Chapter 1

Introduction to Programming Paradigms

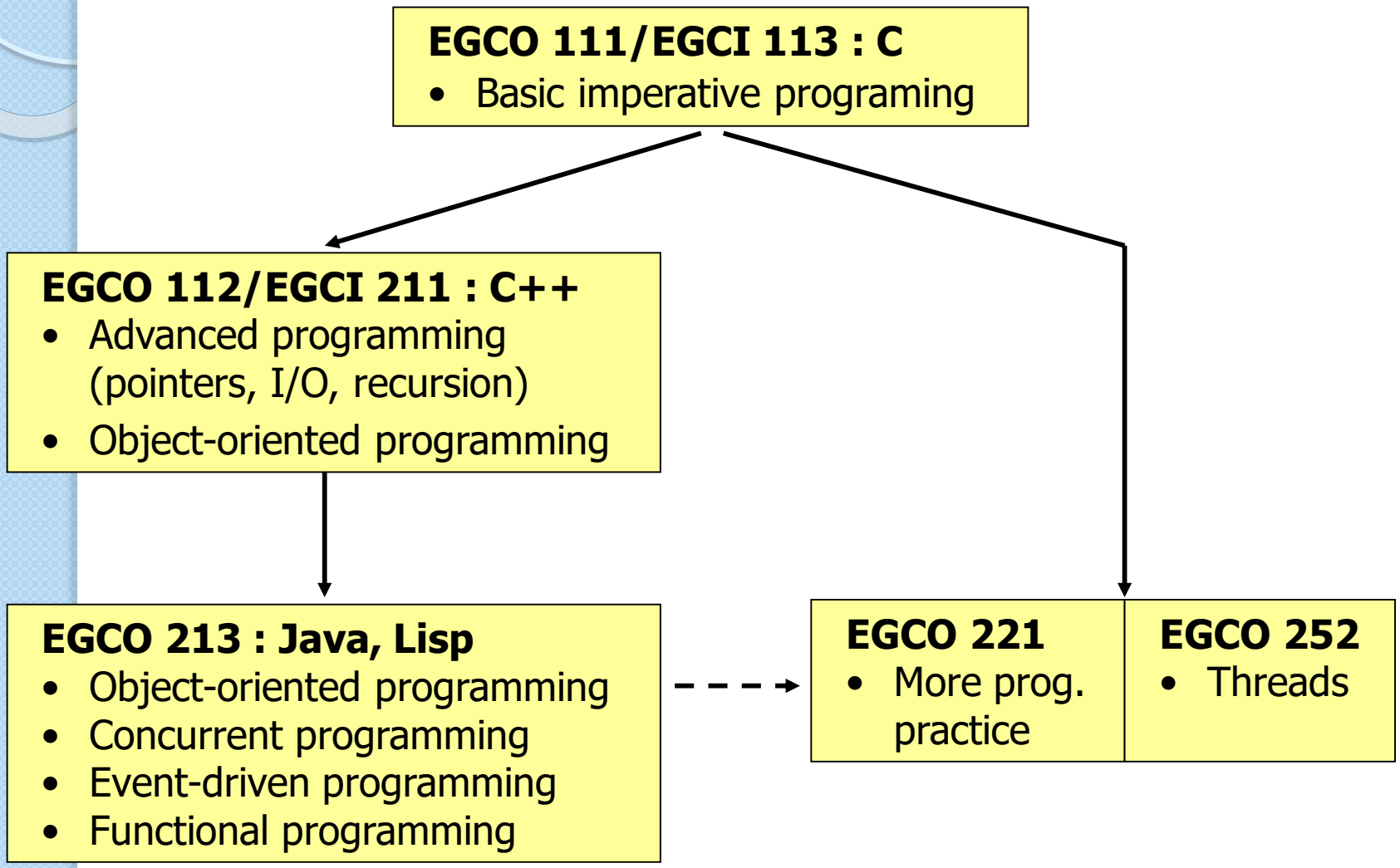
References :

[1] รังสิพรรณ มฤคทัต, กระบวนทัศน์ในการเขียนโปรแกรม (บทที่ 1)

[2] Tucker & Noonan, Programming Languages: Principles and Paradigms (Chapter 1)

[3] Sebesta, Concepts of Programming Languages (Chapters 1-2)

Course Outline & Background



References

- รังสิพรรณ มฤคทัต, กระบวนทัศน์ในการเขียนโปรแกรม, 2013.
  - Based on Java 7
  - All contents are already included in course materials
- Theory
  - Allen Tucker & Robert Noonan, Programming languages: principles and paradigms (11th edition), 2019.
  - Robert Sebesta, Concepts of programming languages (2nd edition), 2006.
- Programming : any book on Java & Lisp programming
  - Oracle, Java Documentation (https://docs.oracle.com/en/java/javase/)
  - Guy L. Steele, Common Lisp the Language, 1990.
  - นิตยา นินทรกิจ, เรียนรู้ภาษา Lisp, 2003.

