Preliminaries – Chapter 1

Reasons for Studying Concepts of Programming Languages

Why do we studying programming language concepts?

Increased capacity to express ideas

- Programming languages limit programmers on the kinds of control structures, data structures, and abstractions that they can use; thus the algorithms that they can construct.
- Awareness of a wider variety of programming language features can reduce such limitations in software development.
- Programmers can increase the range of their software-development thought processes by learning new language constructs.
- The study of programming language concepts builds an appreciation for valuable language features and encourages programmers to use them.

Improved background for choosing appropriate languages

- Many professional programmers have had little formal education in computer science and learned their programming on their own or through in-house training.
- Many other programmers received their formal training in the distant past.
- When given a choice, many programmers will continue to use the language with which they are most familiar, even if it is poorly suited to a new project.

Increased ability to learn new languages

- The process of learning a new programming language can be lengthy and difficult, especially for someone who is comfortable with only one or two languages and has never examined programming language concepts in general.
- Once a thorough understanding of the fundamental concepts of languages is acquired, it becomes far easier to see how these concepts are incorporated into the design of the language being learned.
- It is essential that programmers know the vocabulary and fundamental concepts of programming languages so they can read and understand programming language manuals and literature for languages and compilers.

Better understanding of the significance of implementation

- This allows programmers to understand why languages are designed the way they are, which leads to the ability to use the language more intelligently.
- Certain kinds of bugs can only be found and fixed by a programmer who knows related implementation details.
- Understanding implementation issues allows programmers to visualize how a computer executes various language constructs.
- Understanding implementation issues allows programmers to understand the relative efficiency of alternative constructs (e.g., iterative algorithm versus recursive algorithm).

Overall advancement of computing

- Many people believe that the most popular programming languages are not always the best available.
- In some cases, a language might have become widely used because those in position to choose languages were not sufficiently familiar with programming language concepts (e.g., FORTRAN versus ALGOL 60).

Programming Domains

Computers have been applied to a wide variety of different areas, from controlling nuclear power plants to providing video games in mobile phones. Because of this great diversity in computer use, programming languages with very different goals have been developed.

Scientific Applications

- Began in the 1940s
- Used with the first digital computers.
- Typically have simple data structures (e.g., arrays and matrices).
- Require large numbers of floating-point arithmetic computations.
- The most common control structures are counting loops and selection.
- Early high-level languages competed with assembly language, so efficiency was a primary concern.
- The first language used was FORTRAN (efficient).
- ALGOL 60 and its descendents were also intended for use in this area as well as other related areas.

Business Applications

- Began in the 1950s.
- Special computers were developed for this purpose.
- Special languages were developed for this purpose.
- The first successful language used for business was COBOL (1960). COBOL is still the most commonly used language.
- Primary concerns: report generation, precise ways to describe and store decimal numbers and character data, and the ability to specify decimal arithmetic operations.
- Spreadsheet systems and database systems were developed for business and are now widely used both in homes and businesses.

Artificial Intelligence (AI)

- Uses symbolic rather than numeric computations.
- Symbolic computation means that symbols, consisting of names rather than numbers, are manipulated.
- Symbolic computation is more conveniently done with linked lists of data rather than arrays.
- The first widely used programming language used for AI was the functional language LISP (1959).
- During the early 1970s an alternative approach to some of these applications appeared – logic programming using the language Prolog.
- More recently, some AI applications have been written in scientific languages such as C.

Systems Programming

- The operating system and all of the programming support tools are collectively known as systems software.
- Used almost continuously and therefore must be efficient.
- Must have low-level features that allow the software interfaces to external devices to be written.
- IBM, Digital, and Burroughs/UNISYS developed special machine-oriented high-level languages for systems software on their machines (1960s and 1970s).
    - IBM developed PL/S, a dialect of PL/I, for their mainframes.
    - Digital developed BLISS, a language just above assembly language.
    - Burroughs developed Extended ALGOL.

- The UNIX operating system was written almost entirely in C which made it relatively easy to port, or move, to different machines.
    - Some C characteristics make it a good choice for systems programming because it is low-level, it is execution efficient, and it does not burden the user with safety restrictions.
    - Systems programmers are often excellent programmers and do not believe they need such restrictions.
    - Some nonsystems programmers find C to be too dangerous to use on large, important software systems.

Web Software

- Supported by a wide range of languages, ranging from markup languages, such as XHTML, which is not a programming language, to general-purpose programming languages, such as Java.
- XHTML provides a way of embedding presentation instructions in the pages of information, which could be text, images, sound, or animation, that constitute Web content.
- Because of the need for dynamic Web content, computation functionality can be provided by embedding programming code in an XHTML document using a scripting language such as JavaScript and PHP.
- An XHTML document can request the execution of a separate program on the Web server to provide dynamic content which can be written in virtually any programming language.

