Programming Languages

A Brief Review

V Rajaraman

In this article we review the development of programming languages and classify them based on their structures and their applications.

V Rajaraman is with the Jawaharlal Nehru Centre for Advanced Scientific Research and the Indian Institute of Science, Bangalore. Several generations of scientists and engineers in India have learnt computer science using his lucidly written text books on programming and computer fundamentals.

Introduction

Programming languages for computers are developed with the primary objective of facilitating a large number of persons to use computers without the need to know in detail the internal structure of a computer. Languages are matched to the type of applications which are to be programmed using the language. The ideal language would be one which expresses precisely the specification of a problem to be solved, and converts it into a series of instructions for a computer. It is not possible to achieve this ideal as a clear specification of a problem is often not available and developing an algorithm from specifications requires subject knowledge and expertise. In actual practice, a detailed algorithm to solve a problem is the starting point and it is expressed as a program in a programming language. A large number of languages, over a thousand, exist each catering to a different class of applications. All modern programming languages (with one exception) are designed to be *machine independent*. In other words, the structure of the programming language would not depend upon the internal structure of a specified computer; one should be able to execute a program written in the programming language on any computer regardless of who manufactured it or what model it is. Such languages are known as *high level machine independent programming languages*.

In this article we will briefly review various programming languages which are currently used. We will look at a classification of programming languages based on their characteristics and

another classification based on their applications. We will also point out some of the recent developments in programming languages.

Assembly Language

The first step in the evolution of programming languages was the development of what is known as an *assembly language*. In an assembly language, mnemonics are used to represent operations to be performed by the computer and strings of characters to represent addresses of locations in the computer's memory where the operands will be stored. Thus the language is matched to a particular computer's processor structure and is thus *machine dependent*. A translator called an *assembler* translates a program written in assembly language to a set of machine instructions, which can be executed by a computer. Now-a-days programs are written in assembly language only in applications which are cost sensitive or time critical as efficiency of machine code is of paramount importance in these types of applications. A cost sensitive application is one in which microprocessors are used to enhance the functionality of consumer items such as washing machines or music systems. In these cases the program is stored in a *read only memory* and its size is small. Thus code optimisation is important. A time-critical application is use of microprocessors in aircraft controls where real time operation of the system is required. Here again the number of machine instructions executed should be minimised.

In an assembly language, mnemonics are used to represent operations to be performed by the computer and strings of characters to represent addresses of locations in the computer's memory where the operands will be stored.

High Level Languages

During the evolution of computers, till about 1955, computers were slow and had a small memory. Thus programming efficiency was very important and assembly language was dominant. With improvements in technology, computers were designed with larger memory capacity, higher speed and improved reliability. The tremendous potential of computer applications in diverse areas was foreseen. It was evident that this potential could be realised only if a non-expert user could effectively use the computer to solve problems. It was thus clear that a user should

be concerned primarily with the development of appropriate algorithms to solve problems and not with the internal logical structure of a computer. Consequently a good notation to express algorithms became an essential requirement. For algorithms to be executed by computers, the notation to express them should be simple, concise, precise and unambiguous. The notation should also match the type of algorithm. For example, programming languages to solve science and engineering problems should support arithmetic using wide ranging, high precision real and complex numbers and should have features to express operations with arrays and matrices. On the other hand, algorithms for processing business data would have operations to be performed on massive amounts of organised data known as files. The notation, in this case, must facilitate describing files and formatting and printing intricate reports. Such notations to express algorithms are known as *high level, machine independent, programming languages*. High level programming languages are further classified as *procedural* and *non-procedural*. Languages which express step-by-step algorithms written to solve a problem are known as procedural languages whereas those which express specifications of a program to be solved are known as non-procedural. We will first discuss the common features of procedural languages.

Languages which express step-by-step algorithms written to solve a problem are known as procedural languages whereas those which express specifications of a program to be solved are known as non-procedural.

Procedural Languages

Procedural languages have facilities to:

i) specify data elements such as real, integer, boolean, characters and data structures such as arrays, matrices, stacks, records, sets, strings of characters, lists, trees, etc.,

ii) control structures to sequence operations to be performed. An *if then else* structure is necessary to allow programs to follow different sequences of statements based on testing a condition. For example, the following statement:

```
if ( a > b ) then
      x = y + z ;
```

Procedural languages are designed using a set of *syntax rules*, which precisely specify the 'words' of the language, and how they may be combined legally.

```
        p = q + t
else
        x = y - z ;
        p = q * t
endif
```

commands that the statements x = y + z and p = q + t are to be executed if (a > b) is *true*. If (a > b) is *false* x = y - z and p = q * t are executed.

iii. Repetition structures which carry out a group of statements again and again while a condition is *true* as shown below

```
while (a > b ) do
        x = y - z ;
        p = q * r
end while
```

iv. Statements to input and output data.