Principles of Programming Languages

- Syntax : rules of grammar
  - Influence how programs are written by a programmer, read by other programmers, and parsed by the computer
- Semantics : meaning
  - Determine how programs are composed by a programmer, understood by other programmers, and interpreted by the computer
- Names (identifiers) & Types
  - Names of elements such as variables, functions, classes is associated with properties (scope, visibility, etc.)
  - Types : kinds of values, operations on the values, memory

Programming Paradigms

- School of thought that underlies a particular genre of programs and languages
  - Different paradigms are based on different concepts, making each of them suitable for certain classes of problems
- Some programming languages may support multiple paradigms
- Choosing paradigms ➔ application domain e.g. scientific, business, AI, system software, web-based

Major Paradigms

- **Imperative programming**
  - Series of steps : input → process → output
  - Assignments, loops, conditional statements, procedures
  - Examples : Fortran, Pascal, C
- **Object-oriented programming**
  - Objects interacting with each other by passing messages
  - Object modelling, encapsulation, classification, inheritance, polymorphism
  - Examples : Smalltalk, Eiffel, C++, Java
- **Event-driven programming**
  - Continuous loop responding to events that are generated in an unpredictable order
  - Examples : Java, Visual Basic

Concurrent programming

- Asynchronous elements executing concurrently
- Involve communication and synchronization
- Examples : Occam, High Performance Fortran

Functional programming

- Collection of math functions (domain → range)
- Function composition, condition, recursion
- Examples : Lisp, Scheme, ML, Haskell

Logic (declarative) programming

- Declare what outcome the program should accomplish, rather than how it should accomplish
- Execution requires facts, rules, inferencing engine
- Examples : prolog

