Chapter 1

Preliminaries

Chapter 1 Topics

1. Reasons for Studying Concepts of Programming Languages
2. Programming Domains
3. Language Evaluation Criteria
4. Influences on Language Design
5. Language Categories
6. Language Design Trade-Offs
7. Implementation Methods
8. Programming Environments

1.1 Reasons for Studying Concepts of Programming Languages

- Increased ability to express ideas
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Better understanding of significance of implementation
- Overall advancement of computing
1.2 Programming Domains

- Scientific applications
  - Large number of floating point computations
  - Fortran
- Business applications
  - Produce reports, use decimal numbers and characters
  - COBOL
- Artificial intelligence
  - Symbols rather than numbers manipulated
  - LISP
- Systems programming
  - Need efficiency because of continuous use
  - C
- Web Software
  - Eclectic collection of languages: markup (e.g., XHTML), scripting (e.g., PHP), general-purpose (e.g., Java)

1.3 Language Evaluation Criteria

- Readability: the ease with which programs can be read and understood
- Writability: the ease with which a language can be used to create programs
- Reliability: conformance to specifications (i.e., performs to its specifications)
- Cost: the ultimate total cost
Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Few feature multiplicity (means of doing the same operation)
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in a relatively small number of ways
  - Every possible combination is legal
- Control statements
  - The presence of well-known control structures (e.g., while statement)
- Data types and structures
  - The presence of adequate facilities for defining data structures
- Syntax considerations
  - Identifier forms: flexible composition
  - Special words and methods of forming compound statements
  - Form and meaning: self-descriptive constructs, meaningful keywords

Evaluation Criteria: Writability

- Simplicity and orthogonality
  - Few constructs, a small number of primitives, a small set of rules for combining them
- Support for abstraction
  - The ability to define and use complex structures or operations in ways that allow details to be ignored
- Expressivity
  - A set of relatively convenient ways of specifying operations
  - Example: the inclusion of for statement in many modern languages

Evaluation Criteria: Reliability

- Type checking
  - Testing for type errors
- Exception handling
  - Intercept run-time errors and take corrective measures
- Aliasing
  - Presence of two or more distinct referencing methods for the same memory location
- Readability and writability
  - A language that does not support “natural” ways of expressing an algorithm will necessarily use “unnatural” approaches, and hence reduced reliability
Evaluation Criteria: Cost

- Training programmers to use language
- Writing programs (closeness to particular applications)
- Compiling programs
- Executing programs
- Language implementation system: availability of free compilers
- Reliability: poor reliability leads to high costs
- Maintaining programs

Evaluation Criteria: Others

- Portability
  - The ease with which programs can be moved from one implementation to another
- Generality
  - The applicability to a wide range of applications
- Well-definedness
  - The completeness and precision of the language’s official definition

1.3 Language Evaluation Criteria

- Readability: the ease with which programs can be read and understood
- Writability: the ease with which a language can be used to create programs
- Reliability: conformance to specifications (i.e., performs to its specifications)
- Cost: the ultimate total cost
Evaluation Criteria: Readability

- Overall simplicity
  - A manageable set of features and constructs
  - Few feature multiplicity (means of doing the same operation)
  - Minimal operator overloading
- Orthogonality
  - A relatively small set of primitive constructs can be combined in
    a relatively small number of ways
  - Every possible combination is legal
- Control statements
  - The presence of well-known control structures (e.g., while
    statement)
- Data types and structures
  - The presence of adequate facilities for defining data structures
- Syntax considerations
  - Identifier forms: flexible composition
  - Special words and methods of forming compound statements
  - Form and meaning: self-descriptive constructs, meaningful
    keywords

Evaluation Criteria: Writability

- Simplicity and orthogonality
  - Few constructs, a small number of primitives, a small
    set of rules for combining them
- Support for abstraction
  - The ability to define and use complex structures or
    operations in ways that allow details to be ignored
  - Process and Data Abstraction
- Expressivity
  - A set of relatively convenient ways of specifying
    operations
  - Example: the inclusion of for statement in many
    modern languages

Evaluation Criteria: Reliability

- Type checking
  - Testing for type errors
- Exception handling
  - Intercept run-time errors and take corrective measures
- Aliasing
  - Presence of two or more distinct referencing methods for
    the same memory location
- Readability and writability
  - A language that does not support "natural" ways of
    expressing an algorithm will necessarily use "unnatural"
    approaches, and hence reduced reliability
Evaluation Criteria: Cost

- Training programmers to use language
- Writing programs (closeness to particular applications)
- Compiling programs
- Executing programs
- Language implementation system: availability of free compilers
- Reliability: poor reliability leads to high costs
- Maintaining programs

Evaluation Criteria: Others

- Portability
  - The ease with which programs can be moved from one implementation to another
- Generality
  - The applicability to a wide range of applications
- Well-definedness
  - The completeness and precision of the language’s official definition

