data while an input/output transfer occurs)
  - stream access (allows access to a file without reference to any record structure)
  - user specified transfer operations for derived types
  - user specified control of rounding during format conversions
  - the flush statement
  - named constants for preconnected units
  - regularisation of input/output keywords
  - access to input/output error messages
- Procedure pointers
- Scoping enhancements
  - the ability to rename defined operators (supports greater data abstraction)
  - control of host association into interface bodies
- Support for IEC 60559 (IEEE 754) exceptions and arithmetic (to the extent a processor's arithmetic supports the IEC standard)
- Interoperability with the C programming language (allows portable access to many libraries and the low-level facilities provided by C and allows the portable use of Fortran libraries by programs written in C)
- Support for international usage

  - ISO 10646
  - choice of decimal or comma in numeric formatted input/output
- Enhanced integration with the host operating system
  - access to command line arguments and environment variables
  - access to the processor's error messages (improves the ability to handle exceptional conditions)

The earlier web address has details of Fortran compiler conformance to this standard.

3.16.5 DTR 19767 Enhanced Module Facilities

The module system in Fortran has a number of shortcomings and this DTR addresses some of the issues.

One of the major issues was the so-called recompilation cascade. Changes to one part of a module forced recompilation of all code that used the module. Modula 2 addressed this issue by distinguishing between the definition or interface and implementation. This can now be achieved in Fortran via submodules.

3.16.6 Fortran 2008

The next standard, ISO/IEC 1539-1:2010, commonly known as Fortran 2008, was approved in September 2010. The new features include:

- Submodules
- Coarrays
- Performance enhancements
  - do concurrent
  - Contiguous attribute
  - Simply contiguous arrays
- Data declaration
  - Maximum rank
  - Long integers
  - Allocatable components of recursive type
  - Implied-shape array
  - Pointer initialization
  - Data statement restrictions lifted
  - Kind of a forall index
  - Type statement for intrinsic types
  - Declaring type-bound procedures

  - Extensions to value attribute

- Data usage
  - Omitting an allocatable component in a structure constructor
  - Multiple allocations with source=
  - Copying the properties of an object in an allocate statement
  - Polymorphic assignment
  - Accessing real and imaginary parts
  - Pointer functions
  - Elemental dummy argument restrictions lifted

- Input/Output
  - Finding a unit when opening a file
  - g0 edit descriptor
  - Unlimited format item
  - Recursive input/output

- Execution control
  - The block construct
  - Exit statement
  - Stop code

- Intrinsic procedures and modules
  - Bit processsing
  - Storage size
  - Optional argument radix added to selected real kind
  - Extensions to trigonometric and hyperbolic intrinsic functions
  - Bessel functions
  - Error and gamma functions
  - Euclidean vector norms
  - Parity
  - Execute command line
  - Optional argument back added to maxloc and minloc
  - Find location in an array
  - String comparison
  - Constants
  - Compiler information
  - Function for C sizeof
  - Additional optional argument for ieee_selected_real_kind

- Programs and procedures
  - Save attribute for module and submodule data
  - Empty contains part
  - Form of the end statement for an internal or module procedure
  - Internal procedure as an actual argument or pointer target

  - Null pointer or unallocated allocatable as an absent dummy argument
  - Non-pointer actual for pointer dummy argument
  - Generic resolution by pointer/allocatable or data/procedure
  - Elemental procedures that are not pure
  - Entry statement becomes obsolescent
- Source form
  - Semicolon at line start

A more thorough coverage can be found in John Reid's paper.

```
https://wg5-fortran.org/N1851-N1900/N1891.pdf
```

3.16.7 TS 29113 Further Interoperability of Fortran with C

This TS was published in 2012.

3.16.8 Fortran 2018

According to the current WG5 work schedule it is expected that the Fortran 2018 standard will be published in August 2018.

Here is a short list of some of the changes introduced by this standard. It has been taken from John Reid's paper on the new features of Fortran 2018. The first edition of this paper is N2127 and was published in 2017. The second edition is N2145 and was published in January 2018.

