

Chapter 1 Topics

- Reasons for Studying Concepts of Programming Languages
- Programming Domains
- Language Evaluation Criteria
- Influences on Language Design
- Language Categories
- Language Design Trade-Offs
- Implementation Methods
- Programming Environments

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
- Better use of languages that are already known
- Overall advancement of computing

Copyright © 2012 Addison-Wesley. All rights reserved.

1-3

Programming Domains

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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-4

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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-5

Evaluation Criteria: Readability

- Overall simplicity
 - A manageable set of features and constructs
 - Minimal feature multiplicity
 - 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
- Data types
 - Adequate predefined data types
- Syntax considerations
 - Identifier forms: flexible composition
 - Special words and methods of forming compound statements
 - Form and meaning: self-descriptive constructs, meaningful keywords

Copyright © 2012 Addison-Wesley. All rights reserved.

1-6

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
 - Strength and number of operators and predefined functions

Copyright © 2012 Addison-Wesley. All rights reserved.

1-7

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 require the use of “unnatural” approaches, and hence reduced reliability

Copyright © 2012 Addison-Wesley. All rights reserved.

1-8

Evaluation Criteria: Cost

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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-9

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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-10

Influences on Language Design

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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-11

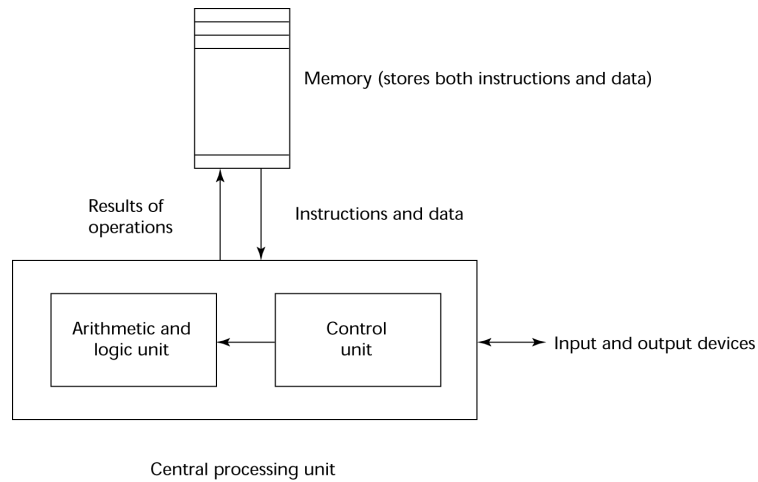
Computer Architecture Influence

- 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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-12

The von Neumann Architecture



Copyright © 2012 Addison-Wesley. All rights reserved.

1-13

The von Neumann Architecture

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

```

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

```

Copyright © 2012 Addison-Wesley. All rights reserved.

1-14

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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-15

Language Categories

- Imperative
 - Central features are variables, assignment statements, and iteration
 - Include languages that support object-oriented programming
 - Include scripting languages
 - Include the visual languages
 - Examples: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- Functional
 - Main means of making computations is by applying functions to given parameters
 - Examples: LISP, Scheme, ML, F#
- Logic
 - Rule-based (rules are specified in no particular order)
 - Example: Prolog
- Markup/programming hybrid
 - Markup languages extended to support some programming
 - Examples: JSTL, XSLT (extending HTML, XHTML, XML)

Copyright © 2012 Addison-Wesley. All rights reserved.

1-16

Language Design Trade-Offs

- **Reliability vs. Cost of Execution**
 - Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs

- **Readability vs. Writability (Expressiveness)**

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**
 - Example: C++ pointers are powerful and very flexible but are unreliable (prone to programmer-error/mistake)

Copyright © 2012 Addison-Wesley. All rights reserved.

1-17

Implementation Methods

- **Compilation**
 - Programs are translated into machine language; includes JIT systems
 - Use: Large commercial applications

- **Pure Interpretation**
 - Programs are interpreted by another program known as an interpreter
 - Use: Small programs or when efficiency is not an issue

