1.1 What is a Programming Language

A language is a systematic set of rules for communicating ideas. With a natural language, such as English, this communication is between people, and the language is used in both spoken and written forms.

Programming languages differ from natural languages in several important ways. First, whereas a natural language allows one person to communicate with other people, the primary communication with a programming language is between a person and a computer. Programming languages also prove useful for communication between people, however, as a means of communicating algorithms and processes to other people.

The second major difference between natural and programming languages is in the content of the communication. In the case of programming languages, this communication is known as a program. Programs are expressions of solutions to problems that are specific enough to give the receiver of the program sufficient information to carry out the solution. While natural languages can be used to communicate solutions to problems, they are also useful for many other types of communication, such as expression of feelings, statements of facts, etc.

A third unique feature of communication via a programming language is the medium used. Because a computer is the intended receiver, this has traditionally meant that programs are represented symbolically as strings of characters as opposed, for example, to audible sounds. This restriction of programming languages is a result of the limitations on the media that computers can accept as input. Although modern programming environments have somewhat released programmers from this restriction, all languages that we will discuss in this book have been designed to be communicated by a string of characters.

A fourth difference between natural languages and programming languages is the formal, static nature of programming languages compared to the informal, evolving nature of natural languages. Whereas natural languages lack a formal complete definition of their grammar and vocabulary, a programming language is formally and unambiguously defined by the program that is used to translate it. This does not mean that all translators define the language in the same way. Most languages have standard formal definitions that each translator strives to implement, however.

Our working definition for a programming language is:

*A programming language is a language that is intended to be used by a person to express a process by which a computer can solve a problem.*

The four key components in this definition of a programming language are
1. processor - the computing device that will carry out the process described by the program
2. person - the programmer who serves as the source of the process and who wishes to communicate that process to the processor
3. process - the activity being described by the program
4. problem - the problem that the process is intended to solve or the actual system it is intended to model.

Four major models for programming languages are described in this text, which represent the point of view of each of the preceding four components.

The imperative model is based on expressing the solution from the processor’s perspective. This outlook is reflected in the sequential execution of commands and the use of a changeable data store, concepts that are based on the way computers execute programs at the machine language level. In the past, imperative was the predominant paradigm for languages, because such languages were easiest to translate into a form suitable for machine execution. Programs written in a language of this model consist of a sequence of modifications to the processor's storage.

The logic-oriented model is most closely related to the perspective of the person who is the problem-solver. This model looks at the problem from the logical point of view. The program is a logical description of the problem expressed in a formal way, similar to the manner that a human brain would reason about the problem.

The functional model focuses on the process of solving the problem. The functional view results in programs that describe, in an abstract way, the operations that must be performed to solve the problem. In this view, a program is considered as a function that maps its input into the corresponding output.

The object-oriented model reflects most closely the actual problem in that it represents the problem in a way similar to its real-world representation. A program in a language of this model consists of objects that send messages to each other. These objects in the program correspond directly to actual objects in the problem space, such as people, machines, departments, documents, and so on.

In this book, we will look at all four of these models, or paradigms, and the ways they are represented in programming languages. We will also find that most programming languages provide some combination of these viewpoints to allow for efficiency in the construction and execution of programs. Also, we will briefly look at other models, some of which intersect with the above four points of view. Included among these are Non-Procedural, Visual Programming, Scripting, and Stack-Oriented models.

1.2 Why Study Programming Languages?

You will receive five major benefits from the study of the structure and models of programming languages.
1. You will improve your problem-solving ability.
   Experts tell us that facility and understanding of natural language affects our ability to think and form ideas. Similarly, a thorough understanding of programming languages can increase our ability to think about approaches to problems. This is especially true when we have the ability to think about a problem from a number of different perspectives using the various models of languages described in Section 1.1. Even when the language you are using is not based on a given model, your familiarity with that model will enable you to think about the problem in different ways that may permit a better expression of the solution in the language that is available to you.
2. You will be able to make better use of a programming language.

The study of programming language structures will give you a better understanding of the function and implementation of these structures. Then, when you are programming, you will be better able to use the language to its greatest potential and to do so in an efficient way. Understanding the power of a language will enable you to better utilize that power.