Procedural languages are designed using a set of *syntax rules*, which precisely specify the 'words' of the language, and how they may be combined legally. The rules of syntax are specified using a notation called Backus–Naur Form (BNF) which recursively defines various syntactic units of the language. These rules are similar to the ones used by the great Sanskrit grammarian Panini. A sample BNF definition of a variable name is

```
< variable name > : = < letter >
< variable name > : = < letter > < digit >
< variable name > : = < variable name > < variable name >
```

where < letter > is any upper case English letter A to Z and < digit > is any digit between 0 and 9.

Observe the third line in the above definition, which is a recursive definition.

Each syntactically correct structure should have one and only one semantic interpretation.

Besides rules of syntax each language has *semantic rules*. Each syntactically correct structure should have one and only one semantic interpretation.

Associated with each high level language is an elaborate computer program which translates it into the machine language of the computer in which it is to be executed. There are two types of translators. One of them takes each statement of the high level language, translates it and immediately executes it. This is called an *interpreter*. Interpreters are easy to write but the translated programs' execution is slow. The other approach is to scan the whole program and translate it into an equivalent machine language program. Such a translator is called a *compiler*. A compiler is a complex program but the compiled machine code takes lesser time to execute compared to an interpreted program.

Interpreters are easy to write but the translated programs' execution is slow.

A Classification of Programming Languages

We give in *Figure 1* a classification of programming languages. We have classified high level machine independent languages into three groups, namely, procedural, non-procedural and problem-oriented. Procedural languages have as their starting

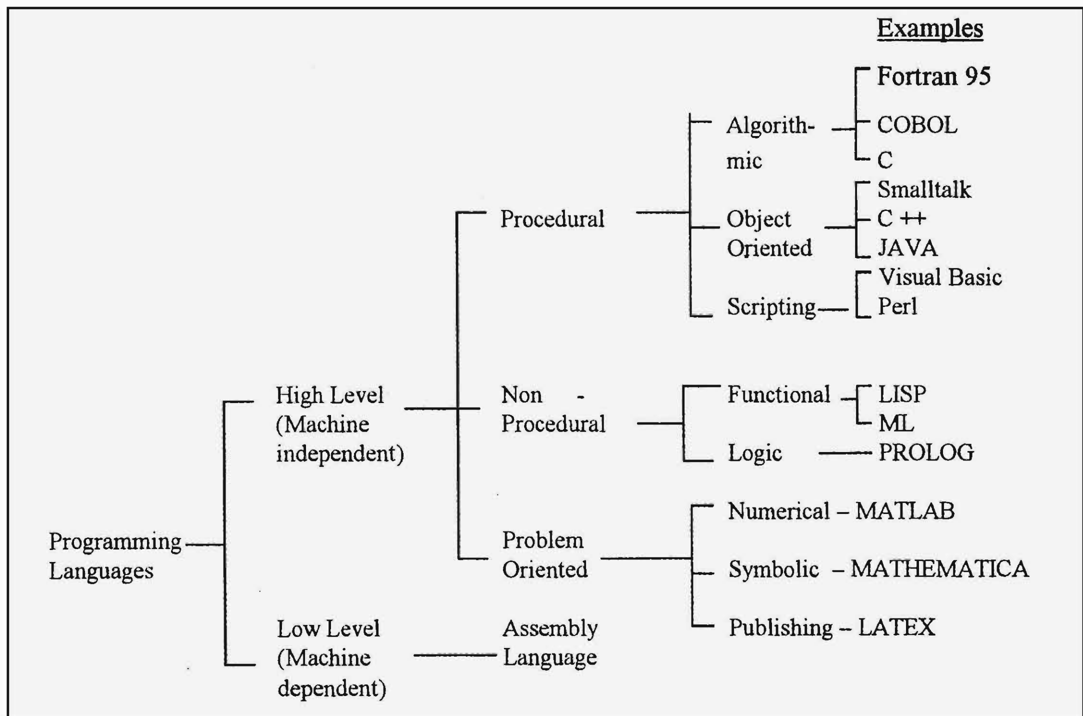

Figure 1. Classification of Programming Languages.