- Additional parallel features in Fortran
  - Teams
  - Image failure
  - Form team statement
  - Change team construct
  - Coarrays allocated in teams
  - Critical construct
  - Lock and unlock statements
  - Sync team statement
  - Image selectors
  - Intrinsic functions get team and team number
  - Intrinsic function image index
  - Intrinsic function num images
  - Intrinsic function this image
  - Intrinsic function move alloc

  - Fail image statement
  - Detecting failed and stopped images
  - Collective subroutines
  - New and enhanced atomic subroutines
  - Failed images and stat=specifiers
  - Events
- Conformance with ISO/IEC/IEEE 60559:2011
  - Subnormal values
  - Type for floating-point modes
  - Round away from zero
  - Decimal rounding mode
  - Rounded conversions
  - Fused multiply-add
  - Test sign
  - Conversion to integer type
  - Remainder function
  - Maximum and minimum values
  - Adjacent machine numbers
  - Comparisons
- Removal of deficiencies and discrepancies
  - Default accessibility for entities accessed from a module
  - Implicit none enhancement
  - Enhancements to inquire
  - d0.d, e0.d, es0.d, en0.d, g0.d and ew.d e0 edit descriptors
  - Formatted input error conditions
  - Rules for generic procedures
  - Enhancements to stop and error stop
  - Intrinsics that access the computing environment
  - New elemental intrinsic function out of range
  - New reduction intrinsic reduce
  - Intrinsic function coshape
  - Intrinsic subroutine random init
  - Intrinsic function sign
  - Intrinsic functions extends type of and same type as
  - Nonstandard procedure from a standard intrinsic module
  - Kind of the do variable in implied do
  - Locality clauses in do concurrent
  - Control of host association
  - Connect a file to more than one unit
  - Advancing input with size=
  - Extension to the generic statement
  - Removal of anomalies regarding pure procedures
  - Recursive and non-recursive procedures

- Simplification of calls of the intrinsic cmplx
- Removal of the restriction on argument dim of many intrinsic functions
- Kinds of arguments of intrinsic and IEEE procedures
- Hexadecimal input/output
- Deletions
  - Arithmetic if
  - Nonblock do construct
- New obsolescences
  - common and equivalence
  - Labelled do statements
  - Specific names for intrinsic functions
  - The forall construct and statement

Both N2127 and N2145 can be found on the WG5 site.

```
https://wg5-fortran.org/documents.html
```

Both versions can also be found at the ACM Fortran Forum site.

```
http://dl.acm.org/citation.cfm?id=J286
```

N2127 was published in the August 2017 edition, and N2145 can be found in the April 2018 edition.

Table 3.2 summarises the Fortran standardisation history.

Fortran 2018 is currently on schedule for a 2018 publication date.

Table 3.2 Fortran standardisation history

Year	Fortran standard	Informal name
1966	Ansi x3.9-1966	Fortran 66
1978	Ansi x3.9-1977	Fortran 77
1978	ISO 1539-1980	Fortran 77
1991	ISO/IEC 1539:1991	Fortran 90
1997	ISO/IEC 1539-1:1997	Fortran 95
1998	ISO/IEC TR 15580:1998	Floating-point exception handling
1998	ISO/IEC TR 15581:1998	Enhanced data type facilities
1999	ISO/IEC 1539-3:1999	Conditional compilation
2000	ISO/TEC 1539-2:2000	Part 2: varying length character strings
2001	ISO/TEC TR 15580:2001	Floating-point exception handling
2004	ISO/IEC 1539-1:2004	Fortran 2003
2009	ISO/IEC 1539-1	Module TSR
2010	1539-1:2010	Fortran 2008
2012	ISO/TEC TS 29113:2012 ISO/TEC NP TS 18508	Further interoperability of fortran with C Additional parallel features in fortran
201?	1539-1:2018	Fortran 2018

3.17 Fortran Discussion Lists

The first to look at is the Fortran 90 list. Details can be found at

```
http://www.jiscmail.ac.uk/lists/COMP-FORTRAN-90.html
```

If you subscribe you will have access to people involved in Fortran standardisation, language implementers for most of the hardware and software platforms, people using Fortran in many very specialised areas, people teaching Fortran, etc.

There is also a comp.lang.fortran list available via USENET news. This provides access to people worldwide with enormous combined expertise in all aspects of Fortran. Invariably someone will have encountered your problem or one very much like it and have one or more solutions.

Here is an extract from Wikipedia.

> Usenet is a worldwide distributed Internet discussion system. It was developed from the general purpose UUCP dial-up network architecture. Tom Truscott and Jim Ellis conceived the idea in 1979 and it was established in 1980. Users read and post messages (called articles or posts, and collectively termed news) to one or more categories, known as newsgroups. Usenet resembles a bulletin board system (BBS) in many respects, and is the precursor to Internet forums that are widely used today. Usenet can be superficially regarded as a hybrid between email and web forums. Discussions are threaded, as with web forums and BBSes, though posts are stored on the server sequentially.
>
> One notable difference between a BBS or web forum and Usenet is the absence of a central server and dedicated administrator. Usenet is distributed among a large, constantly changing conglomeration of servers that store and forward messages to one another in so-called news feeds. Individual users may read messages from and post messages to a local server operated by a commercial usenet provider, their Internet service provider, university, employer, or their own server.