In addition, your understanding of the structure of programming languages will give you new insights into their implementation and the better you understand the implementation, the more you will be able to utilize that understanding to increase the efficiency of your programs. Also, improved understanding of implementations can assist you in debugging your code.

3. You will be able to choose an appropriate language more intelligently.

In the 1960s and 1970s, programmers had very few choices on the language they would use. All programmers working for a given organization were expected to program in the language that was the "standard" for that site. In many cases, only one computer was available, and only one language was implemented on that computer. Frequently, the local programmers knew how to program in only that one language as well, because that was all that was necessary.

This situation changed dramatically in the succeeding decades. In the first place, the improved technology of computers and language translators has made many languages available on present-day machines, even on the smallest handheld computers. In addition, programmers who have completed courses such as the one for which this book is intended have experience with and an understanding of a variety of languages.

Therefore, it is common practice today for the programmer to choose a language for a given project from among several possibilities. Whereas in the past this choice was determined by local requirements, limited availability, or knowledge of the programmer, today a programmer with an understanding of programming languages can choose the language that makes the problem solution easiest and most cost effective.

4. You will find it easier to learn new programming languages.

As we study the development of programming languages, we find that new languages and enhancements to present languages are continually being introduced. These changes have been driven by new technologies, by new approaches to problem-solving, and by a better understanding of programming language principles. We also see that throughout this development, there are key concepts that remain constant as well as new facilities that are added.

An important benefit of the study of programming languages is that it enables you to learn new languages and new capabilities of existing languages as they are developed. Through a thorough understanding of programming language models, you can quickly assess a new language in comparison with those models and determine the ways in which the new language adheres to a model and ways in which it differs. This greatly enhances the speed and agility with which you learn new languages.

5. You will become a better language designer.

This benefit is more important than it first appears. Few people ever have the desire or the opportunity to design their own programming language. Although you may be one of these people, most likely you are not. However, if we hold to our view that language is a means of communication between a person and a computer, then every computer system that is developed must have a language embedded within it to provide for human/machine interaction. A good
understanding of programming language principles can greatly assist the system developer in designing the language the user will use to interact with the machine.

In addition, many modern languages are extendable in a variety of ways. This means that the programmer can enhance the language through the addition of new data types and operators. With these languages, every program is actually a new language design in the sense that the programmer has the power to enhance the original language by adding features to it.

Discussion: What other reasons might you have for studying programming languages?

Discussion: Have you had experience at designing a programming language in the sense described above? Describe your experience.

Discussion: How do you personally go about learning a new programming language? How does this differ from the way you go about learning a new natural language like French?

1.3 A Brief History of Programming Languages

It helps in understanding programming languages to have some appreciation of their history. Much has been written on this topic including Sammet (1969), Wexelblat (1981), and Bergin (1996).

In the historical overview which follows, we have limited our consideration to those languages that are either used extensively today or that originated important concepts for today's languages. Thousands of languages have been implemented and most of these have made important contributions to the field. We have necessarily limited our consideration to the few that we feel have had the greatest impact.

We have structured this history into four periods. The first period began in 1955 and lasted until the latter part of the 1960s. During this period the first higher-level languages were being developed with a wide variety of philosophies and concepts. The second period, roughly covering the 1970s, was a time of consolidation around the model of one language, ALGOL 60, with the development of a number of new languages derived from ALGOL 60 but extending it by adding important new features. The third period corresponds to the decade of the 1980s, when the results of earlier research on languages were pulled together to introduce new models and approaches for programming languages. Finally, the decade of the 1990s, our fourth period, has seen its important language advances related to important technological advances in computer systems, through the adaptation of earlier language models to facilitate the use of these new technologies.

1.3.1 The Early Languages

1.3.1.1 FORTRAN

The distinction of being the first widely used higher-level programming language is held by FORTRAN. Prior to its implementation, many people were skeptical about the possibility of a language being compiled successfully and efficiently. FORTRAN quickly erased such skepticism and became a very popular language; it is still in use more than 40 years later.
Most early languages were named by acronyms. We indicate this in a language's name by writing it in all uppercase letters. We also display the full name in parentheses after the first mention of the language's name. FORTRAN (FORmula TRANslation) was designed in 1954 and implemented several years later by John Backus at IBM. It was specifically designed for a single machine, the IBM 704, and still bears the marks of some of the idiosyncrasies of that machine. For example, the control structures of FORTRAN are based on the branch statements contained in the IBM 704 machine language. The arithmetic if statement and the computed goto statement that determine the next statement to be executed based on the result of a calculation, are examples of this correspondence. FORTRAN also supports implicit typing of variables, where the type of an undeclared variable is determined by the first letter of its name, making declaration of type unnecessary when names are chosen appropriately.