Language Evaluation Criteria

The table below lists common criteria used for evaluating languages along with the language characteristics that affect them.

	CRITERIA		
Characteristic	**Readability**	**Writability**	**Reliability**
Simplicity/orthogonality	X	X	X
Control structures	X	X	X
Data types and structures	X	X	X
Syntax design	X	X	X
Support of abstraction		X	X
Expressivity		X	X
Type checking			X
Exception handling			X
Restricted aliasing			X

Readability

- Perhaps one of the most important criteria for judging a programming language is the ease with which programs can be read and understood.
- Before 1970, software development was largely thought of in terms of writing code. The primary positive characteristics of programming languages were efficiency and machine readability.
- After 1970, the software life cycle concept was developed; coding was relegated a much smaller role and maintenance was recognized as a major part of the cycle, particularly in terms of cost.
- Because ease of maintenance is determined in large part by the readability of programs, readability became an important measure of the quality of programs and programming languages.

Overall Simplicity

- A language with a large number of basic constructs is more difficult to learn. Programmers will often learn a subset of the constructs. A program reader may not know the same subset as the program author.
- A second complicating characteristic is "feature multiplicity" – when a language offers more than one way to accomplish a particular operation.
    - count = count + 1 or count += 1 or count++ or ++count
- A third potential problem is "operator overloading" – when a single operator symbol has more than one meaning. What if a programmer overloads a symbol with a new and completely unrelated meaning.

Orthogonality

- A relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language.
- Every possible combination of primitives is legal and meaningful.
- Closely related to simplicity – the more orthogonal, the fewer the exceptions, the higher the degree of regularity in the design – this makes the language easier to learn, read, and understand.
- Too much combinational freedom can allow for extremely complex constructs. This extreme form of orthogonality can lead to unnecessary complexity.
- A functional language, such as Lisp, offers a good combination of simplicity and orthogonality because it can accomplish everything with a single construct, the function call, which can be combined with other function calls in simple ways.

Example 1:

IBM – not orthogonal – only two of four combinations legal – more restrictive

A	Reg1, memory_cell	– fixed format statements
AR	Reg1, Reg2	– additional instructions required

Equivalent to: Reg1 ← contents(Reg1) + contents(memory_cell)
Reg1 ← contents(Reg1) + contents(Reg2)

VAX – orthogonal – all four combinations legal

ADDL	operand1, operand2	– operand can be Reg or memory_cell

Equivalent to: operand2 ← contents(operand1) + contents(operand2)

Example 2:

The C programming language rules show a lack of orthogonality in a high-level language. Compare the rules that apply to the following two data structures:

Arrays
- Cannot be returned by functions
- Elements can be any data type except void or function
- Passed by reference

Records (structs)
- Can be returned by functions
- Members can be any data type except void or a structure of the same type
- Passed by value

Control Statements
- The structured programming revolution of the 1970s was in part a reaction to the poor readability caused by the inadequate control statements.
- The goto statement severely reduced program readability.
- A program that can be read from top to bottom is much easier to understand than one that requires the programmer to jump from one statement to some nonadjacent statement.
- Early versions of BASIC and FORTRAN lacked the control statements that allow strong restrictions on the use of the goto statement.

C with the while statement

```
while ( incr < 20 )
{
        while ( sum <= 100 )
        {
                sum = sum + incr;
        }
        ++incr;
}
```

C without the while statement

```
loop1:
        if ( incr >= 20 ) go to out;
loop2:
        if ( sum > 100 ) go to next;
                sum = sum + incr;
                go to loop;
next:
        ++incr;
        go to loop1;
out:
```

Data Types and Structures

- The presence of adequate facilities for defining data types and data structures in a language is another significant aid to readability.

Boolean data type:

timeOut = 1 versus timeOut = true

Parallel arrays versus arrays of records:

Fortran 95

```
Character (Len = 30) Name (100)
Integer Age (100), Employee_Id (100)
Real Salary (100)
```

Pascal

```
TYPE
        Employee_Record = RECORD
                Name : String[30];
                Age, Employee_Id : Integer;
                Salary : Real;
        END;
        Employee_List = ARRAY[1..100] OF Employee_Record;
VAR
        Employees : Employee_List;
```

Syntax Considerations

- The syntax of the elements of a language has a significant effect of the readability of programs.
- Identifier forms – short identifiers detracts from readability
    - 1 character max – BASIC (ANSI, 1978b)
    - 6 characters max – FORTRAN 77
- Special words (sometimes symbols)
    - Reserved words or key words: class, for, while, etc…
    - Used in forming compound statements, or statement groups
        - Ada uses “end if” and “end loop”
    - Conflict between simplicity (fewer – Java) and readability (more – Ada)
    - Can special words be used as identifiers? May create confusion.
- Form and meaning
    - Designing statements so that their appearance indicates their purpose is an aid to readability.
    - Semantics (meaning) should follow directly from syntax (form).
    - C – static
        - Compile time variable if declared in a function
        - File variable (not exportable) if declared outside functions

