

编译原理

第2讲 高级程序设计语言概述

高级程序设计语言概述

- ▶ 常用的高级程序设计语言
- ▶ 程序设计语言的定义
- ▶ 高级程序设计语言的一般特性

编译原理

高级程序设计语言概述
——常用的高级程序设计语言

常用的高级程序设计语言

语言	特点
FORTRAN	数值计算
COBOL	事务处理
PASCAL	结构化程序设计
LISP	函数式程序设计
PROLOG	逻辑程序设计
C	系统程序设计
Smalltalk	面向对象程序设计
Java	Internet应用, 可移植性
Python	解释型

A language that doesn't affect the way you think about programming, is not worth knowing.

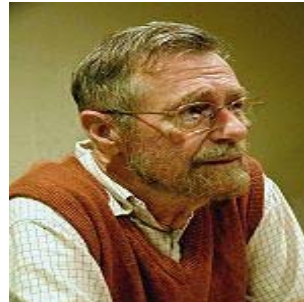


Alan J. Perlis

ACM图灵奖



Alan J. Perlis



Edsger W. Dijkstra



John W. Backus



Kenneth E. Iverson

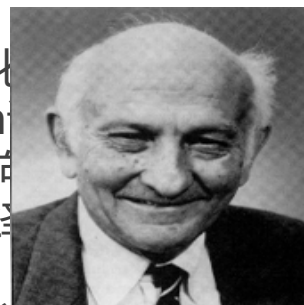
- ▶ Alan J. Perlis (1966) -- ALGOL
- ▶ Edsger Wybe Dijkstra (1972) -- ALGOL
- ▶ Michael O. Rabin & Dana S. Scott (1976) -- 非确定自动机
- ▶ John W. Backus (1977) -- FORTRAN
- ▶ Kenneth Eugene Iverson (1979) -- APL程序语言
- ▶ Niklaus Wirth (1984) -- PASCAL
- ▶ John Cocke (1987) -- RISC & 编译优化
- ▶ O. Dahl, K.Nygaard (2001) -- Simula
- ▶ Alan Kay(2003) -- SmallTalk语言和面向对象
- ▶ Peter Naur(2005) -- ALGOL60以及编译器
- ▶ Frances E. Allen(2006)-- 优化编译器
- ▶ Barbara Liskov(2008)-- 编程语言和系统设计的实践与理论



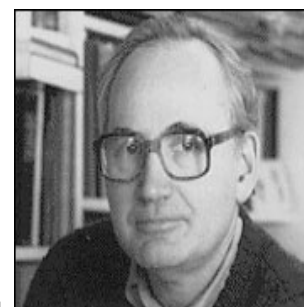
K.Nygaard



O. Dahl



John Cocke



Dana S. Scott



Michael O. Rabin



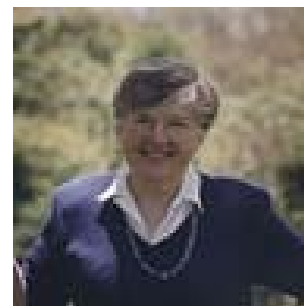
Niklaus Wirth



Donald E. Knuth



Barbara Liskov



Frances E. Allen



Peter Naur



Alan Kay

高级程序设计语言的优点

- ▶ 相对机器语言或汇编语言，高级程序设计语言
 - ▶ 更接近于数学语言和工程语言，更直观、自然和易于理解
 - ▶ 更容易验证其正确性、改错
 - ▶ 编写程序的效率更高
 - ▶ 更容易移植

