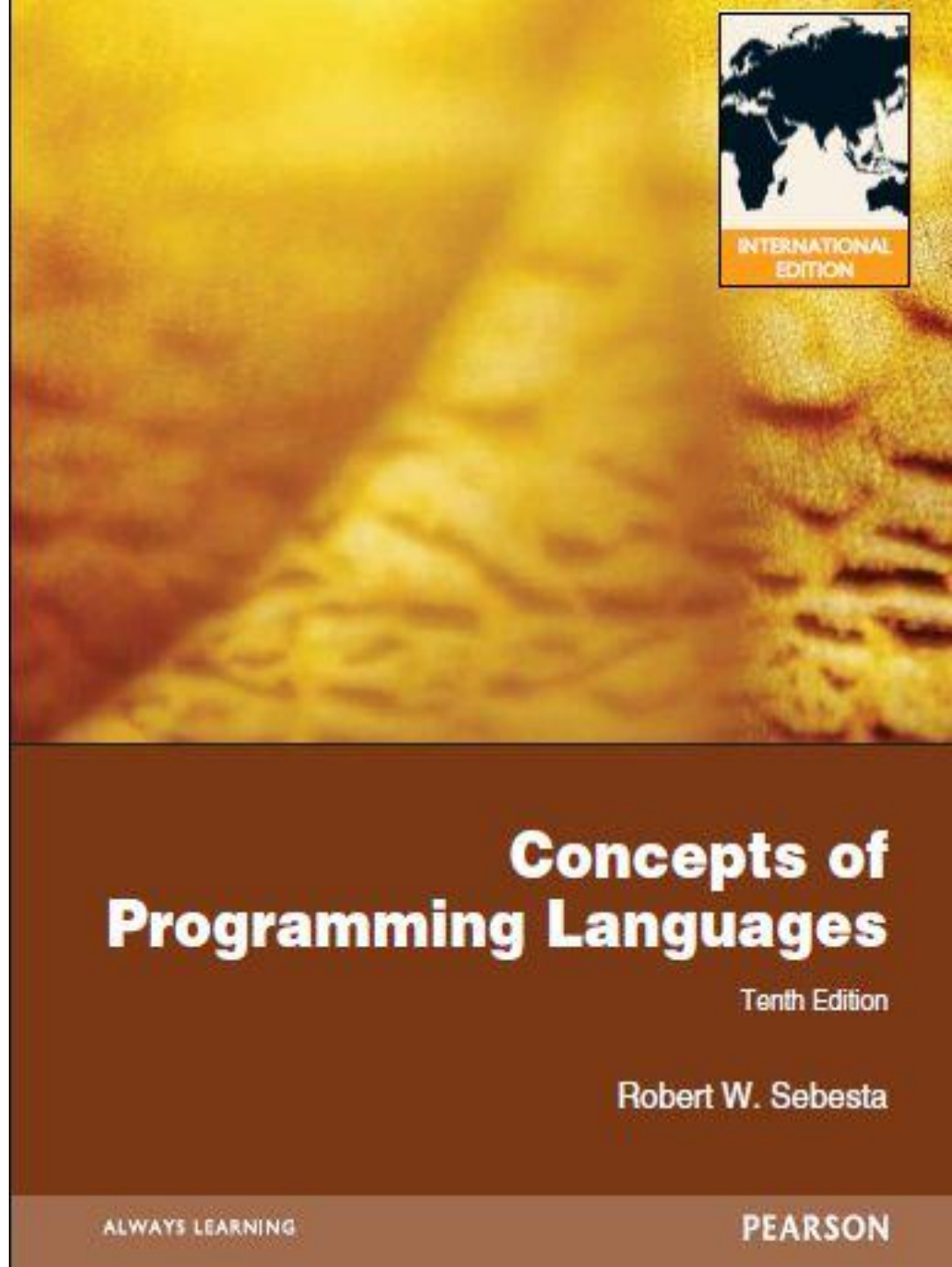


Programming Language

Instructor:

Min-Chun Hu

anita_hu@mail.ncku.edu.tw



Lecture 1

Introduction

- ❑ Reasons for studying Concepts of Programming Languages
- ❑ Programming Domains
- ❑ Language Evaluation Criteria
- ❑ Influences on Language Design
- ❑ Evolution of the Major Programming Languages

Why to learn PL?

- Increase capacity to express ideas
- Improve background for choosing appropriate languages
- Better use of languages that are already known
- Increased ability to learn new languages
- Better understanding of the significance of implementation
- Overall advancement of computing

Programming Domains

- Scientific Application
 - ▣ Fortran, ALGOL60
- Business Applications
 - ▣ COBOL
- Artificial Intelligence
 - ▣ LISP, Prolog (Symbolic rather than numeric computations)
 - ▣ Python (Deep Learning)
- Systems Programming
 - ▣ PL/S, PL/I, BLISS, C, C++
- Web Software
 - ▣ JAVA, Markup language such as HTML, Scripting language such as JavaScript

Language Evaluation Criteria

Language evaluation criteria and the characteristics that affect them.

Characteristic	CRITERIA		
	READABILITY	WRITABILITY	RELIABILITY
Simplicity	•	•	•
Orthogonality	•	•	•
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

Simplicity for Readability

- The overall simplicity of a programming language strongly **affects its readability**
- Potential problems for readability
 - The program's author has learned a different subset from that subset with which the reader is familiar
 - **Feature multiplicity**: having more than one way to accomplish a particular operation
 - **Operator overloading**: a single operator symbol has more than one meaning

Orthogonality for Readability

- 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.