1.4 Influences on Language Design

- Computer Architecture
  - Languages are developed around the prevalent computer architecture, known as the von Neumann architecture
- Programming Methodologies
  - New software development methodologies (e.g., object-oriented software development) led to new programming paradigms and by extension, new programming languages
Computer Architecture Influence

- Well-known computer architecture: Von Neumann
- Imperative languages, most dominant, because of von Neumann computers
  - Data and programs stored in memory
  - Memory is separate from CPU
  - Instructions and data are piped from memory to CPU
  - Basis for imperative languages
    - Variables model memory cells
    - Assignment statements model piping
    - Iteration is efficient

The von Neumann Architecture

Programming Methodologies Influences

- 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
  - Data abstraction = inheritance + polymorphism
1.5 Language Categories

- Imperative
  - Central features are variables, assignment statements, and iteration
  - Examples: C, Pascal
- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, Scheme
- Logic
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog
- Object-oriented
  - Data abstraction, inheritance, late binding
  - Examples: Java, C++
- Markup
  - New; not a programming per se, but used to specify the layout of information in Web documents
  - Examples: XHTML, XML

1.6 Language Design Trade-Offs

- Reliability vs. cost of execution
  - Conflicting criteria
  - Example: Java demands all references to array elements be checked for proper indexing but that leads to increased execution costs
- Readability vs. writability
  - Another conflicting criteria
  - Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability
- Writability (flexibility) vs. reliability
  - Another conflicting criteria
  - Example: C++ pointers are powerful and very flexible but not reliably used

1.7 Implementation Methods

- Compilation
  - Programs are translated into machine language
- Pure Interpretation
  - Programs are interpreted by another program known as an interpreter
- Hybrid Implementation Systems
  - A compromise between compilers and pure interpreters
Layered View of Computer

The operating system and language implementation are layered over the machine interface of a computer.

Compilation

- Translate high-level program (source language) into machine code (machine language)
- Slow translation, fast execution
- Compilation process has several phases:
  - Lexical analysis: converts characters in the source program into lexical units
  - Syntax analysis: transforms lexical units into parse trees which represent the syntactic structure of the program
  - Semantics analysis: generate intermediate code
  - Code generation: machine code is generated

The Compilation Process
Additional Compilation Terminologies

- **Load module** (executable image): the user and system code together
- **Linking and loading**: the process of collecting system program and linking them to user program

Execution of Machine Code

- **Fetch-execute-cycle (on a von Neumann architecture)**

  initialize the program counter
  repeat forever
  fetch the instruction pointed by the counter
  increment the counter
  decode the instruction
  execute the instruction
  end repeat

Von Neumann Bottleneck

- Connection speed between a computer's memory and its processor determines the speed of a computer
- Program instructions often can be executed a lot faster than the above connection speed; the connection speed thus results in a *bottleneck*
- Known as von Neumann bottleneck; it is the primary limiting factor in the speed of computers
Pure Interpretation

- No translation
- Easier implementation of programs (runtime errors can be easily and immediately displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Becoming rare on high-level languages
- Significant comeback with some Web scripting languages (e.g., JavaScript)

Pure Interpretation Process

Hybrid Implementation Systems

- A compromise between compilers and pure interpreters
- A high-level language program is translated to an intermediate language that allows easy interpretation
- Faster than pure interpretation
- Examples
  - Perl programs are partially compiled to detect errors before interpretation
  - Initial implementations of Java were hybrid; the intermediate form, byte code, provides portability to any machine that has a byte code interpreter and a run-time system (together, these are called Java Virtual Machine)
Hybrid Implementation Process

1. Initially translate programs to an intermediate language
2. Then compile intermediate language into machine code
3. Machine code version is kept for subsequent calls
4. JIT systems are widely used for Java programs
5. .NET languages are implemented with a JIT system

Just-in-Time Implementation Systems

Preprocessors

1. Preprocessor macros (instructions) are commonly used to specify that code from another file is to be included
2. A preprocessor processes a program immediately before the program is compiled to expand embedded preprocessor macros
3. A well-known example: C preprocessor
   - expands #include, #define, and similar macros
1.8 Programming Environments

- The collection of tools used in software development
- UNIX
  - An older operating system and tool collection
  - Nowadays often used through a GUI (e.g., CDE, KDE, or GNOME) that run on top of UNIX
- 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++

Summary

- The study of programming languages is valuable for a number of reasons:
  - Increase our capacity to use different constructs
  - Enable us to choose languages more intelligently
  - Makes learning new languages easier
- Most important criteria for evaluating programming languages include:
  - Readability, writability, reliability, cost
- Major influences on language design have been machine architecture and software development methodologies
- The major methods of implementing programming languages are: compilation, pure interpretation, and hybrid implementation

Genealogy of Common Languages