编译原理

高级程序设计语言概述 ——程序设计语言的定义

高级程序设计语言概述

- ▶ 常用的高级程序设计语言
- ▶ 程序设计语言的定义
- ▶ 高级程序设计语言的一般特性

测试

▶ 下面哪种说法正确？（ ）

A. 标识符是语义概念，名字是语法概念

B. 标识符是语法概念，名字是语义概念

程序语言的定义

- ▶ 语法
- ▶ 语义
- ▶ 语用

语法

- ▶ 程序本质上是一定字符集上的字符串
- ▶ **语法**：一组规则，用它它可以形成和产生一个**合式(well-formed)**的程序

语法

▶ 词法规则：单词符号的形成规则

- ▶ 单词符号是语言中具有独立意义的最基本结构
- ▶ 一般包括：常数、标识符、基本字、算符、界符等
- ▶ 描述工具：有限自动机

▶ 语法规则：语法单位的形成规则

- ▶ 语法单位通常包括：表达式、语句、分程序、过程、函数、程序等;
- ▶ 描述工具：上下文无关文法

语法

$E \rightarrow i$

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

- ▶ 语法规则和词法规则定义了程序的形式结构
- ▶ 定义语法单位的意义属于语义问题

语义

▶ 语义

- ▶ 一组规则，用它可以定义一个程序的意义

▶ 描述方法

▶ 自然语言描述

- ▶ 二义性、隐藏错误和不完整性

▶ 形式描述

- ▶ 操作语义
- ▶ 指称语义
- ▶ 代数语义

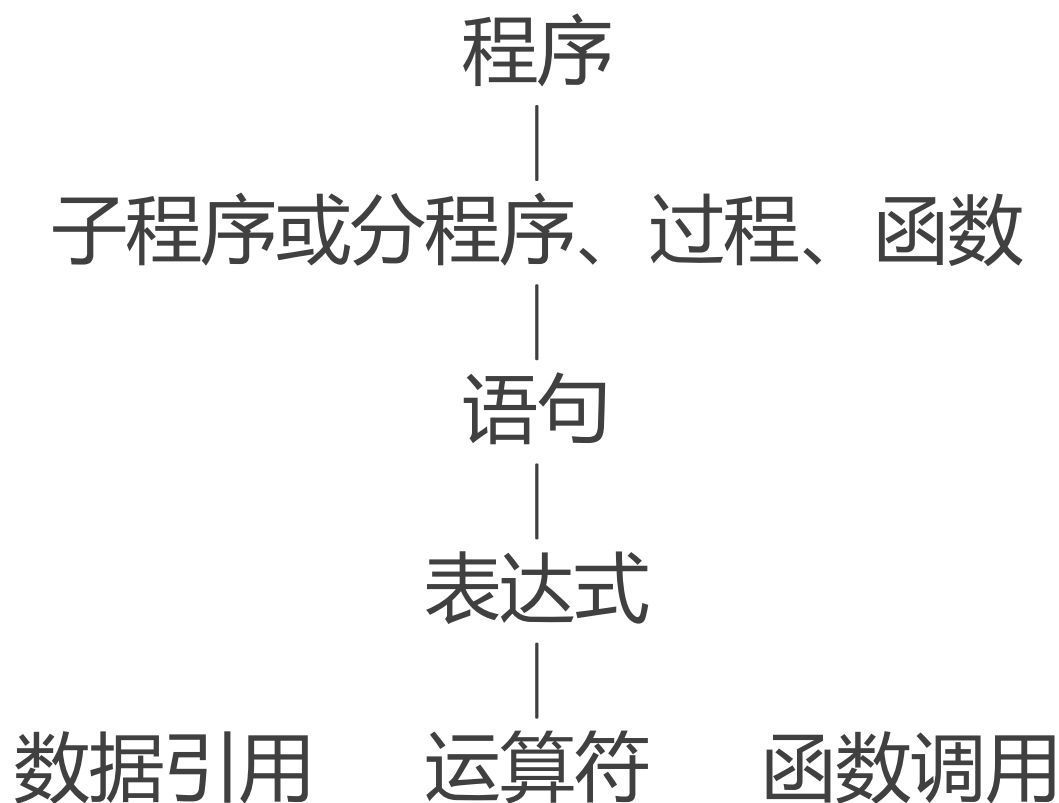
测试

- ▶ 下面哪些属于程序语言的语义定义？（ ）
- A. 表达式中圆括号必须匹配
 - B. 类的声明必须以class开头
 - C. 关于函数调用时参数传递方法的描述
 - D. 函数体必须用return语句结尾