IBM mainframe:

A Reg1, memory_cell

AR Reg1, Reg2

VAX

ADDL operand_1, operand_2

- A lack of orthogonality leads to **exceptions** to the rules of the language.

Data Types for Readability

- The presence of adequate facilities for defining data types and data structures is another significant aid to readability.
 - `timeOut = 1`
 - `timeOut = true`
 - With Boolean type, the meaning of the statement would be clear

Syntax Design for Readability

- Special words (**class**, **while**, **for**)
 - **}**, **end** → **end loop**, **end if**
 - can be legal variable name or not ?
- Form and meaning
 - In C, the reserved word **static** inside/outside a function means different semantics
 - In UNIX, the commands can be only deciphered by the one familiar with UNIX

Support of Abstraction for Writability

- **Abstraction** means the ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored.
 - Process abstraction
 - subprogram
 - Data abstraction

Expressiveness for Writability

- **Expressiveness** means that a language has relatively convenient ways of specifying computations.
 - In C, `count++` is more convenient and shorter than `count = count+1`
 - `for` makes writing counting loops easier than with the use of `while`
 - In APL, expressiveness means there are very **powerful operators** allowing a great deal of computation to be accomplished with a very small program.

Type Checking for Reliability

- Run-time type checking is expensive, compile-time type checking is more desirable.
- Checking type consistency between actual parameters and expected parameters can avoid following errors which might be difficult to diagnose.
 - ▣ In Java, check of the types of nearly all variables and expressions at compile time.

Exception Handling for Reliability

- **Exception handling** is the ability to
 - ▣ intercept run-time errors and other unusual conditions detectable by the program
 - ▣ take corrective measures and then continue
- Ada, C++, Java, C# have extensive capabilities for exception handling

Aliasing for Reliability

- **Aliasing** is having two or more names that can be used to access the same memory cell.
 - ▣ A dangerous feature
 - ▣ Two pointers set to point to the same variable
- How to avoid/restrict aliasing by the design of language?

Cost

- The total cost of a language is a function of many of its characteristics.
 - Cost of training programmers to use the language
 - A function of the simplicity and orthogonality of the language, and the experience of the programmer
 - Cost of writing programs in the language
 - A function of the writability of the language depending on its closeness in purpose to a particular application
 - Cost of compiling programs in the language
 - Cost of execution programs written in a language
 - Run-time type check will prohibit fast code execution
 - Trade-off between compilation cost and execution cost

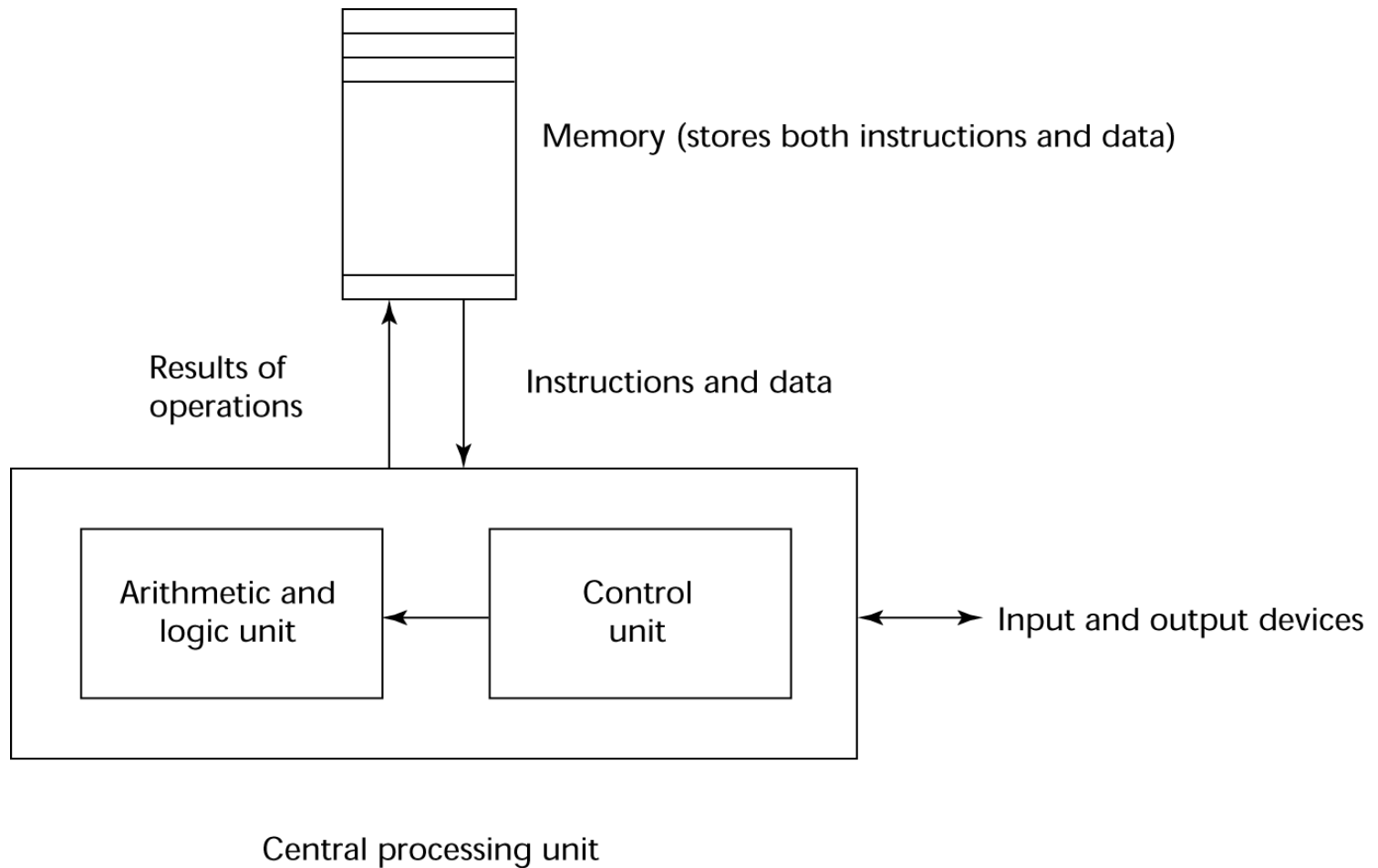
Cost (Cont.)