FORTRAN has evolved through the years into a language that incorporates many modern language facilities while still maintaining its original character. The latest version, FORTRAN 90, implements many of the features that were introduced by later languages.

FORTRAN was designed for solving scientific problems and therefore adopted an algebraic notation. Because it was a pioneer language, it made many contributions to language development. Included among these were (1) variables and assignment statements, (2) the concept of types, (3) modularity through the use of subprograms, (4) conditional and iterative control structures, and (5) formatted input/output.

Discussion: What are some of your favorite acronyms (both computer-related and non-computer-related).


Research: In what additional ways did FORTRAN reflect the structure of the IBM 704?

[1.3.1.2 COBOL]

Although much early programming was scientific in nature, in the 1950s many business applications were being programmed as well, and the requirements of such programming were not handled well by FORTRAN. There was an evident need for a common language suitable for these business applications. In 1959, through the initiative of the U.S. Department of Defense, a committee was formed to develop a language to meet these needs. This committee, called the CODASYL Committee, consisted of representatives from computer manufacturers and the Department of Defense. The resulting language, COBOL (COmmon Business Oriented Language), was first implemented in 1960 and soon became the standard for business data processing applications.

Beyond the difference in their intended applications, there were two major differences in the ways FORTRAN and COBOL were developed. FORTRAN was the effort of one organization, IBM, whereas COBOL resulted from the cooperative effort of many organizations. Furthermore, FORTRAN was designed to be run on a single machine, the IBM 704; in fact, the architecture of that machine affected many of the design decisions. COBOL, on the other hand, was designed independently of any specific computer with the intention that it be implemented on all computers.
COBOL programs are divided into four divisions. The identification division contains the name of the program and any comments that describe it. The data division describes the structure and name of all data in the program. The procedure division contains the executable part of the COBOL program. And the environment division contains machine-dependent information.

Like FORTRAN, COBOL has evolved over the years as new standards have been developed. It has also continued to be used extensively over a 40-year period. Its strengths are in the manipulation of files and in handling fixed-decimal data. It is not currently used heavily in the development of new systems, but much legacy code exists that is written in COBOL. Much of this code was revisited in the latter part of the 1990s as a part of the preparation for Y2K.

One of the primary objectives of COBOL was that its code be English-like. For this reason, COBOL programs tend to be very wordy, and many programmers find this cumbersome. However, the English-like property of COBOL was an early attempt at designing a language to facilitate the readability of programs and thus was an important contribution to the development of later languages. The other major contribution of COBOL was the introduction of a heterogeneous data structure, the record, which became an important component of later languages.

Discuss: How do you attempt to make your programs more easily readable by others? What language features assist you in doing that?


Research: What praise and critique have been leveled at the COBOL language? Why is this language still in use today?

1.3.1.3 ALGOL 60

The ALGOL (ALGorithmic Oriented Language) 60 language had a European origin and was designed by an international committee. The first version was called ALGOL 58 after the year of its introduction, but in 1960 the finished product, ALGOL 60, was published. Although it never enjoyed the commercial popularity of FORTRAN and COBOL, it is the most important language of this era in terms of its influence on later language development.

Like FORTRAN, ALGOL 60 was designed for use in scientific problem solving. Unlike FORTRAN, it was not designed to be implemented on one specific machine. Rather, the objective was for the language’s design to free from restrictions imposed by concern for implementation. This was both a major asset and a major liability of the ALGOL 60 language. Its machine independence permitted the designers to be more creative, but it made implementation much more difficult.

One of the greatest impacts ALGOL 60 had was a result of its description, as found in Naur (1963). This report became the accepted definition of the language and was a model of clarity and completeness. A major contribution from this report was the introduction of BNF notation for defining the syntax of the language. This notation is described in Chapter 2 and is used throughout this book.

