Chapter 1 "Preliminaries"

Reasons for Studying Concepts of Programming Languages

- *Increased ability to express ideas*.
    - It is believed that the depth at which we think is influenced by the expressive power of the language in which we communicate our thoughts. It is difficult for people to conceptualize structures they can't describe, verbally or in writing.
    - Language in which they develop S/W places limits on the kinds of control structures, data structures, and abstractions they can use.
    - Awareness of a wider variety of P/L features can reduce such limitations in S/W development.
    - Can language constructs be simulated in other languages that do not support those constructs directly?

- *Improved background for choosing appropriate languages*
    - Many programmers, when given a choice of languages for a new project, continue to use the language with which they are most familiar, even if it is poorly suited to new projects.
    - If these programmers were familiar with other languages available, they would be in a better position to make informed language choices.

- *Greater ability to learn new languages*
    - Programming languages are still in a state of continuous evolution, which means continuous learning is essential.
    - Programmers who understand the concept of OO programming will have easier time learning Java.
    - Once a thorough understanding of the fundamental concepts of languages is acquired, it becomes easier to see how concepts are incorporated into the design of the language being learned.

- *Understand significance of implementation*
    - Understanding of implementation issues leads to an understanding of why languages are designed the way they are.
    - This in turn leads to the ability to use a language more intelligently, as it was designed to be used.

- *Ability to design new languages*
    - The more languages you gain knowledge of, the better understanding of programming languages concepts you understand.

- *Overall advancement of computing*
    - In some cases, a language became widely used, at least in part, b/c those in positions to choose languages were not sufficiently familiar with P/L concepts.

- Many believe that ALGOL 60 was a better language than Fortran; however, Fortran was most widely used. It is attributed to the fact that the programmers and managers didn't understand the conceptual design of ALGOL 60.
- Do you think IBM has something to do with it?

Programming Domains

- **Scientific applications**
  - In the early 40s computers were invented for scientific applications.
  - The applications require large number of floating point computations.
  - Fortran was the first language developed scientific applications.
  - ALGOL 60 was intended for the same use.
- **Business applications**
  - The first successful language for business was COBOL.
  - Produce reports, use decimal arithmetic numbers and characters.
  - The arrival of PCs started new ways for businesses to use computers.
  - Spreadsheets and database systems were developed for business.
- **Artificial intelligence**
  - Symbolic rather than numeric computations are manipulated.
  - Symbolic computation is more suitably done with linked lists than arrays.
  - LISP was the first widely used AI programming language.
- **Systems programming**
  - The O/S and all of the programming supports tools are collectively known as its system software.
  - Need efficiency because of continuous use.
- **Scripting languages**
  - Put a list of commands, called a script, in a file to be executed.
  - PHP is a scripting language used on Web server systems. Its code is embedded in HTML documents. The code is interpreted on the server before the document is sent to a requesting browser.
- Special-purpose languages
  - RPG is an example of these languages.

Language Evaluation Criteria

Readability

- Software development was largely thought of in term of writing code "**LOC**".
- Language constructs were designed more from the point of view of the computer than the users.
- 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. The result is a crossover from focus on machine orientation to focus on human orientation.
- The most important criterion "ease of use"
- **Overall simplicity** "Strongly affects readability"

- Too many features make the language difficult to learn. Programmers tend to learn a subset of the language and ignore its other features. "ALGOL 60"
- Multiplicity of features is also a complicating characteristic "having more than one way to accomplish a particular operation."
- Ex "**Java**":

```
count = count + 1
count += 1
count ++
++count
```

- Although the last two statements have slightly different meaning from each other and from the others, all four have the same meaning when used as stand-alone expressions.
- Operator overloading where a single operator symbol has more than one meaning.
- Although this is a useful feature, it can lead to reduced readability if users are allowed to create their own overloading and do not do it sensibly.

- **Orthogonality**
  - Makes the language easy to learn and read.
  - Meaning is context independent. Pointers should be able to point to any type of variable or data structure. The lack of orthogonality leads to exceptions to the rules of the language.
  - 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 is legal and meaningful.
  - Ex: page 11 in book.
  - The more orthogonal the design of a language, the fewer exceptions the language rules require.
  - The most orthogonal programming language is ALGOL 68. Every language construct has a type, and there are no restrictions on those types.
  - This form of orthogonality leads to unnecessary complexity.

- **Control Statements**
  - It became widely recognized that indiscriminate use of goto statements severely reduced program readability.
  - Ex: Consider the following nested loops written in C

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

if C didn't have a loop construct, this would be written as follows:

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

- Basic and Fortran in the early 70s lacked the control statements that allow strong restrictions on the use of gotos, so writing highly readable programs in those languages was difficult.
- Since then, languages have included sufficient control structures.
- The control statement design of a language is now a less important factor in readability than it was in the past.

- **Data Types and Structures**
  - The presence of adequate facilities for defining data types and data structures in a language is another significant aid to reliability.
  - Ex: Boolean type.
    timeout = 1 or timeout = true

- **Syntax Considerations**
  - The syntax of the elements of a language has a significant effect on readability.
  - The following are examples of syntactic design choices that affect readability:
    - *Identifier forms*: Restricting identifiers to very short lengths detracts from readability. ANSI BASIC (1978) an identifier could consist only of a single letter of a single letter followed by a single digit.
    - *Special Words*: Program appearance and thus program readability are strongly influenced by the forms of a language's special words. Ex: **while**, **class**, **for**. C uses braces for pairing control structures. It is difficult to determine which group is being ended. Fortran 95 allows programmers to use special names as legal variable names.
    - *Form and Meaning*: Designing statements so that their appearance at least partially indicates their purpose is an obvious aid to readability.
    - Semantic should follow directly from syntax, or form.
    - Ex: In C the use of **static** depends on the context of its appearance.
      If used as a variable inside a function, it means the variable is created at compile time.
      If used on the definition of a variable that is outside all functions, it means the variable is visible only in the file in which its definition appears.

