Reasons for studying Concepts of Programming Languages

1. Increased capacity to express ideas
   (To increase one's vocabulary of useful programming concepts)

   - more language features known, more used
   - allows flexible thoughts

2. Improved background for choosing appropriate languages

   Simulation: GPSS, SLAM II,
   AI: LISP
   Database: SQL, Ingres,

3. Increased ability to learn new languages

   Fortran + Cobol → PL/I
   Lisp → Prolog (∵ functional language)

4. Better understanding of the significance of implementation

   Recursion: Why recursive version is slower?
   Static vs. dynamic memory allocation

   in Fortran,

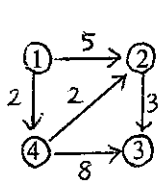
   ```
   DIMENSION A(N)
         :
   READ, N
   ```

5. Increased ability to design new language

6. Overall advancement of computing

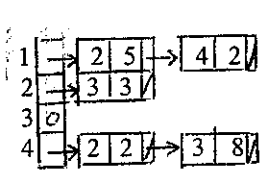
   Algol 60 could not replace Fortran, because at that time programmers/managers did not understand the benefit of block structure, recursion, structured control statements.

Ex. Data structures



	1	2	3	4
1	0	5	0	2
2	0	0	3	0
3	0	0	0	0
4	0	2	8	0

Adjacency matrix



Adjacency list

1	2	5
1	4	2
2	3	3
4	2	2
4	3	8

Triplet

Forward star

Application areas and associated languages

Scientific (numerical) applications
- **Fortran, Algol, PL/I, APL**
- **Mathematica, Matlab**

Business applications
- **RPG**
- **Cobol, PL/I**

Artificial Intelligence
- **Lisp**
- **Prolog**

Systems programming
- **Assembly language**
- **C**

Instruction
- **Watfor/Watfiv, PL/C**
- **Basic, Pascal, Alice**

String manipulation
- **Snobol**
- **Perl**

Social science
- **SPSS (Statistical Package for Social Sciences) 1968**

Database
- **SQL, Ingres, DBASE, Paradox, Prolog**

Simulation
- **GPSS, Simscript, SLAM II, Simulink**

Web software – Script languages
- **JavaScript, PHP, Ruby, Python**

Programming languages evolve continuously.

Trend

- more features (built-in function)
- support for abstraction
- support for modularization
- support for linear flow of control
- exception handling
- support for concurrency

