http://www.cs.nthu.edu.tw/~king/courses/cs2403.html

CS2403 Programming Languages

Preliminaries

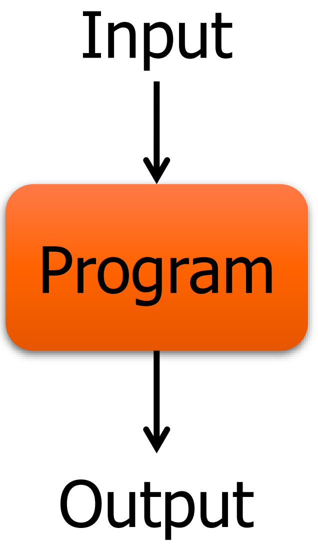
Chung-Ta King
Department of Computer Science
National Tsing Hua University

(Slides are adopted from *Concepts of Programming Languages*, R.W. Sebesta; *Modern Programming Languages: A Practical Introduction,* A.B. Webber)

國立清華大學
National Tsing Hua University

Programming Language

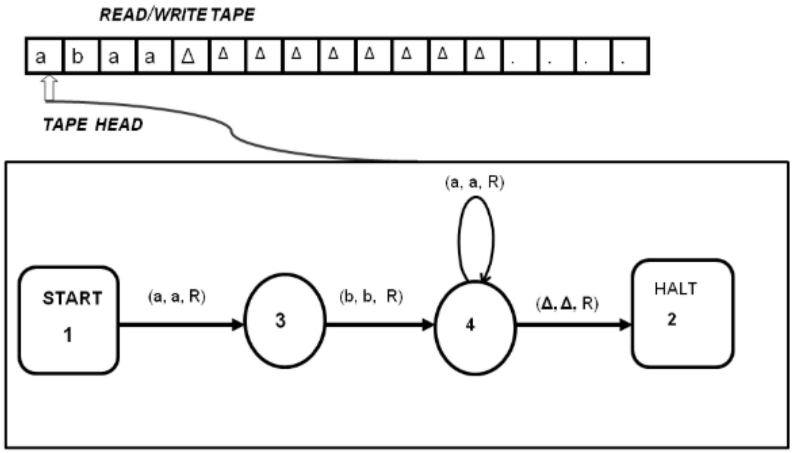
- A programming language is an artificial language designed to express computations or algorithms that can be performed by a computer -- Wikipedia
- A program is computer coding of an algorithm that
  - Takes input
  - Performs some calculations on the input
  - Generates output



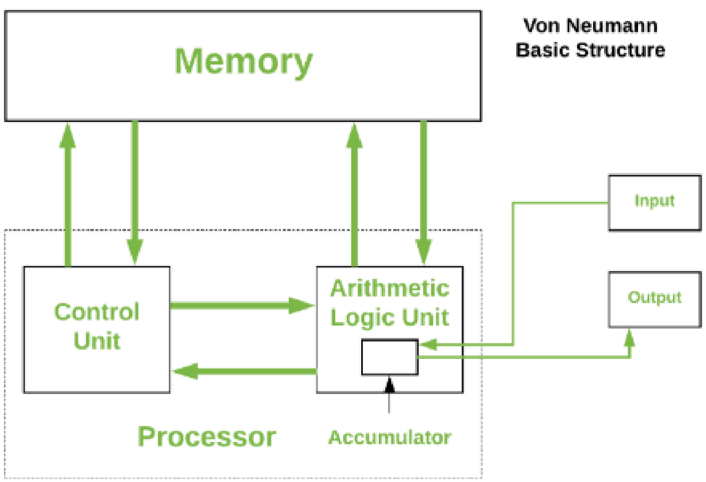
Outline

- Three models of programming languages
- A brief history
  - Programming design methodologies in perspective (Sec. 1.4.2)
- Language evaluation criteria (Sec. 1.3)
  - Language design trade-offs (Sec. 1.6)
- Implementation methods (Sec. 1.7)