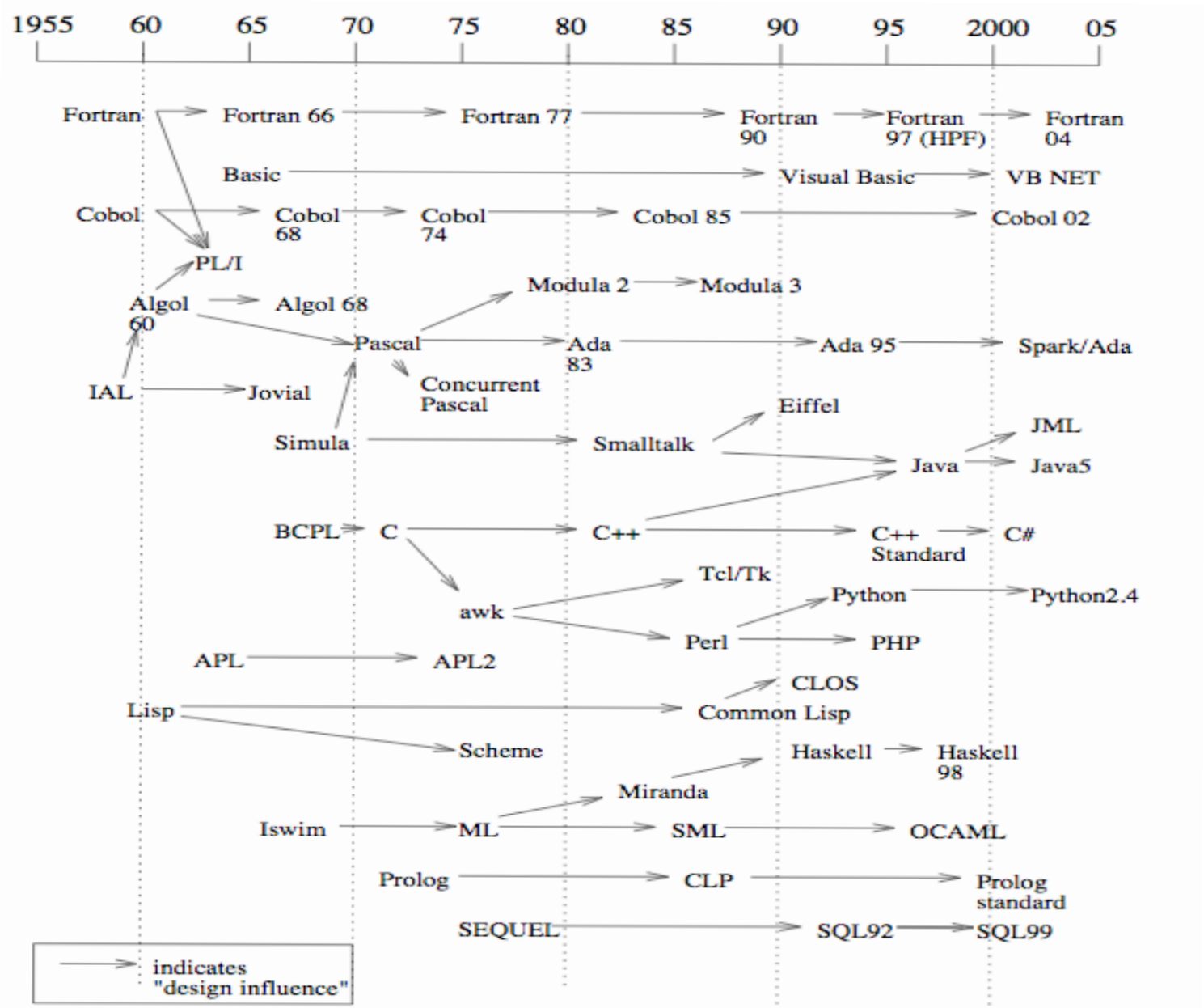


Figure 1.2: A Snapshot of Programming Language History

(From Tucker & Noonan [2], Ch.1)

1	Programming Language	Dec 2023	Dec 2022	Ratings(%)	Change(%)
2	Python	1	1	13.86	-2.80
3	C	2	2	11.44	-5.12
4	C++	3	3	10.01	-1.92
5	Java	4	4	7.99	-3.83
6	C#	5	5	7.30	+2.38
7	JavaScript	6	7	2.90	-0.30
8	PHP	7	10	2.01	+0.39
9	Visual Basic	8	6	1.82	-2.12
10	SQL	9	8	1.61	-0.61

- Tiobe index (https://www.tiobe.com/tiobe-index/) : based on the number of skilled engineers, courses, third-party vendors retrieved by popular search engines
- Tiobe index archive (https://github.com/toUpperCase78/tiobe-index-ratings)

- IEEE Spectrum, The Top Programming Languages 2024 (https://spectrum.ieee.org/top-programming-languages-2024)

- IEEE Spectrum, The Top Programming Languages 2024 (https://spectrum.ieee.org/top-programming-languages-2024)

Choosing languages

- Writability : rules, syntax, semantics of the language
- Readability → maintenance
- Reliability : type checking, exception mechanisms
- Cost
  - Compiler & runtime system
    - Nonproprietary : C, C++, Java
    - Proprietary : Visual Basic
  - Learning, availability of documents
  - Implementation on contemporary platforms

Imperative Family

- Fortran (Formular Translator)
  - Developed by IBM
  - Motivation
    - Scientific application : floating-point processing
    - Speed of compilation & code generation
  - Prior to Fortran 90, dynamic allocation & recursion are not included
- COBOL (Common Business Oriented Language)
  - Motivation
    - Business application : simple computation & reporting
    - Easy to use and understand, even by managers
  - Very similar to English
  - Main components are data description & operation

Example : Fortran code

```
C     EXAMPLE FORTRAN CODE
      INTEGER INTLIST(99)
      INTEGER LIST_LEN, COUNTER, SUM, AVERAGE
      RESULT = 0
      SUM = 0
      READ *, LIST_LEN
      IF ( (LIST_LEN .GT. 0) .AND. (LIST_LEN .LT. 100) ) THEN
          DO 10 COUNTER = 1, LIST_LEN
              READ *, INTLIST(COUNTER)
              SUM = SUM + INTLIST(COUNTER)
10        CONTINUE
          AVERAGE = SUM / LIST_LEN
          PRINT *, ' AVERAGE IS ', AVERAGE
      ELSE
          PRINT *, ' ILLEGAL LENGTH '
      END IF
      STOP
      END
```