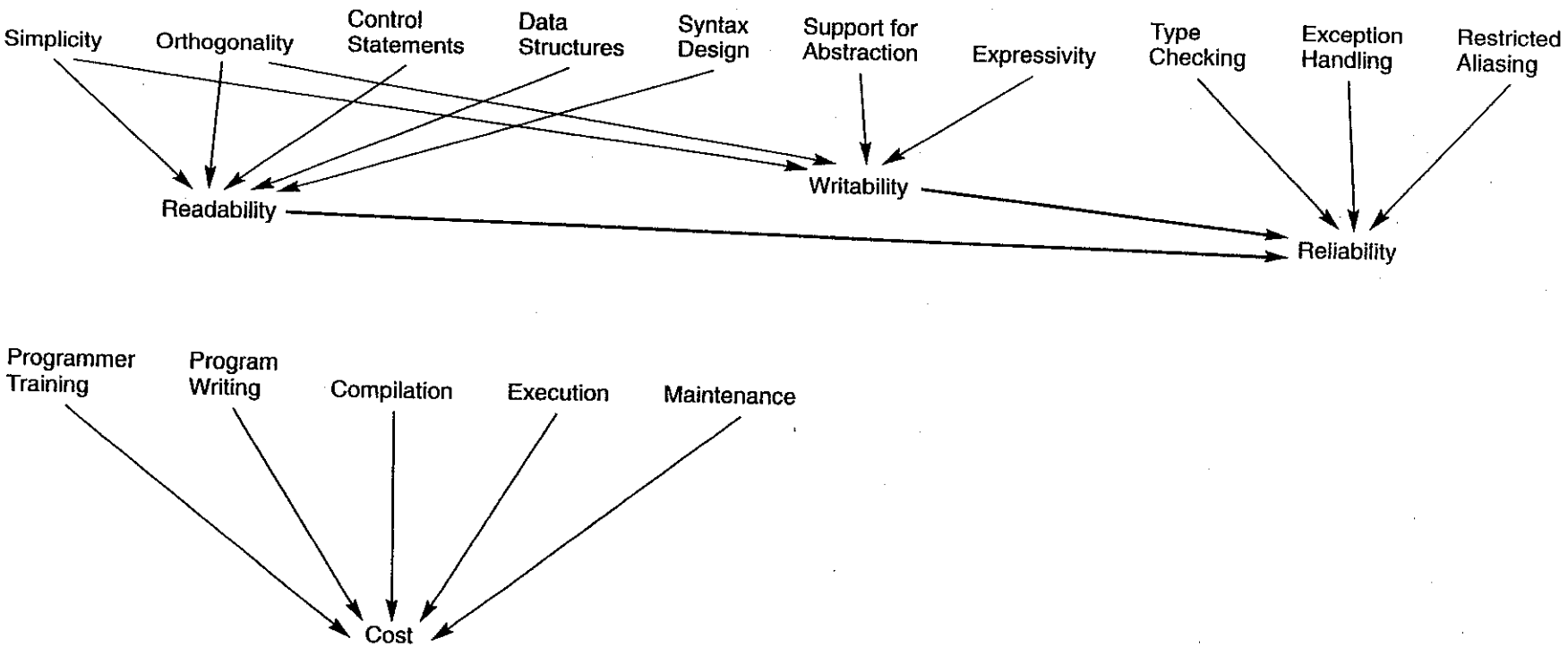
Language Evaluation Criteria

Readability

Writability

Reliability

Cost



Programming language evaluation criteria

Readability

- clear, simple, unified

Overall Simplicity

- Feature multiplicity
- Over-simplification → lack of suitable control structures, large programs, hard to read

Orthogonality (Regularity) †

- more orthogonal, fewer exceptions
- too much → complex constructs, poor readability (Algol 68)

Control Structures

- Sequence, Selection, Iteration => structured program

Adequate Data Structures and Data Types

- Example. Boolean
  Fortran: flag = 1
  Pascal: flag:= true;

Syntax Considerations

- well-structured syntax and semantic description
- BNF, Attribute grammars, …

naming: Basic: A-Z, A1, A2,…,Z9
Fortran: 6 characters
PL/I: FED-TAX, STATE-TAX

- Consistency with commonly used notations and conventions
  Ex. +: addition
  ↑, ^, or **
  * : multiplication (cf. x)

- Pretest vs. posttest Ex. DO 25 I = 5,0

Orthogonality Example

Values stored in registers or memory cells

4 options
- reg + reg
- reg + memory
- memory + reg
- memory + memory

In VAX (orthogonal)

```
ADDL operand_1,operand_2
```

In IBM (not orthogonal)

```
A  reg1, memory_cell
AR reg1, reg2
```

Writability

Cobol vs. APL

Simplicity

allows correct use of all features

Orthogonality

Support for Abstraction

Abstraction is the ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored.

- process abstraction (subprograms)
- data abstraction (array, records, classes)

Expressivity

Convenient ways of expressing computations

Examples

1. if ~ then ~
   else if then ~ => case n of
   else if then ~ 1: -
   else ~ 2: -
   3: -

2. for loop
   while ~ do ~
   repeat ~ until ~

Today's Fortune Cookie:

APL is a write-only language. I can write programs in APL, but I can't read any of them.

-- Roy Keir

Reliabilty

Program is reliable if it performs to specification under all conditions.

Type-Checking

Compile-time checking rather than run-time checking
- Ada
No parameter/argument checking – Fortran

Exception Handling

Run-time errors intercepted, corrections made, and program continues.
PL/I, Ada, C++, Java

Aliasing

More than one name for the same memory location.
EQUIVALENCE (Fortran), DEFINES (Cobol)

Readability and Writabilty

Fortran's ignoring blanks causes reliability problem.