Computer architecture



A Turing Machine for aba*

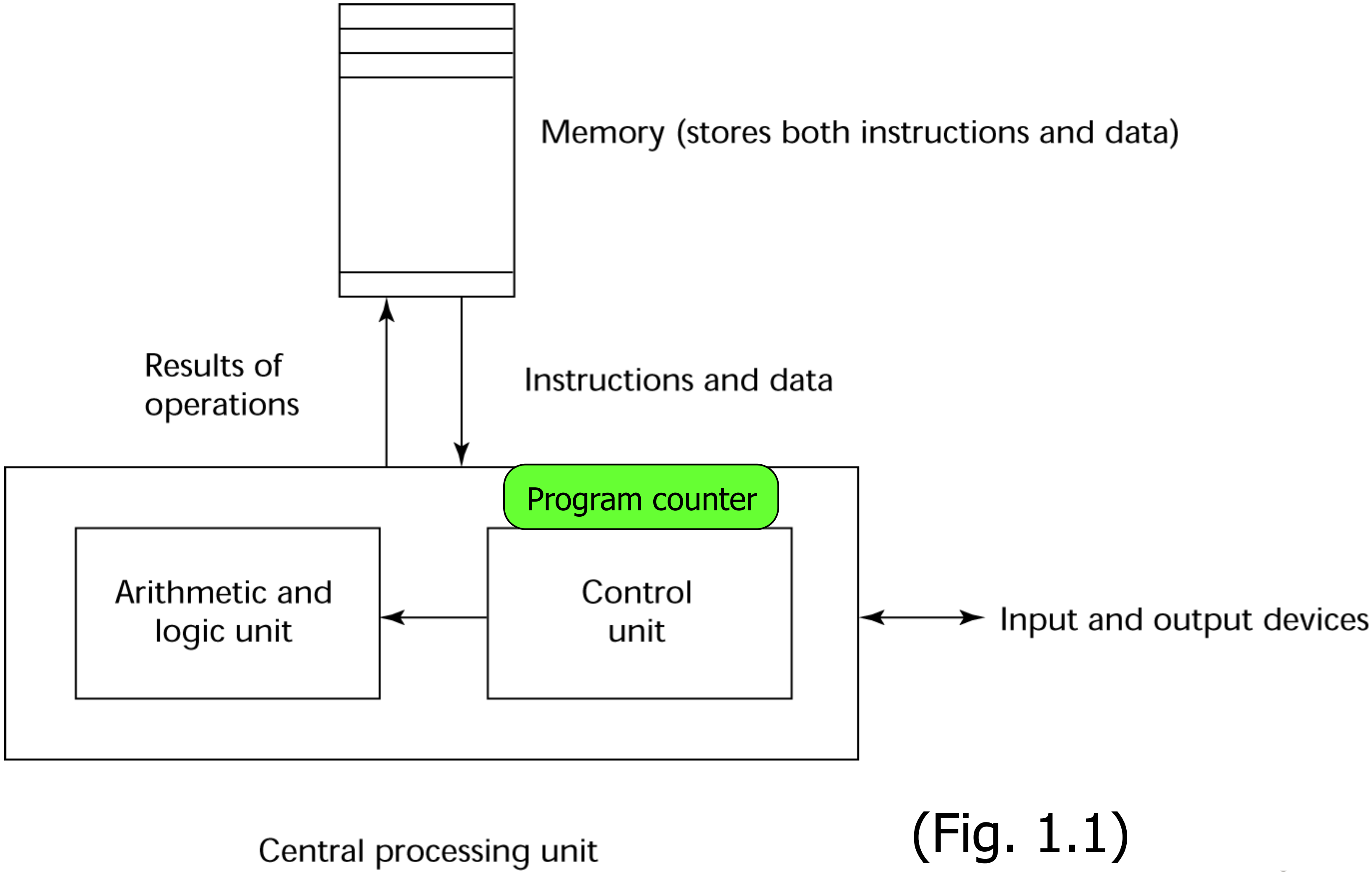


von Neumann

Models of Programming Languages

Programming is like …

The von Neumann Architecture



(Fig. 1.1)

The von Neumann Architecture

- Key features:
  - Data and programs stored in memory
  - Instructions and data are piped from memory to CPU
  - *Fetch-execute-cycle* for each machine instruction

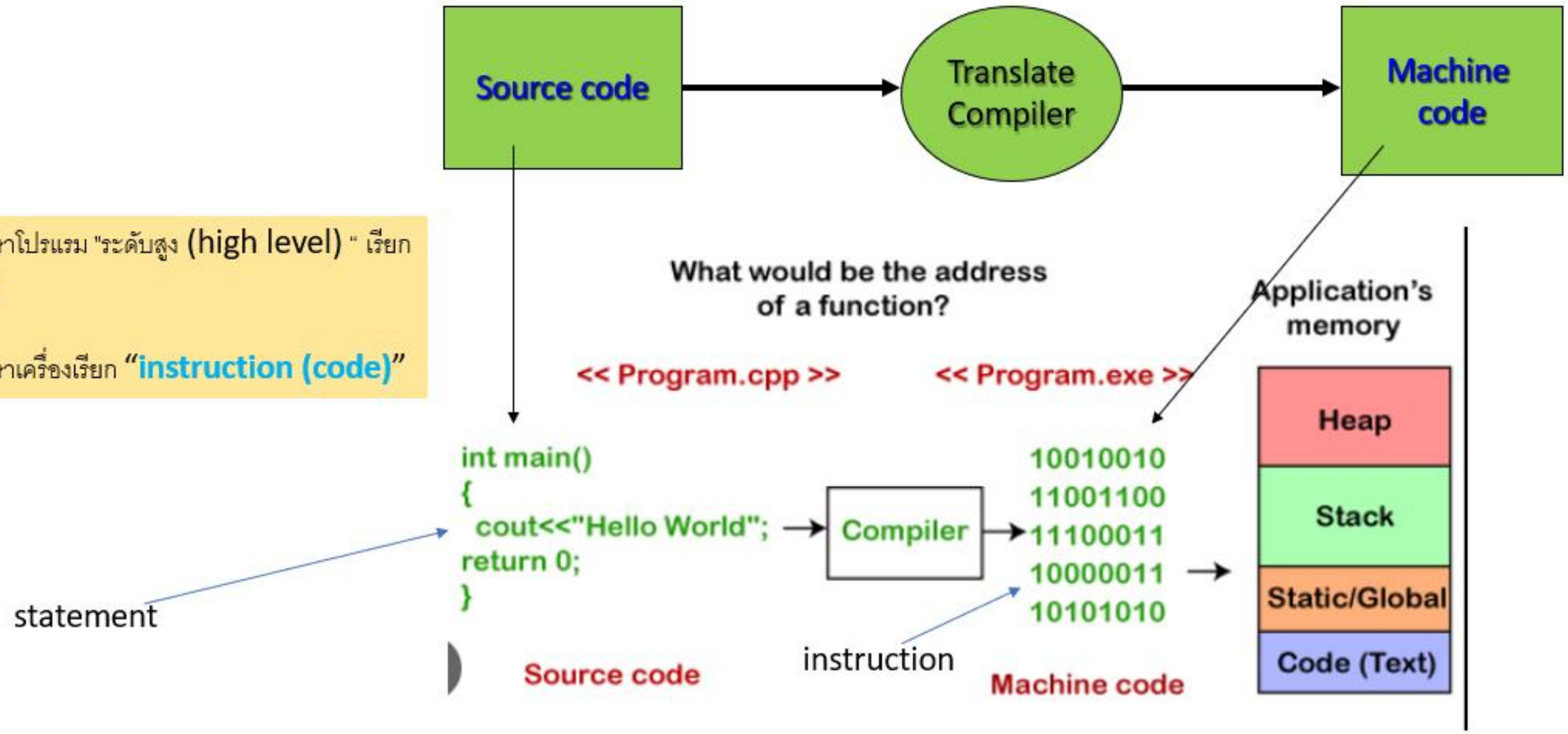
```
initialize the program counter (PC)
repeat forever
  fetch the instruction pointed by PC
  increment the counter
  decode the instruction
  execute the instruction
end repeat
```

add A,B,C	0100 1001
sub C,D,E	0110 1010
br LOOP	1011 0110
…	…
Assembly code	Machine code