Example : COBOL code

```
FILE SECTION.
FD    BAL-FWD-FILE
      LABEL RECORDS ARE STANDARD
      RECORD CONTAINS 80 CHARACTERS.
01    BAL-FWD-CARD.
      …
000-REORDER-MAIN.
      OPEN INPUT BAL-FWD-FILE.
      …
      PERFORM 100-PRODUCE-REORDER-LINE
          UNTIL CARD-EOF-SWITCH IS EQUAL TO "Y".
      CLOSE BAL-FWD-FILE.
      STOP RUN.
100-PRODUCE-REORDER-LINE.
      …
      ADD BAL-ON-HAND BAL-ON-ORDER GIVING AVAILABLE.
      IF AVAILABLE IS LESS THAN REORDER-POINT
          PERFORM 130-PRINT-REORDER-LINE.
```

- Algol (Algorithmic Language)
  - Algol 60 is the first language to have formal syntax
  - The language is unpopular mostly due to I/O problems
  - But its concept is followed by many imperative & object-oriented languages
- Pascal
  - Popular use : academic / language teaching
  - Simple and expressive syntax
  - Nearly strongly type, except for variant record structures
- C
  - Develped by Bell Laboratories, for Unix-based software
  - Lack of complete type checking (weakly type) ➔ array subscription, pointer arithmetic, parameter passing, union

Example : Algol 60 code

```
comment Example Algol 60 code;
begin
    integer array intlist [1:99];
    integer listlen, counter, sum, average;
    sum := 0;
    readint (listlen);
    if (listlen > 0) ^ (listlen < 100) then
        begin
            for counter := 1 step 1 until listlen do
                begin
                    readint ( intlist[counter] );
                    sum := sum + intlist[counter];
                end;
            average := sum / listlen;
            printstring (“average is “);
            printint (average);
        end;
end;
```

Example : Pascal code

```
program findavg (input, output);
  var
     intlist : array [1..99] of integer;
     listlen, counter, sum, average : integer;
  begin
     sum := 0;
     readln (listlen);
     if ( (listlen > 0) and (listen < 100) ) then
        begin
           for counter := 1 to listlen do
              begin
                 readln( intlist[counter] );
                 sum := sum + intlist[counter];
              end;
           average := sum / listlen;
           writeln ('average is ', average);
        end;
  end.
```

Ada

- Most extensive & expensive language design effort
- Implement US military software
  - Complex
  - Embedded systems
  - Reliability is critical
- Important features
  - Data abstraction & encapsulation
  - Exception handling
  - Generic programming
  - Concurrent execution (message passing style)
- But turn out to be too large & too complex

Example : Ada code

```
with Ada.Text_IO, Ada.Integer.Text_IO;
use Ada.Text_IO, Ada.Integer.Text_IO;
procedure Ada_Ex is
    type Int_List_Type is array (1..99) of Integer;
    Int_List : Int_List_Type;
    List_Len, Sum, Average : Integer;
    begin
        Sum := 0;
        Get (List_Len);
        if (List_Len > 0) and (List_Len < 100) then
            for Counter := 1 .. List_Len loop
                Get ( Int_List(Counter );
                Sum := Sum + Int_List(Counter);
            end loop;
            Average := Sum / List_Len;
            Put (“average is “);
            Put (Average)
        end if;
    end Ada_Ex;
```

Object-Oriented Family

- Simula (Simulation Language) : successor of Algol 60
  - Features for simulation application : coroutines can resume at the position where they previously stopped
  - Class & object concepts are followed by other OO languages
- C++
  - Hybrid imperative-OO language
  - Inherit type unsafe from C
- Smalltalk
  - Target desktop environment ➔ focus on Window-GUI
  - Pure OO language
- Essential basis for event-driven programming

Functional Family

- Symbolic computation such as calculus, math logic, electrical circuit theory, game playing, AI
- Lisp (List Processor)
  - Single linked-list structure
  - Popular dialects are Scheme, Common Lisp
  - Scheme is typically used for academic / teaching
  - Common Lisp combines features from many dialects
- ML (Meta Language)
  - Popular use : program verification & formal proof
  - Syntax is closer to imperative language (Pascal) than Lisp

- Haskell
  - Pure functional language without any imperative feature
  - ML and Haskell are not as widespread as Lisp since they are mainly used in research / academic

Logic / Declarative Family

- Prolog (Programming Logic)
  - Developed for natural language processing
  - Program consists of facts, rules, and queries
  - Query is processed by drawing inference from known facts and rules
- SQL (Structured Query Language)

Example : Lisp / ML / Prolog code

Lisp

```
(defun factorial (x)
    (if  (= x 0)
         1
         (* x (factorial (- x 1)))
    )
)
```