Writability

- Measures how easily a language can be used to create programs for a chosen domain.
- Most of the characteristics that affect readability also affect writability. The process of writing a program requires the programmer frequently to reread the part of the program that is already written.
- Writability must be considered in the context of the target problem domain of the language. For example, COBOL and Fortran cannot be compared for creating programs that deals with:
    - two-dimensional arrays – Fortran is ideal for this
    - producing complex financial reports – COBOL is idea for this

Simplicity and Orthogonality

- Misuse or disuse of features may arise if a language has a large number of constructs that a programmer is not familiar with.
- A smaller number of primitive constructs and a consistent set of rules for combining them is much better than having a large number of primitives.
- Too much orthogonality can lead to errors in programs that go undetected when any combination of primitives is legal.

Support for Abstraction

- The ability to define complex structures or operations in ways that allow implementation details to be ignored.
- The degree of abstraction allowed by a programming language and the naturalness of its expression are very important to its writability.
- Two categories of abstraction:
    - Process – use subprograms to implement algorithms in a program. The function main is used to call the other functions in the program – the implementation details for these functions are not stated in main.
    - Data – consider a binary tree that stores one integer value it each node.
        - It is natural to use an abstraction of a tree node in the form of a simple class/struct with two pointers and an integer value in Java and C++.
        - It is not natural to use three parallel arrays, two for subscripts indicating child nodes and one for the integer value, in Fortran.
- This is an important factor in the writability of a language.

Expressivity

- This can refer to several different characteristics.
- In APL it means that there are very powerful operators that allow a great deal of computation to be accomplished with a small program – allows for long, elaborate expressions.
- Commonly means that a language has relatively convenient ways of specifying computations.
    - C – count++ is more convenient and shorter than count = count + 1
    - Ada – the “and then” Boolean operator is a convenient way to specify short-circuit evaluation of a Boolean expression.
    - Java – the “for” statement makes writing counting loops easier that using the “while” statement.
- All of these increase the writability of a language.

Reliability

A program is reliable if it performs to its specifications under all conditions.

Type Checking

- Testing for type errors in a given program – by the compiler or during program execution.
- Run-time type checking is expensive – compile-time type checking is preferred.
- The earlier the errors are detected the less expensive it is to make repairs.
- Java checks nearly all variables and expressions at compile time. This eliminates virtually all type errors at run time.
- The original version of C did absolutely no type checking of function arguments. There was no guarantee that mismatched arguments would yield meaningful results. This also made it very difficult to diagnose problems.
- This is an important factor in the reliability of a programming language.

Exception handling

- Ability of a program to intercept run-time errors (as well as other unusual conditions detected by the program) and apply corrective measures.
- Ada, C++, and Java include extensive exception handling capabilities.
- C and Fortran include virtually no exception handling capabilities.
- This is an aid to the reliability of a programming language.

Aliasing

- Having two or more distinct referencing methods, or names, for the same memory cell.
- Widely accepted that aliasing is a dangerous feature in a programming language.
- Most languages support some kind of aliasing – such as having two pointers pointing to the same variable.
- Sometimes used to overcome deficiencies in the language's abstraction facilities.
- Sometimes greatly restricted to increase reliability.

Cost

Total cost of a programming language is a function of many of its characteristics.

- Cost of training programmers – function of simplicity and orthogonality.
- Cost of writing programs – function of writability.
- Cost of compiling programs – the first-generation Ada compilers were prohibitively high.
- Cost of executing programs – a large number of run-time checks prohibits fast code execution.
- Compiling/Execution trade-off: Optimization is the name given to a collection of techniques that compilers may use to decrease size and/or increase the execution speed of the code produced. No optimization means quick compile. Lots of compilation means quick execution.
- Cost of the language implementation system.
    - Java has been rapidly accepted because of the availability of its free compiler/interpreter system.
    - Ada did not become popular in the early days because of expensive first-generation compilers.
- Cost of reliability – when systems fail, critical or not, the cost can be very high.
- Cost of maintenance – includes both corrections and modifications – function on several characteristics, but primarily readability. Maintenance costs can be two to four times as much as development.

Additional criteria for evaluating programming languages:

- Portability – the ease with which programs can be moved from one implementation to another. This is strongly influenced by language standardization.
- Generality – the applicability to a wide range of applications.
- Welldefinedness – the completeness and precision of the language's official definition document.

Most criteria are neither precisely defined nor exactly measured.

Language design criteria are weighed differently from different perspectives:

- Designers – elegance and the attraction of widespread use.
- Implementors – difficulty of implementing constructs and features.
- Users – writability first and readability later.
- These characteristics sometimes conflict with one another.