National Tsing Hua University 國立清華大學

ภาษาโปรแกรม (Programming Languages)

- หนึ่งคำสั่งของภาษาโปรแรม "ระดับสูง (high level) " เรียก “**statement**”
- หนึ่งคำสั่งของภาษาเครื่องเรียก “**instruction (code)**”



Statement vs Instruction

- หนึ่ง Statement อาจประกอบด้วย หลาย statement ย่อยขึ้นอยู่โครงสร้างภาษาโปรแกรม

```
x = x + 1;
if (x > 0) {
    printf("จำนวนเป็นบวก");
    x = x + 2;
}
```

```
MOV AX, x      ; instruction
ADD AX, 1      ; instruction
MOV x, AX      ; instruction

CMP x, 0          ; เปรียบเทียบ x กับ 0
JLE skip_print    ; ถ้า x <= 0 ให้ข้าม
CALL print_pos    ; เรียกฟังก์ชันพิมพ์
MOV AX, x      ; instruction
ADD AX, 2      ; instruction
MOV x, AX      ; instruction

skip_print:
```

Statement : หนึ่งคำสั่งในภาษาระดับสูง

Instruction : หนึ่งคำสั่งในภาษา machine code

Programming language design paradigm

```
1950s-1960s
│
├── Imperative Paradigm
│   └── FORTRAN (1957), COBOL (1959), C (1972)
│       - Based on Von Neumann architecture
│       - Focus on changing state using instructions
│
├── Procedural (subset of Imperative)
│   └── Pascal (1970), C (1972), Ada (1980)
│       - Organize code into reusable procedures/functions
│
├── Functional Paradigm
│   └── LISP (1958), Scheme (1975), Haskell (1990)
│       - No mutable state, based on mathematical functions
│
├── Logic Paradigm
│   └── Prolog (1972)
│       - Based on formal logic, rules, and queries
│
└── Object-Oriented Paradigm
    └── Simula (1967), Smalltalk (1972), C++ (1983), Java (1995), Python (1991)
        - Objects = data + behavior, supports encapsulation/inheritance
```

(Immutable variable)

If, goto, single variable, array

Subroutine, structure, while, for

Inheritance , polymorphism

```
1990s-present
│
├── Multi-Paradigm Languages
│    └── Python, JavaScript, Scala, Kotlin, Rust
│         - Support multiple paradigms (OO + functional + imperative)
│
└── Reactive/Concurrent Paradigms
     └── Erlang, Elixir, Go, Rust, RxJS
          - Designed for scalable, concurrent systems
```

Parallel execution function

2. Imperative Programming (การเขียนโปรแกรมแบบคำสั่ง)

นิยาม:

การเขียนโปรแกรมโดยการ **สั่งให้คอมพิวเตอร์ทำงานตามลำดับขั้นตอน** อย่างชัดเจน

ลักษณะสำคัญ:

- ควบคุม flow อย่างละเอียด (เช่น `if`, `for`, `while`)
- เน้นการเปลี่ยนแปลงของ state หรือค่าของตัวแปร Mutable variable

ภาษาตัวอย่าง: C, Python, Java

ตัวอย่าง:

python

```python
for student in students:
    if student.age > 18:
        print(student.name)
```

國立清華大學
National Tsing Hua University

3. Concurrent Programming (การเขียนโปรแกรมแบบขนาน/พร้อมกัน)

นิยาม:

การเขียนโปรแกรมที่สามารถ **รันหลายงานพร้อมกัน** (หรือดูเหมือนพร้อมกัน) เพื่อเพิ่มประสิทธิภาพ

ลักษณะสำคัญ:

- งานหลายงาน (tasks/threads/processes) ทำงานไปพร้อม ๆ กัน
- ใช้ในระบบที่ต้องการตอบสนองเร็ว หรือทำหลายอย่างในเวลาเดียวกัน เช่น เกม, เว็บเซิร์ฟเวอร์

ตัวอย่าง:

```python
import threading

def task1():
    print("Doing task 1")

def task2():
    print("Doing task 2")

threading.Thread(target=task1).start()
threading.Thread(target=task2).start()
```

ฟังก์ชันแฟกทอเรียลได้นิยามเชิงรูปนัยไว้ดังนี้

$$n! = \prod_{k=1}^{n} k$$

หรือนิยามแบบเวียนเกิดได้ดังนี้

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

นิยามด้านบนทั้งสองได้รวมกรณีนี้เข้าไปด้วย

$$0! = 1$$

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \cdots \cdot 3 \cdot 2 \cdot 1 \, .$$

ตัวอย่างเช่น

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Prolog (Declarative / Logic)

```prolog
factorial(0, 1).
factorial(N, F) :-
    N > 0,
    N1 is N - 1,
    factorial(N1, F1),
    F is N * F1.

?- factorial(5, F).
F = 120.
```

C (Imperative)

```c
#include <stdio.h>

int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

int main() {
    int n = 5;
    printf("Factorial of %d is %d\n", n, factorial(n));
    return 0;
}
```

