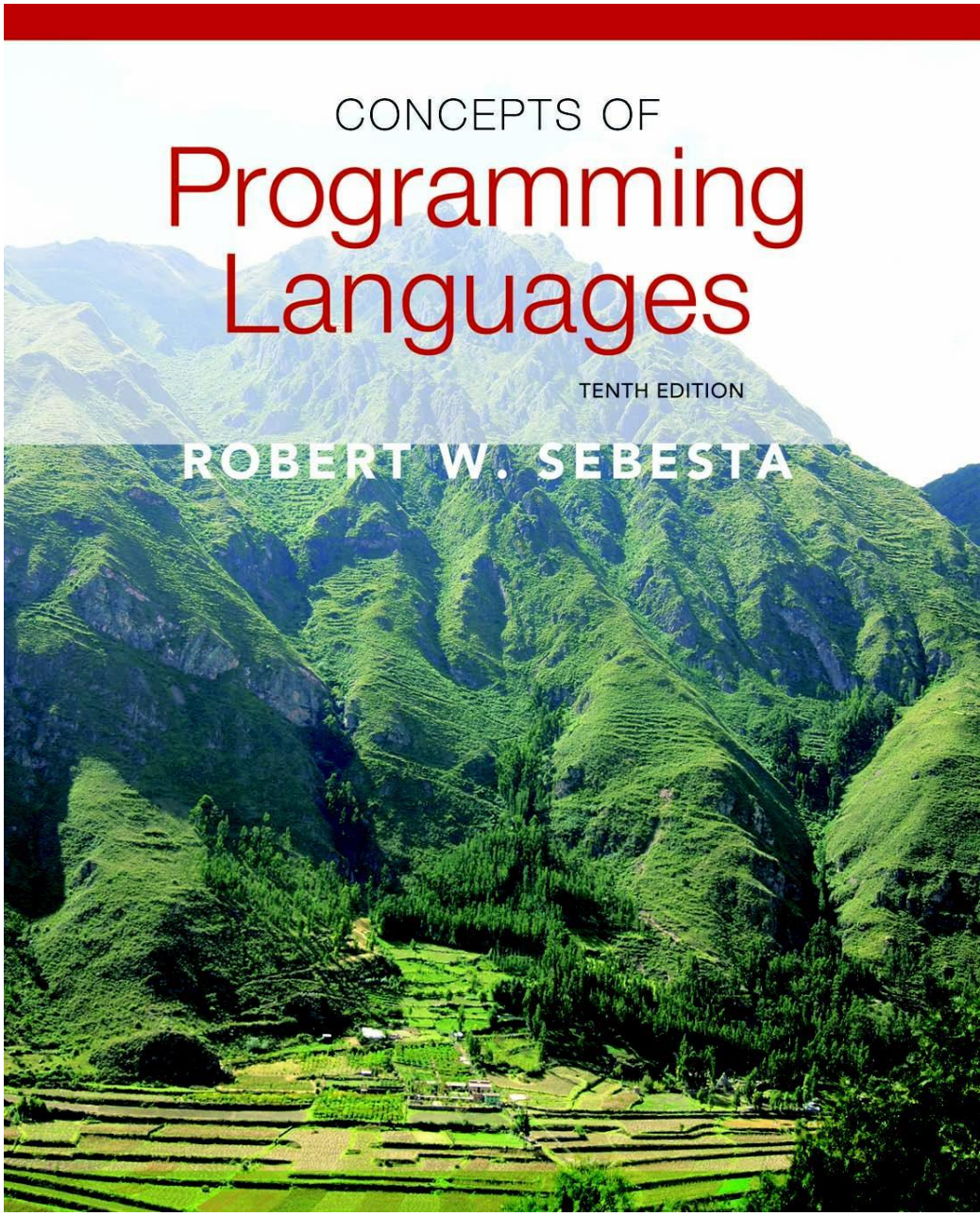
Chapter 2

Evolution of the Major Programming Languages



Chapter 2 Topics

- Zuse's Plankalkül
- Minimal Hardware Programming: Pseudocodes
- The IBM 704 and Fortran
- Functional Programming: LISP
- The First Step Toward Sophistication: ALGOL 60
- Computerizing Business Records: COBOL
- The Beginnings of Timesharing: BASIC

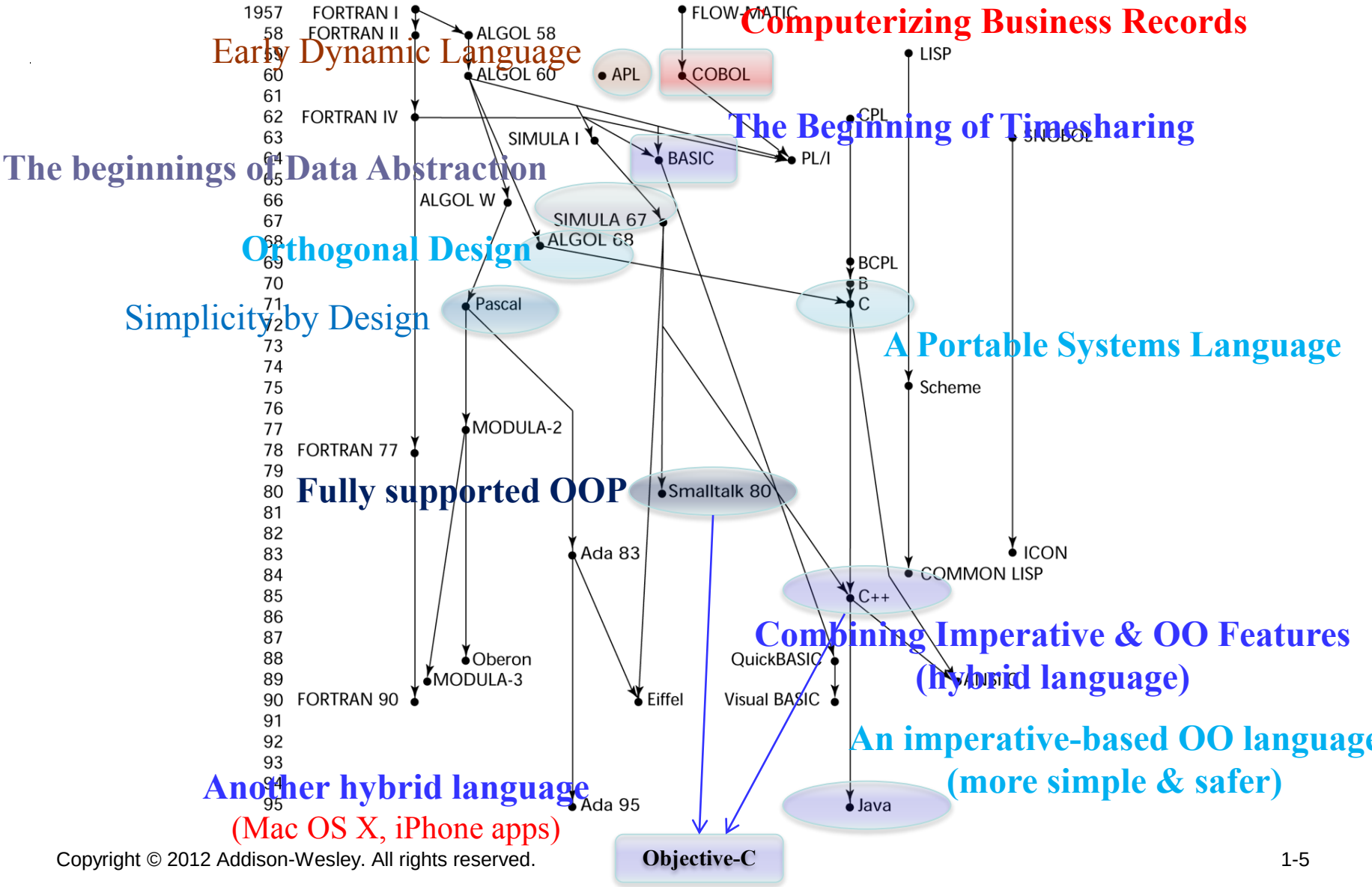
Chapter 2 Topics (continued)

