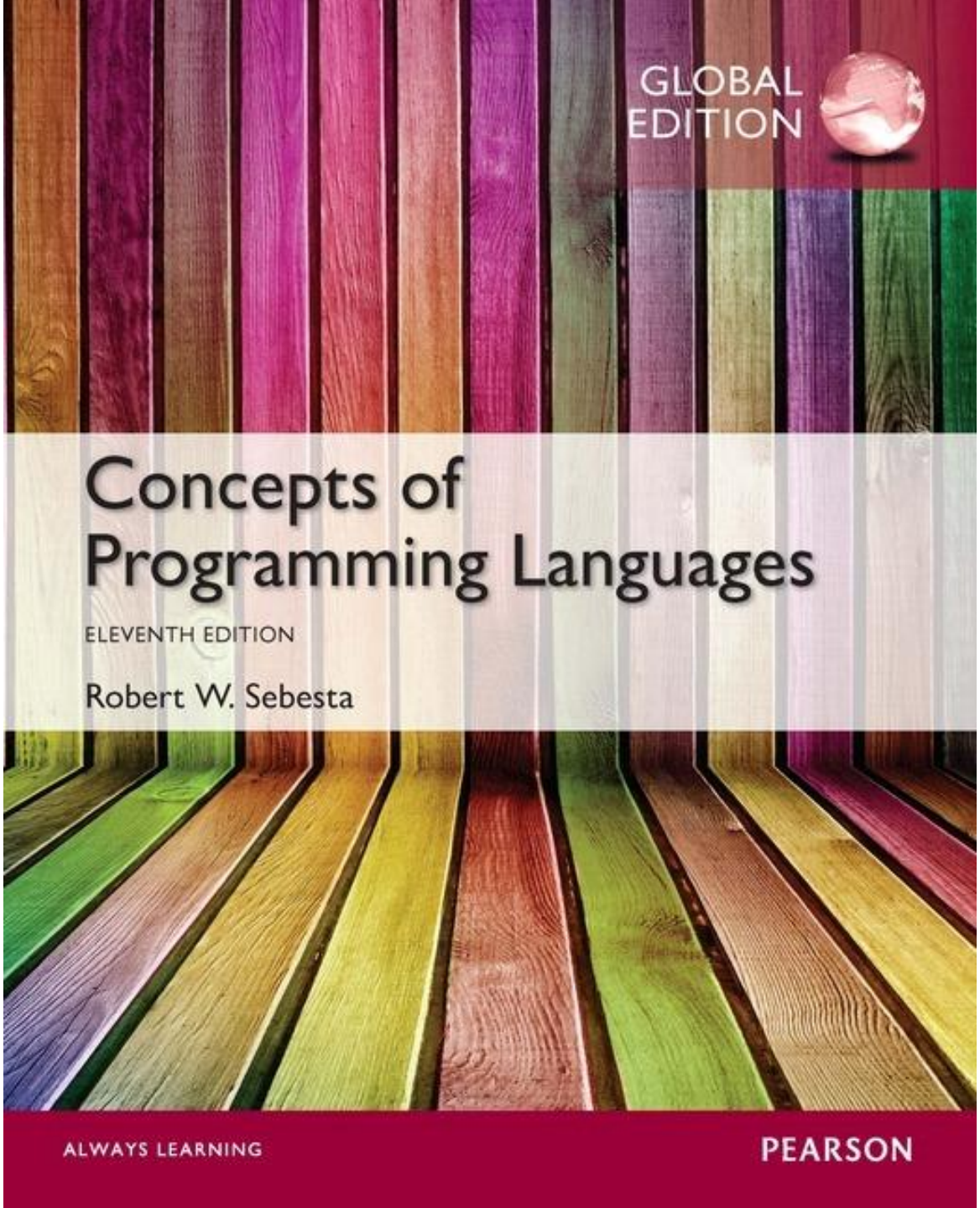

GLOBAL
EDITION
Concepts of
Programming Languages
ELEVENTH EDITION
Robert W. Sebesta
ALWAYS LEARNING
PEARSON