$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \cdots \cdot 3 \cdot 2 \cdot 1 \,.$

ตัวอย่างเช่น

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

ฟังก์ชันแฟกทอเรียลได้นิยามเชิงรูปนัยไว้ดังนี้

$$n! = \prod_{k=1}^{n} k$$

หรือนิยามแบบเวียนเกิดได้ดังนี้

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

นิยามด้านบนทั้งสองได้รวมกรณีนี้เข้าไปด้วย

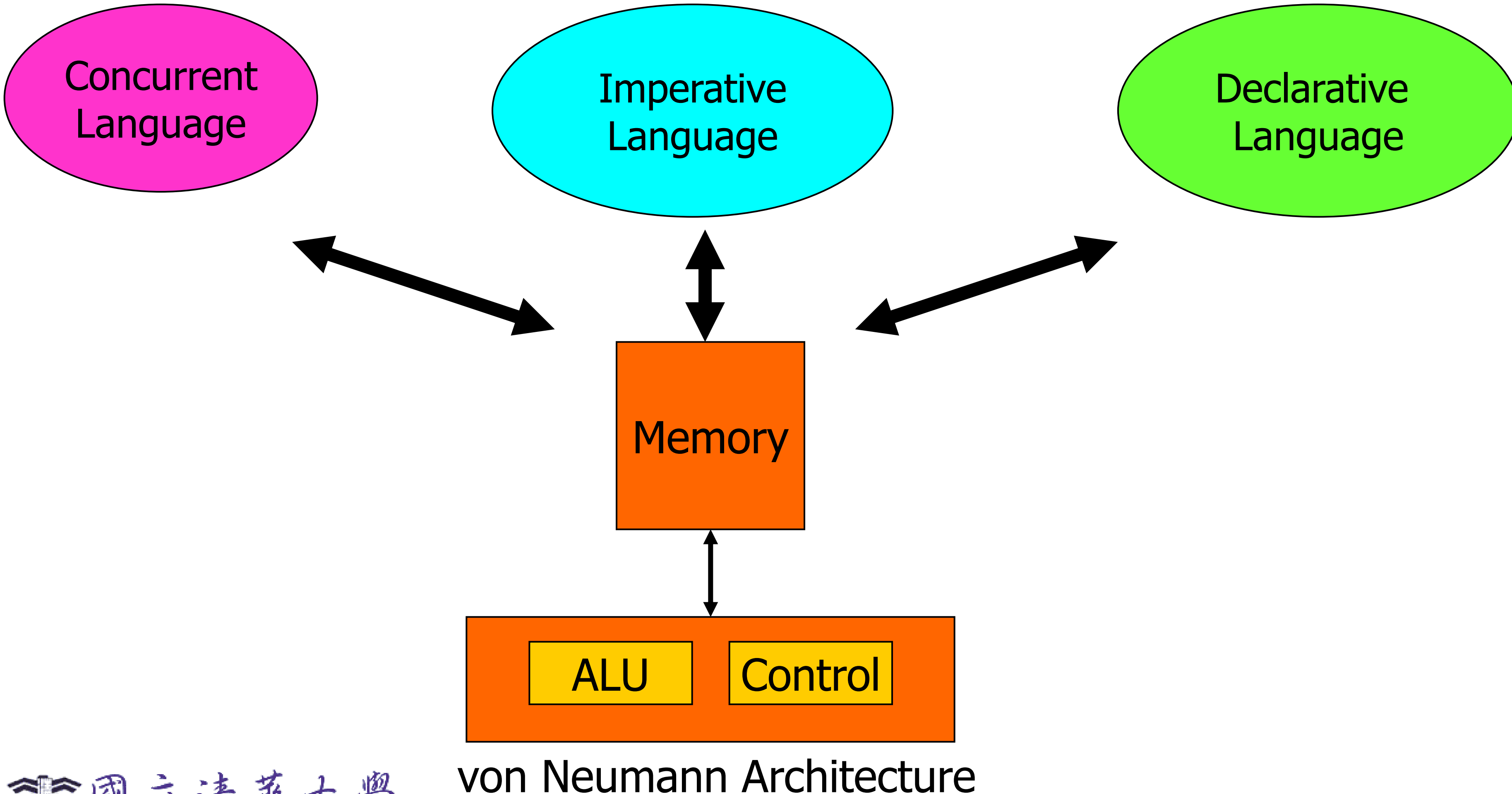
$$0! = 1$$

Lisp (Functional)

```lisp
(defun factorial (n)
  (if (<= n 0)
      1
      (* n (factorial (- n 1)))))

(print (factorial 5))  ; แสดงผล 120
```