- Everything for Everybody: PL/I
- Two Early Dynamic Languages: APL and SNOBOL
- The Beginnings of Data Abstraction: SIMULA 67
- Orthogonal Design: ALGOL 68
- Some Early Descendants of the ALGOLs
- Programming Based on Logic: Prolog
- History's Largest Design Effort: Ada

Chapter 2 Topics (continued)

- Object-Oriented Programming: Smalltalk
- Combining Imperative ad Object-Oriented Features: C++
- An Imperative-Based Object-Oriented Language: Java
- Scripting Languages
- The Flagship .NET Language: C#
- Markup/Programming Hybrid Languages

Genealogy of Common Languages



Gene ages

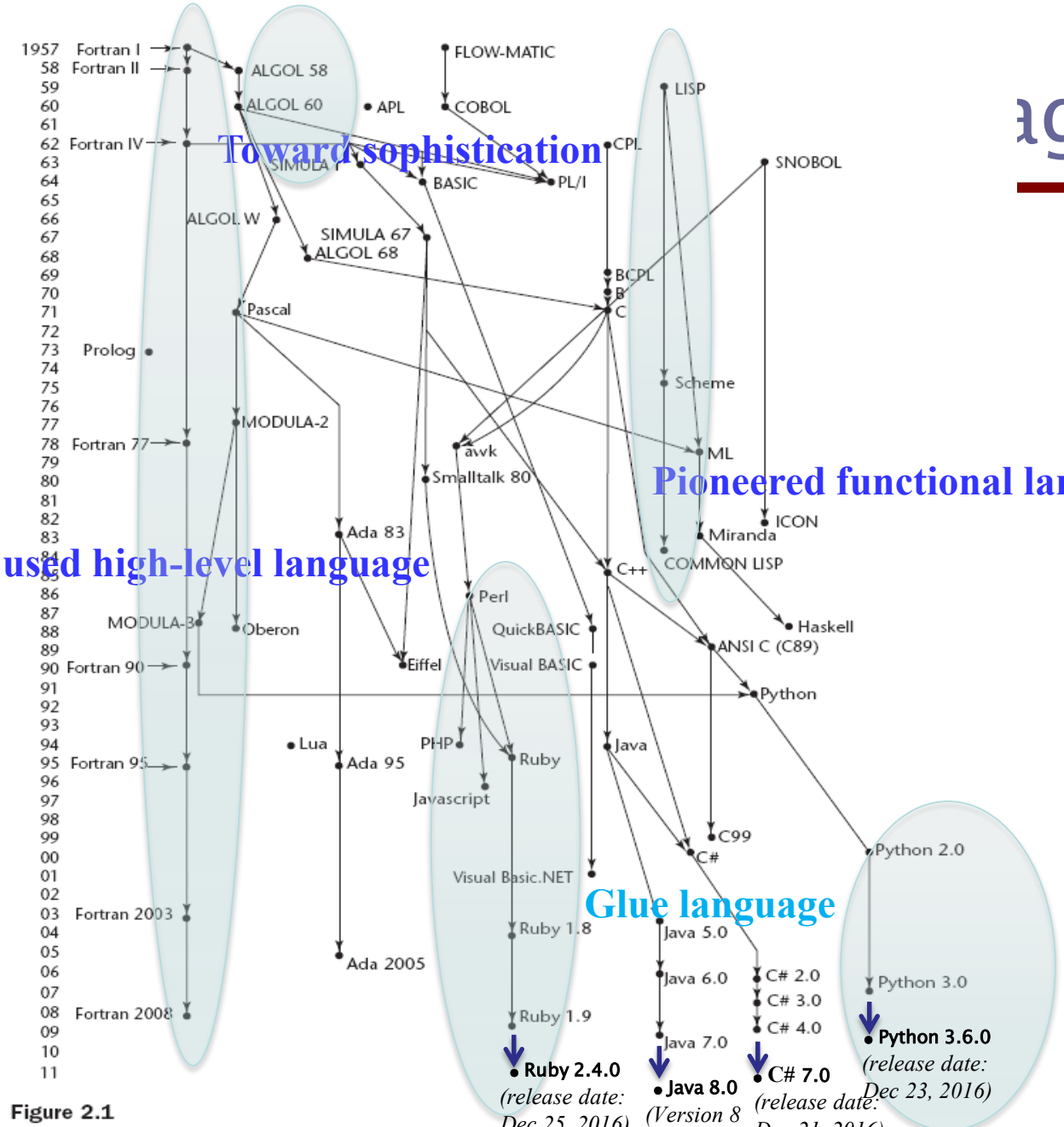


Figure 2.1

Genealogy of common high-level programming languages

Zuse's Plankalkül

- Means "*program calculus*"
- Designed in 1945, but not published until 1972
- Never implemented
- Advanced data structures
  - floating point, arrays, records
- Invariants
  - Invariant (computer science), an expression whose value doesn't change during program execution (ref. wikipedia)
  - In computer science, an **invariant** is a condition that can be relied upon to be true during execution of a program, or during some portion of it. (ref. wikipedia)