- ❑ Trade-off between compilation cost and execution speed
 - Optimization can be used to decrease the size of the code and increase the execution speed of the code
 - Language with optimization will slow down the compilation time
- ❑ Cost of the language implementation system
- ❑ Cost of poor reliability
- ❑ Cost of maintaining
 - Correction and modification to add new functions

Other evaluation criteria

- Portability
 - The ease that the program 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 defining document

The von Neumann Architecture



The von Neumann Architecture

- Fetch–execute–cycle
 - ▣ the execution of a machine code program (on a von Neumann architecture computer)

initialize the program counter

repeat forever

 fetch the instruction pointed by the counter

 increment the counter

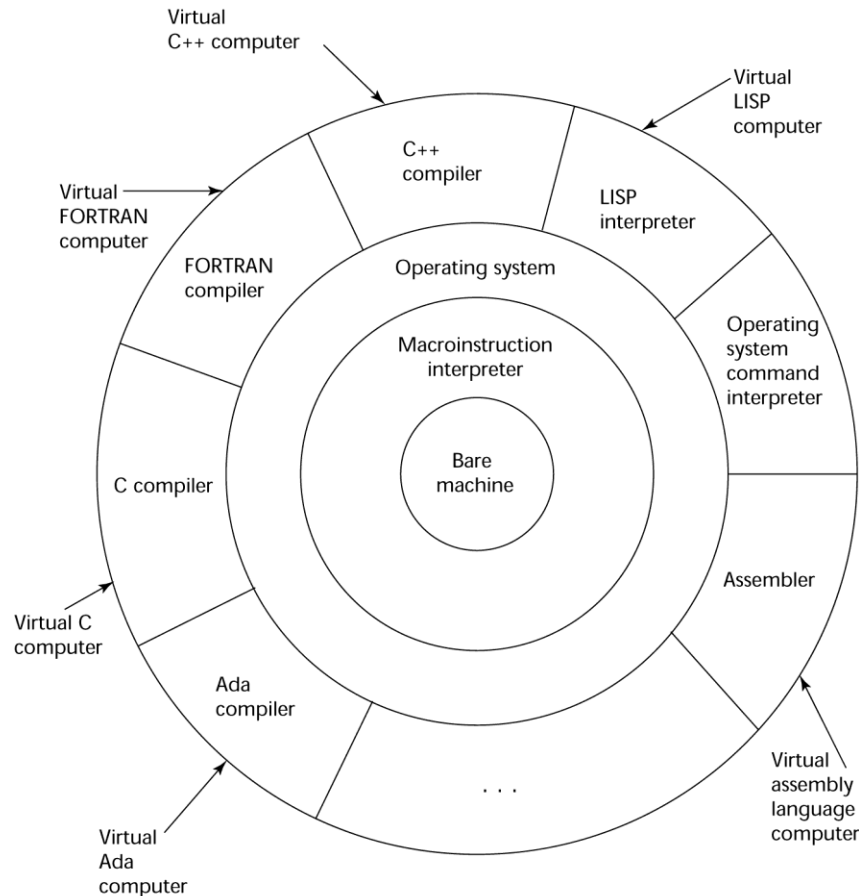
 decode the instruction

 execute the instruction

end repeat

Layered View of Computer

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



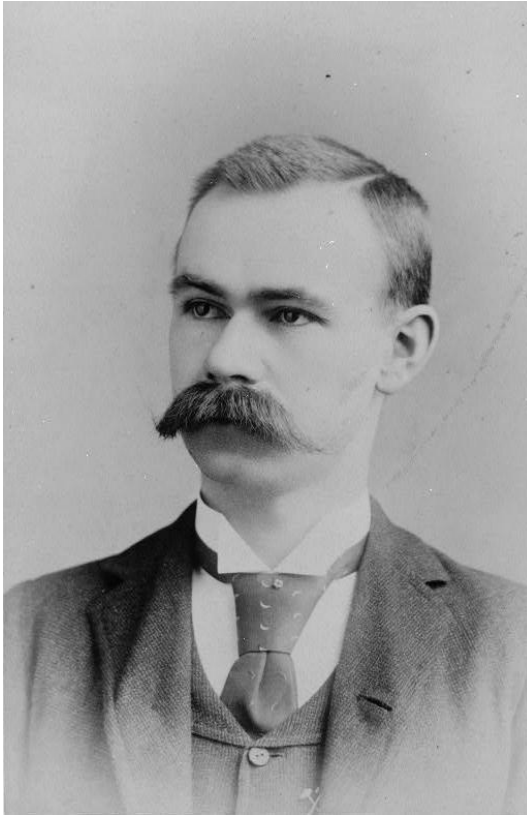
Four Paradigms of High-Level Programming Languages

- Imperative Language
- Functional Language
- Logical Language
- Object-Oriented Language

Evolution of Programming Language

- Machine Language
- Assembly Language
- High Level Language
 - ▣ BASIC, FORTRAN, COBOL, PASCAL, C, ...
- Query Language
 - ▣ Oracle, Informix,...
- Object–Oriented Programming (OOP)
 - ▣ Visual BASIC, Visual C++, Delphi, ...