程序语言的基本功能和层次结构

- ▶ 程序，本质上说是描述一定数据的处理过程
- ▶ 程序语言的基本功能
 - ▶ 描述数据和对数据的运算

程序的层次结构




程序语言成分的逻辑和实现意义

- ▶ 抽象的逻辑的意义

 - ▶ 数学意义

- ▶ 计算机实现的意义

 - ▶ 具体实现



计算思维与数学思维
的不同

编译原理

高级程序设计语言概述 ——高级程序设计语言的一般特性

高级程序设计语言概述

- ▶ 常用的高级程序设计语言
- ▶ 程序设计语言的定义
- ▶ 高级程序设计语言的一般特性

高级语言的一般特性

- ▶ 高级语言的分类
- ▶ 程序结构
- ▶ 数据结构与操作
- ▶ 语句与控制结构

编译原理

高级程序设计语言概述 ——高级语言的分类

高级语言的一般特性

- ▶ 高级语言的分类
- ▶ 程序结构
- ▶ 数据结构与操作
- ▶ 语句与控制结构

高级语言的分类

- ▶ 强制式语言(Imperative Language)/过程式语言
- ▶ 应用式语言(Applicative Language)
- ▶ 基于规则的语言(Rule-based Language)
- ▶ 面向对象语言(Object-Oriented Language)

高级语言的分类

- ▶ 强制式语言(Imperative Language)/过程式语言
 - ▶ 命令驱动，面向语句
 - ▶ FORTRAN、C、Pascal, Ada

高级语言的分类

- ▶ 强制式语言(Imperative Language)/过程式语言
- ▶ 应用式语言(Applicative Language)
 - ▶ 注重程序所表示的功能，而不是一个语句接一个语句地执行
 - ▶ LISP、ML

高级语言的分类

- ▶ 强制式语言(Imperative Language)/过程式语言
- ▶ 应用式语言(Applicative Language)
- ▶ 基于规则的语言(Rule-based Language)
 - ▶ 检查一定的条件，当它满足值，则执行适当的动作
 - ▶ Prolog

高级语言的分类

- ▶ 强制式语言(Imperative Language)/过程式语言
- ▶ 应用式语言(Applicative Language)
- ▶ 基于规则的语言(Rule-based Language)
- ▶ 面向对象语言(Object-Oriented Language)
 - ▶ 封装、继承和多态性
 - ▶ Smalltalk, C++, Java

编译原理

高级程序设计语言概述 ——程序结构

高级语言的一般特性

- ▶ 高级语言的分类
- ▶ 程序结构
- ▶ 数据结构与操作
- ▶ 语句与控制结构

程序结构

► FORTRAN

- 一个程序由一个主程序段和若干辅程序段组成
- 辅程序段可以是子程序、函数段或数据块
- 每个程序段由一系列的说明语句和执行语句组成，各段可以独立编译
- 模块结构，没有嵌套和递归
- 各程序段中的名字相互独立，同一个标识符在不同的程序段中代表不同的名字

主程序

```
PROGRAM ...  
...  
end
```

辅程序1

```
SUBROUTINE ...  
...  
end
```

辅程序2

```
FUNCTION ...  
...  
end
```

程序结构

▶ PASCAL

- ▶ PASCAL程序本身可以看成是一个操作系统调用的过程，过程可以嵌套和递归
- ▶ 一个PASCAL过程

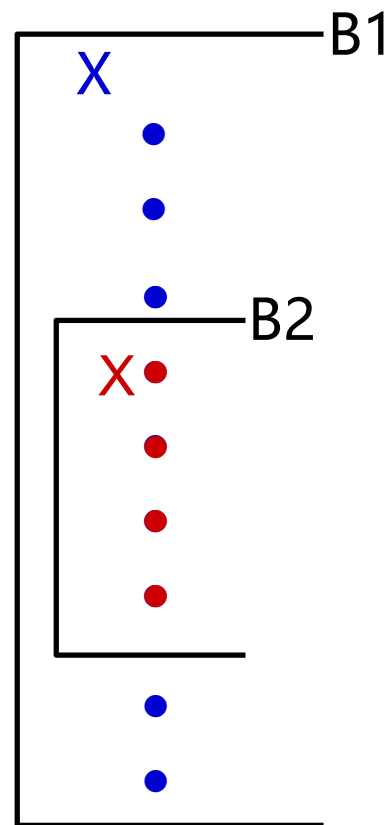
```
过程头;  
    说明段（由一系列的说明语句组成）;  
begin  
    执行体（由一系列的执行语句组成）;  
end
```

作用域

- ▶ 同一个标识符在不同过程中代表不同的名字
- ▶ **作用域**：一个名字能被使用的区域范围
- ▶ 名字作用域规则——"**最近嵌套原则**"