Plankalkül Syntax

- An assignment statement to assign the expression A[4] + 1 to A[5]

	|	A + 1 => A	
V	|	4 5	(subscripts)
S	|	1.n 1.n	(data types)

Minimal Hardware Programming: Pseudocodes

- What was wrong with using machine code?
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Machine deficiencies--no indexing or floating point
    - Neither of these capacities, however, was included in the architecture of the computers of the late 1940s and early 1950s
    - Led to the development of somewhat higher-level lanaguage

Pseudocodes: Short Code

- Short Code developed by Mauchly in 1949 for BINAC(**BIN**ary **A**utomatic **C**omputer) computers
  - Not translated into machine code, Implemented by **the pure interpretation method** ***(50 times slower than machine code)***
  - words of **UNIVAC I's memory** (12 six-bit bytes = 72 bites) (1byte=6 bits, a word=12 bytes)
  - Expressions were coded, left to right
  - **(The statement)**

X0 = SQRT(ABS(Y0))

(1 word) : 00 X0 03 20 06 Y0 (combination of bytes)

- initial 00 : used as padding to fill the word
- 03 : =
- 20 : SQRT
- 06 : ABS
- X0, Y0 : variable

Pseudocodes: Speedcoding

- Speedcoding developed by Backus in 1954 for IBM 701
  - **Interpretive system** that extended machine language to include floating-point operations
    - Pseudo ops for arithmetic and math functions (by the interpretation methods)
  - Conditional and unconditional branching
  - Auto-increment registers for array access
  - Only 700 words left for user program
  - Slow (Machine language programming – heavy cost)
    - **Improved productivity of the programmer (by speedcoding programming)**

Pseudocodes: Related Systems

- The UNIVAC Compiling System (1951~53)
  - Developed by a team led by Grace Hopper
  - Pseudocode expanded into machine code
    - Still quite primitive
    - Although even this was a great improvement
      - It made source programs **much shorter**

IBM 704 and Fortran

- FORTRAN (1954)
  - The IBM Mathematical **FOR**mula **TRAN**slating system
  - Not implemented
- Fortran I (1957)
  - 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**

Design process

- Environment of development
  - Computers were small and unreliable
  - Applications were scientific
  - No programming methodology or tools
  - Machine efficiency was the most important concern
    - Because of the high cost of computers compared to the cost of programmers, **speed of the generated object code** was the primary goal of the fist Fortran compilers

컴퓨터 비용이 프로그래머의 비용보다 컸으므로
효율적인 목적코드가 중요시됨

Fortran I Overview

- First implemented version of Fortran
  - Formatted I/O (형식화된 입출력)
  - Variable names could have up to six characters
  - No data typing statements
    :I, J, K, L, M, and N (implicitly integer type)
    :others (implicitly floating-point type)
  - User-defined subprograms
  - Three-way selection statement (arithmetic **IF**)
  - Post-test counting loop (**DO**)

Ref) Selection statements(one-way, two-way, n-way)

IF selection statement: IF (expr) N1, N2, N3
expr < 0 ➔ GOTO N1
expr = 0 ➔ GOTO N2
expr > 0 ➔ GOTO N3

Fortran I Overview (continued)

- First implemented version of FORTRAN
  - **No separate compilation (subprograms)**
  - Compiler released in April 1957, after 18 worker-years of effort (Fortran 0의 수정 구현)
  - Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of 704
  - Code was very fast
  - Quickly became widely used

Early success: IBM 704를 위한 코드의 절반이상이 Fortran I으로 대체됨

Fortran II & III

- Fortran II
  - Distributed in 1958
    - Fixed the bugs that was included in Fortran I
    - **Independent compilation of subprograms**

Including precompiled machine language versions of subprograms shortened the compilation process

실질적인 대규모 프로그래밍이 가능하게 됨

- Fotran III
  - It was developed, but it was never widely distributed

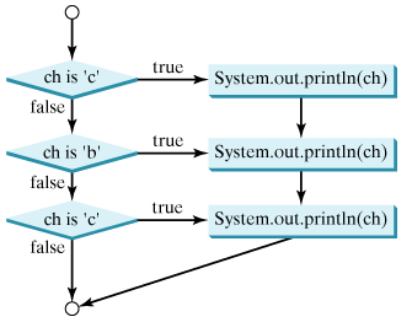
Fortran IV & 66

- Evolved during 1960-62
  - Standard of Fortran (1962~1978)
  - Improvement over Fortran II
    - Explicit type declarations
    - Logical selection statement
    - **Subprogram names could be parameters**

- When there's a need for explicit type system:

```
let x:int = 0
let piConstant:float = 3.141
let Name:String = 'Eriawan'
```

- Fortran 66
  - ANSI standard in 1966
    (Fortran 66, 이름은 거의 사용되지 않음)



Fortran 77

- Became the new standard in 1978
- Features added to the Fotran IV
  - Character string handling
  - Logical loop control statement
  - **IF-THEN-ELSE** statement

4. Character string functions:

The functions below perform operations from and to character strings. Please note that ACHAR works with the standard ASCII character set while CHAR works with the representation in the computer you are using.

```
ACHAR(I)          Returns the ASCII character which has number I
ADJUSTL(STRING)   Adjusts to the left
ADJUSTR(STRING)   Adjusts to the right
CHAR(I, kind)     Returns the character that has the number I
IACHAR(C)         Returns the ASCII number of the character C
ICHAR(C)          Returns the number of character C

INDEX(STRING, SUBSTRING, back)  Returns the starting position for a
    substring within a string.  If BACK is true then you get the
    last starting position, in the other case, the first one.

LEN_TRIM(STRING)  Returns the length of the string without the possibly
    trailing blanks.

      LGE(STRING_A, STRING_B)
      LGT(STRING-A, STRING_B)
      LLE(STRING_A, STRING_B)
      LLT(STRING_A, STRING_B)
```

Looping Control Structures

Control structures alter the normal sequential flow of a statement execution. Loops allow the a block of statements to be executed repeatedly without actually writing them down numerous times.