An object models a complex real world or an abstract object.

point an algorithm to solve the problem. Languages such as FORTRAN, COBOL and C are purely algorithmic. These languages provide a methodology to break up a large job into a number of tasks and programming the tasks independently as functions or subroutines. These functions or subroutines are then combined to form a program. The general idea is to simplify debugging a program and to reuse the procedures in other programs which may need them. Over the years it was realised that this was not sufficient to enable re-use of programs. Subroutines and functions are too rigid. They require the specification of the type of data to be used a priori and the data to be passed to them in a pre-specific order. As the cost of programming continually increased it was realised that 'building' programs using a library of reusable 'components' was imperative. This led to the emergence of the so-called *object-oriented languages*. In these languages the concept of subroutine/functions is extended to that of an object. An object models a complex real world or an abstract object. A real world object, for example, is a student whereas an abstract object is a course taken by a student. In an object oriented (OOP) program an object is modelled by a collection of data structures and a set of procedures that can be performed on this data structure. A program consists of a collection of objects, each object providing a service when it is invoked and all the objects co-operating to get the job done. Objects are invoked by sending messages to them and objects return messages when the job is done.

The advantages of object oriented programming (OOP) accrue only when a large software project is undertaken. The methodology of OOP enables a programmer to remain close to the conceptual higher level model of the real world problem.

The action performed in response to a message can vary depending on the data and type of parameters. This is called *polymorphism*. Objects form class hierarchy with super class (parent) and subclass (child) relationship. An object can use procedures and data defined on objects in its superclass through *inheritance*.

The advantages of object oriented programming (OOP) accrue only when a large software project is undertaken – also known as 'programming in the large'. The methodology of OOP enables a programmer to remain close to the conceptual higher level

model of the real world problem. One of the earliest OOP languages to be developed was Smalltalk. It, however, did not become popular. Currently an object oriented version of C known as C++ is the most popular OOP language.

JAVA is getting wide acceptance now as a programming language to write applications for a network of heterogeneous computers.

Another development, which has taken place in the last few years, is the *internet* – an international network of a very large number of national computer networks. The technology developed in creating the internet has been adapted for networking computers within an organization. A computer network within an organization using protocols and providing services similar to an internet is called an *intranet*. In both inter and intranet small application programs (agents or objects to perform some services – known as *applets*) may be developed at any one of the computers connected to the network. One would like to create a new application by using these applets by either importing them to one's own computer or using them via the network. A language known as JAVA, which is an object oriented language achieves this. This language achieves machine independence by defining a JAVA virtual machine for which the compiler is written. The JAVA code compiled for the virtual machine is then executed on any machine by an interpreter which generates machine code from the compiled code. This technique makes it easy to port JAVA language to any machine quickly (see *Figure 2*). JAVA is getting wide acceptance now as

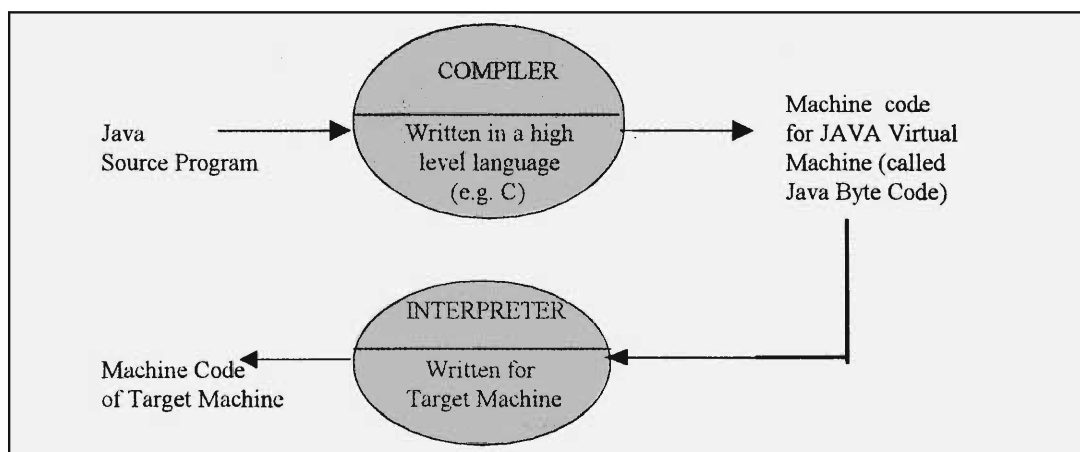

Figure 2. Illustrating portability of JAVA.