Chapter 1 Preliminary

1.1 Reasons for Studying Concepts of Programming Languages

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

1.2 Programming Domains

- Scientific Applications
  - Fortran?
- Business Applications
  - COBOL (appeared in 1960)

1.2 Programming Domains

- Artificial Intelligence
  - Symbolic but not numeric
  - Linked list but not array
  - Functional language : LISP
  - Logic programming language: Prolog

1.2 Programming Domains

- Systems Programming
  - OS and tools
  - Machine-dependent
  - C language

1.2 Programming Domains

- Web Software
  - Markup languages
    - HTML, XML…
  - Scripting langages
    - JavaScript or PHP

1.3 Language Evaluation Criteria

- Impact on the software development process
- Maintenance

Readability

Maintenance and readability

- Overall Simplicity
  - Readability problems occur
    - Authors had learned a different subsets
    - Feature multiplicity
    - Operator overloading
  - Simplicity in languages can be carried too far
    - Result in less readable
      - Assembly language

Readability

- Orthogonality
  - Orthogonal
    - 直角的、正交的
    - Easier use (in mathematics)
    - Non-overlapping, uncorrelated, independent object
  - Definition of orthogonality in PL
    - First para. of Section 1.3.1.2

Readability

- Data Types
  - The presence of adequate facilities for defining data types and data structures in a language is another significant aid to readability

Readability

- Syntax Design
  - Special words
  - Form and meaning
    - Syntax and semantics

Writability

- Simplicity and Orthogonality
- Expressivity

Reliability

- Type checking
- Exception handling
  - Intercept run-time error
- Aliasing
  - A dangerous feature
  - E.g., Union & pointer in C
    - See next slice.
- Readability and Writability

Union of C

```
typedef struct a {
 int i;
 union {
     float x;
     int y; }
 } r1;
  r1.x=5.1;
  printf(“%d”,k.y);
```

Cost

- Cost of training programmers
- Cost of writing programs
- Cost of compiling programs
- Cost of executing programs
- Cost of the language implementation system
- Cost of poor reliability

Cost

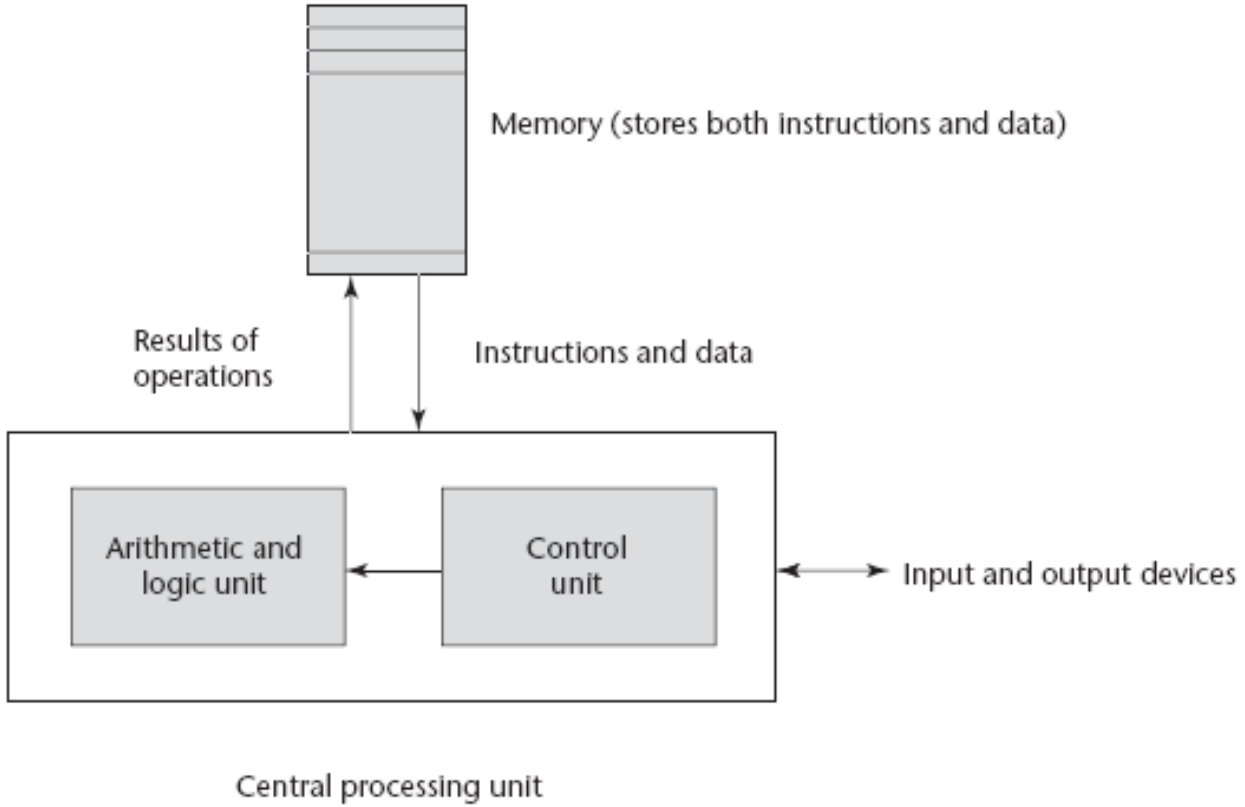
- Cost of maintaining programs
  - Maintenance costs can be as high as two to four times as much as development costs (Sommerville, 2005)
- Portability
- Generality and Well-definedness