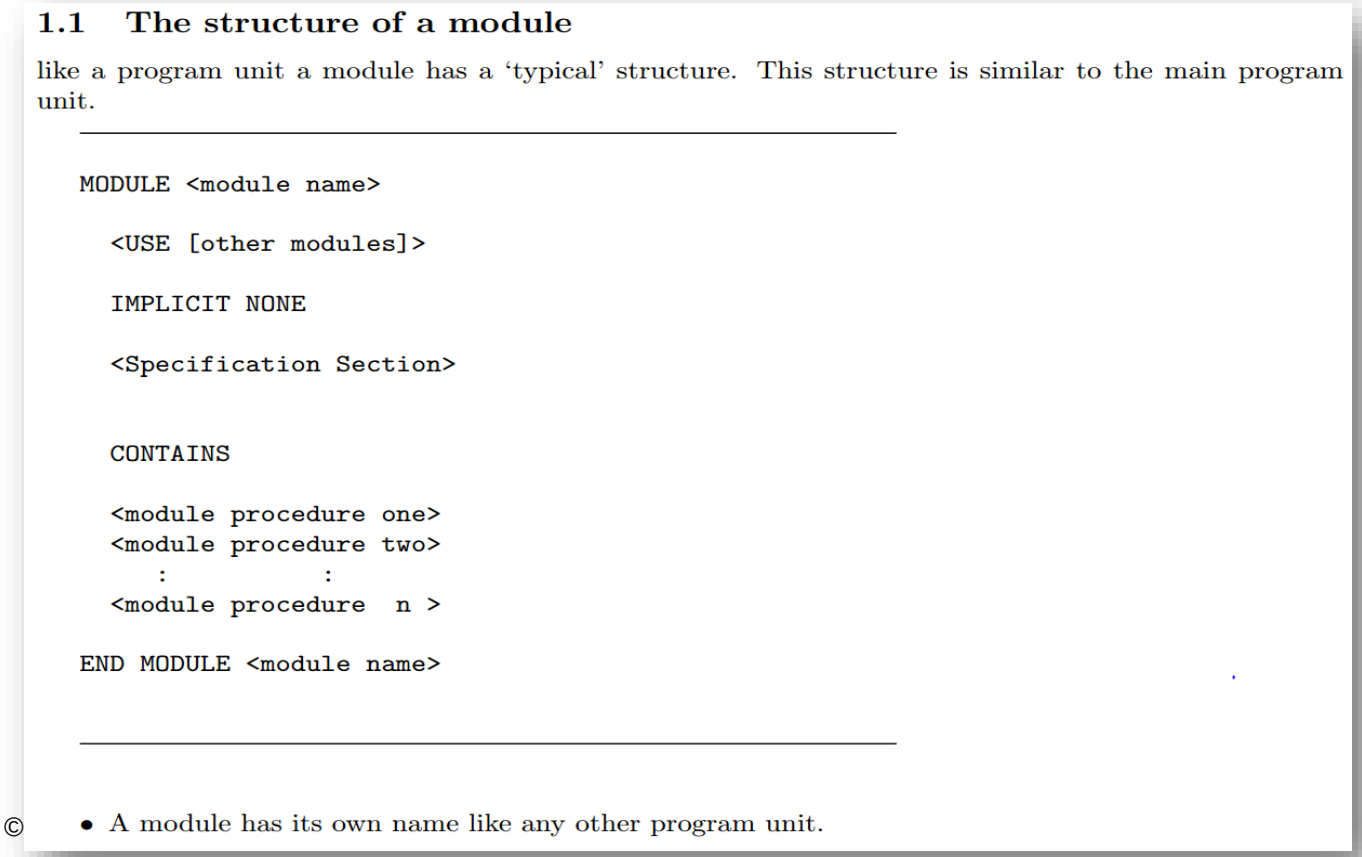
DO Loop

A DO loop allows a block of statements to be executed repeatedly.

```
        DO label, loop-control-variable = initial-value, final-value, step-size
           statement1
           statement2
...
           statementn
label   CONTINUE
```

Fortran 90

- Most significant changes from Fortran 77
  - **Modules,** Dynamic arrays, Pointers, Recursion
  - **`CASE`** statement, **Parameter type checking**



1.1 The structure of a module

like a program unit a module has a 'typical' structure. This structure is similar to the main program unit.

```
MODULE <module name>

  <USE [other modules]>

  IMPLICIT NONE

  <Specification Section>


  CONTAINS

  <module procedure one>
  <module procedure two>
     :           :
  <module procedure  n >

END MODULE <module name>
```

- A module has its own name like any other program unit.

Latest versions of Fortran

- Fortran 95 – relatively minor additions, plus some deletions
  - New iteration construct (**Forall**)
    (the task of parallelizing Fotran programs)
- Fortran 2003 – support for OOP, procedure pointers, interoperability with C
- Fortran 2008 – blocks for local scopes, co-arrays, Do Concurrent (to specify loops without interdependencies)

Features in Versions

Version

FORTRAN 77 : Character-based data , Array Programming , Modular programming

FORTRAN 90 : Generic Programming – Specific Type

FORTRAN 95 : High performance Fortran – Parallel Computing

FORTRAN 2003 : Object-Oriented Programming

FORTRAN 2008 : Concurrent Programming – Thread Base

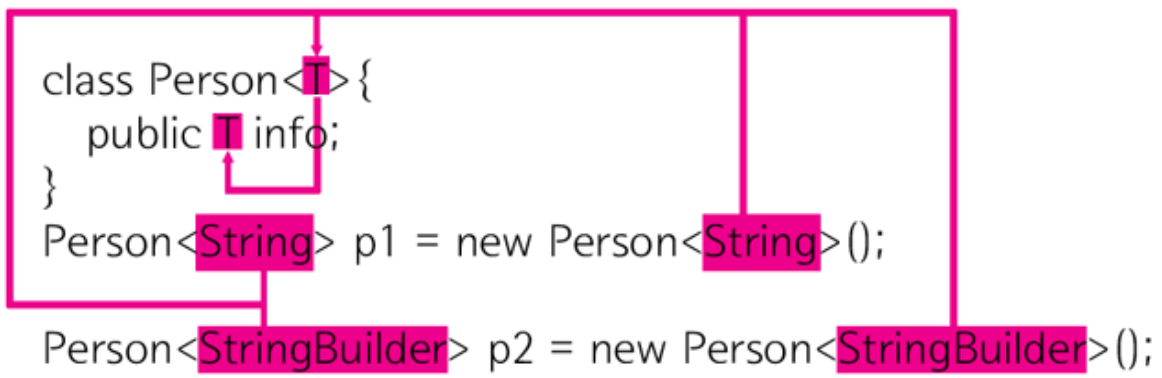
In the simplest definition, **gen** programming in which algorit *specified-later* that are then *in* provided as parameters

Ref.) http://en.wikipedia.org/wiki/Ge

(참고자료)

제네릭(Generic)은 클래스 내부에서 사용할 데이터 타입을 외부에서 지정하는 기법을 의미한다. 말이 어렵다. 아래 그림을 보자.

