

第二章 程序设计语言

学习内容

2.1 语言及其文法

2.2 程序语言定义

2.3 程序语言的构造基础

学习内容

2.1 语言及其文法

2.2 程序语言定义

2.3 程序语言的构造基础

字母表

- ▶ 字母表 Σ 是一个非空有穷符号集合
符号：字母、数、标点符号 ...

- ▶ 以下是不同的字母表：

$\{a, b, c, d\}$

$\{a, b, c, \dots, z\}$

$\{0, 1\}$

ASCII字母表

字母表上的运算

- 字母表 Σ_1 与 Σ_2 的乘积:

$$\Sigma_1\Sigma_2=\{ab|a\in\Sigma_1, b\in\Sigma_2\}$$

例: $\Sigma_1=\{0,1\}$, $\Sigma_2=\{a,b\}$, $\Sigma_1\Sigma_2=\{0a,0b,1a,1b\}$

- 字母表 Σ 的 n 次幂递归地定义为:

(1) $\Sigma^0=\{\epsilon\}$

(2) $\Sigma^n=\Sigma^{n-1}\Sigma \quad n\geq 1$

例: $\Sigma_1^3=\{000,001,010,011,100,101,110,111\}$

字母表上的运算

- 字母表 Σ 的**正闭包**(Positive Closure):

$$\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4 \cup \dots$$

例: $\{0,1\}^+ = \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \dots\}$

- 字母表 Σ 的**克林闭包**(Kleene Closure)为:

$$\Sigma^* = \Sigma^0 \cup \Sigma^+ = \Sigma^0 \cup \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \dots$$

例: $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \dots\}$

串

- **串：** 是字母表中符号的一个有穷序列
 Σ 是一个字母表， $\forall x \in \Sigma^*$ ， x 是 Σ 上的一个串。
- 串 s 的长度： s 中符号的个数，通常记作 $|s|$ ，例： $|aab|=3$
- 空串是长度为0的串，用 ε (*epsilon*) 表示， $|\varepsilon|=0$

串上的运算

- 连接

- ▶ 串 x 和 y 的连接：把 y 附加到 x 后面而形成的串，记作 xy
- ▶ 例如，如果 $x=dog$ 且 $y=house$ ， $xy=doghouse$
- ▶ 空串是连接运算的单位元，对于任何串 s 都有， $\epsilon s = s\epsilon = s$

设 x, y, z 是三个字符串，如果 $x=yz$ ，则称 y 是 x 的前缀， z 是 x 的后缀

串上的运算

- 幂

- 串 s 的 n 次幂:

- (1) $s^0 = \varepsilon$;

- (2) $s^n = s^{n-1}s$ 。

设 x, y, w 是三个字符串，如果 $w = xy$ ，则称 y 是 w 的子串

语言

设 Σ 是一个字母表, $\forall L \subseteq \Sigma^*$, L 称为字母表 Σ 上的一个语言

$\forall x \in L$, x 叫做 L 的一个句子

- 例: 字母表 $\{0, 1\}$ 上的语言

$\{0, 1\}$

$\{00, 11\}$

$\{0, 1, 00, 11\}$

$\{0, 1, 00, 11, 01, 10\}$

$\{00, 11\}^*$

$\{01, 10\}^*$

语言的运算

- Σ_1 和 Σ_2 是字母表, $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, 语言 L_1 与 L_2 的乘积是字母表 $\Sigma_1 \cup \Sigma_2$ 上的一个语言, 该语言定义为:

$$L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\}$$

- 字母表 Σ , $\forall L \in \Sigma^*$, L 的 n 次幂是一个语言, 该语言定义为:
 - (1) 当 $n=0$ 时, $L^n = \{\varepsilon\}$;
 - (2) 当 $n \geq 1$ 时, $L^n = L^{n-1}L$ 。

语言的运算

- ▶ L 的正闭包 L^+ 是一个语言，该语言定义为：
$$L^+ = L \cup L^2 \cup L^3 \cup L^4 \cup \dots$$
- ▶ L 的克林闭包 L^* 是一个语言，该语言定义为：
$$L^* = L^0 \cup L \cup L^2 \cup L^3 \cup L^4 \cup \dots$$