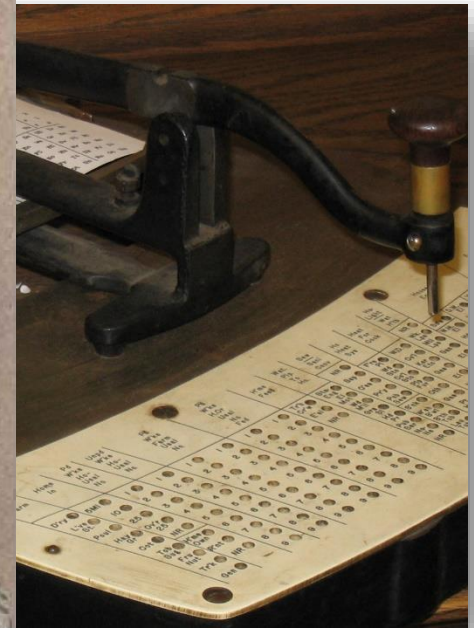
The 1st Program



Herman Hollerith



Ada Lovelace



Zuse's Plankalkül

- The world's **first complete high-level language**
- Designed in 1945, but not published until 1972
- Never implemented
- With **advanced data structures**
 - ▢ Integer, floating point, arrays, records
- Includes a **iterative statement** similar to Ada **for**
- Includes a **selection statement**, but does not allow **else** clause
- Includes **mathematical expressions** showing the current relationships between variables
- Syntax: e.x. assign the expression $A[4] + 1$ to $A[5]$



		$A + 1 \Rightarrow A$		
V		4	5	(subscripts)
S		1.n	1.n	(data types)

Pseudocodes

- Drawbacks of using machine code:
 - ❑ Expression coding is tedious
 - ❑ Poor readability
 - ❑ Use numerical codes for specifying instructions
 - ❑ Poor modifiability and error prone
 - ❑ Machine deficiencies--no indexing or floating point

Pseudocodes: Short Code

- Short Code was developed by John Mauchly in 1949 for BINAC computers, and was later transferred to UNIVAC I computer.
- Expressions were coded, left to right
- Example of operation codes:

01 -	06 abs value	1n (n+2)nd power
02)	07 +	2n (n+2)nd root
03 =	08 pause	4n if <= n
04 /	09 (58 print and tab

- Example: the statement $X0 = \text{SQRT}(\text{ABS}(Y0))$ would be coded in a word as 00 X0 03 20 06 Y0

Pseudocodes: Speedcoding

- Speedcoding was developed by John Backus in 1954 for IBM 701
- Includes pseudoinstructions for **arithmetic operations** (+ - x /) and **math functions** (root, sine, arc tangent, exponent, logarithm)
- Includes conditional/unconditional **branching**
- Includes input/output **conversions**
- Includes **auto-increment registers** for array access (matrix multiplication could be done in 12 Speedcoding instructions)
- Slow!
- Only 700 words were left for user program after loading the interpreter

Pseudocodes: Related Systems

- The UNIVAC Compiling Systems
 - ▣ Developed by a team led by Grace Hopper between 1951 and 1953
 - ▣ Expands Pseudocode into machine code
- David J. Wheeler (Cambridge University)
 - ▣ Developed a method of using blocks of re-locatable addresses to solve the problem of absolute addressing

IBM 704 and Fortran

- Fortran 0: 1954 – not implemented
 - ▣ Designed for the new IBM 704, which had index registers and floating point hardware
 - This led to the idea of compiled programming languages, because there was no place to hide the cost of interpretation (no floating-point software)
 - ▣ Environment of developing Fortran
 - Computers had small memories and unreliable
 - Applications were scientific computation
 - No efficient/effective programming methodology or tools
 - Computers are much more expensive than programmers

Fortran I Overview

- First implemented version of Fortran
 - ❑ Formatted I/O
 - ❑ Variable names could have up to six characters
 - ❑ User-defined subprograms, but no separate compilation
 - ❑ Post-test counting loop (**DO**)
 - ❑ Three-way selection statement (arithmetic **IF**)
 - ❑ No data typing statements
 - Variable name begins with I, J, K, L, M, and N are integer type, and all others are floating-point

Fortran I Overview (continued)

- First implemented version of FORTRAN
 - ▣ Compiler released in April 1957, after 18 worker-years of effort
 - ▣ Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of 704
 - ▣ Code was very fast
 - ▣ Quickly became widely used

Fortran II & IV

- Fortran II was distributed in 1958
 - Independent compilation
 - Fixed the bugs
- Fortran IV was evolved during 1960–62
 - Explicit type declarations
 - Logical selection statement
 - Subprogram names could be parameters
 - ANSI (American National Standards Institute) standard in 1966