- Generic Types in Java -



Fortran Evaluation (google trends – Fortran)

- Highly optimizing compilers (all versions before 90)
  - Simplicity & efficiency
  - Types and storage of all variables are fixed before run time (**in compile time**)
  - Recursive programs (X)
    - No new variables or space could be allocated during execution time (can not use the dynamic allocation such as – `alloc(), calloc(), free(), realloc() in C`)

- Dramatically changed the way computers are used
  - **Interpretation → compilation**
  - The first widely used high-level language
  - Early versions of Fortran (various problems)
  - Huge investment in Fortran software

Functional Programming: LISP

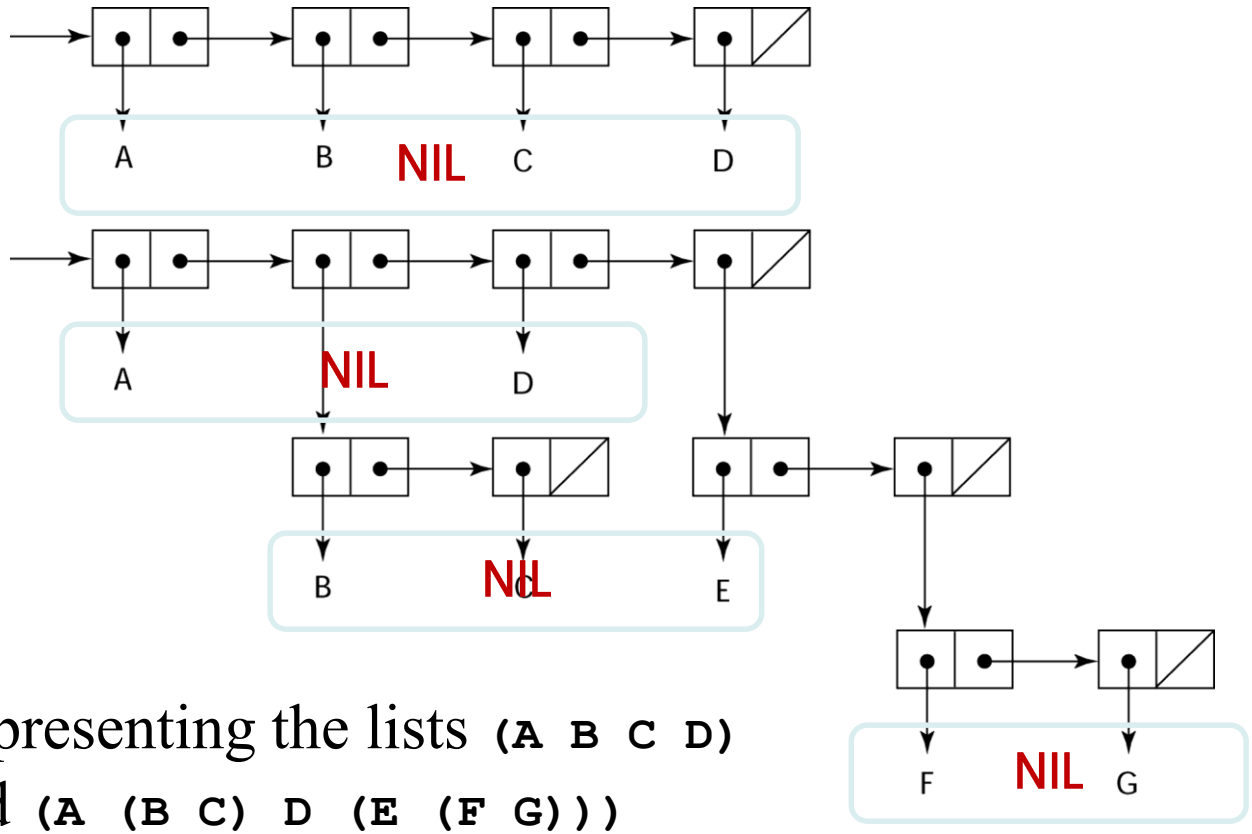
- LIST Processing language (Designed at MIT by McCarthy)
- **AI** research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)

1950년대 중반 인공지능에 관한 관심 태동

- 언어학자: 자연언어처리
- 심리학자: 인간의 정보처리능력의 형상화
- 수학자: 정리의 증명
- ➔ 기호적인 데이터를 연결 리스트로 처리할 수 있는 능력을 포함하는 언어의 필요성 대두

  - Only two data types: atoms and lists
  - Syntax is based on *lambda calculus*

Representation of Two LISP Lists



Representing the lists `(A B C D)` and `(A (B C) D (E (F G)))`

Internally, lists are stored as single-linked list structures

LISP Evaluation

- Pioneered functional programming
  - No need for variables or assignment
  - Control via recursion and conditional expressions
- Still the dominant language for AI
- COMMON LISP and Scheme are contemporary dialects of LISP
- ML, Haskell, and F# are also functional programming languages, but use very different syntax

Scheme & COMMON LISP

(Descendants of LISP, Dialect of the LISP)

- **Scheme**
  - Developed at MIT in mid 1970s
  - **Small**
  - Extensive use of static scoping
  - 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**

Scheme & COMMON LISP

(Descendants of LISP, Dialect of the LISP)

Scheme

- 1970년 중반에 MIT에서 개발
- 비교적 적은 규모의 lisp 구현
- 교육분야에 적합

COMMON LISP

- 1990년대 중반에 개발(Graham, 1996)
- 여러 변형의 lisp을 통합
- 대규모의 복잡한 응용 프로그램을 사용하는 산업용으로 적합

The First Step Toward Sophistication: ALGOL 60

- **Environment of development**
  - FORTRAN had (barely) arrived for IBM 70x
  - Many other languages were being de
    machines
  - No portable language; all were machine
  - No universal language for communicating algorithms

ALGOL 60 > FORTRAN

- **ALGOL 60** was the result of efforts to design **a univ**
  and machine-independent algorithmic language for
  scientific applications
  - **Fortran was solely owned by IBM**

Early Design Process

- ACM(Americans) and GAMM(Europeans) met for four days for design (May 27 to June 1, 1958)

- **Goals of the language**
  - Close to **mathematical notation (easy readability)**
  - Good for **describing algorithms**
  - Must be **translatable to machine code**

ALGOL 58

- Concept of type was formalized
- Names could be any length
- Arrays could have any number of subscripts
- 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
  - IBM 70X 계열의 machine에 대한 개발 언어로써 사용자의 ALGOL 교육 및 구현에 드는 개발비용이 많이 듦 (ALGOL 58을 포기하고 Fortran을 그대로 유지하게 된 배경)

ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris
- New features
  - Block structure (local & global scope)
  - Two parameter passing methods
    (pass by value, pass by name) – (details in Ch.9)
  - Subprogram recursion
  - Stack-dynamic arrays
    - The subscript range or ranges are specified variables
    - 첨자범위 동적 바인딩 및 동적 메모리 할당 (details in ch.6)
  - Still no I/O and no string handling