文法定义

文法是用于描述语言的语法结构的形式规则。

任何一种语言都有它自己的文法，不管它是机器语言还是自然语言。

自然语言的文法：主 谓 宾

机器语言也有描述它语言构成的特定文法

文法可以定义为一个四元组。

文法定义

四元组 (V_T, V_N, S, P)

1. 一个终结符号集合 V_T
2. 一个非终结符号集合 V_N
3. 一个产生式集合 P ，定义语法范畴
产生式: $\alpha \rightarrow \beta$ (α 定义为 β)
4. 一个特定的非终结符——开始符号 S

$$V_T \cap V_N = \Phi$$

$V_T \cup V_N$: 文法符号集

产生式: 描述了将终结符和非终结符组合成串的方法

$\alpha \in (V_T \cup V_N)^+$: 称为产生式的头或左部, α 中至少包含 V_N 中的一个元素

$\beta \in (V_T \cup V_N)^*$: 称为产生式的体或右部

符号约定

终结符:

- 字母表中排在前面的小写字母, 如 a 、 b 、 c
- 运算符, 如 $+$ 、 $*$ 等
- 标点符号, 如括号、逗号等
- 数字0、1、...、9
- 粗体字符串, 如 id 、 if 等

非终结符:

- 字母表中排在前面的大写字母, 如 A 、 B 、 C
- 字母 S 。通常表示开始符号
- 小写、斜体的名字, 如 expr 、 stmt 等
- 代表程序构造的大写字母。如 E (表达式)、 T (项)和 F (因子)

符号约定

文法符号:

- 字母表中排在后面的大写字母（如 **X** 、 **Y** 、 **Z** ）表示文法符号（即终结符或非终结符）

终结符号串:

- 字母表中排在后面的小写字母（主要是 **u** 、 **v** 、 **\dots** 、 **z** ）表示终结符号串（包括空串）

文法符号串:

- 小写希腊字母，如 **α** 、 **β** 、 **γ** ，表示文法符号串（包括空串）

除非特别说明，第一个产生式的左部就是开始符号

产生式的简写

对一组有相同左部的 α 产生式 $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$

可以简记为:

$$\alpha \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

读作:

α 定义为 β_1 , 或者 β_2 , ..., 或者 β_n

$\beta_1, \beta_2, \dots, \beta_n$ 称为 α 的候选式

例:

$\begin{aligned} E &\rightarrow E + E \\ E &\rightarrow E * E \\ E &\rightarrow (E) \\ E &\rightarrow \text{id} \end{aligned}$	\Longrightarrow	$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$
--	-------------------	--

产生式设计练习

- 首先是“人会做”

- 我们自己先把语法概念“什么模样”搞清楚

- 然后是“让计算机做”——符号化

- 为这个语法概念起个名字，“模样”中的其他语法概念、单词也都有相应的名字
 - 将语法概念放在产生式左部
“模样”放在产生式右部
都是用名字替换掉语法概念和单词——

产生式设计练习

- **while** (expression) statement

对应的产生式

$stmt \rightarrow \mathbf{while} (expr) stmt$

- **for** (expression₁; expression₂; expression₃)
statement

对应的产生式

$stmt \rightarrow \mathbf{for} (expr ; expr ; expr) stmt$

- **int(float, double, char)** id₁, id₂, ...;

$type \rightarrow \mathbf{int} \mid \mathbf{float} \mid \mathbf{double} \mid \mathbf{char}$