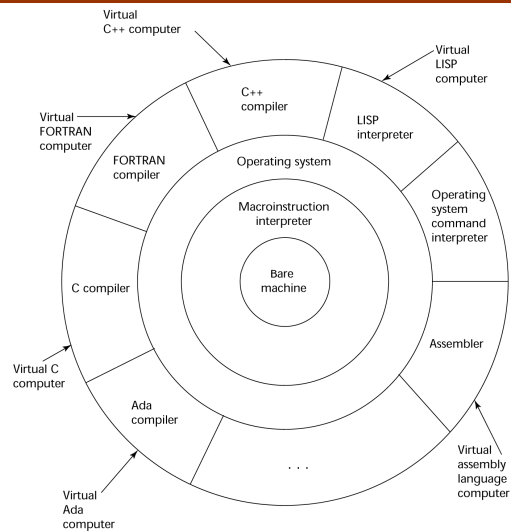
- **Hybrid Implementation Systems**
 - A compromise between compilers and pure interpreters
 - Use: Small and medium systems when efficiency is not the first concern

Copyright © 2012 Addison-Wesley. All rights reserved.

1-18

Layered View of Computer

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



Copyright © 2012 Addison-Wesley. All rights reserved.

1-19

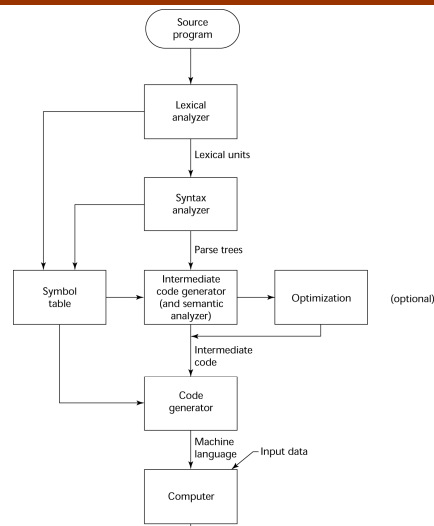
Compilation

- 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 program
 - Semantics analysis: generate intermediate code
 - code generation: machine code is generated

Copyright © 2012 Addison-Wesley. All rights reserved.

1-20

The Compilation Process



Copyright © 2012 Addison-Wesley. All rights reserved.

1-21

Additional Compilation Terminologies

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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-22

Von Neumann Bottleneck

- Connection speed between a computer's memory (RAM, Disk, ...) and its processor determines the speed of a computer
- Program instructions often can be executed much faster than the speed of the connection; the connection speed thus results in a *bottleneck*
- Known as the *von Neumann bottleneck*; it is the primary limiting factor in the speed of computers

Copyright © 2012 Addison-Wesley. All rights reserved.

1-23

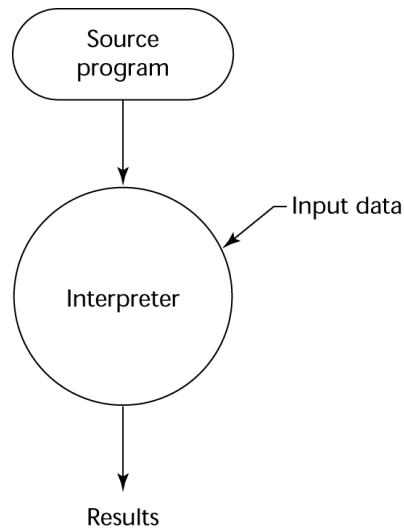
Pure Interpretation

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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-24

Pure Interpretation Process



Copyright © 2012 Addison-Wesley. All rights reserved.

1-25

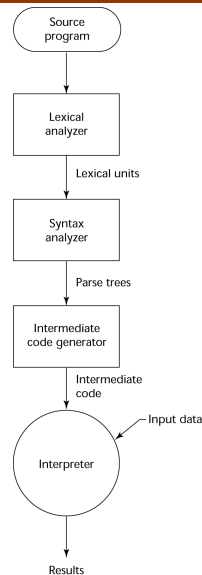
Hybrid Implementation Systems

- 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*)

Copyright © 2012 Addison-Wesley. All rights reserved.

1-26

Hybrid Implementation Process



Copyright © 2012 Addison-Wesley. All rights reserved.

1-27

Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile the intermediate language of the subprograms into machine code when they are called
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system
- In essence, JIT systems are delayed compilers

Copyright © 2012 Addison-Wesley. All rights reserved.

1-28

Preprocessors

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

Copyright © 2012 Addison-Wesley. All rights reserved.

1-29

Programming Environments

- A 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 runs on top of UNIX
- Microsoft Visual Studio.NET
 - A large, complex visual environment
- Used to build Web applications and non-Web applications in any .NET language
- NetBeans
 - Related to Visual Studio .NET, except for applications in Java
- Eclipse, ...

Copyright © 2012 Addison-Wesley. All rights reserved.

1-30

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