最近嵌套原则

- ▶ 一个在子程序B1中说明的名字X只在B1中有效（局部于B1）
- ▶ 如果B2是B1的一个内层子程序且B2中对标识符X没有新的说明，则原来的名字X在B2中仍然有效
- ▶ 如果B2对X重新作了说明，那么，B2对X的任何引用都是指重新说明过的这个X



	A(real)	A(integer)	B(real)	B(bool)
program main				
var A, B:real;				
...				
procedure P1				
var B:boolean;				
...				
begin				
...				
end				
procedure P2				
var A:integer;				
...				
begin				
...				
end				
begin				
...				
end				

测试

```
program main
  var A, B:real;
  ...
  procedure P1
    var B:boolean;
    ...
    begin
    ...
    end
  procedure P2
    var A:integer;
    ...
    begin
    ...
    end
begin
  ...
end
```

P2的代码能够调用
P1吗?

- A. 可以
- B. 不可以
- C. 说不清

程序结构

▶ JAVA

▶ 面向对象的高级语言

- ▶ 类 (Class)

- ▶ 继承(Inheritance)

- ▶ 多态性(Polymorphism)和动态绑定(Dynamic binding)

JAVA 程序示例

```
class Car{
    int color;
    int door;
    int speed;
    ...
    public push_break ( ) {
    ...
    }
    public add_oil ( ) {
    ...
    }
}

class Trash_Car extends car {
    double amount;
    public fill_trash ( ) {
    ...
    }
}
```

编译原理

高级程序设计语言概述
——数据结构与操作

高级语言的一般特性

- ▶ 高级语言的分类
- ▶ 程序结构
- ▶ 数据结构与操作
- ▶ 语句与控制结构

数据类型与操作

- ▶ 数据类型通常包括三要素
 - ▶ 用于区别这种类型数据对象的属性
 - ▶ 这种类型的数据对象可以具有的值
 - ▶ 可以作用于这种类型的数据对象的操作

数据类型与操作

▶ 初等数据类型

▶ 数值类型

- ▶ 整型、实型、复数、双精度

- ▶ 运算：+，-，*，/等

▶ 逻辑类型

- ▶ true、false

- ▶ 布尔运算： \vee ， \wedge ， \neg 等

▶ 字符类型：符号处理

▶ 指针类型

编译原理

高级程序设计语言概述 ——标识符与名字

测试：标识符与名字

▶ 下面哪种说法正确？（ ）

- A. 标识符是语义概念，名字是语法概念
- B. 标识符是语法概念，名字是语义概念

标识符与名字

- ▶ 标识符

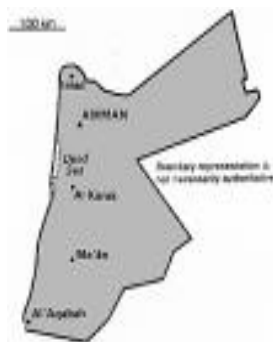
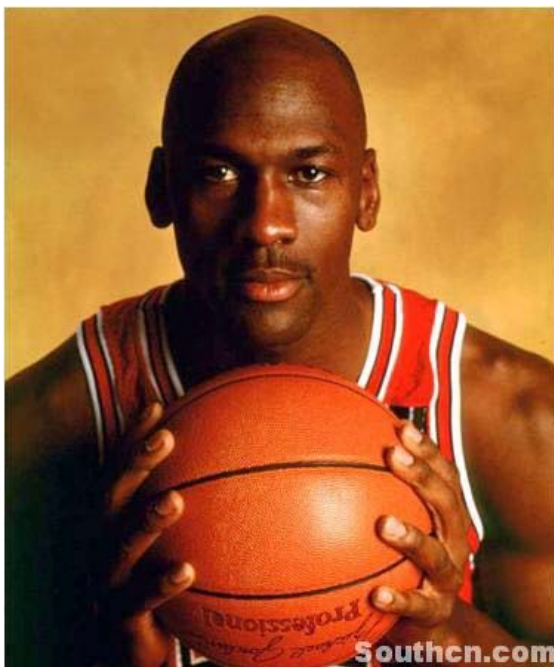
- ▶ 以字母开头的，由字母数字组成的字符串

- ▶ 名字

- ▶ 标识程序中的对象

Jordan

标识符 $\xrightarrow{\text{binding}}$ 名字
绑定



标识符与名字

- ▶ 名字的意义和属性

- ▶ 值：单元中的内容
- ▶ 属性：类型和作用域

- ▶ 名字的说明方式

- ▶ 由说明语句来明确规定的
 - ▶ `int score`
- ▶ 隐含说明
 - ▶ FORTRAN 以I,J,K,...N为首的名字代表整型,否则为实型
- ▶ 动态确定
 - ▶ 走到哪里，是什么，算什么

测试

▶ 下面说法的是错误的是()

A. 名字的绑定(binding)是指将标识符与所代表的程序数据或代码进行关联

B.名字的绑定总是发生在编译过程中

C.名字的绑定可以发生在运行过程中

▶ 你能举几个静态绑定和动态绑定例子吗?