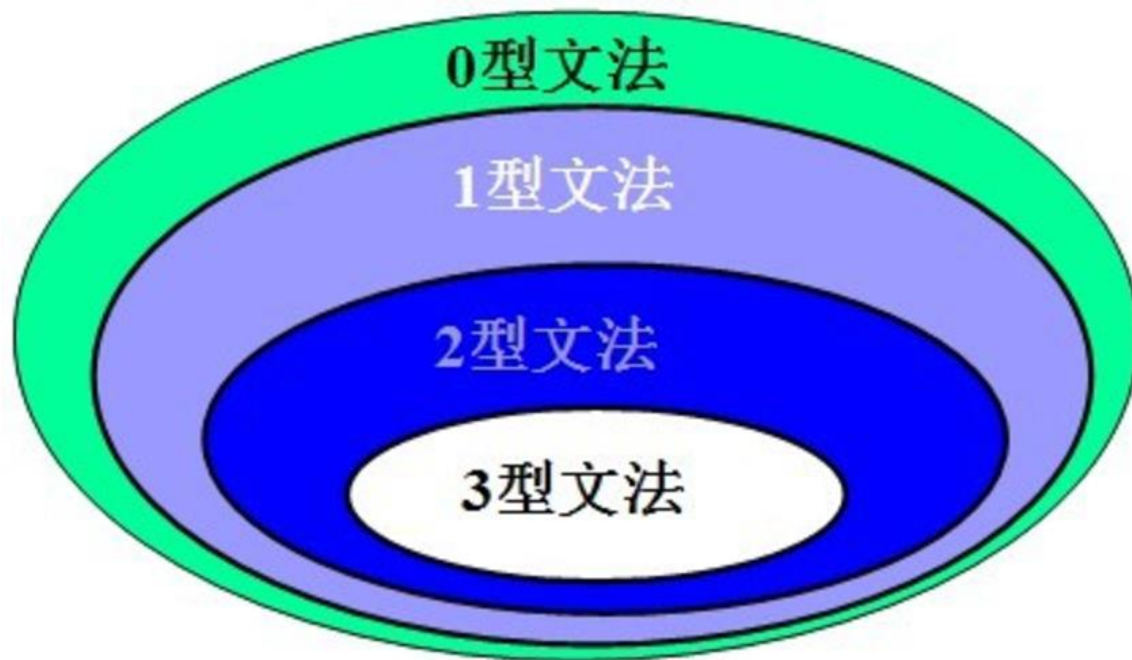
$idlist \rightarrow idlist , \mathbf{id} \mid \mathbf{id}$

$decl \rightarrow type idlist ;$

文法的 4 种类型

1956年，Chomsky建立形式语言的描述。

通过对产生式的施加不同的限制，Chomsky把文法分为4种类型

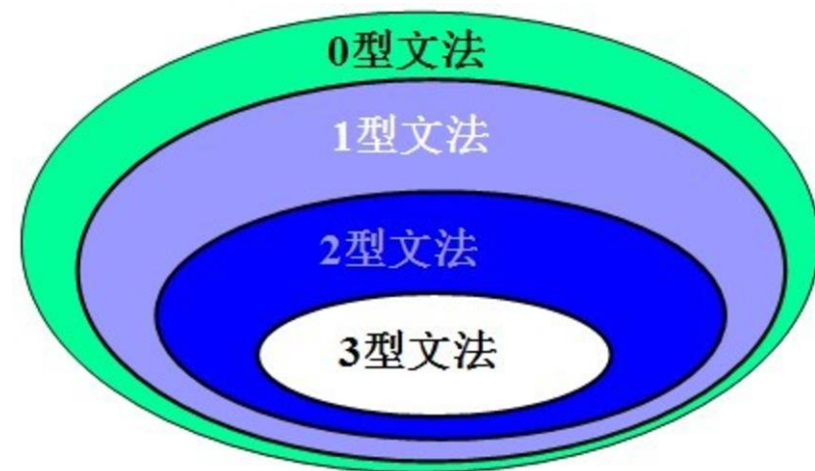


文法的 4 种类型

- 0型文法：也称为短语文法
 $\forall \alpha \rightarrow \beta \in P, \alpha$ 中至少包含1个非终结符
- 0型文法生成的语言：0型语言

任何0型语言都是递归可枚举的；反之，递归可枚举集必定是一个0型语言。

- 0型文法产生式的形式 + 某些限制 = 1型、2型、3型



文法的 4 种类型

- 1型文法（上下文有关文法**context-sensitive**）：

- $\forall \alpha \rightarrow \beta$ ，都有 $|\alpha| \leq |\beta|$ ，仅 $S \rightarrow \varepsilon$ 除外
- 产生式的形式描述：

$$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2 \quad (\alpha_1, \alpha_2, \beta \in (V_N \cup V_T)^*, \beta \neq \varepsilon, A \in V_N)$$

即：A 只有出现在 α_1, α_2 的上下文中，才允许用 β 替换。

- 产生的语言称“上下文有关语言”
- 例如： $0 A 0 \rightarrow 0 1100 0$
- $1 A 1 \rightarrow 1 0101 1$