Writability

- It is a measure of how easily a language can be used to create programs for a chosen problem domain.
- Most of the language characteristics that affect readability also affect writability.
- Simplicity and orthogonality
  - A smaller number of primitive constructs and a consistent set of rules for combining them is much better than simply having a large number of primitives.
- Support for abstraction
  - **Abstraction** means the ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored.
  - A process abstraction is the use of a subprogram to implement a sort algorithm that is required several times in a program instead of replicating it in all places where it is needed.
- Expressivity
  - It means that a language has relatively convenient, rather than cumbersome, ways of specifying computations.
  - Ex: ++count ⇔ count = count + 1 // more convenient and shorter

Reliability

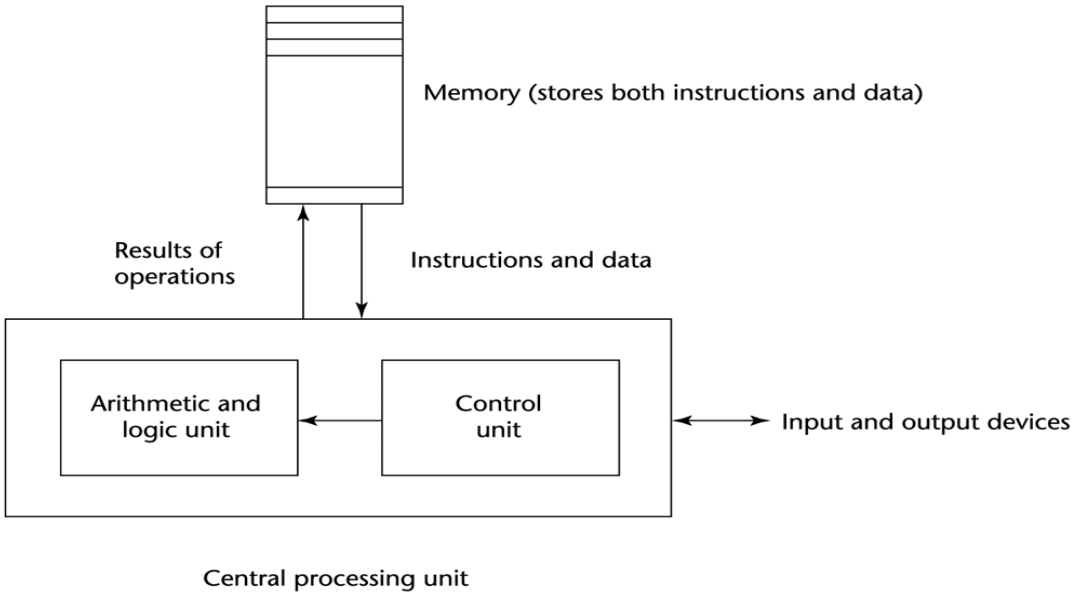
- A program is said to be **reliable** if it performs to its specifications under all conditions.
- **Type checking**: is simply testing for type errors in a given program, either by the compiler or during program execution.
  - The earlier errors are detected, the less expensive it is to make the required repairs. Java requires type checking of nearly all variables and expressions at compile time.
- **Exception handling**: the ability to intercept run-time errors, take corrective measures, and then continue is a great aid to reliability.
- **Aliasing**: it is having two or more distinct referencing methods, or names, for the same memory cell.
  - It is now widely accepted that aliasing is a dangerous feature in a language.
- **Readability and writability**: Both readability and writability influence reliability.

Cost

- Categories
  - Training programmers to use language
  - **Writing programs** "Writability"
  - Compiling programs
  - Executing programs
  - Language implementation system "Free compilers is the key, success of Java"
  - **Reliability**, does the software fail?
  - **Maintaining** programs: Maintenance costs can be as high as two to four times as much as development costs.
  - Portability "standardization of the language"
  - Generality (the applicability to a wide range of applications)

Influences on Language Design

- **Computer architecture**: Von Neumann
- We use imperative languages, at least in part, because we use von Neumann machines
  - Data and programs stored in same memory
  - Memory is separate from CPU
  - Instructions and data are piped from memory to CPU
  - Results of operations in the CPU must be moved back to memory
  - Basis for imperative languages
    - Variables model memory cells
    - Assignment statements model piping
    - Iteration is efficient



Programming methodologies

- 1950s and early 1960s: Simple applications; worry about machine efficiency
- Late 1960s: People efficiency became important; readability, better control structures
  - Structured programming
  - Top-down design and step-wise refinement
- Late 1970s: Process-oriented to data-oriented
  - data abstraction
- Middle 1980s: Object-oriented programming

Language Categories

- Imperative
  - Central features are variables, assignment statements, and iteration
  - C, Pascal
- Functional
  - Main means of making computations is by applying functions to given parameters
  - LISP, Scheme
- Logic
  - Rule-based
  - Rules are specified in no special order
  - Prolog
- Object-oriented
  - Encapsulate data objects with processing
  - Inheritance and dynamic type binding
  - Grew out of imperative languages
  - C++, Java

Programming Environments

- The collection of tools used in software development
- UNIX
  - An older operating system and tool collection
- Borland JBuilder
  - An integrated development environment for Java
- Microsoft Visual Studio.NET
  - A large, complex visual environment
  - Used to program in C#, Visual BASIC.NET, Jscript, J#, or C++
  -

Chapter 2

- Genealogy of common high-level programming languages