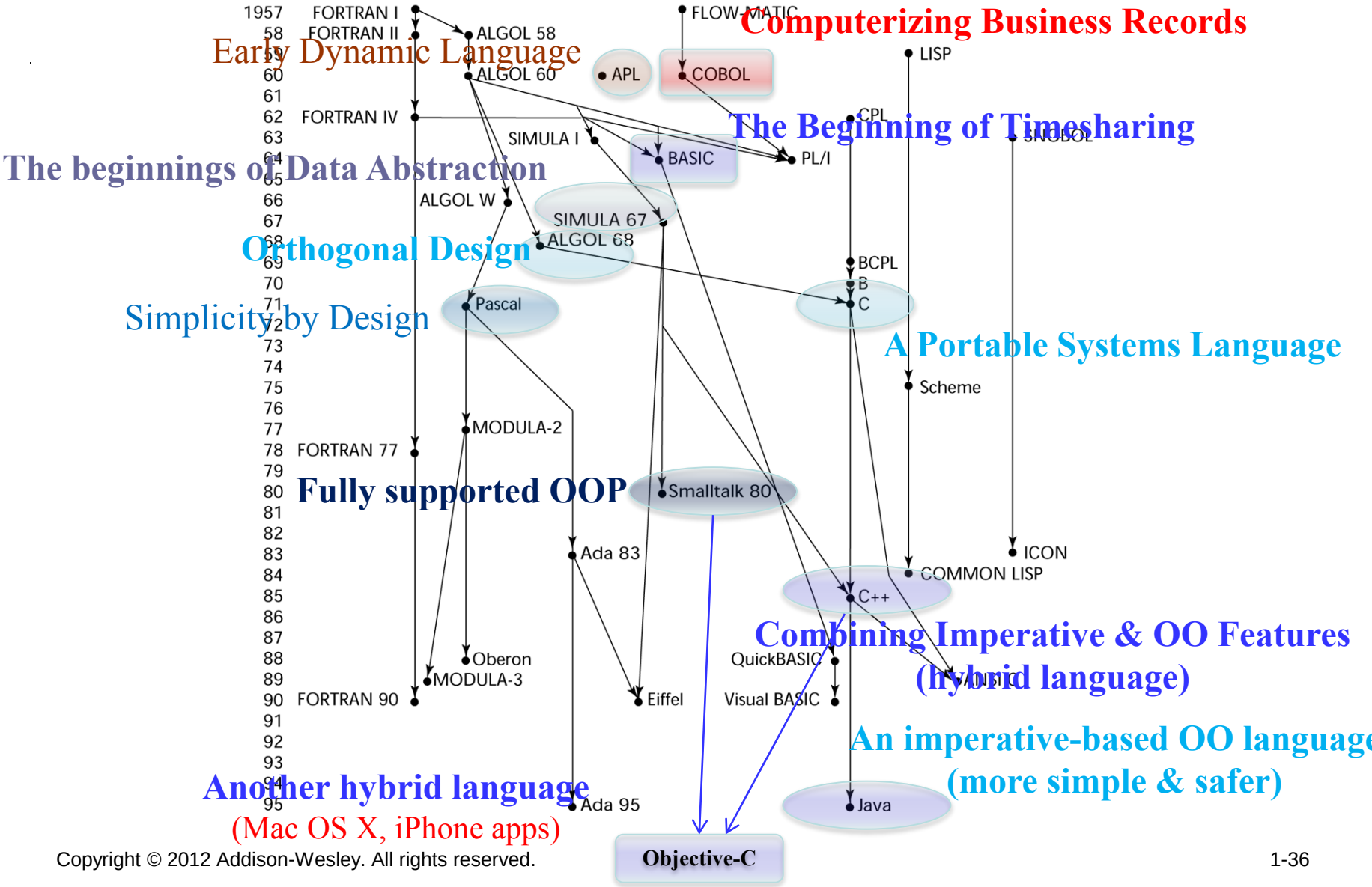
ALGOL 60 Evaluation

- Successes
  - It was ***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)
    - Context-Free-Grammar (CFG)의 유사형태 *(details in ch. 3)*

ALGOL 60 Evaluation (continued)

- Failure
  - Never widely used, especially in U.S.
  - Reasons
    - Lack of I/O and the character set made programs non-portable
    - Too flexible--hard to implement
      (difficult to understand → implementation inefficient)
    - Formal syntax description
      (BNF-당시에는 난해하게 보임)
    - Entrenchment of Fortran
    - Lack of support from IBM

Genealogy of Common Languages



Computerizing Business Records: COBOL

- Environment of development
  - UNIVAC was beginning to use FLOW-MATIC
  - USAF was beginning to use AIMACO
  - IBM was developing COMTRAN

여러 언어들의 설계 프로젝트가 계획되고 있었음

COBOL Historical Background

- Based on FLOW-MATIC
- FLOW-MATIC features
  - 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? (불편?) subscripts? (복잡?)
  - 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**

COBOL: 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
- 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

  } *time shared concept*

- Current popular dialect: Visual BASIC
- First widely used language with **time sharing**

With individual access through terminals by numerous simultaneous users

2.8 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 (Management Information Systems)
  - 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
  (large number of fixed structures)
  - First unit-level concurrency
  - First exception handling
  - Switch-selectable recursion
  - First pointer data type
  - First array cross sections
- Concerns
  - Many new features were poorly designed

많은 구조들의 성공적인 결합에 실패
70년대 사무 및 과학 분야에 부분적으로 많이 사용됨

Two Early Dynamic Languages: APL and SNOBOL

- Characterized by **dynamic typing** and **dynamic storage allocation (공통점)**
  - **Variables are untyped** : A variable acquires a type when it is assigned a value
  - 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
  - Highly expressive (many operators, for both scalars and arrays of various dimensions)
  - Many powerful operators, but poor readability

- Still in use; minimal changes
  - "Throw-away" programming (일회성)
  - Discarded after use because programs are difficult to maintain

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)
- A supported language that 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
- Extension of ALGOL 60
  - Taking both block structure and the control statements from ALGOL60
- Primary Contributions
  - **Coroutines** – a kind of subprogram
  - **Classes, objects, and inheritance** for supporting coroutines
  - **Data abstraction** provides the foundation for **objected-oriented programming** (OOP)

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 few basic concepts, plus a few combining mechanisms
  - A few primitive types and structures and allow the user to combine those primitives into a large number of different structures (**user-defined data structures**)

ALGOL 68 Evaluation (**Orthogonal design**)