Summary: Language Categories

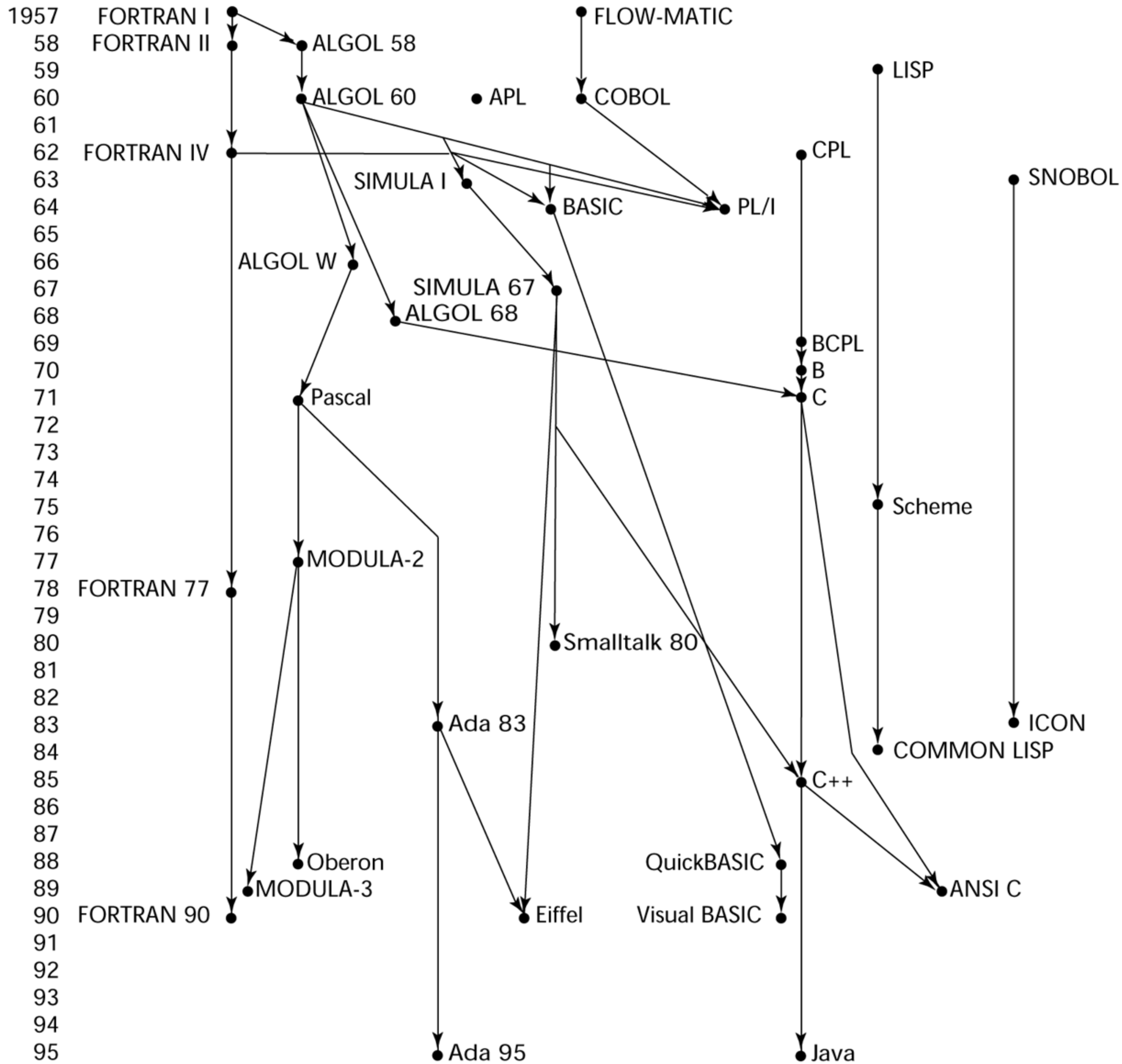


Outline

- Three models of programming languages
- A brief history
  - Programming design methodologies in perspective (Sec. 1.4.2)
- Language evaluation criteria (Sec. 1.3)
  - Language design trade-offs (Sec. 1.6)
- Implementation methods (Sec. 1.7)

Genealogy of Common Languages

(Fig. 2.1)



國立清

National Tsing

Outline

- Three models of programming languages
- A brief history
  - Programming design methodologies in perspective (Sec. 1.4.2)
- Language evaluation criteria (Sec. 1.3)
  - Language design trade-offs (Sec. 1.6)
- Implementation methods (Sec. 1.7)

What Make a Good PL?

Language evaluation criteria:

- **Readability**: the ease with which programs can be read and understood
- **Writability**: the ease with which a language can be used to create programs
- **Reliability**: a program performs to its specifications under all conditions
- **Cost**

Readability & Writability

```
void example() {
  int[] intList = new int[100];
  String[] strList =  new  String[100];
  . . .
  for  (index = 0; index < 100; index++) {
    . . .
  }
  . . .
  for  (index = 0; index < 100; index++) {
    . . .
  }
  . . .
  average = sum / 100;
  . . .
}
```

```
void example() {
  final int  len = 100;
  int[] intList =  new int[len];
  String[] strList =  new  String[len];
  . . .
  for (index = 0; index < len; index++) {
    . . .
  }
  . . .
  for (index = 0; index < len; index++) {
    . . .
  }
  . . .
  average = sum / len;
  . . .
}
```

View of Analysis Readability and writability

1.Developer’s Perspective

2.Caller’s Perspective

main.cpp

```
#include <stdio.h>

int addInt(int a, int b) {
    return a + b;
}

double addDouble(double a, double b) {
    return a + b;
}

int main() {
    int x = addInt(3, 4);
    double y = addDouble(2.5, 1.2);
    return 0;
}
```

```
int add(int a, int b) {
    return a + b;
}

double add(double a, double b) {
    return a + b;
}

int main() {
    int x = add(3, 4);
    double y = add(2.5, 1.2);
    return 0;
}
```

Overload function

```
#include <stdio.h>

// Template function for adding tw
template <typename T>
T add(T a, T b) {
    return a + b;
}

int main() {
    int x = add(3, 4);          //
    double y = add(2.5, 3.7);   //
    return 0;
}
```

template

國立清華大學
National Tsing Hua University

Reliability

```cpp
#include <iostream>


int main() {
    int x =  20;
    int *px = &x;
    std::cout << "\n1.  x = "  << x;
    std::cout << "\n2. *x = "  << *px;
    *px = 100;
    std::cout << "\n3.  x = "  << x;
    std::cout << "\n4. *x = "  << *px;

    return 0;
}
```

```
1.  x = 20
2. *x = 20
3.  x = 100
4. *x = 100
```

Report template

- อธิบาย code (อาจจะไม่มีก็ได้)
- วิเคราะห์ readability and writability (developer views) (กรณีมี subroutine) in source code context.
- วิเคราะห์ readability and writability (caller views) in source code context.