ALGOL 60 was used on only a limited basis, mostly by Europeans and research computer scientists in the United States. Its use in commercial applications was hindered by the
absence of standard input/output facilities in its specification and the lack of interest in the language by large computer vendors. ALGOL 60 did, however, become the standard for the publication of algorithms and had a profound effect on future language development. In that sense, it was in the arena of person-to-person communication that ALGOL 60 had its greater impact, not in person-to-machine communication.

Some of the major contributions of ALGOL 60 to later languages were (1) block structure: the ability to create blocks of statements for the scope of variables and the extent of influence of control statement; (2) structured control statements: if-then-else and the use of a general condition for iteration control; and (3) recursion: the ability of a procedure to call itself.

**Discuss:** What are the limiting factors of a language that runs on only one platform?

**Discuss:** What are some advantages and disadvantages of a language designed by a diverse committee versus one designed by an individual or team directed by an individual?

**Research:** Find some programs or algorithms expressed in ALGOL 60. Examine the features of the language via that example. What do you find clear and what are you unable to understand?

### 1.3.1.4 LISP

Like FORTRAN and COBOL, LISP (LISt Processing) is a language that was developed for a specific application and is still extensively used today. LISP was developed by John McCarthy in the Artificial Intelligence Group at M.I.T. in the late 1950s as a language to support artificial intelligence research. It was first implemented in 1960 on the IBM 704. It has remained the primary programming language for artificial intelligence through the years. Common LISP was defined in 1981 as an informal standard and has since become officially recognized.

LISP pioneered the idea of nonnumeric, or symbolic, computing. It also introduced as its basic data structure the concept of the linked list. The LISP language is functional in nature. This means that rather than specifying operations as a sequential set of statements, LISP specifies the invocation of a function, using composition of functions as the main device for specifying multiple actions and recursion to specify repetition of the same action. This model of computation is defined and explored in Chapter 12, along with a description of the Scheme dialect of the LISP language.

LISP also uses the same basic linked list construct, the S-expression, to represent both data and program, thus allowing a program to be accessed as data and data to be executed as a program.

**Research:** What is artificial intelligence and what are some important applications of it?

### 1.3.1.5 APL

Still another language designed in the late 1950s was APL (A Programming Language), which was the creation of Kenneth Iverson. Iverson did his initial work on the language at Harvard and later continued development at IBM. APL was enthusiastically received by a
number of programmers. It consisted of many powerful operators and a simple, mathematical notation. The availability of this large number of operators required a large character set and made implementation of the language difficult. The mathematical nature of APL discouraged programmers who were not adept at mathematics. The definition of APL was specified in Iverson (1962).

The primary data structure of APL is the array, and the language features operators that apply to an entire array. Iterative processing is accomplished by placing the data to be iteratively operated upon into an array and applying a single operator to that entire array. The variables of APL are untyped, taking on the type of the objects assigned to them.

APL is especially useful for mathematical and array processing applications. Because of its powerful operators and compact notation, a great pastime among APL programmers is the construction of one-line programs. Such programs actually use APL in a purely functional manner that very closely matches the functional model.

Discuss: What are advantages and disadvantages of compact language notation?

Research: Prepare a report on the language features of APL. Include at least one example program.

1.3.1.6. BASIC

The BASIC (Beginners All-purpose Symbolic Instruction Code) language was developed at Dartmouth College by Thomas Kurtz and John Kemeny in the mid 1960s. Its objective was to be easy for undergraduate students to learn and to use the interactive programming environment that was also under development at Dartmouth at that time.

BASIC was quite popular in academic circles over the next decade, but its greatest popularity came with the arrival of the microcomputer in the mid 1970s. The marketers of microcomputers needed a language that would be useful to the consumer. The two major criteria were that the language be easy to learn and that it exploit the interactive environment provided by the microcomputer. Because BASIC was designed a decade earlier to meet these same two objectives, it was chosen as the language that was provided with all of the early microcomputers.

BASIC received another revival in the 1990s when Microsoft used it as the basis for Visual Basic, a language that combined BASIC’s ease of learning with the new Visual element that promoted the quick development of graphical user interfaces.

Although the microcomputer gave BASIC an important place in the history of programming languages, BASIC contributed little to the development of programming language technology. Perhaps its greatest contribution was that it was one of the first languages to provide an interactive programming environment, including the interpretive execution of programs, as a part of the language.

Discuss: BASIC is a language that permits programs to be developed interactively as well as executed interactively. Discuss the distinction between these two ideas.