标识符与名字

▶ 标识符

- ▶ 以字母开头的，由字母数字组成的字符串

▶ 标识符与名字两者有本质区别

- ▶ 标识符是语法概念
- ▶ 名字有确切的意义和属性

编译原理

高级程序设计语言概述
——数据结构

数据结构

▶ 数组

- ▶ 逻辑上，数组是由同一类型数据组成的某种n维矩形结构，沿着每一维的距离，称为下标
- ▶ 数组可变与不可变
 - ▶ 编译时能否确定其存储空间的大小
- ▶ 访问
 - ▶ 给出数组名和下标值，如 $A[10, i + j]$
- ▶ 存放方式
 - ▶ 按行存放,按列存放

数组元素地址计算

- ▶ 数组A[10,20]的A[1, 1]的地址为a, 每个元素占1字节, 各维下标从1开始, 按行存放, 那么A[i, j]地址为:

$$a + (i-1)*20 + (j-1)$$

- ▶ 通用的数组元素地址计算公式

数组元素地址计算

- ▶ 设A为n维数组，按行存放，每个元素宽度为w
 - ▶ low_i 为第i维 的下界
 - ▶ up_i 为第i维 的上界
 - ▶ n_i 为第i维 可取值的个数($n_i = up_i - low_i + 1$),
 - ▶ base为A的第一个元素相对地址

- ▶ 元素 $A[i_1, i_2, \dots, i_k]$ 相对地址公式

$$((\dots i_1 n_2 + i_2) n_3 + i_3) \dots n_k + i_k) \times w +$$

$$\text{base} - \frac{((\dots ((low_1 n_2 + low_2) n_3 + low_3) \dots) n_k + low_k) \times w}{\text{Con}}$$

Con

内情向量

$$\frac{((\dots i_1 n_2 + i_2) n_3 + i_3) \dots) n_k + i_k) \times w +$$

$$\text{base} - \frac{((\dots ((\text{low}_1 n_2 + \text{low}_2) n_3 + \text{low}_3) \dots) n_k + \text{low}_k) \times w}{\text{Con}}$$

▶ 内情向量

- ▶ 登记维数，各维的上、下限，首地址，以及数组（元素）的类型等信息

low_1	up_1	n_1
low_2	up_2	n_2
...
low_k	up_k	n_k
k	Con	
type	base	

记录

- ▶ 由已知类型的数据组合在一起的一种结构
- ▶ 记录或者结构的元素，也叫做域(field)

```
record { char name[20];
```

```
        integer age;
```

```
        bool married;
```

```
    } cards[1000]
```

- ▶ 访问：复合名 cards[k].name
- ▶ 存储：连续存放
- ▶ 域的地址计算
 - ▶ 相对于记录结构起点的相对数OFFSET

字符串、表格、栈

- ▶ 字符串：符号处理、公式处理
- ▶ 表格：本质上是一种记录结构
- ▶ 线性表：一组顺序化的记录结构
- ▶ 栈：一种线性表，后进先出，POP, PUSH

抽象数据类型

- ▶ 抽象数据类型(Abstract Data Type)
 - ▶ A set of data values and associated operations that are precisely specified independent of any particular implementation.
 - ▶ 抽象数据类型由数据集合、及其相关的操作组成，这些操作有明确的定义，而且定义不依赖于具体的实现。

抽象数据类型

- ▶ 一个抽象数据类型包括
 - ▶ 数据对象集合
 - ▶ 作用于这些数据对象的抽象运算的集合
 - ▶ 这种类型对象的封装，即，除了使用类型中所定义的运算外，用户不能对这些对象进行操作
- ▶ 程序设计语言对抽象数据类型的支持
 - ▶ Ada通过程序包(package)提供了数据封装的支持