- Contributions
  - **User-defined data structures**
    - By using a few primitives which compose of basic data types and structures
    - Allow the user to design data abstractions that fit particular problems very closely
  - **Dynamic arrays** (called flex arrays)
    - Not specify subscript bounds
    - Assignments to a dynamic array cause allocation of required storage (기억장소 할당: 값이 **assign** 될 때)
- Comments
  - Less usage than ALGOL 60 (more complicated)
  - Had strong influence on subsequent languages, especially Pascal, C, and Ada

Simplicity of Design: Pascal

- Developed by Wirth (a former member of the ALGOL 68 committee) – 1971

- Designed for teaching structured programming
  - Pascal lacks several features that are essential for many kinds of applications
    - The impossibility of writing a subprogram that takes as a parameter an array of variable length (가변길이 배열 매개변수)
    - No independent compilation of subprograms (터보 Pascal 파생 원인)

- Largest impact was on teaching programming
  - From mid-1970s until the late 1990s, it was the most widely used language for teaching programming

A Portable Systems Language: C

- Designed for systems programming (at Bell Labs by Dennis Richie) – 1972
- Evolved primarily from BCLP and B, but also ALGOL 68
  - BCLP & B : No type, only machine word
  - Complicated and instable
- Powerful set of operators, but poor type checking
  (C99 이전 parameter type checking 없이 함수 작성)
- Initially spread through **UNIX**
- Although originally designed as a systems programming, it has been used in many application areas (portability)
- The first standard for C was published by ANSI (1989)
  - often referred to as "ANSI C" or "C89
  - Current standard: C89 → C99 → C11(ISO/IEC 9899:2011)

Bell Labs (AT &T 산하 연구소)
- 1925년: 최초의 팩스 기능 시연
- 1927년: 장거리 TV 데이터 전송 기능
- 1947년: 트랜지스터 개발
- 1969년: 유닉스 운영체제 개발
- 1978년: 이동 전화 기술 개발

Programming Based on Logic: Prolog

- Developed, by Comerauer and Roussel (University of Aix-Marseille), with help from Kowalski (University of Edinburgh)
- 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
  (An example of fact statements and a rule statement – p.100)
- Comparatively inefficient
- Few application areas

History's Largest Design Effort: Ada

- Developed for the DoD (Department of Defense)
  - Over half of the applications of computers in DoD were embedded systems
  - Increasing complexity of systems → no standardization
- Huge design effort, involving hundreds of people, much money, and about eight years
- Sequence of requirements (1975–1978)
  - (Strawman, Woodman, Tinman, Ironman, Steelman)
- Named Ada after Augusta Ada Byron, the first programmer (1980)

Ada Evaluation

- Contributions
  - Packages – support for data abstraction
  - Exception handling – elaborate
  - Generic program units (알고리즘의 일반화)
    - 같은 알고리즘 기반의 유사한 구현에서 공통점을 찾음
  - Concurrency – through the tasking model

- Comments
  - Competitive design (No limits on participation)
  - 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 (inheritance, polymorphism)
  - 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 design (Window systems)
- Promoted OOP

Combining Imperative and Object-Oriented Programming: C++

- Developed at Bell Labs by Stroustrup in 1980
- Evolved from C and SIMULA 67 (Goals of C++)
  - Classes & inheritance
    (facilities of OOP taken partially from SIMULA 67)
  - No performance penalty relative to C
  - Could be used for every application for which C could be used
- 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's version: MC++
  - An extension of the C++ programming language designed for concurrent programming

Related OOP Languages

- Objective-C (by Brad Cox–early 1980s), Hybrid Language
  - Based on Smalltalk & C++
  - Initially, C plus the classes and message passing of Smalltalk
  - Uses Smalltalk syntax to support hybrid features
  - Used by Apple for systems programs (MAC OS X & iPhone SW)

- Delphi (Borland), Hybrid Language **(fast implementation!)**
  - **Pascal** plus features to support **OOP**
  - More elegant and safer than C++
    - No array subscript range checking **(in C)**
    - user-defined operator overloading**(in C++)**

(Not allowed)

- 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
  - **Reliability** is an important characteristic of the software in consumer electronic products (many products recall → significant cost)
  - C and C++ were not satisfactory for embedded electronic devices
    - C – relatively small, but not supporting OOP
    - C++ – supporting OOP, but too large & complex
  - Merged into Oracle Corporation(Jan 27, 2010)
    - Free download(**OTN**-Oracle Technology Network)

An Imperative-Based Object-Oriented Language: Java (continued)

- Based on C++
  - **more simple and reliable than 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
    - Java applets are relatively small Java programs
    - **Synchronized** modifier (**thread locking** for shared resources)

Java Evaluation

- Eliminated many unsafe features of C++
  - int → float (o), float → int (x) : assignment coercions
  - Index range checking of array accesses
- Supports concurrency (**synchronized**)
- Libraries for applets, GUIs, database access
- **Portable**: Java Virtual Machine concept, **JIT compilers**
  - At least 10 times slower than equivalent compiled C programs (initial java interpreter – JVM)
  - By using JIT compilers, Java programs are translated to **machine code before being executed**
- Widely used for Web programming (applet)
- Use increased faster than any previous language
- Most recent version, 8, released in March18, 2014 (OTN)

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 (타입의 강제변환에 따른 오류)
  - 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(Live sciript → Javascript, Javascript v1.1 java 연동 지원)
  - 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