Fortran 77 & 90

- Fortran 77 became the new standard in 1978
 - Character string handling
 - Logical loop control statement
 - **IF-THEN-ELSE** statement
- Fortran 90 had significant changes from Fortran 77
 - Modules
 - Dynamic arrays
 - Pointers
 - Recursion
 - **CASE** statement
 - Parameter type checking

Latest versions of Fortran

- Fortran 95 – relatively minor additions, plus some deletions
- Fortran 2003 – support for OOP, procedure pointers, interoperability with C
- Fortran 2008 – blocks for local scopes, co-arrays, `Do Concurrent`

Functional Programming: LISP

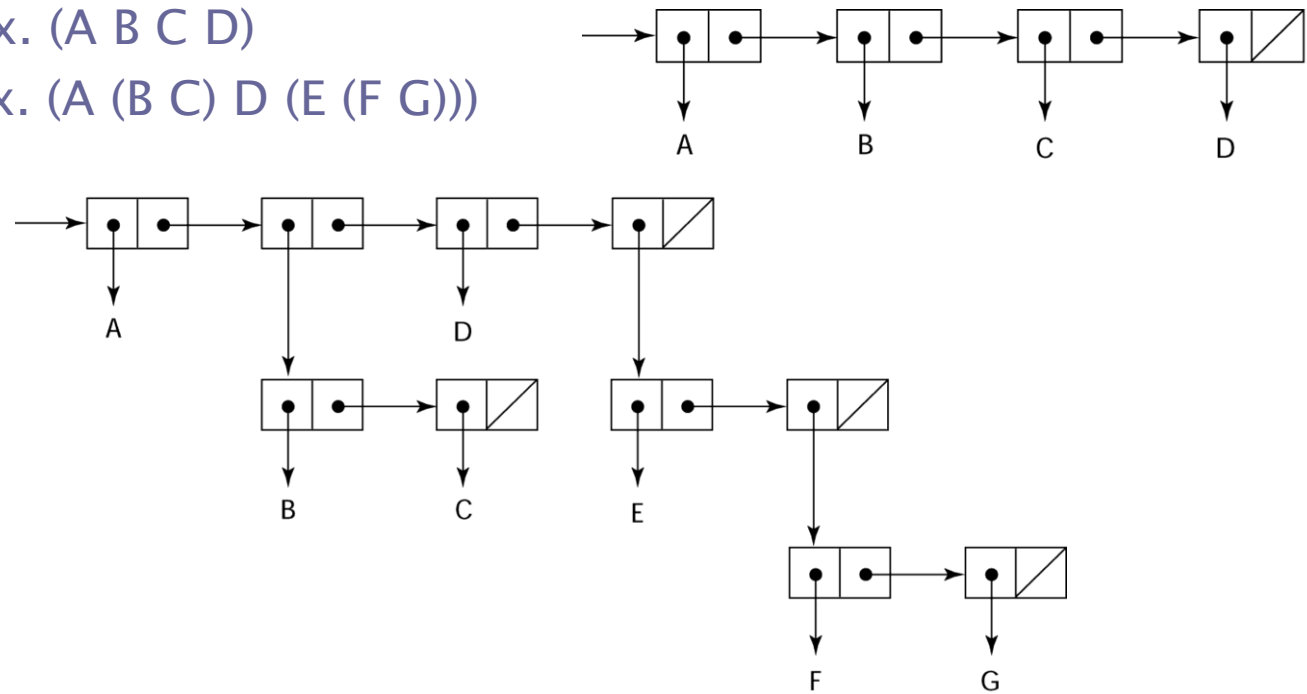
- The first functional programming language
- Designed by McCarthy, MIT, 1958
- List processing language:
 - ❑ Process symbolic data in linked lists rather than numeric data in arrays
 - ❑ LISP includes recursion, conditional expression, dynamic storage allocation
 - ❑ Applied in the area of Artificial Intelligence
 - Linguists: Natural language processing
 - Psychologists: Modeling human information storage and retrieval
 - Mathematicians: mechanizing certain intelligent process, e.g. theorem proving

Data Structure and Syntax of LISP

- Pure LISP only two data types
 - ▣ atoms : symbols or numerical literals
 - ▣ lists : specified by delimiting elements with parentheses

➤ ex. (A B C D)

➤ ex. (A (B C) D (E (F G)))



- LISP Syntax is based on *lambda calculus*

An Example of LISP program

```
( DEFUN equal_lists (lis1 lis2)
  ( COND
    ( (ATOM lis1) ( EQ lis1 lis2) )
    ( (ATOM lis2) NIL )
    ( (equal_lists (CAR lis1) (CAR lis2) )
      (equal_lists (CDR lis1) (CDR lis2) ) )
    ( T NIL )
  )
)
```

Two Descendants of LISP

- Scheme

- Developed at MIT in mid 1970s
- Small
- Extensive use of static scoping
- Treat functions as first-class entities
- Simple syntax (and small size) make it ideal for educational applications

- Common LISP

- An effort to combine features of several dialects of LISP into a single language
- Large, complex, used in industry for some large applications

The First Step Toward Sophistication: ALGOL 60

- Environment of development
 - ▣ FORTRAN had (barely) arrived for IBM 70x
 - ▣ Many other languages were being developed, all for specific machines
 - ▣ No portable language; all were machine-dependent
 - ▣ No universal language for communicating algorithms
- ALGOL 60 was the result of efforts to design a universal language