DO 45 I = 1.10 → **DO45I = 1.10**
for-loop **assignment**

Cost

Training Programmers

Simplicity and orthogonality
experience of programmer

Writing Programs

closeness to purpose of application
Development environment affects training as well

Compilation Costs

Execution Costs

trade-off

EX WATFOR vs. Fortran IV

Compiler Costs

PL/I vs. Subset PL/I

Cost of Poor Reliability

Maintenance Costs

corrections and modification to add capabilities

Other Criteria

Portability

machine independence
Java – bytecode for JVM

Generality

PL/I vs. GPSS

Naturalness for the application

Engineering	--	Fortran
Simulation	--	GPSS
Data processing	–	Cobol
Database	--	SQL

Provabilty

ease of programming verification

Restrictabilty

PL/I - subset PL/I
Ada - no subset

Extensibility

maintenability

Uniformity

Lisp - Cambridge Polish Notation (prefix)

Influences on Language Design

Computer Architecture

Von Neumann architecture

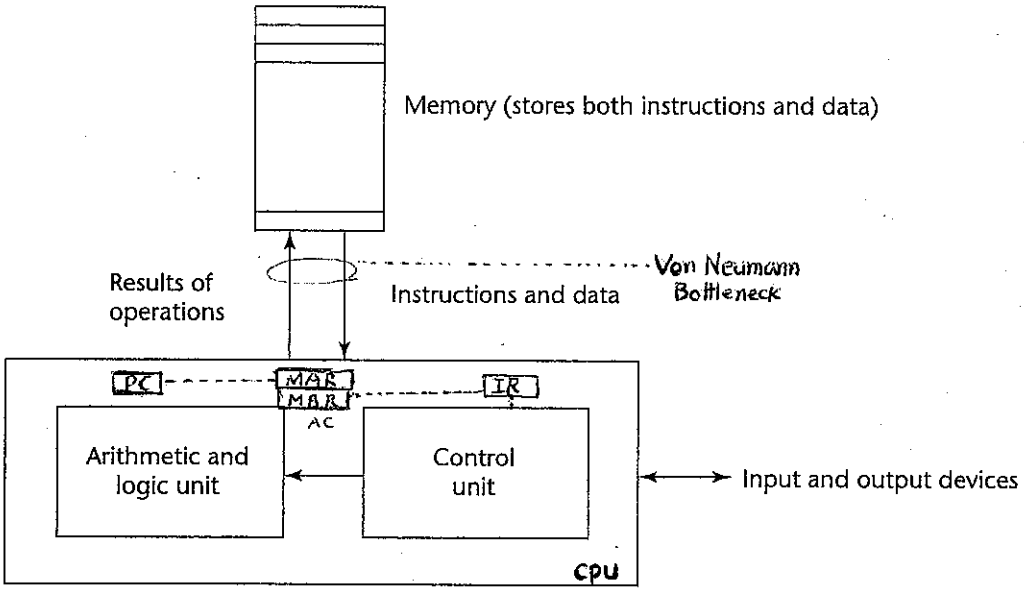


Figure 1.1
The von Neumann computer architecture

Instruction cycle

fetch→(indirect)→execute→(interrupt)

Imperative languages – variables, assignment statements, iteration

Functional languages (Lisp) don't fit von Neumann

- applies functions to parameters

no variables
no assignment statements
no iteration (recursion instead)

Programming Methodologies

Structured Programming ('70)

top-down design and step-wise refinement

Data-Oriented Program Design

Data abstraction
Simula 67, Modula 2, Ada, C++

Object-Oriented Programming ('80)

Adds inheritance and dynamic binding to data Encapsulation
Smalltalk, Ada95, C++, Java

Process-Oriented Programming

Concurrency
Ada, Java, Fortran 90/HPF

Language Categories

Imperative (Procedural) – Visual : Visual Basic
Functional
Logic · rule-based
Object-oriented

· Scripting – Perl, JavaScript, Ruby

Language Design Trade-offs

Compilation vs. Interpretation
Reliability vs. Cost of execution
Expressivity vs. Readability (APL)
Flexibility vs. Safety