Research: Investigate the language Visual BASIC. In particular, how is it related to the original BASIC language?
1.3.1.7. SNOBOL

Another language that was developed in the early 1960s is SNOBOL (StriNg Oriented symBOlic Language). SNOBOL was developed by Griswold in the early 1960s while he was working at Bell Labs. As its name implies, SNOBOL is a string-oriented language where every action is placed into the context of pattern-matching. The language contains many powerful pattern matching operators. It was useful for text processing and other string manipulation activities. The most popular version was SNOBOL4, which is described in Griswold (1971).

Although it is seldom used anymore, SNOBOL has a number of unique features that make it an interesting language to study, including its flow of control, its array operators, its use of patterns as a data type, and its ability to execute string data as SNOBOL code. We examine these and other interesting aspects of SNOBOL in Chapter 18.

Discuss: What types of applications might be inconvenient to implement in a string-oriented language like SNOBOL?

1.3.2 The Seventies: ALGOL-based Languages

The six languages described in the preceding section represent the first wave of programming language development. The next wave built on the ideas and concepts of that first wave. The most important languages to appear in the latter half of the 1960s, and to see extensive use in the 1970s, were based on the key concepts of the ALGOL 60 language. Four of these ALGOL-based languages are described in the following paragraphs.

1.3.2.1 PL/I

The philosophy behind PL/I (Programming Language/I), developed at IBM in the mid 1960s, was the replacement of the multitude of languages that were in use for specific applications with one general-purpose language. The approach used was to incorporate features from each of the earlier languages into PL/I. For example, PL/I included the block structure, control structures, and recursion from ALGOL 60, subprograms and formatted input/output from FORTRAN, file manipulation and the record structure from COBOL, dynamic storage allocation and linked structures from LISP, and the array operations from APL.

PL/I, although highly promoted by IBM, never became as popular as its designers hoped. The major difficulty was a lack of cohesiveness in the language design, which contained many different features implemented in many different ways. The language was complex, difficult to learn, and difficult to implement. Two possible remedies for these problems were included in the language: The use of many defaults that could remain transparent to the user, and the intention that a programmer needed to learn only a subset of the language for a given application. These remedies proved to be inadequate, however, and PL/I never realized the popularity its designers had envisioned.

Two features of PL/I that have had a significant impact on later language development. These are interrupt-handling, the ability to execute specified procedures when an exceptional
condition occurs, and multitasking, the specification of tasks that can be performed concurrently. These topics are explored in Chapters 7 and 9, respectively.

Discuss: What examples can you give of one language deriving features directly from another language?

Research: Investigate the PL/I language. Include an example program in PL/I

1.3.2.2 Simula 67

Ole-Johan Dahl and Kristen Nygaard developed Simula 67 at the Norwegian Computing Center in the early 1960s. The original work was based on ALGOL 60 and was intended to be a language for system description and simulation programming. The first version was called Simula 1. The designers soon discovered that this language had potential beyond simulation; to realize this potential they extended the original design to Simula 67.

The major contribution of Simula 67 is the concept of class. A class is an encapsulation of data and procedures, which can be instantiated in a number of objects. The class of Simula 67 is the forerunner of abstract data types as implemented in Ada and Modula-2 (see Chapter 8) and of classes from the object-oriented languages C++, Java, and Smalltalk (see Chapters 11 and 21). The latter three languages also adopted from Simula 67 the hierarchy of classes with inheritance of components. Classes are useful in the simulations for which Simula 67 was intended, because the components of a simulation can be considered as objects whose behavior is determined by the classes to which they belong.

1.3.2.3 ALGOL 68

Although its name implies that it is an improved version of ALGOL 60, ALGOL 68 is actually a rather radical departure from its predecessor. It was designed to be a general-purpose language as opposed to having the scientific orientation of ALGOL 60.

ALGOL 68 never gained acceptance by programmers even to the limited level attained by ALGOL 60. This was, in part, because the original description (van Wijngaarden and others, 1969) was difficult to understand, using notation and terminology that was foreign to many of its readers.

The major design philosophy of ALGOL 68 is also its major contribution, namely, orthogonality. A language that is orthogonal has a relatively small number of basic constructs and a set of rules for combining those constructs. It is then possible to combine these constructs using any of the rules with predictable results. This approach is in opposition to that of PL/I which included a large number of independent constructs.