Scripting languages assume that a collection of useful programs, each performing a task, already exists in other languages. It has facilities to combine these components to perform a complex task.

a programming language to write applications for a network of heterogeneous computers.

Scripting Languages: Programming languages such as C and JAVA are also known as system programming languages as they have been used to develop large systems. For example C has been used to write the Unix operating system. System programming languages are strongly typed, that is, each variable must be declared as a particular type – real, integer, pointer etc. Typing is used both for easy readability and enabling more efficient compilation and error detection. Another class of languages, which are gaining wider acceptance is called *scripting language* [3]. Scripting languages assume that a collection of useful programs, each performing a task, already exists in other languages. It has facilities to combine these components to perform a complex task. A scripting language may be thus thought of as a *gluing language*, which glues together components. One of the earliest scripting languages is *Unix Shell.* Unix shell filter programs, read a stream of bytes from an input and write a stream of bytes on to an output. Any two programs can be connected by attaching the output of one program to the input of the other. The following shell commands stack three filters to count the number of lines in the selection that contains the word 'language'.

```
select | grep language | wc
```

The program select reads the given text that is currently on the display and prints the text on its output; the grep program reads its input and prints as its output the lines containing the word 'language'; the wc program counts the number of lines on its input. Each of these programs select, grep and wc are independent programs which could be combined with other programs also in many ways. Another popular scripting language is *Visual Basic,* which is used to develop *Graphical User Interfaces* (GUI) on the screen of a *Visual Display Unit.* It is expected that with increasing complexity of applications it will be more cost effective

	Assembly	*System Programming (e.g. C)*	*Scripting (e.g. PERL)*
No.of instructions per statement of language	1	5	100
Degree of typing	None	Strong	Weak
Applications	Time Critical, Cost Critical	Routine applications	GUI, Gluing components

Table 1. Comparison of Languages.

to glue together existing 'program components' using scripting languages [3]. In *Table 1* we give a comparison of some of the languages.

Non-procedural Languages: In procedural languages (also known as imperative languages) each statement causes the values stored in one or more memory locations to change. Program design consists of writing a sequence of statements, which transform the 'state' of the memory from an initial state to a final state which is the solution to the problem.

Non-procedural functional languages solve a problem by applying a set of functions to the initial variables in specific ways to get the answer. The syntax of such languages is similar to

$$f_n (f_{n-1} (f_{n-2} \ldots\ldots\ldots f_1 (\text{data}))\ldots\ldots)$$

where f's are the successive function applications which transform their arguments which, at the start, is the initial data. LISP and ML are two languages in use which support this model. LISP has been widely used to program artificial intelligence applications.

Non-procedural functional languages solve a problem by applying a set of functions to the initial variables in specific ways to get the answer.

Another non-procedural class of languages is called *rule based languages* or *logic programming languages.* A logic program is expressed as a set of atomic sentences (known as facts) and Horn clauses (*if then* rules). A query is then posed. Execution of the program now begins and the system tries to find out if the answer to the query is *true* or *false* for the given facts and rules. PROLOG is the best known language of this type.

Languages known as 4GLs (Fourth Generation Languages) are also used which provide query languages to access data from data bases and manipulate them.

Problem Oriented Languages: Problem oriented languages are designed to solve a narrow class of problems. A user of such a language need not express in detail the procedure used to solve a problem. Readymade procedures are pre-programmed. The user merely presents the data in a flexible 'language'. MATLAB is a very popular language among scientists and engineers to solve a wide class of problems in digital signal processing, control systems, modelling systems described by differential equations, matrix computations etc.

Another class of problem oriented languages is for symbolic manipulation, for example, simplifying a complex algebraic expression or getting the indefinite integral of a complex expression. MATHEMATICA is a popular language of this type.