ML

```
fun   fac  0  =  1
  |   fac  x  =  x * fac (x - 1)
```

Haskell

```
fac 0 = 1
fac 1 = 1
fac n = n * fac (n - 1)
```

Prolog

```
parents (william, diana, charles).
parents (harry, diana, charles).
parents (charles, elizabeth, philip).
parents (diana, frances, edward).

parent (C, M) := parents (C, M, D).
parent (C, D) := parents (C, M, D).
grandparent (C, GP)  :=  parent (C, P),
                         parent (P, GP).

?- grandparent (william, harry).
   no

?- grandparent (william, philip).
   yes
```

Concurrent Family

- Ada 95
- HPF (High-Performance Fortran), C*
  - Based on sequential language, with additional instructions for concurrent execution
  - Data parallelism
- Occam
  - Developed for transputers
  - Collection of processes communicating with each other via channels
  - Few basic instructions but very strict syntax (indentation)
  - Popular use : academic, industrial
- Sequential language equipped with thread libraries

Example : HPF code

Sequential execution

```
REAL, DIMENSION(N) :: A, B
…
DO I = 1, N
   A(I)  =  A(I) + B(I)
END DO
```

Parallel execution

```
REAL, DIMENSION(N) :: A, B
…
FORALL (I = 1:N)
   A(I)  =  A(I) + B(I)
END FORALL
```

```
REAL, DIMENSION(N) :: A, B
    …
!HPF$  INDEPENDENT
    DO I = 1, N
        A(I) = A(I) + B(I)
    END DO
```

Example : Occam code

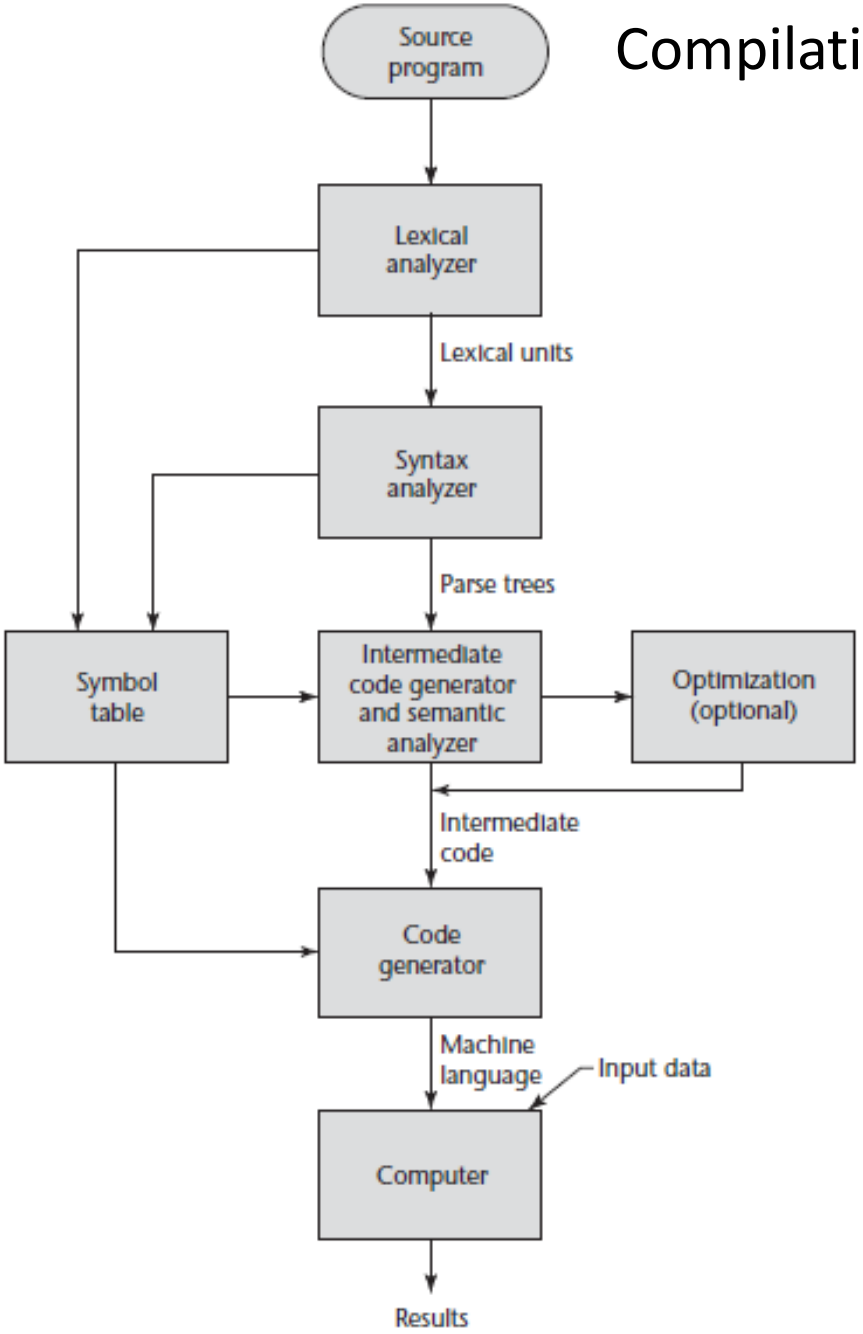
```
PROC producer (CHAN INT out!)
  INT x:
  SEQ
    x := 0
    WHILE TRUE
      SEQ
        out ! X
        x := x + 1
:
PROC consumer (CHAN INT in?)
  INT y:
  SEQ
    WHILE TRUE
      in ? y
      y := y - 10
:
```

```
PROC network ()
  CHAN INT c:
  PAR
    producer(c!)
    consumer(c?)
:
```