文法的 4 种类型

- 2型文法(上下文无关文法, context-free grammar. CFG)
 - $\forall \alpha \rightarrow \beta$, 都有 $\alpha \in V_N$, $\beta \in (V_N \cup V_T)^*$
 - 产生式的形式描述: $A \rightarrow \beta$ ($A \in V_N$)

即: β 取代 A 时, 与 A 所处的上下文无关。
- 产生的语言称“上下文无关语言”
- 例如: $S \rightarrow 01$ $S \rightarrow 0S1$

文法的 4 种类型

- 3型文法(regular grammar, RG): 也称正则文法
- 每个产生式均为 “ $A \rightarrow aB$ ”或 “ $A \rightarrow a$ ” —— 右线性

or “ $A \rightarrow Ba$ ”或 “ $A \rightarrow a$ ” —— 左线性,

($A, B \in V_N, a \in V_T^*$)

- 产生的语言称 “正则语言”
- 例如: $S \rightarrow 0A \mid 0$

$A \rightarrow 1B \mid B$

$B \rightarrow 1 \mid 0$

学习内容

2.1 语言及其文法

2.2 程序语言定义

2.3 程序语言的构造基础

程序语言定义

- 任何语言实现的基础是语言定义
- 语言的定义决定了该语言具有什么样的语言功能、什么样的程序结构、以及具体的使用形式等细节问题。

程序语言定义

语法与语义

- 语法：是指一组规则，用它可产生一个程序。

- 规则：

词法规则 + 语法规则

- 语义：定义语言的单词符号和语法单位的意义。对于编译程序来说，只有了解程序的语义，才知道应把它翻译成什么样的目标指令代码

词法规则

C语言的字母表为：

$$\Sigma = \{a-z, A-Z, 0-9, (,), [,], \rightarrow, ., !, \sim, +, -, *, /, \&, \%, <, >, =, ^, |, ?, ,, ;\}$$

词法规则是指单词符号的形成规则

C语言的标识符的构成规则：

字母、下划线打头的字母、数字和下划线构成的符号串。如：a1、ave、_day

形式语言

上述的定义是用文字来描述的，当设计编译程序时，就要把它用形式的方式描述出来，就要用到形式语言。

在现今多数程序设计语言中，单词符号一般包括：

标识符、 关键字、 各类型的常数、算符和界符等

状态转换图、正则表达式和有穷自动机是描述词法结构和进行词法分析的有效工具

语法规则

语法规则规定了如何从单词符号形成更大的结构（即语法单位），换言之，语法规则是语法单位的形成规则

一般的程序设计语言的语法单位有：

表达式、语句、分程序、函数、过程和程序等

下推自动机理论和上下文无关文法是讨论语法分析的理论基础

语义

语义是指这样一组规则，使用它可以定义一个程序的意义。

采用的方法：属性方法和基于属性文法的语法制导翻译方法

学习内容

2.1 语言及其文法

2.2 程序语言定义

2.3 程序语言的构造基础

程序语言的构造基础

A. 程序结构简介

B. 构造基础

C. 参数传递

A. 程序结构简介

一个高级语言程序通常由若干子程序段（过程、函数等）组成，许多语言还引入了类、程序包等更高级的结构

FORTRAN

一个FORTRAN 程序由一个主程序和若干个辅助程序段组成

PROGRAM MAIN

.

END

SUBROUTINE SUB1

.

END

FUNCTION FUN1

.