Another to consider is the Fortran group on ‘linkedin’ The address is

```
https://www.linkedin.com/
```

3.18 ACM Fortran Forum

Ian Chivers is also Editor of Fortran Forum, the SIGPLAN Special Interest Publication on Fortran, ACM Press. Visit

```
http://portal.acm.org/citation.cfm?id=J286
```

for more information.

3.19 Other Sources

The following URLs are very useful:
Our Fortran web site.

```
https://www.fortranplus.co.uk
```

The Fortran Company, maintained by Walt Brainerd.

```
http://www.fortran.com/
```

3.20 Summary

It is hoped that you now have some idea about the wide variety of uses that programming languages are put to.

3.21 Bibliography

Fortran 2008 Standard, ISO/IEC 1539-1:2010, price CHF 338. Publication date: 2010-10-06.

```
http://www.iso.org/iso/home/store.htm
```

Fortran 2003 Standard, ISO/IEC DIS 1539-1:2004(E)
DTR 19767: Enhanced module Facilities: ISO/IEC TR 19767:2004(E)
The Fortran 77 and 66 standards are available from the WG5 site.

```
https://wg5-fortran.org/fearlier.html
```

The ISO home page is

```
http://www.iso.org/
```

The J3 home page is:

```
https://j3-fortran.org
```

The WG5 home page is:

https://wg5-fortran.org/

Both have copies of working documents.
Adobe Systems Incorporated, Postscript Language:
Tutorial and Cookbook, Addison-Wesley, 1985; Reference Manual, Addison-Wesley, 1985; Program Design, Addison-Wesley, 1985.

- The three books provide a comprehensive coverage of the facilities and capabilities of Postscript.

They third edition of the reference manual is available online.

http://www.adobe.com/products/postscript/pdfs/PLRM.pdf

ACM SIG PLAN, History of programming Languages Conference — HOPL-II, ACM Press, 1993.

- One of the best sources of information on C++, CLU, Concurrent Pascal, Formac, Forth, Icon, Lisp, Pascal, Prolog, Smalltalk and Simulation Languages by the people involved in the original design and or implementation. Very highly recommended. This is the second in the HOPL series, and the first was edited by Wexelblat. Details are given later.

Adams J.C., Brainerd W.S., Hendrickson R.A., Maine R.E., Martin J.T., Smith B.T., The Fortran 2003 Handbook, Springer, 2009.

- Their most recent version, and a complete coverage of the 2003 standard. As with the Metcalf, Reid and Cohen book some of the authors were on the J3 committee. Very thorough.

Annals of the History of Computing, Special Issue: Fortran's 25 Anniversary, ACM, Article 6,1, 1984.

- Very interesting comments, some anecdotal, about the early work on Fortran.

Barnes J., Programming in Ada 95, Addison-Wesley, 1996.

- One of the best Ada books. He was a member of the original design team.

Bergin T.J., Gibson R.G., History of Programming Languages, Addison-Wesley, 1996.

- This is a formal book publication of the Conference Proceedings of HOPL II. The earlier work is based on preprints of the papers.

Birtwistle G.M., Dahl O. J., Myhrhaug B., Nygaard K.,
SIMULA BEGIN, Chartwell-Bratt Ltd, 1979.

- A number of chapters in the book will be of interest to programmers unfamiliar with some of the ideas involved in a variety of areas including systems and models, simulation, and co-routines. Also has some sound practical advice on problem solving.

Brinch-Hansen P., The Programming Language Concurrent Pascal, IEEE Transactions on Software Engineering, June 1975, 199-207.

- Looks at the extensions to Pascal necessary to support concurrent processes.

Cannan S., Otten G., SQL — The Standard Handbook, McGraw-Hill, 1993.

- Very thorough coverage of the SQL standard, ISO 9075:1992(E).

Chivers I.D., Clark M.W., History and Future of Fortran, data Processing, vol. 27 no 1, January/February 1985.

- Short article on an early draft of the standard, around version 90.

Chivers Ian, Essential C# Fast, Springer, ISBN 1-85233-562-9.

- A quick introduction to the C# programming language.

Chivers I.D., A Practical Introduction to Standard Pascal, Ellis Horwood, 1986.

- A short introduction to Pascal.