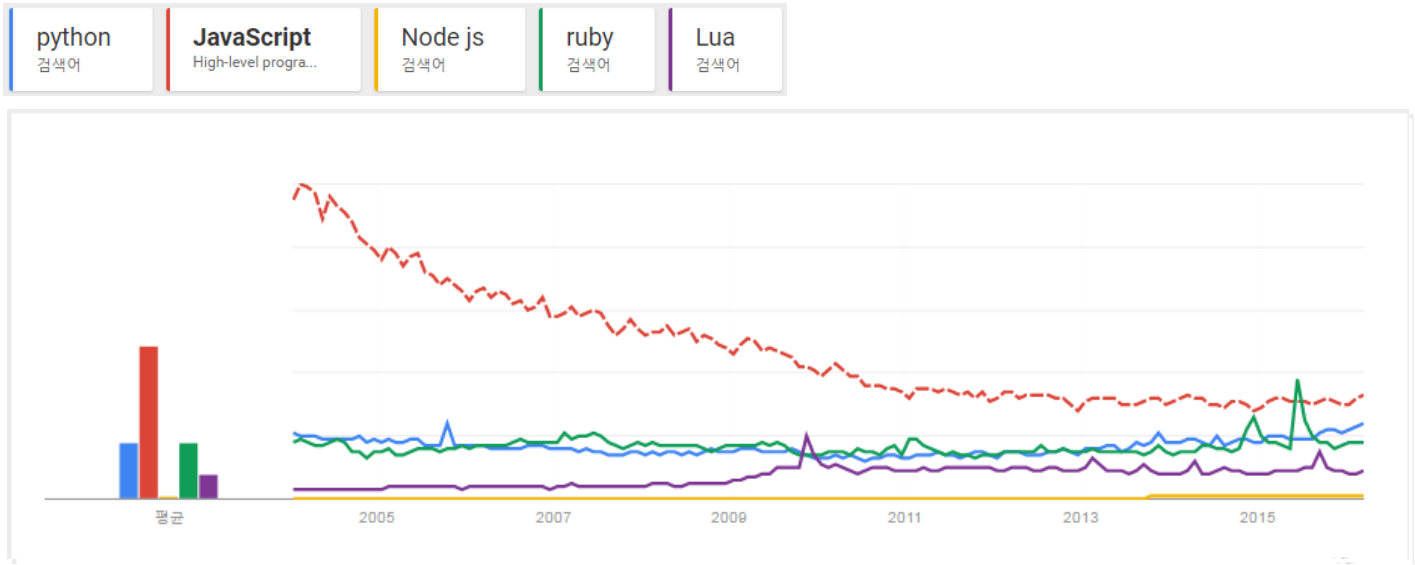
Scripting Languages for the Web

- **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
  - Cphyton, Stackless Phython, Jython, IronPython(.Net)
- **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
  - 공식 구현
    - MRI 루비 인터프리터 → (YARV: Yet Another Ruby VM) Bytecode Interpreter (속도 개선)
  - 그외
    - Jruby (JVM), IronRoby (.NET), MacRuby(Mac OS X)
    - Rubinius : bytecode interpreter based on JIT compiling method

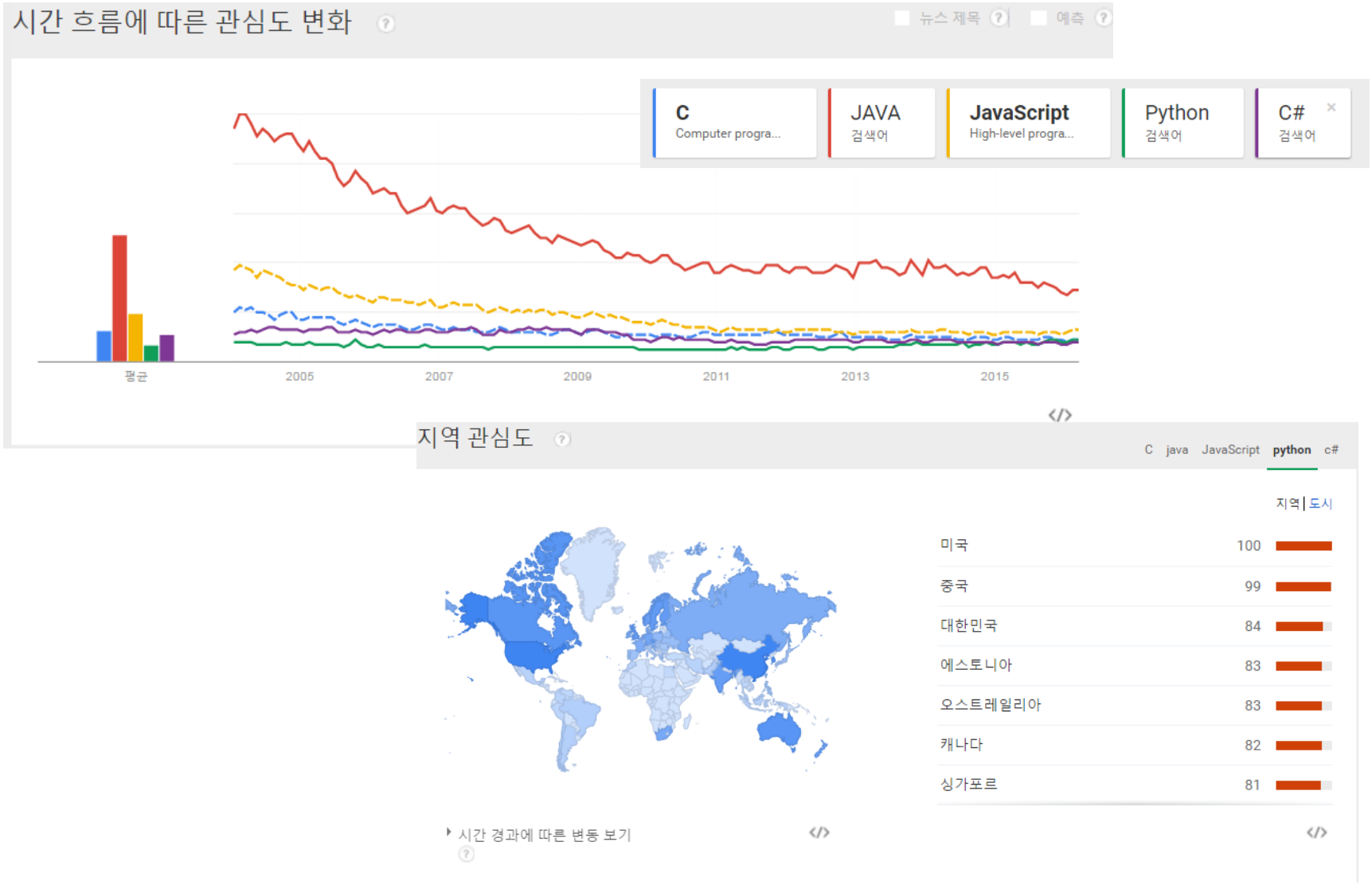
Scripting Languages for the Web

- **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
  - Use in the gamming industry (2006~2007) (the small size of its interpreter, 150Kbytes)

- 확장언어, 스크립트 언어
  - 게임 개발 용
  - Python 보다 성능 우수
  - C/C++과 연동
  - 게임엔진, 인공지능 C++ 개발
  - 성능위주 Lua 개발
  - 강력한 자료 구조 (연관배열 – table)



Google Trends (global)



The Flagship .NET Language: C#

- Part of the .NET development platform (2000)
  - Component-based software development
  - CTS(Common Type System) provides common class library
    - Creating objects from CTS
  - CLR(Common Language Runtime) virtual machine
- Based on C++ , Java, and Delphi
- Includes pointers, delegates, properties, enumeration types, a limited kind of dynamic typing, and anonymous types
- Is evolving rapidly
- JIT compiler

  (Intermediate Language(IL) → machine code)

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