END



它的定义是
并列的

FORTRAN的构成特点

同一名字在不同的程序段中一般都代表不同的对象，也就是说代表不同的存贮单元

PROGRAM MAIN

integer x

x=100

END

SUBROUTINE SUB1

integer x

x=9999

END



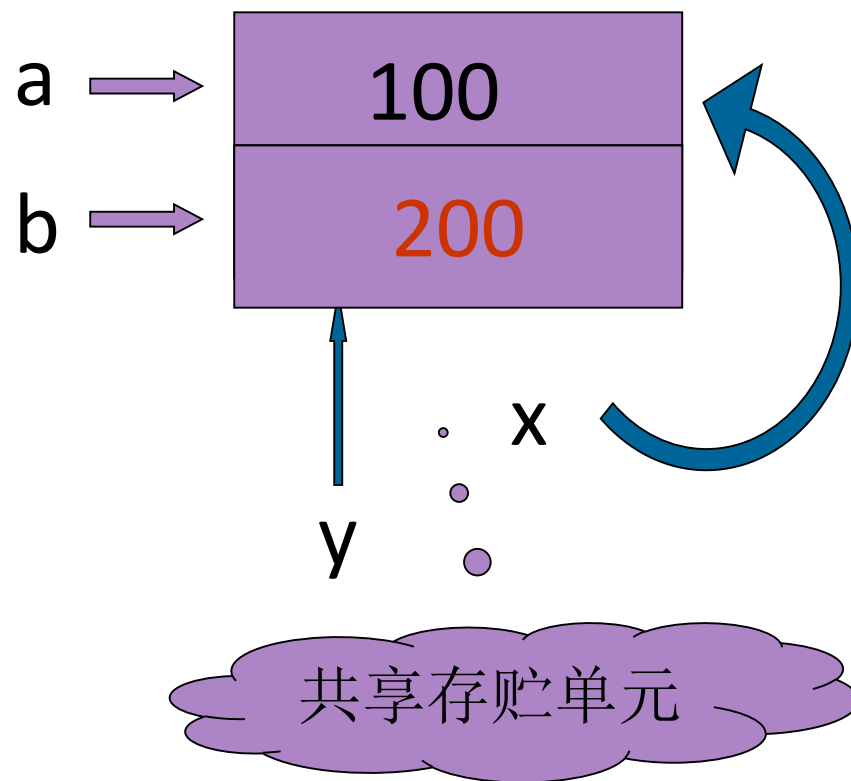
一个名字对应多个对象

FORTRAN的构成特点

不同程序段里的同名公用块却代表同一个存储区域

多个名字对应一个对象

```
PROGRAM MAIN
  common a,b
  a=100
  b=50
END
SUBROUTINE SUB1
  common x,y
  common x,y
  y=x+100
END
END
```



Pascal

Pascal的程序结构

Pascal 允许子
程序嵌套定义

也允许并列定义

Program main

Procedure P1
说明部分
Procedure P11

Begin
Begin
end

可执行部分
Begin
end

end
Procedure P2
Begin
end

Begin

end

关于名字的作用域的规定

标识符X的任意一次出现（除去说明语句中）都意味着对某个说明语句中说明的这个变量X的引用

此时，说明语句同标识符X应共处一个最小程序中，即：

P1中说明的X只在P1中有效

P11是P1的内层子程序，P11中没有再对X作新的说明，则在P11中对X的引用，实际上引用的就是P1中说明的X，

即内部过程可以引用外部过程中定义的量

B. 构造基础

一、名字

程序设计语言的数据对象：数据、函数、过程
常用能反映其本质的、有助于记忆的名字来表示

特性：

- 一个名字对应一个对象，普通变量
- 多个名字对应一个对象，common
- 一个名字对应多个对象，数组、重载、局部变量、重写

B. 构造基础

名字具有属性，通常由说明语句给出

一个名字的属性，包括：类型和作用域

类型决定了它有什么样的值，值在计算机内的表示，以及对它能施加什么样的运算

作用域规定了值的存在范围

B. 构造基础

二. 数据类型

1. 初等数据类型

- ①数值数据： 整型、实型、双精度等，可施行算术运算
- ②逻辑数据： 可施行逻辑运算
- ③字符数据：
- ④指针类型：

B. 构造基础

三、数据结构 ① 数组

从逻辑上讲，一个数组是由同类型数据所组成的 n 维矩形结构

一个数组所需的存贮空间大小在编译时就已知道的，则称此数组是一个确定的数组；否则称为可变数组

设 $\text{int } A[l_1..u_1,][l_2..u_2]..... [l_n..u_n]$ 为 n 维数组

各维的长度： $d_i = u_i - l_i + 1 \quad (1 \leq i \leq n)$

数据结构-数组

任一数组元素 $A[i_1, i_2, \dots, i_n]$ 的地址为:

$$D = a + (i_1 - l_1) d_2 d_3 \dots d_n + (i_2 - l_2) d_3 d_4 \dots d_n \dots + (i_{n-1} - l_{n-1}) d_n + (i_n - l_n)$$