Early Design Process

- ACM and GAMM met for four days for design International **Algorithmic** Language (May 27 to June 1, 1958)
- Goals of the language
 - ▣ Close to mathematical notation and readable with little further explanation
 - ▣ Good for describing algorithms in printed publications
 - ▣ Must be translatable to machine code

ALGOL 58

- Concept of data type was formalized
- Variable names could be any length
- Allow any number of arrays dimensions
- Parameters were separated by mode (in & out)
- Subscripts were placed in brackets
- Compound statements (**begin ... end**)
- Semicolon as a statement separator
- Assignment operator was **:=**
- **if** had an **else-if** clause
- No I/O – “would make it machine dependent”

ALGOL 58 Implementation

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
- Although IBM was initially enthusiastic, all support was dropped by mid 1959

ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris, 1960
- New features
 - Block structure (local scope)
 - Two parameter passing methods
 - Pass by value and Pass by name
 - Subprogram recursion
 - Stack-dynamic arrays
 - Still no I/O and no string handling

ALGOL 60 Evaluation

- Successes

- ❑ The standard way to publish algorithms for over 20 years
- ❑ All subsequent imperative languages are based on it
- ❑ First machine-independent language
- ❑ First language whose syntax was formally defined (BNF, Backus–Naur form)

- Failure

- ❑ Never widely used, especially in U.S.
 - Lack of I/O and the character set made programs non-portable
 - Too flexible--hard to implement
 - Entrenchment of Fortran
 - Formal syntax description
 - Lack of support from IBM

Computerizing Business Records: COBOL

- Environment of development
 - ▣ UNIVAC was beginning to use FLOW-MATIC
 - ▣ U.S. Air Force was beginning to use AIMACO
 - ▣ IBM was developing COMTRAN
- Based on FLOW-MATIC
 - ▣ Names up to 12 characters, with embedded hyphens
 - ▣ English names for arithmetic operators (no arithmetic expressions)
 - ▣ Data and code were completely separate
 - ▣ The first word in every statement was a verb

COBOL Design Process

- First Design Meeting (Pentagon) – May 1959
- Design goals
 - ▣ Must look like simple English
 - ▣ Must be easy to use, even if that means it will be less powerful
 - ▣ Must broaden the base of computer users
 - ▣ Must not be biased by current compiler problems
- Design committee members were all from computer manufacturers and DoD branches
- Design Problems:
 - ▣ arithmetic expressions?
 - ▣ subscripts?
 - ▣ Fights among manufacturers

COBOL Evaluation

- Contributions
 - ▣ First macro facility in a high-level language
 - ▣ Hierarchical data structures (records)
 - ▣ Nested selection statements
 - ▣ Long names (up to 30 characters), with hyphens
 - ▣ Separate data division
- DoD Influence
 - ▣ First language required by DoD
 - ▣ would have failed without DoD
- Still the most widely used business applications language

The Beginning of Timesharing: BASIC

- Designed by Kemeny & Kurtz at Dartmouth College
- Design Goals:
 - Easy to learn and use for non-science students
 - Must be “pleasant and friendly”
 - Fast turnaround for homework
 - Free and private access
 - User time is more important than computer time
- Current popular dialect: Visual BASIC
- First widely used language with time sharing

Everything for Everybody: PL/I

- Designed by IBM and SHARE
- Computing situation in 1964 (IBM's point of view)
 - Scientific computing:
 - IBM 1620 and 7090 computers
 - FORTRAN
 - SHARE user group
 - Business computing
 - IBM 1401, 7080 computers
 - COBOL
 - GUIDE user group

PL/I: Background

- By 1963
 - Scientific users began to need more elaborate I/O, like COBOL had; business users began to need floating point and arrays for MIS
 - It looked like many shops would begin to need two kinds of computers, languages, and support staff--too costly
- The obvious solution
 - Build a new computer to do both kinds of applications
 - Design a new language to do both kinds of applications

PL/I: Design Process

- Designed in five months by the 3 X 3 Committee
 - ▣ Three members from IBM, three members from SHARE
- Initial concept
 - ▣ An extension of Fortran IV
- Initially called NPL (New Programming Language)
- Name changed to PL/I in 1965

PL/I: Evaluation

- PL/I contributions
 - ❑ First one that allows to create concurrently executing subprograms
 - ❑ Detect and handle 23 different types of exceptions or run-time errors
 - ❑ Switch-selectable recursion
 - ❑ First pointer data type
 - ❑ First one that allows cross-sections of array to be referenced.
- Concerns
 - ❑ Many new features were poorly designed
 - ❑ Too large and too complex

Two Early Dynamic Languages: APL and SNOBOL

- Characteristics shared by APL and SNOBOL:
 - Dynamic typing
 - Variables are untyped: A variable acquires a type when it is assigned a value
 - Dynamic storage allocation
 - Storage is allocated to a variable when it is assigned a value

APL: A Programming Language

- Designed as a **hardware description language** at IBM by Ken Iverson around 1960
- Includes many operators, for both scalars and arrays of various dimensions
 - Highly expressive
 - Programs are very difficult to read
- Still in use, but not widely

SNOBOL

- Designed as a **string manipulation language** at Bell Labs by Farber, Griswold, and Polensky in 1964
- Powerful operators for string pattern matching
- Slower than alternative languages (and thus no longer used for writing editors)
- Still used for certain text processing tasks