Cost

- Training programmers to use language
- Writing programs (closeness to particular applications)
- Compiling programs
- Executing programs: run-time type checking
- Language implementation system: availability of free compilers
- Reliability: poor reliability leads to high costs
- Maintaining programs

Outline

- Three models of programming languages
- A brief history
  - Programming design methodologies in perspective (Sec. 1.4.2)
- Language evaluation criteria (Sec. 1.3)
  - Language design trade-offs (Sec. 1.6)
- Implementation methods (Sec. 1.7)

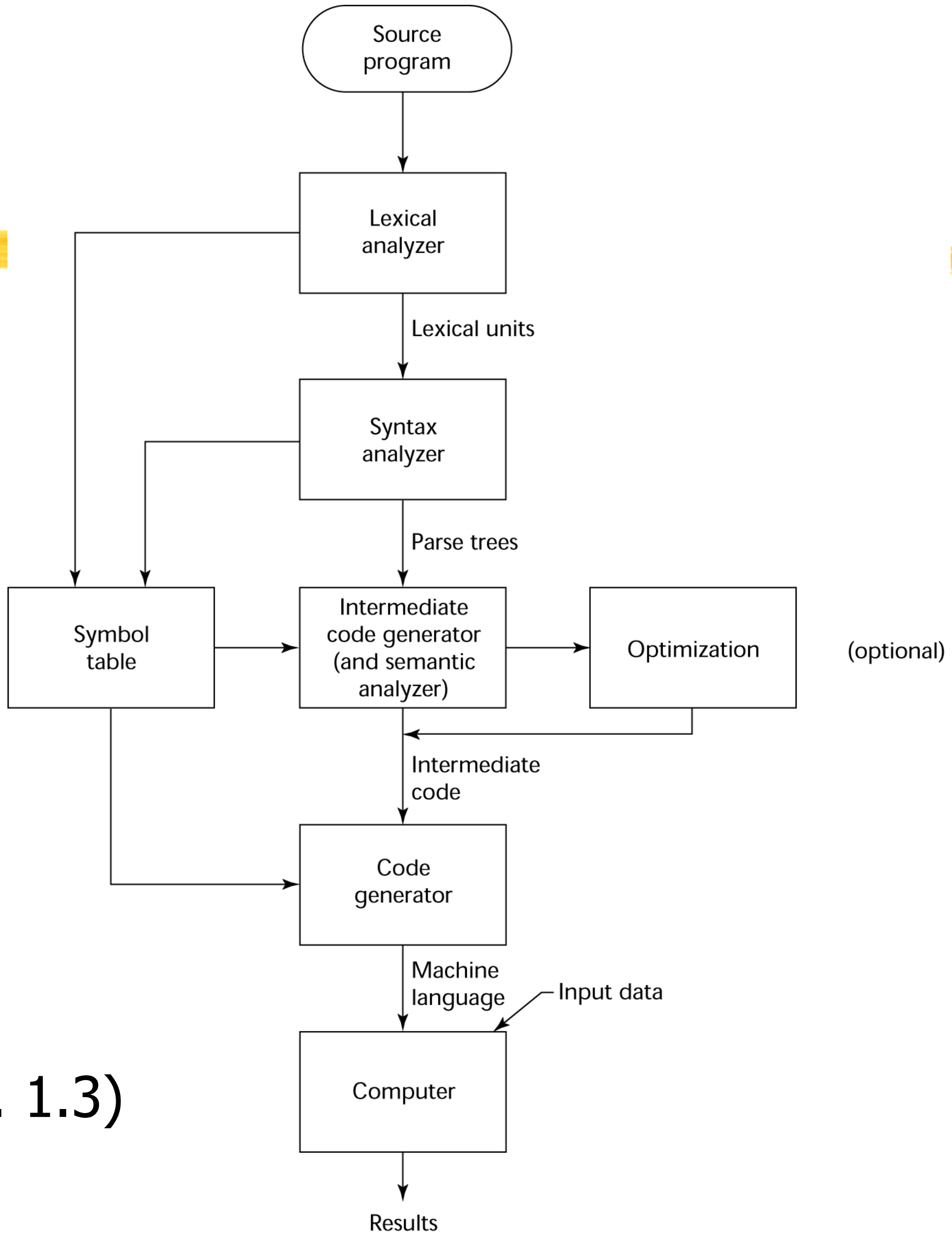
Implementations of PL

- It is important to understand how features and constructs of a programming language, e.g., subroutine calls, are implemented
  - Implementation of a PL construct means its realization in a lower-level language, e.g. assembly → mapping/translation from a high-level language to a low-level language
  - Why the need to know implementations? Understand whether a construct may be implemented efficiently, know different implementation methods and their tradeoffs, etc.

Implementation by Compilation

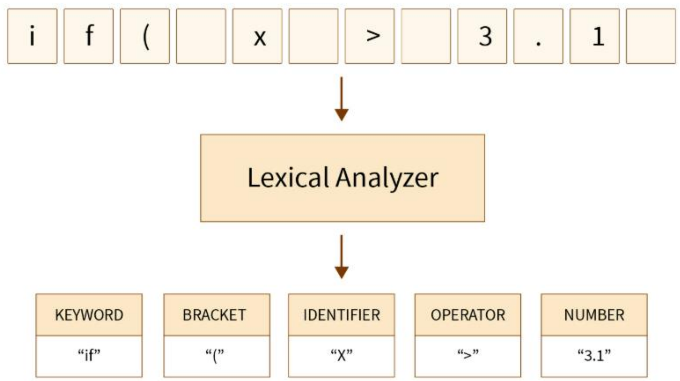
- Translate a high-level program into equivalent machine code automatically by another program (compiler)
- Compilation process has several phases:
  - Lexical analysis: converts characters in the source program into lexical units
  - Syntax analysis: transforms lexical units into parse trees which represent syntactic structure of program
  - Semantics analysis: generate intermediate code
  - Code generation: machine code is generated
  - Link and load

