CMPS 3500

Programming Languages

Dr. Chengwei Lei
CEECS
California State University, Bakersfield
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
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

- Memory (stores both instructions and data)
- Results of operations
- Instructions and data

- Arithmetic and logic unit
- Control unit

Central processing unit

Input and output devices
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**
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
Coding in metal

- https://www.youtube.com/watch?v=_8aH-M3PzM0
The Main Battery Plot room aboard the USS Missouri, where the Rangekeeper Mark 8 and its associated analog computing hardware was tended to. The switchboards on the wall controlled which turrets and guns were under the system's control.

- The Rangekeeper Mark 8 still gave operators the option of manually entering data just in case a connection to a sensor failed,
- and they could also override or adjust data based on observation of shot and other corrections.
- The machine could even be operated without electricity by turning a hand crank.
The Institute of Advanced Study (IAS) computer is a multi-year research project conducted under the overall supervision of world-famous mathematician John von Neumann.

The notion of storing both data and instructions in memory became known as the ‘stored program concept’ to distinguish it from earlier methods of instructing a computer.

The IAS computer was designed for scientific calculations and it performed essential work for the US atomic weapons program.
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
DEC PDP-1 computer system (1960)

- The typical PDP-1 computer system, which sells for about $120,000,
- includes a cathode ray tube graphic display, paper tape input/output, needs no air conditioning and requires only one operator;
- all of which become standards for minicomputers.

- Its large scope intrigued early hackers at MIT, who wrote the first computerized video game, SpaceWar!, as well as programs to play music. More than 50 PDP-1s were sold.
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
Amdahl Corporation introduces the Amdahl 470

- Gene Amdahl, father of the IBM System/360, starts his own company, Amdahl Corporation, to compete with IBM in mainframe computer systems.

- The 470V/6 was the company’s first product and ran the same software as IBM System/370 computers but cost less and was smaller and faster.
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
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
  - 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
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
Figure: Compiler

Figure: Interpreter
Layered View of Computer

The operating system and language implementation are layered over 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 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 units and linking them to a user program
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 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
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)
Pure Interpretation Process

Source program

Input data

Interpreter

Results
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
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
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.
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
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