The Beginning of Data Abstraction: SIMULA 67

- Designed primarily for **system simulation** in Norway by Nygaard and Dahl
- Based on ALGOL 60 and SIMULA I
- Primary Contributions
 - Coroutines
 - Classes, objects, and inheritance

Orthogonal Design: ALGOL 68

- From the continued development of ALGOL 60 but not a superset of that language
- Source of several new ideas (even though the language itself never achieved widespread use)
- Design is based on the concept of **orthogonality**
 - A relative small set of primitive constructs can be combined in a relative small number of ways to build a large number of control and data structures of the language

ALGOL 68 Evaluation

- Contributions
 - ▣ User-defined data structures
 - ▣ Reference types
 - ▣ Dynamic arrays (called flex arrays)
- Comments
 - ▣ Less usage than ALGOL 60
 - ▣ Had strong influence on subsequent languages, especially Pascal, C, and Ada

Pascal – 1971

- Developed by Wirth (a former member of the ALGOL 68 committee)
- Designed for teaching structured programming
- Small, simple, nothing really new
- Largest impact was on teaching programming
 - From mid-1970s until the late 1990s, it was the most widely used language for teaching programming

Portable System Language : C

- Designed for **systems programming** (at Bell Labs by Dennis Richie, 1972)
- Evolved primarily from BCLP and B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX
- Though designed as a systems language, it has been used in many application areas

Programming Based on Logic: Prolog

- Developed, by Comerauer and Roussel (University of Aix–Marseille), with help from Kowalski (University of Edinburgh)
- Name Prolog based on *Programming Logic*
- Designed based on formal logic
- Non–procedural
- Can be summarized as being an intelligent database system that uses an inferencing process to infer the truth of given queries
- Comparatively inefficient
- Few application areas

Two kinds of Statements

- Facts

- mother(mary, jake)
- father(vern, mary)

- Rules

- grandparent(X,Z) :- parent(X, Y), parent(Y,Z)

History's Largest Design Effort: Ada

- Huge design effort, involving hundreds of people, much money, and about eight years
- Sequence of requirements (1975–1978)
 - ▣ Strawman, Woodman, Tinman, Ironman, Steelman
- High-level language for **embedded systems**
- Named Ada after Augusta Ada Byron, the first programmer

Ada Evaluation

- Contributions
 - ▣ Packages – support for data abstraction
 - ▣ Exception handling – elaborate
 - ▣ Generic program units
 - ▣ Concurrency – through the tasking model
- Comments
 - ▣ Competitive design
 - ▣ Included all that was then known about software engineering and language design
 - ▣ First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed

Ada 95

- Ada 95 (began in 1988)
 - ▣ Support for OOP through type derivation
 - ▣ Better control mechanisms for shared data
 - ▣ New concurrency features
 - ▣ More flexible libraries
- Ada 2005
 - ▣ Interfaces and synchronizing interfaces
- Popularity suffered because the DoD no longer requires its use but also because of popularity of C++

Object–Oriented Programming: Smalltalk

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg
- First full implementation of an object–oriented language
 - ▣ data abstraction, inheritance, and dynamic binding
- Pioneered the **graphical user interface** (GUI) design
- Promoted **OOP**

Combining Imperative and Object-Oriented Programming: C++

- Developed at Bell Labs by Stroustrup in 1980
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67
- A large and complex language, in part because it supports both procedural and OO programming
- Rapidly grew in popularity, along with OOP
- ANSI standard approved in November 1997
- Microsoft release .NET computing platform in 2002
 - Including a new version of C++: Managed C++ (MC++)
 - Properties, delegates, interfaces, a reference type for garbage-collected objects
 - Does not support multiple inheritance