Date C.J., A Guide to the SQL Standard, Addison-Wesley, 1997.

- Date has written extensively on the whole database field, and this book looks at the SQL language itself. As with many of Date's works quite easy to read.

Deitel H.M., Deitel P.J., Java: How to program, 10th Edition Pearson Education

- A good introduction to Java and programming for people with little or no background in programming.

Deitel H.M., Deitel P.J., Visual Basic How to Program, Pearson Education, 2014.

- Good practical introduction to VB .NET.

Dyson G., Turing's Cathedral, The origins of the Digital Universe, Pantheon Books, 2012.

- The following is taken from the books blurb. ... *Dyson focuses on a small group of men and women, led by John von Neuman at the Institute of Advanced Study in Princeton, New Jersey, who build one of the first computers to realise Alan Turing's vision of a Universal Machine.*

Eckstein R., Loy M., Wood D., Java Swing, O'Reilly, 1998.

- Comprehensive coverage of the visual interface features available in Java.

Geissman L.B., Separate Compilation in Modula 2 and the Structure of the Modula 2 Compiler on the Personal Computer Lilith, Dissertation 7286, ETH Zurich.

- Fascinating background reading concerning Modula 2 and the Lilith architecture.

Goldberg A., Robson D., Smalltalk 80: The Language and its Implementation, Addison-Wesley, 1983.

- Written by some of the Xerox PARC people who have been involved with the development of Smalltalk. Provides a good introduction (if that is possible with the written word) of the capabilities of Smalltalk.

Goos G., Hartmanis J. (Eds), The programming Language Ada — Reference Manual, Springer Verlag, 1981.

- The definition of the language.

Goossens M., Mittelbach F., Rahtz S., Roegel D., Voß H. The LaTeX Graphics Companion, second edition, Addison Wesley, 2007.

- Another essential LaTeX book.

Griswold R.E., Poage J.F., Polonsky I.P., The Snobol4 programming Language, Prentice-Hall, 1971.

- The original book on the language. Also provides some short historical material on the language.

Griswold R.E., Griswold M.T., The Icon programming Language, Prentice-Hall, 1983.

- The definition of the language with a lot of good examples. Also contains information on how to obtain public domain versions of the language for a variety of machines and operating systems.

Harbison S.P., Steele G.L., A C Reference Manual, Prentice-Hall, 2002.

- Very good coverage of the various flavours of C, including K&R C, Standard C 1989, Standard C 1995, Standard C 1999 and Standard C++

Hellman D., The Python Standard Library by Example, Addison-Wesley, 2011.

- Good introduction to the Python standard library.

Hoare C.A.R., Hints on programming Language Design, SIGACT/SIGPLAN Symposium on Principles of programming Languages, October 1973.

- The first sentence of the introduction sums it up beautifully: “I would like in this paper to present a philosophy of the design and evaluation of programming languages which I have adopted and developed over a number of years, namely that the primary purpose of a programming language is to help the programmer in the practice of his art.”

Jacobi C., Code Generation and the Lilith Architecture, Dissertation 7195, ETH Zurich

- Fascinating background reading concerning Modula 2 and the Lilith architecture.

Jenson K., Wirth N., Pascal: User Manual and Report, Springer-Verlag, 1975.

- The original definition of the Pascal language. Understandably dated when one looks at more recent expositions on programming in Pascal.

Kemeny J.G., Kurtz T.E., Basic programming, Wiley, 1971.

- The original book on Basic by its designers.

Kernighan B.W., Ritchie D.M., The C programming Language, Prentice-Hall; first edition 1978; second edition 1988.

- The original work on the C language, and thus essential for serious work with C.

Kowalski R., Logic programming in the Fifth Generation, The Knowledge Engineering Review, The BCS Specialist Group on Expert Systems.

- A short paper providing a good background to Prolog and logic programming, with an extensive bibliography.

Knuth D. E., The TEXbook, Addison-Wesley, 1986.

- Knuth writes with an tremendous enthusiasm and perhaps this is understandable as he did design TEX. Has to be read from cover to cover for a full understanding of the capability of TEX.

Lamport L., LATEX User's Guide and Reference Manual, 2005, Addison Wesley, ISBN 0201529831.

- The original LATEX book. Essential reading.

Lyons J., Chomsky, Fontana/Collins, 1982.

- A good introduction to the work of Chomsky, with the added benefit that Chomsky himself read and commented on it for Lyons. Very readable.

Malpas J., Prolog: A Relational Language and its Applications, Prentice-Hall, 1987.

- A good introduction to Prolog for people with some programming background. Good bibliography. Looks at a variety of versions of Prolog.