Classification Based on Applications

Another method of classifying computer languages is by applications. The major applications of computers are in the following areas:

i. Business Data Processing where large files are to be processed. COBOL has been the dominant language in this area. We have seen, however, the emergence of spreadsheet based 'languages' for answering 'what if' type questions. Languages known as 4GLs (Fourth Generation Languages) are also used which provide query languages to access data from data bases and manipulate them. 4 GLs also have special features like 'fill in the blanks' to obtain answers to queries and for designing good looking forms.

ii. Scientific applications require numeric intensive computing such as those used to solve problems in science and engineering. Fortran 90 is the dominant language in this area. C is making inroads. Recently Fortran 95 standard has been published which incorporates features to write Fortran programs for parallel computers.

iii. System programs such as those used to write compilers and

Box 1. Ada Language

Most computer languages evolve from the work of a small group and take years to get standardised. Through an initiative from the United States Department of Defence in early 1970s a standard for a programming language called Ada was approved in 1983 before a working compiler was written. Ada was designed based on world wide competition, where a French entry by Jean Ichbiah won in 1979. Initially the language was named DOD-1 but the name was changed later to Ada in honour of Lady Ada Augusta Lovelace who is reputed to have programmed an early mechanical computer designed by Charles Babbage in UK in the 1850s. Ada is a large complex language, which includes the concept of tasks, concurrent execution, real-time execution of tasks, exception handling and abstract data types. Due to its complexity, compilers did not appear till 1987 in spite of support and funding by US Department of Defence. It was revised in 1995 to include better object orientation and better tasking models for processes. It is, however, more or less dead today probably due to its complexity and strong competition from C and later C++ and Java.

operating systems. In this area C and more recently C++ dominate. A language known Ada was specially designed to write programs for these applications but did not become popular. (See *Box 1*)

iv. Scripting programs: Another class of applications is to combine 'program components' to build large programs. Examples of these are: commands to 'back up' files at specified times, sending replies automatically to email messages and invoking certain processes automatically when some conditions are satisfied. Languages have been developed to specify such tasks and sequence them to execute automatically. In UNIX operating system the user command language is called the *shell* and command programs as *shell scripts*. This class of languages is called *scripting languages*. One such language is called PERL (Practical Extraction and Report Language). Visual Basic is used to develop graphical user interfaces.

v. Artificial intelligence applications are characterised by algorithms, which search large data spaces for specific patterns. Typical examples are chess playing programs which generate many potential moves and search for the 'best' move within a given time using heuristic rules. LISP and Prolog are preferred languages in this area.

Box 2. Survival of Programming Languages

A language does not seem to survive even if it is very good and has strong Government support. Complex interplay of vendor support, committed user groups, corpus of existing application programs, ease of learning and use, good compilers giving efficient object code and popular hype seem to prop up some languages at the expense of others. Except for perennials such as FORTRAN and COBOL most other languages have a life of less than a decade. Even in these cases only their names have survived! The modern versions of FORTRAN and COBOL are quite different from the versions of the 60s.

vi. Publishing has become an important application of computers. Languages for word-processing are proliferating and have special formatting commands, print commands etc. TEX is a popular language used to typeset material with complicated mathematical equations. The TEX translator produces a program in the Postscript page description language for printing the material using a laser printer.

Conclusions

The area of programming languages is dynamic, even somewhat chaotic (See *Box 2*). As more sophisticated hardware systems appear in the market new computer applications emerge. These applications spawn new languages to solve such applications. Another trend is the continuous increase in complexity of applications as hardware become more sophisticated and cheaper. The increase in size of programs needs new methods of tackling complexity while keeping the cost of program development low and ensuring correctness of programs.

Address for Correspondence
V Rajaraman
IBM Professor of Information Technology,
Jawaharlal Nehru Centre for Advanced Scientific Research
and
Hon.Professor, Supercomputer Education & Research Centre
Indian Institute of Science,
Bangalore 560 012, India
email: rajaram@serc.iisc.ernet.in

Suggested Reading

[1] Pratt T W and Zelkowitz M V. *Programming Languages*. (3rd edition), Prentice Hall of India, New Delhi, 1996.
[2] Bird R and Wadler P. *Introduction to Functional Programming*. Prentice Hall Inc., Englewood–Cliffs, N J, USA, 1988.
[3] Ousterhout J K. Scripting: High level Programming for the 21st Century. *Computer*. (IEEE, U.S.A), Vol.31, No.3, 23–30, March 1998.
[4] Wilkes M V. A Revisionist Account of Early Language Development. *Computer*. (IEEE, U.S.A.), Vol.31, No.4, 22–26, April 1998.
[5] Mohan T S. The Java Internet. *Resonance*. Vol.1.No.5,1996.
[6] Shyamsundar R K. Introduction to Algorithms–4 Turtle graphics. *Resonance*. Vol.1.No.9,1996.