不变部分: $c = (\dots (l_1 d_2 + l_2) d_3 + l_3) d_4 + \dots + l_{n-1}) d_n + l_n$

变量部分: $v = (\dots (i_1 d_2 + i_2) d_3 + i_3) d_4 + \dots + i_{n-1}) d_n + i_n$

任一数组元素 $A[i_1, i_2, \dots, i_n]$ 的地址:

$$\text{addr} = a - c + v$$

数据结构-数组

在编译时，当遇到说明时，必须把数组的有关信息记录在一个“**内情向量**”之中，用于数组元素的地址计算。

数组的内情向量包括：

维数，各维的上、下限，首地址及数组的类型

l_1	u_1	d_1	
l_2	u_2	d_2	
$\dots\dots$	$\dots\dots$	$\dots\dots$	
l_n	u_n	d_n	
N维数	C常数	T类型	A首地址

数据结构-数组

- 对于确定数组来说，内情向量可登记在符号表中
- 对于可变数组，内情向量的信息在编译时无法全部知道，只有到运行阶段才能全部确定下来，存贮分配也要等到运行时方能进行

数据结构

② 记录（结构）

从逻辑上讲，记录是由已知的数据组合的一种结构

```
Struct student
```

```
{
```

```
    char  name[20];
```

```
    boolean partmember;
```

```
    int age;
```

```
} stu;
```

数据结构-记录

```
Struct student
{
    char name[20];
    boolean partmember;
    int age;
} stu;
```

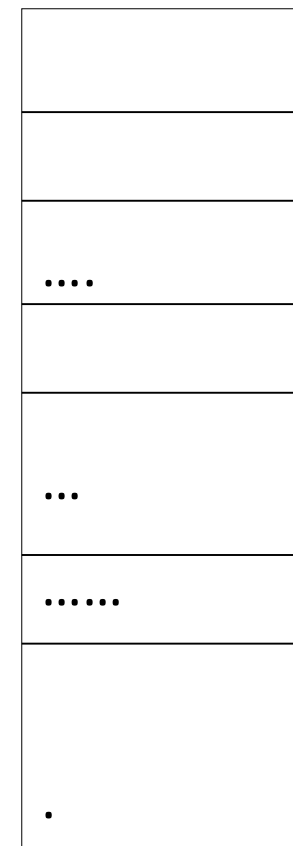
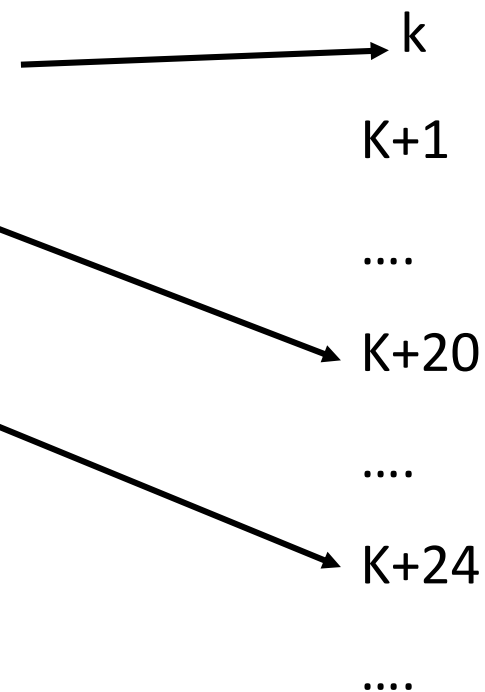
记录结构最简单的存贮方式是连续存放