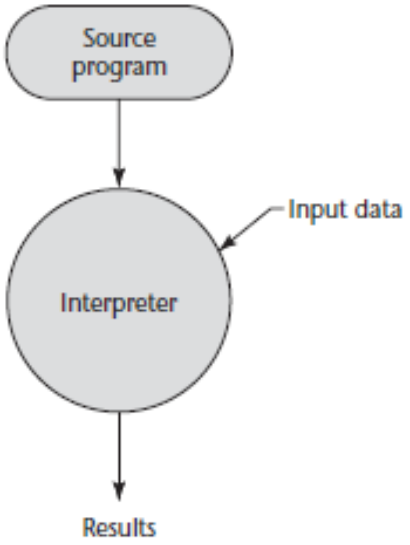
Language Processors

- Compiler
  - Translate high-level to low-level (machine dependent) code
  - Translation at compile-time
  - Machine code executes directly on hardware, hence fast execution
- Interpreter
  - Act as a software simulation of a machine, a virtual machine
  - Read, interpret, and process instructions one-by-one

Compilation

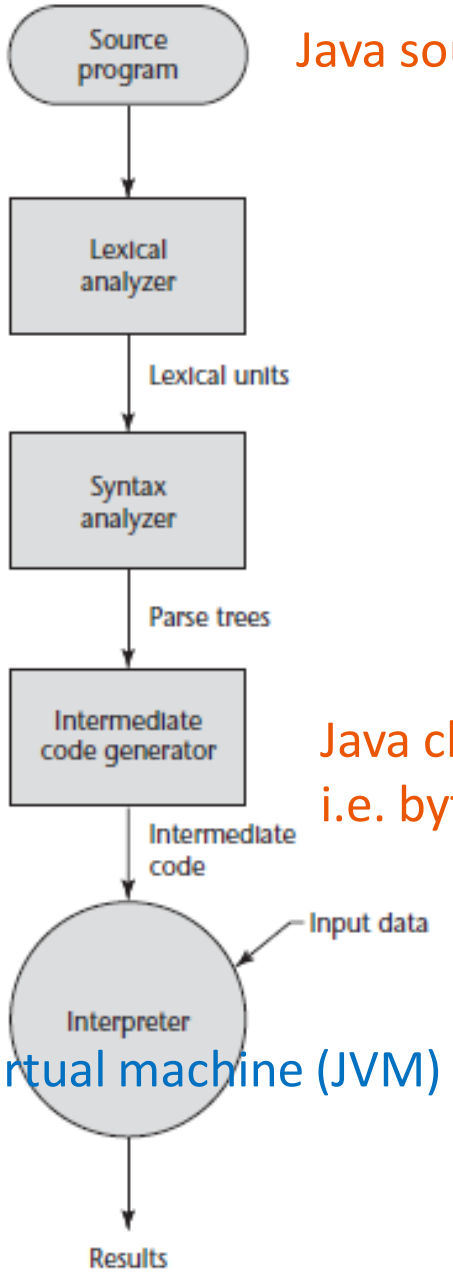


Interpretation



From Sebesta [3], Ch.1

Figure 1.5
Hybrid implementation system



Java source file (test.java)

Hybrid Approach

Java compiler

Java class file (test.class), i.e. byte code

Java virtual machine (JVM)

From Sebesta [3], Ch.1