Compilation Process

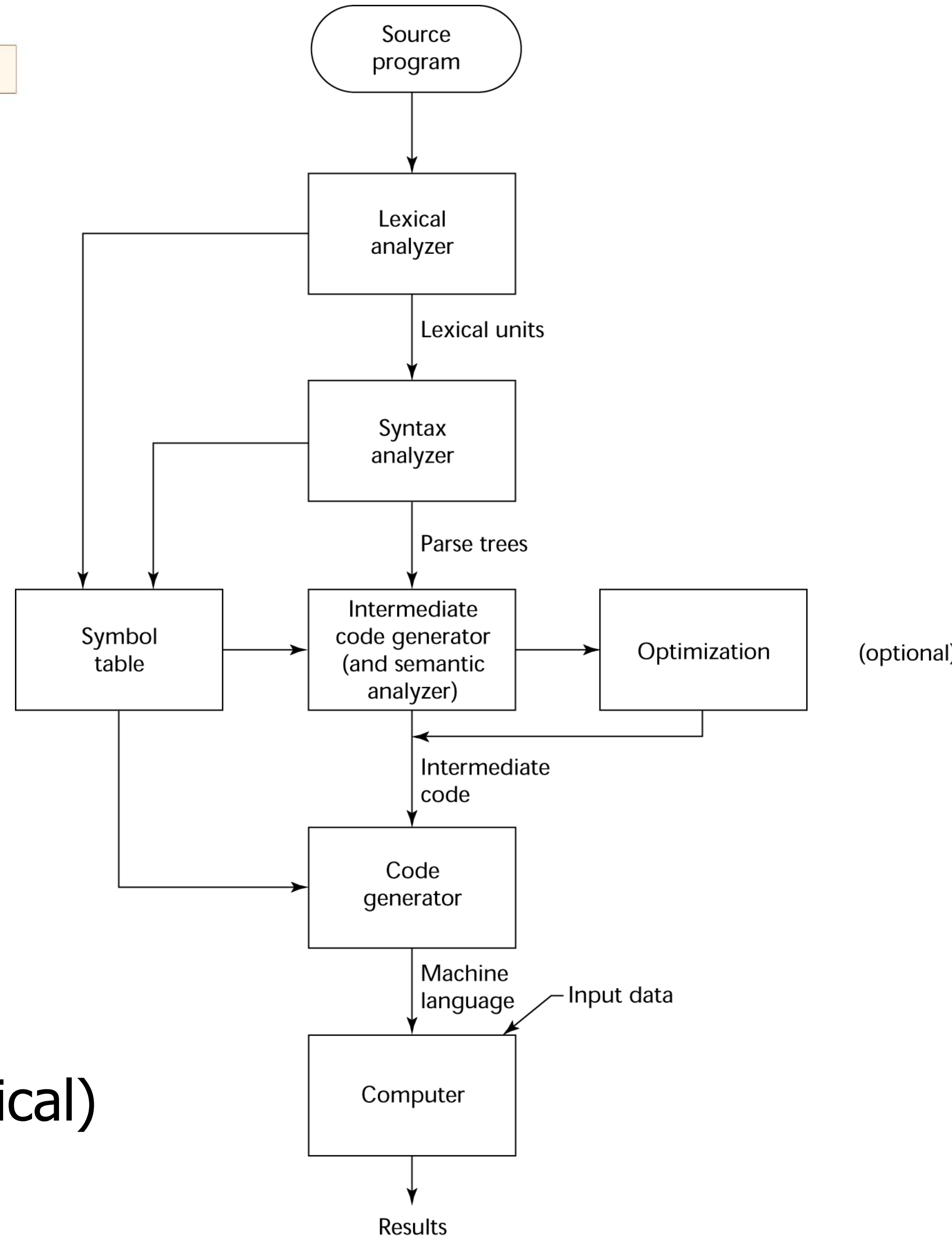


(Fig. 1.3)

國立清華大學 National Tsing Hua University



Token
1.Identifer
2.Constant value
3.Reserved word
4.Operator symbol



(lexical)