1.4 Influences on Language Design

- Computer architecture
  - A profound effect on language design
  - Von Meumann architecture
    - Imperative languages
    - Central features
      - Variables
      - Assignment statements
      - Iterative form

Figure 1.1

The von Neumann computer architecture



1.4 Influences on Language Design

- Computer architecture
  - Languages that are not imperative
    - Functional language
      - Without assignment statements and without iteration
  - Imperative languages dominate!

1.4 Influences on Language Design

- Programming design methodologies
  - Trend
    - HW cost ↓
    - SW cost ↑

1.4 Influences on Language Design

- Programming design methodologies
  - 1970s
    - Top-down design and stepwise refinement
  - Late 1970s
    - A shift from procedure-oriented to data-oriented program design and methodologies
      - Abstract data types
  - 1980
    - Object-oriented design

1.5 Language Categories

- Four bins:
  - Imperative, functional, logic, and object-oriented.
- Others:
  - Scripting language
    - By interpretation
    - E.g., Perl, JavaScript, Ruby (still imperative)

1.5 Language Categories

- Recently days
  - Markup language
    - HTML, XML, XSLT, etc.

1.6 Language Design Trade-Offs

- What is the meaning of trade-off?
- Trade-offs
  - Reliability and cost of execution
  - Design trade-off
    - How about APL? (See next slice)
  - Writability and reliability

How does a really complicated APL routine look like?

Performing a fast Fourier transformation (FFT)

```
     ∇ Z←FFT X;C;D;E;J;K;LL;M;N;O
[1]    LL←⌊2*-O-⍳M←⌊2⊛N,0⍴E←1-2×~O←⍳1.J←⍳L
       ←0,0⍴K←⍳N←¯1↑⍴X
[2]    →(M>L←L+1)/1+⍴⍴J←J,N⍴ 0 1 ∘.=(
       2*L)⍴1
[3]    Z←X[;(L←0)+(⌽LL)+.×J←(M,N)⍴J]
[4]    X← 2 1 ∘.○○(-O-K)÷¯1↑LL
[5]    Z←Z[;K-,LL[L]×J[L;]]+(⍴Z)⍴(-⌿X[;D]×
       Z[;C]),+⌿X[;D←O+N⍴LL[E+M-L]×-O-⍳
       2×LL[L]]×⊖Z[;C←K+,LL[L]×0=J[L;]]
[6]    →((M+O)>L←L+1)/5
     ∇
```

1.7 Implementation Methods

- Compilations
- Pure Interpretation

Figure 1.4

Pure interpretation