Marcus C., Prolog programming: Applications for Database Systems, Expert Systems and Natural Language Systems, Addison-Wesley.

- Coverage of the use of Prolog in the above areas. As with the previous book aimed mainly at programmers, and hence not suitable as an introduction to Prolog as only two chapters are devoted to introducing Prolog.

Metcalf M. and Reid J., Cohen M., Modern Fortran Explained, Oxford University Press, 2011

- A clear compact coverage of the main features of Fortran. John Reid is Convener of the WG5 committee and Malcolm Cohen was the editor of Fortran 2008.

Mittelbach F., Goossens M., Braams J., Carlisle D., and Rowley C., The LATEX Companion, 2005, Addison Wesley, ISBN 0201362996.

- The LATEX book. It is required if you are setting a book using LATEX.

Mossenbeck H., Object-Orientated programming in Oberon-2, Springer-Verlag, 1995.

- One of the best introductions to object oriented programming. Uses Oberon-2 as the implementation language. Highly recommended.

Papert S., Mindstorms - Children, Computers and Powerful Ideas, Harvester Press, 1980.

- Very personal vision of the uses of computers by children. It challenges many conventional ideas in this area.

Sammet J., programming Languages: History and Fundamentals, Prentice-Hall, 1969.

- Possibly the most comprehensive introduction to the history of program language development — ends unfortunately before the 1980s.

Sethi R., programming Languages: Concepts and Constructs, Addison-Wesley, 1989.

- The annotated bibliographic notes at the end of each chapter and the extensive bibliography make it a useful book.

Reiser M., Wirth N., programming in Oberon — Steps Beyond Pascal and Modula, Addison-Wesley, 1992.

- Good introduction to Oberon. Revealing history of the developments behind Oberon.

Reiser M., The Oberon System: User Guide and programmer's Manual, Addison-Wesley, 1991.

- How to use the Oberon system, rather than the language.

Stroustrup B., The C++ Programming Language, Addison-Wesley; third edition 1997; fourth edition 2014. 1997.

- The C++ book. Written by the designer of the language. The third edition is a massive improvement over the earlier editions. The fourth edition covers C++11. One of the best books on C++ and C++11 in particular.

Young S. J., An Introduction to Ada, 2nd Edition, Ellis Horwood, 1984.

- A readable introduction to Ada. Greater clarity than the first edition.

Wexelblat, History of programming Languages, HOPL I, ACM Monograph Series, Academic Press, 1978.

- Very thorough coverage of the development of programming languages up to June 1978. Sessions on Fortran, Algol, Lisp, Cobol, APT, Jovial, GPSS, Simula, JOSS, Basic, PL/I, Snobol and APL, with speakers involved in the original languages. Very highly recommended.

Wiener R., Software development using Eiffel, Prentice Hall, 1995.

- The book's subtitle is *There can be life other than C++* The book gives a good introduction to object oriented analysis and design using the Booch 94 method using Eiffel.

Wirth N., An Assessment of the Programming Language Pascal, IEEE Transactions on Software Engineering, June 1975, 192-198.

- Short paper by Wirth on his experience with Pascal.

Wirth N., History and Goals of Modula 2, Byte, August 1984, 145-152.

- Straight from the horse's mouth!

Wirth N., On the Design of programming Languages, Proc. IFIP Congress 74, 386-393, North-Holland.

- Short paper given in 1974 on designing programming languages.

Wirth N., The programming Language Pascal, Acta Informatica 1, 35-63, 1971.

- Short paper on the development of Pascal from Algol 60.

Wirth N., Modula: a language for modular multiprogramming, Software Practice and Experience, 7, 3–35, 1977.

- Short paper on Modula, the precursor of Modula 2.

Wirth N., Programming in Modula 2, Springer-Verlag, 1983.

- The original definition of the language. Essential reading for anyone considering programming in Modula 2 on a long term basis.

Wirth N. Type Extensions, ACM Trans. on Prog. Languages and Systems, 10, 2 (April 1988), 2004-214

- Short paper on type extension.

Wirth N. From Modula 2 to Oberon, Software — Practice and Experience, 18,7 (July 1988), 661–670

- Brief paper on the move from Modula 2 to Oberon, looking at features that were removed and added.

Wirth N., Gutknecht J., Project Oberon: The Design of an Operating System and Compiler, Addison-Wesley, 1992.

- Fascinating background to the development of Oberon. Highly recommended for anyone involved in large scale program development, not only in the areas of programming languages and operating systems, but more generally.