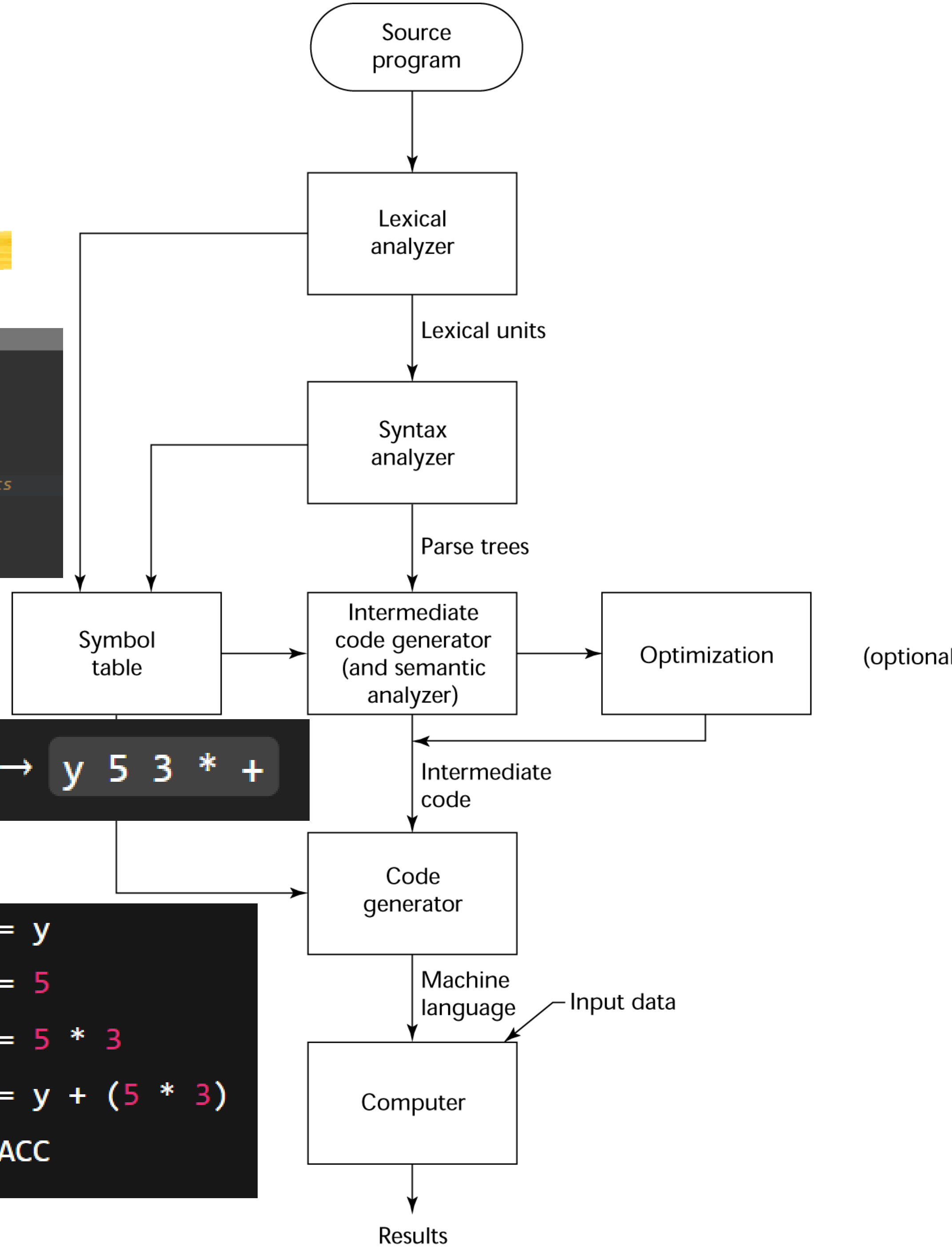
National Tsing Hua University

```
main.cpp
#include <stdio.h>

int main() {

    int x = 10;         // 1) Correct grammar + correct semantics
    int y = ;           // 2) Incorrect grammar (syntax error)
    int z = "123";      // 2) Correct grammar but incorrect semantics

    return 0;
}
```




```
y + 5 * 3 → y 5 3 * +
```

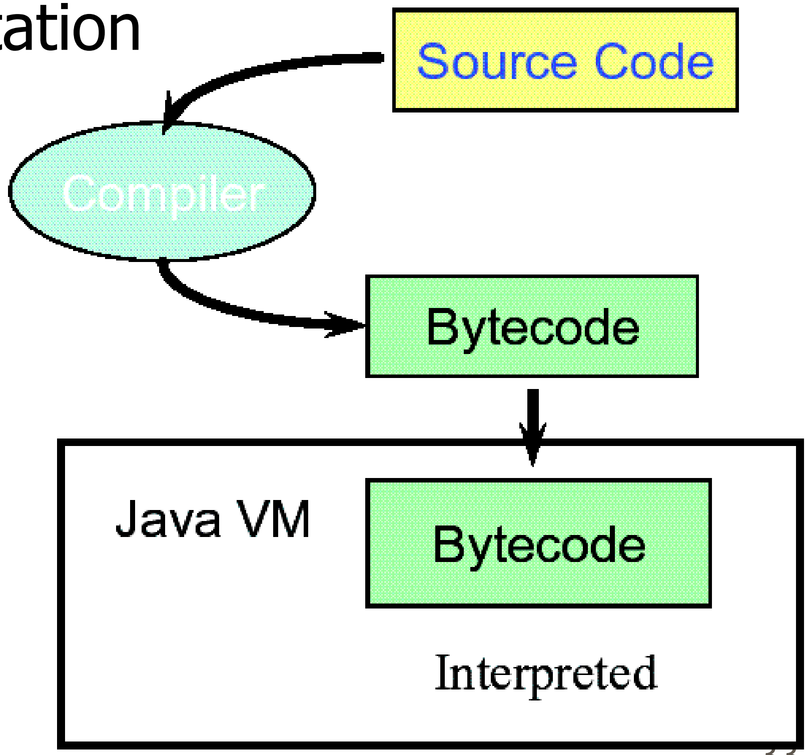
```
LOAD   y    ; ACC = y
LOAD   5    ; ACC = 5
MUL    3    ; ACC = 5 * 3
ADD    y    ; ACC = y + (5 * 3)
STORE  x    ; x = ACC
```

Implementation by Interpretation

- Program interpreted by another program (interpreter) without translation
  - Interpreter acts a simulator or virtual machine
- Easier implementation of programs (run-time errors can easily and immediately displayed)
- Slower execution (10 to 100 times slower than compiled programs)
- Often requires more space
- Popular with some Web scripting languages (e.g., JavaScript)

Hybrid Implementation Systems

- A high-level language program is translated to an intermediate language that allows easy interpretation
  - Faster than pure interpretation



Summary

- Most important criteria for evaluating programming languages include:
  - Readability, writability, reliability, cost
- Major influences on language design have been application domains, machine architecture and software development methodologies
- The major methods of implementing programming languages are: compilation, pure interpretation, and hybrid implementation