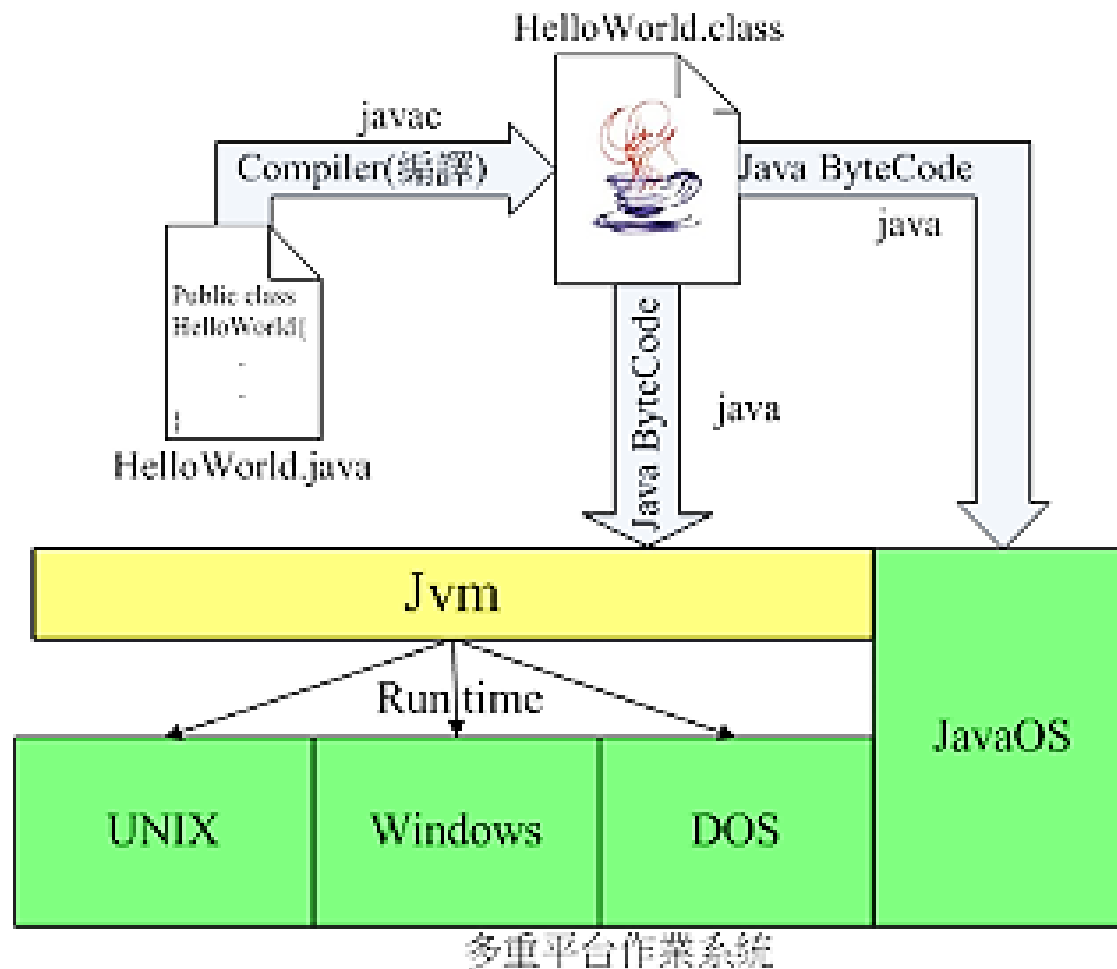
Related OOP Languages

- Objective-C (designed by Brad Cox – early 1980s)
 - C plus support for OOP based on Smalltalk
 - Uses Smalltalk's method calling syntax
 - Used by Apple for systems programs
- Delphi (Borland)
 - Pascal plus features to support OOP
 - More elegant and safer than C++
- Go (designed at Google – 2009)
 - Loosely based on C, but also quite different
 - Does not support traditional OOP

An Imperative–Based Object–Oriented Language: Java

- Developed at Sun in the early 1990s
 - C and C++ were not satisfactory for embedded electronic devices
- Based on C++
 - Significantly simplified (does not include `struct`, `union`, `enum`, pointer arithmetic, and half of the assignment coercions of C++)
 - Supports *only* OOP
 - Has references, but not pointers
 - Includes support for applets and a form of concurrency

How a JAVA Program Works?



Java Evaluation

- Eliminated many unsafe features of C++
- Supports concurrency
- Libraries for applets, GUIs, database access
- **Portable**: Java Virtual Machine concept, JIT compilers
- Widely used for Web programming
- Use increased faster than any previous language
- Most recent version, 7, released in 2011

Scripting Languages for the Web

- Perl
 - ❑ Designed by Larry Wall—first released in 1987
 - ❑ Variables are statically typed but implicitly declared
 - ❑ Three distinctive namespaces, denoted by the first character of a variable's name
 - ❑ Powerful, but somewhat dangerous
 - ❑ Gained widespread use for CGI programming on the Web
 - ❑ Also used for a replacement for UNIX system administration language
- JavaScript
 - ❑ Began at Netscape, but later became a joint venture of Netscape and Sun Microsystems
 - ❑ A client-side HTML-embedded scripting language, often used to create dynamic HTML documents
 - ❑ Purely interpreted
 - ❑ Related to Java only through similar syntax
- PHP
 - ❑ PHP: Hypertext Preprocessor, designed by Rasmus Lerdorf
 - ❑ A server-side HTML-embedded scripting language, often used for form processing and database access through the Web
 - ❑ Purely interpreted

Scripting Languages for the Web

- Python
 - ❑ An OO interpreted scripting language
 - ❑ Type checked but dynamically typed
 - ❑ Used for CGI programming and form processing
 - ❑ Dynamically typed, but type checked
 - ❑ Supports lists, tuples, and hashes
- Ruby
 - ❑ Designed in Japan by Yukihiro Matsumoto (a.k.a, “Matz”)
 - ❑ Began as a replacement for Perl and Python
 - ❑ A pure object-oriented scripting language
 - All data are objects
 - ❑ Most operators are implemented as methods, which can be redefined by user code
 - ❑ Purely interpreted

Scripting Languages for the Web

- Lua
 - ❑ An OO interpreted scripting language
 - ❑ Type checked but dynamically typed
 - ❑ Used for CGI programming and form processing
 - ❑ Dynamically typed, but type checked
 - ❑ Supports lists, tuples, and hashes, all with its single data structure, the table
 - ❑ Easily extendable

The Flagship .NET Language: C#

- Part of the .NET development platform (2000)
- Based on C++ , Java, and Delphi
- Includes pointers, delegates, properties, enumeration types, a limited kind of dynamic typing, and anonymous types
- Is evolving rapidly

Markup/Programming Hybrid Languages

- XSLT
 - eXtensible Markup Language (XML): a metamarkup language
 - eXtensible Stylesheet Language Transformation (XSTL) transforms XML documents for display
 - Programming constructs (e.g., looping)
- JSP
 - Java Server Pages: a collection of technologies to support dynamic Web documents
 - JSTL, a JSP library, includes programming constructs in the form of HTML elements

Summary

- Development, development environment, and evaluation of a number of important programming languages
- Perspective into current issues in language design

Turing Machine