Discuss: How is your ability to use a language effected by the way the language is described? What kind of a language description is most useful to you?

Research: What are some major differences between ALGOL 60 and ALGOL 68?
1.3.2.4 Pascal

The most popular of this second wave of ALGOL-based languages is Pascal, developed by Niklaus Wirth in 1969 and named for the mathematician Blaise Pascal. Wirth's goal was to provide a language that is simple to learn, supportive of structured programming, and easily implemented. He intended it to be a language suitable for use in the teaching of programming. The defining document is provided by Jensen & Wirth (1985).

By the early 1980s, Pascal had become by far the most commonly used language for teaching programming at the college level. By the middle 1980s, it also had become popular as a production language on microcomputers. Much of the popularity of Pascal was a result of its implementation along with a convenient development environment called Turbo Pascal. Wirth later developed a descendent of Pascal called Modula-2 that gave additional data abstraction capability. Modula-2 never enjoyed the popularity of Pascal, however.

Pascal's flexible control structures, user-defined data types, and file, record, and set data structures have made it a model for many of the languages of the next stage of development.

Research: Report on the life and accomplishments of Blaise Pascal.


Research: How is Modula-2 different from Pascal?

Discuss: Should a different language be used for teaching that is used for production programming? Why or why not?

1.3.3 Languages of the Eighties

Although each of the following four languages was actually designed in the 1970s, their major impact on computing occurred in the 1980s. The designers of these languages benefited greatly from experience with earlier languages, and all the languages include features that take advantage of then current hardware and software technology.

The first two languages use entirely different models of computing than the earlier languages whereas the last two continue development in the ALGOL line, following what we call the imperative model.

1.3.3.1 Prolog

Prolog was developed at the University of Marseilles in France in 1977. Prolog was designed for artificial intelligence applications and is based on formal logic. The logic-oriented model of programming served as a basis for Prolog, but Prolog falls short of the model's ideal of clauses that describe the problem and can be expressed in an order-independent way. This model and Prolog are described in Chapters 14 and 15.

Prolog has become a competitor with LISP for artificial intelligence research. It has proven especially useful for the implementation of expert systems and natural language
processing. Prolog received higher visibility in the early 1980s when it was chosen as the language of the Japanese Fifth Generation Project.

Research: What was the Fifth Generation Project and how successful was it?

1.3.3.2 Smalltalk

Alan Kay developed Smalltalk at the Xerox Palo Alto Research Center in the early 1970s as a part of the Dynabook project. The two distinguishing features of Smalltalk are its environment and the strict use of the object-oriented model. The Smalltalk language is embedded within a graphical environment which includes pop-up menus, windows, and the use of a mouse device for input. This environment has served as the prototype for many modern programming environments, including those of the Apple Macintosh and Microsoft Windows.

Smalltalk is designed around the Simula 67 class concept and includes encapsulation, inheritance and instantiation. All operations in Smalltalk consist of objects sending messages to other objects. Smalltalk is described in Chapter 21.

Discuss: What are the key features of a window environment in contrast to a command line system such as DOS? What is your preference and under what circumstances?

1.3.3.3 C

The language C was developed at Bell Laboratories in the early 1970s as a language for implementing the UNIX operating system. C is a powerful language with facilities for accessing raw data stored in memory as well as accessing memory through data types and structures of the language. This language was also designed to fit the architecture of the PDP-8 computer in a manner similar to FORTRAN’s relationship to the IBM 704. The standard for the C language is defined by Kernighan & Ritchie (1988).

The objective of C is to provide a language that has access to low-level data and generates efficient code. The language has an extensive set of operators. As a result, programs often are expressed with compact code at the expense of readability. C grew in popularity in conjunction with the acceptance of UNIX as an operating system and has proven to be an excellent language for the construction of portable system programs.

The features of C, both the beneficial and the problematic, have been carried over into C++, an extension of C that will be highlighted as a language of the nineties.

Discuss: What are the advantages and disadvantages of low-level programming language features such as the ability to access register and memory locations directly and bit-level operations?

Research: What features of C closely reflect the architecture of the PDP-8?
1.3.3.4 Ada

In the early 1970s, the United States Department of Defense initiated a project to obtain a suitable programming language for the development of embedded systems. An embedded system is a computer system that operates as a part of a larger system. A large portion of the work done by the Department of Defense is on these embedded systems.

