

Programming Languages

Preliminaries



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

เป้าหมายของวิชา Programming Language คือการทำให้นักศึกษาเข้าใจหลักการของภาษาโปรแกรม สามารถเปรียบเทียบภาษาได้อย่างมีเหตุผล เรียนรู้แนวคิดการออกแบบภาษา พัฒนาทักษะการเขียนโปรแกรม และเรียนรู้ภาษาใหม่ได้ง่ายขึ้น รวมถึงเข้าใจการทำงานของภาษาในระดับ compiler และ runtime

เป้าหมายของวิชา Programming Language (Course Objectives)

1. ทำให้นักศึกษาเข้าใจหลักการพื้นฐานของภาษาโปรแกรม

เพื่อให้รู้ว่า ทุกภาษา มีโครงสร้างร่วมกัน เช่น

- syntax (ไวยากรณ์)
- semantics (ความหมายของคำสั่ง)
- type system (ระบบชนิดข้อมูล)
- control structures (โครงสร้างควบคุม)
- data abstraction (นามธรรมของข้อมูล)

เป้าหมาย: เข้าใจ “ศาสตร์” ไม่ใช่แค่ “ภาษา”

✓ 2. สามารถวิเคราะห์และเปรียบเทียบภาษาโปรแกรมได้

นักศึกษาควรสามารถตอบได้ว่า

- ภาษาไหนอ่านง่ายกว่า? (readability)
- ภาษาไหนเขียนงานประเภทนี้ได้เหมาะกว่า? (writability)
- ภาษาไหนปลอดภัยกว่า? (safety + type checking)
- ภาษาไหนมีประสิทธิภาพกว่า? (efficiency)

เป้าหมาย: เลือกภาษาให้เหมาะกับงาน ไม่ใช่เขียนตามกระแส

✓ 3. เข้าใจแนวคิดการออกแบบภาษาโปรแกรม (language design principles)

เช่น

- ทำไมภาษา Python ถึงไม่มี type เดิม ๆ แบบ C?
- ทำไม Java ต้องมี garbage collection?
- ทำไม C ต้องกำหนด memory แบบ low-level?

เป้าหมาย: เข้าใจเหตุผลเบื้องหลังการออกแบบ

GLOBAL
EDITION



Concepts of Programming Languages

ELEVENTH EDITION

Robert W. Sebesta



Na

ALWAYS LEARNING

PEARSON

```

1  #include <stdio.h>
2
3
4  //chapter 9
5  double addDouble(double a, double b) {
6      try { //chapter 14 (try-catch)
7          a = a/b;
8      }
9      catch (...) {
10         cout << "divide by zero"
11     }
12     return a + b;
13 }
14
15 int main() {
16
17     //chapter 5+6
18     int x=5;
19     double y;
20
21     //chapter 7
22     x = y*6+1*x;
23
24     //chapter 8
25     if( x<10) {
26         //chapter 9
27         y = addDouble(2.5, 1.2);
28     }
29
30     //chapter 9
31     return 0;
32 }
33
34

```

Chapter 5	Names, Bindings, and Scopes	221
5.1	Introduction.....	222
5.2	Names.....	223
Chapter 6	Data Types	259
6.1	Introduction.....	260
6.2	Primitive Data Types.....	262
6.3	Character String Types.....	266
Chapter 7	Expressions and Assignment Statements	325
7.1	Introduction.....	326
7.2	Arithmetic Expressions.....	326
7.3	Overloaded Operators.....	335
7.4	Type Conversions.....	337
Chapter 8	Statement-Level Control Structures	353
8.1	Introduction.....	354
8.2	Selection Statements.....	356
8.3	Iterative Statements.....	367
Chapter 9	Subprograms	389
9.1	Introduction.....	390
9.2	Fundamentals of Subprograms.....	390
9.3	Design Issues for Subprograms.....	398
Chapter 10	Implementing Subprograms	441
10.1	The General Semantics of Calls and Returns.....	442
10.2	Implementing "Simple" Subprograms.....	443
Chapter 14	Exception Handling and Event Handling	621
14.1	Introduction to Exception Handling.....	622
	History Note.....	626
14.2	Exception Handling in C++.....	628


```

class Stack {
public:
    Stack();           // const
    ~Stack();          // destr

    void push(int value);
    void pop();

    int top() const;
    bool isEmpty() const;
    int size() const;

private:
    int* data;
    int capacity;
    int topIndex;
};

```

Chapter 11	Abstract Data Types and Encapsulation Constructs	471
11.1	The Concept of Abstraction.....	472
11.2	Introduction to Data Abstraction.....	473
11.3	Design Issues for Abstract Data Types.....	476
Chapter 12	Support for Object-Oriented Programming	513
12.1	Introduction.....	514
12.2	Object-Oriented Programming	515
12.3	Design Issues for Object-Oriented Languages.....	519
12.4	Support for Object-Oriented Programming in Specific Languages	524

EXAMPLE 3.1**A Grammar for a Small Language**

```

<program> → begin <stmt_list> end
<stmt_list> → <stmt>
               | <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
               | <var> - <var>
               | <var>

```

Chapter 3 Describing Syntax and Semantics**133**

3.1	Introduction.....	134
3.2	The General Problem of Describing Syntax	135
3.3	Formal Methods of Describing Syntax	137
3.4	Attribute Grammars.....	152
	History Note	152
3.5	Describing the Meanings of Programs: Dynamic Semantics.....	158
	History Note	166
	Summary • Bibliographic Notes • Review Questions • Problem Set.....	179

The language described by the grammar of Example 3.1 has only one statement form: assignment. A program consists of the special word **begin**, followed by a list of statements separated by semicolons, followed by the special word **end**. An expression is either a single variable or two variables separated by either a + or - operator. The only variable names in this language are A, B, and C.

A derivation of a program in this language follows:

```

<program> => begin <stmt_list> end
           => begin <stmt> ; <stmt_list> end
           => begin <var> = <expression> ; <stmt_list> end
           => begin A = <expression> ; <stmt_list> end
           => begin A = <var> + <var> ; <stmt_list> end
           => begin A = B + <var> ; <stmt_list> end
           => begin A = B + C ; <stmt_list> end
           => begin A = B + ; <stmt> end
           => begin A = B + C ; <var> = <expression> end
           => begin A = B + C ; B = <expression> end
           => begin A = B + C ; B = <var> end
           => begin A = B + C ; B = C end

```