ADA 程序示例

```
package STACKS is
  type ELEM is private;
  type STACK is limited private;
  procedure push (S: in out STACK; E: in ELEM);
  procedure pop (S: in out STACK; E: out ELEM);
  ...
end STACK;
```

规范说明

```
package body STACKS is
  procedure push(S: in out STACK; E: in ELEM);
  begin
    .....实现细节
  end push;
  procedure pop (S: in out STACK; E: out ELEM);
  begin
    .....实现细节
  end pop;
end;
```

程序包体

抽象数据类型

- ▶ 一个抽象数据类型包括
 - ▶ 数据对象集合
 - ▶ 作用于这些数据对象的抽象运算的集合
 - ▶ 这种类型对象的封装，即，除了使用类型中所定义的运算外，用户不能对这些对象进行操作
- ▶ 程序设计语言对抽象数据类型的支持
 - ▶ Ada通过程序包(package)提供了数据封装的支持
 - ▶ Smalltalk、C++和Java通过类(Class)对抽象数据类型提供支持

JAVA 程序示例

```
class Car{
    int color_number;
    int door_number;
    int speed;
    ...
    public push_break ( ) {
    }
    public add_oil ( ) {
    }
}

class Trash_Car extends car {
    double amount;
    public fill_trash ( ) {
    }
}
```

编译原理

高级程序设计语言概述 ——语句与控制结构

高级语言的一般特性

- ▶ 高级语言的分类
- ▶ 程序结构
- ▶ 数据结构与操作
- ▶ 语句与控制结构

语句与控制结构

▶ 表达式

- ▶ 表达式由运算量（也称操作数，即数据引用或函数调用）和算符（运算符，操作符）组成

- ▶ 形式：中缀、前缀、后缀

$X*Y$ $-A$ $P\uparrow$ 或者 $p\rightarrow$

- ▶ 表达式形成规则

- ▶ 变量（包括下标变量）、常数是表达式。
- ▶ 若 E_1 、 E_2 为表达式， θ 是一个二元算符，则 $E_1\theta E_2$ 是表达式。
- ▶ 若 E 是表达式， θ 为一元算符，则 θE （或 $E\theta$ ）是表达式。
- ▶ 若 E 是表达式，则 (E) 是表达式。

算符的优先次序

▶ 一般的规定

- ▶ PASCAL: 左结合 $A + B + C = (A + B) + C$
- ▶ FORTRAN: 对于满足左、右结合的算符可任取一种, 如 $A + B + C$ 就可以处理成 $(A + B) + C$, 也可以处理成 $A + (B + C)$

▶ 注意两点

- ▶ 代数性质能引用到什么程度视具体的语言而定
- ▶ 在数学上成立的代数性质在计算机上未必完全成立
 - ▶ $A + B = B + A$

语句

▶ 赋值语句

▶ $A := B$

▶ 名字的左值：该名字代表的存储单元的地址

▶ 名字的右值：该名字代表的存储单元的内容

测试：左值与右值

► 在C语言中，下面选项只具有右值、不具有左值的是（ ）。

A. 变量

B. 下标变量

C. $a + 5$

D. 指针变量P

E. $*P$ (P是指针变量)

语句

► 控制语句

- 无条件转移语句

goto L

- 循环语句

while B do S

repeat S until B

for $i := E_1$ step E_2 until E_3 do S

- 过程调用语句

call $P(X_1, X_2, \dots, X_n)$

- 条件语句

if B then S

if B then S_1 else S_2

- 返回语句

return (E)

语句的分类

▶ 功能

- ▶ 执行语句：描述程序的动作
- ▶ 说明语句：定义各种不同数据类型的变量或运算，定义名字的性质

语句的分类

▶ 形式

- ▶ 简单句：不包含其他语句成分的基本句

`A = B + C ;`

`goto 105 ;`

- ▶ 复合句：句中有句的语句

`while (i >= 0) {`

`j = i * 10;`

`i++;`

`}`

编译原理

高级程序设计语言概述 ——小结

小结

- ▶ 程序语言的定义
 - ▶ 语法
 - ▶ 语义
 - ▶ 程序语言的功能
- ▶ 高级语言的一般特性
 - ▶ 高级语言的分类
 - ▶ 程序结构
 - ▶ 数据结构与操作
 - ▶ 语句与控制结构