After evaluating existing languages against the criteria desired, the Department of Defense decided that no language existed that met the needs and a new language should be designed. In 1977, a competition was initiated among four contractors to design a suitable language. In 1979, the winning design was chosen, and the resulting language was given the name Ada, after Ada Augusta Byron, the Countess of Lovelace and daughter of Lord Byron. She was a collaborator with Charles Babbage in his work on the analytical engine in the nineteenth century and is considered by some to have originated some of the key ideas that led to programming.

The original standard for Ada was published in 1983 and the language was enhanced significantly by new standards that appeared in 1995. This enhancement added object-oriented capabilities to Ada. Every compiler that aspires to be called an Ada compiler must be validated by the Department of Defense. Subsets and supersets of the language are not permitted.

Ada is based primarily on Pascal, but it uses the class concept of Simula 67 in its abstract data type facility called a package, adopts the exception-handling features of PL/I, and provides an extensive tasking facility for concurrent processing. This tasking facility is described in Chapter 20.

Discuss: What are examples of embedded systems that are in common use?

Discuss: What are the advantages and disadvantages of strict language standards.

Research: Find out more information about Ada Augusta Byron’s contributions to Babbage’s work.

1.3.4 Languages of the Nineties

Three languages dominated the interest and attention of programmers during the 1990s. Although only one had its origins in this decade, all three defined the programming landscape over this period. The driving forces behind the prominence of these three languages was the growing popularity of the object-oriented approach to problem-solving and the appearance of the World Wide Web.

1.3.4.1 C++

C++ was designed in the early 1980s by Bjarne Stroustrup at Bell Laboratories. It was a superset of the C language with features added that support object-oriented programming. These features were based on classes similar to those in Simula 67 and Smalltalk. Early implementations of C++ were first available in the mid-1980s and by the end of the decade, the language had achieved a suitable level of stability, as described in Ellis and Stroustrup (1990).
The major contribution of C++ was the marriage of the imperative and object-oriented models in a single language. This proved to be popular as it provided continuity with the predominant model of pre-1990 programming (imperative) while providing the programmer with facilities for utilizing the most popular model of the 1990s (object-oriented). Another important feature of C++ was the use of templates, a mechanism by which both functions and classes can be parameterized.

This workhorse language of the 1990s is described in Chapter 10.

**Research:** Investigate the systems programming languages used when developing utilities for Unix, Microsoft Windows, and other Operating Systems. Report on which systems use C++.

### 1.3.4.2 Perl

Perl is a popular representative of the genre of languages known as scripting languages. The basic purpose of a scripting language is to combine components written in other languages by providing links among these components. Scripting languages, which are described more fully in Chapter 17, often have facilities for text processing and provide tools for the rapid development of applications.

Developed in the early 1990s by Larry Wall, Perl is a highly popular scripting language that has enough power to be used for general applications. Its popularity stems from its arrival at the same time as the Common Gateway Interface (CGI) that is used for creating applications on the World Wide Web. Perl draws heavily on earlier scripting languages in its design, but is known for its combination of simplicity and power. It is defined in Wall, Christiansen, and Schwartz (1996).

**Research:** Report on how the design of earlier languages affected the design of Perl.

### 1.3.4.3 Java

No language better defines programming in the 1990s than Java. Whereas it takes most languages years or decades to become popular, Java caught on right away. Designed at the beginning of the 1990s by a team at Sun Microsystems headed by James Gosling, Java owes much of its immediate success to its simultaneous arrival on the scene with the World Wide Web around 1993.

Java’s design was based on that of C++ and included most of that language’s imperative and object-oriented facilities. In addition, Java contained an extensive built-in class hierarchy with many facilities, but most notably with those for producing graphical user interfaces. The Java language was also heavily tied to its platform independent implementation through the use of an intermediate code. The ability of Java to support multi-threaded processing for concurrent execution was also a Java addition.

During the latter half of the 1990s, Java became an important player among programming languages, especially in education and in programming applications for the World Wide Web. For this reason, Java is used as the primary example language in this textbook.

**Discuss:** Many languages have been designed by corporations. How does this affect those languages use and popularity?

Research: Report on where Java is used today.