- **variant record (Pascal)**
- **range check for array**

C:	**no**
Ada:	**yes**

Compilation

Interpretation: Lisp, Snobol, APL

Hybrid: Java bytecode for JVM

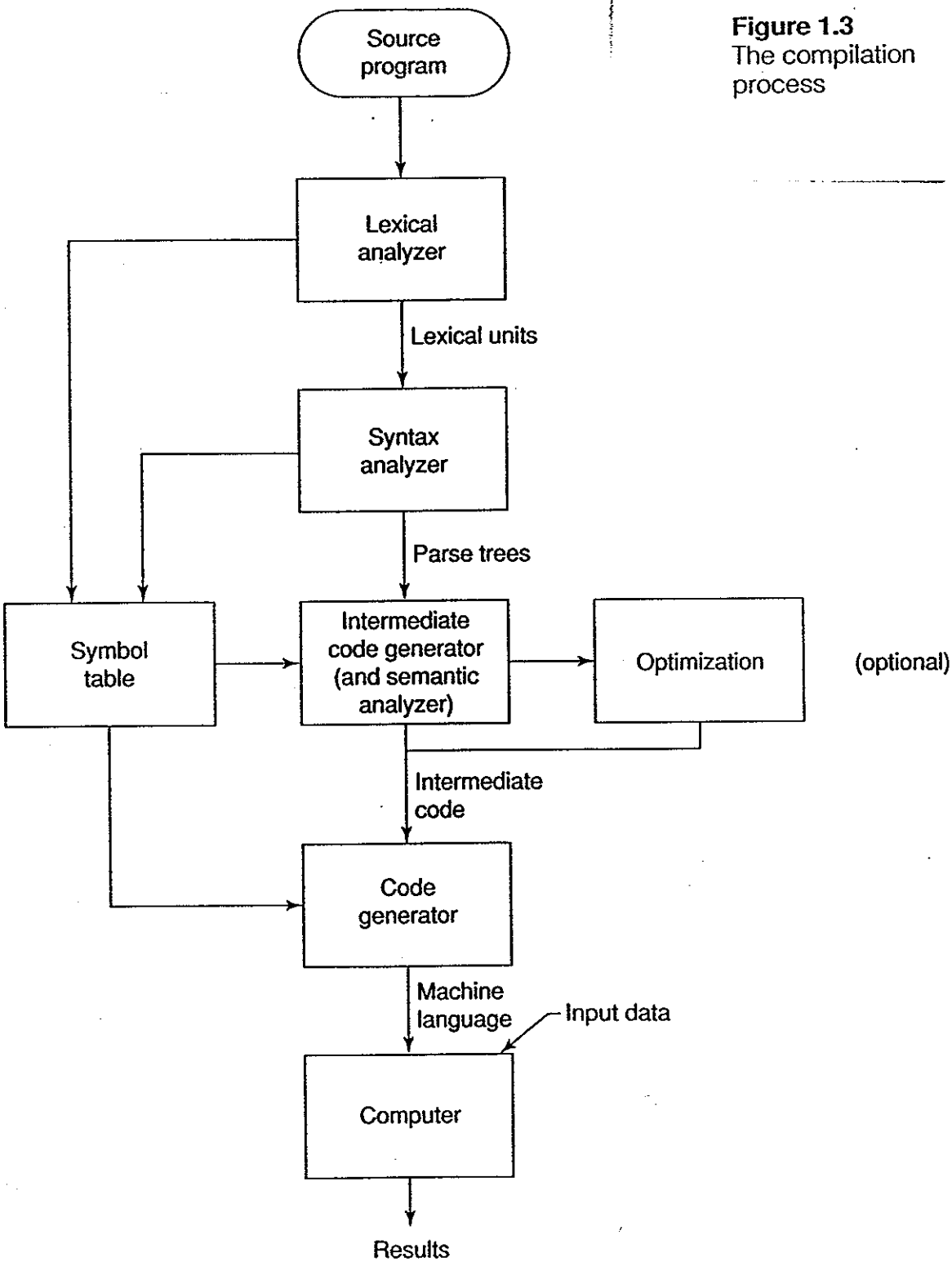


Figure 1.3 The compilation process

Why high-level languages (not assembly languages)?

A:= B + C; vs.

```
Load  R3, B
Add   R3, C
Store R3, A
```

(IBM S/370)

1. Efficiency
2. Easy to learn
3. Easy debugging
4. Good documentation
5. Machine independent
6. Overall code efficiency – Optimizing compiler

Modern Programming Languages

Support for abstraction
Support for modularization
Support for linear flow of control
Exception handling

Steps for Language Development

1. Choose a specific application area.
2. Make the design committee as small as possible.
3. Choose some precise design goal.
4. Release version 1 of the language to a small set of interested people.
5. Revise the language definition.
6. Attempt to build a prototype compiler. Attempt to provide formal definition of the language semantics.
7. Revise the language definition again.
8. Produce a clear, concise language manual and release it.
9. Provide a production quality compiler and distribute it.
10. Write marvelously clear primers explaining how to use the language.