Stu.name

Stu.partmember

Stu.age

上述的变量**stu**共
占**7**个字，共**28**
个字节



③ 字符串、表格和队列

B. 构造基础

四.抽象数据类型

一个抽象数据类型包括：

- (1)数据对象的一个集合
- (2)作用于这些数据对象的抽象运算的集合
- (3)这种类型对象的封装

C++、Java语言通过类对抽象类型提供支持

B. 构造基础

五.语句与控制结构

1.表达式

要解决的问题：

①优先级

②结合律

2.语句， 语句可分为：

①说明语句：定义各种不同数据类型的变量和运算

②可执行语句：描述语句的动作

执行语句分为：赋值、控制和**I/O**语句

表达式

组成：操作数和算符

$$a + b * c$$

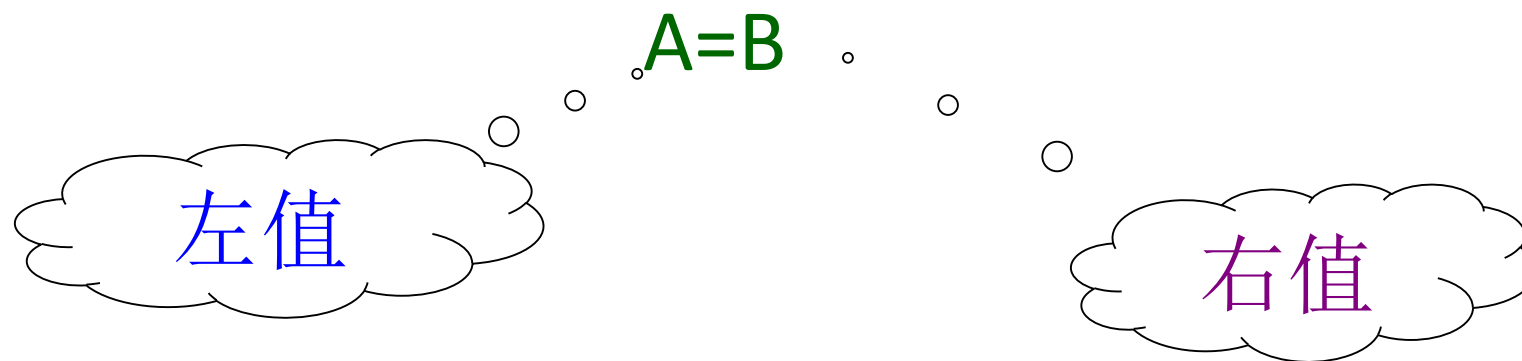
表示形式：

前缀式： $+ a * b c$

中缀式： $a + b * c$

后缀式： $a b c * +$

赋值语句



名字的左值指它所代表的存贮单元地址

名字的右值指该单元的内容

控制语句

无条件转移语句: Goto lable

条件语句: If B then S

 If B then S else S

循环语句: While B do S

 Repeat S until B

 For I=e₁ to e₂ step e₃

过程调用语句: Call P(x₁, x₂, ..., x_n)

返回语句: Return(E)

说明语句

说明语句用于定义名字的性质

编译程序把这些性质登记在符号表中，并检查程序中名字的引用和说明是否一致

许多说明语句不产生目标代码

但有的说明语句，如过程说明和可变数组说明，则要产生相应的目标代码

简单句和复合句

简单句：不包含其它语句成分的基本句。赋值、goto语句等

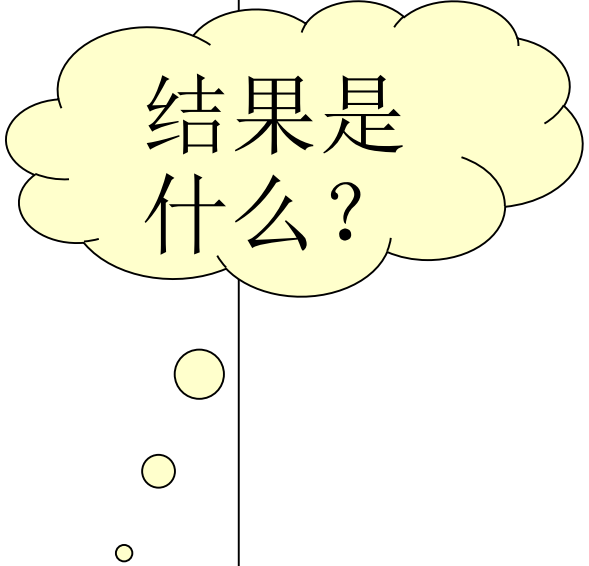
复合句则指那些句中有句的语句

```
if (x==0) then x=1
```

```
{x=1;y=2;goto l1;}
```

C. 参数传递

```
program reference(input, output);  
  var a, b : integer;  
  procedure swap(x, y : integer);  
    var temp: integer;  
    begin temp := x;  
           x := y;  
           y := temp  end;  
  begin a:=1;   b:=2;  
        swap(a, b);  
        writeln( 'a=', a, 'b= ', b )  
  end.
```



结果是什么?

C. 参数传递

实在参数和形式参数结合的方法：

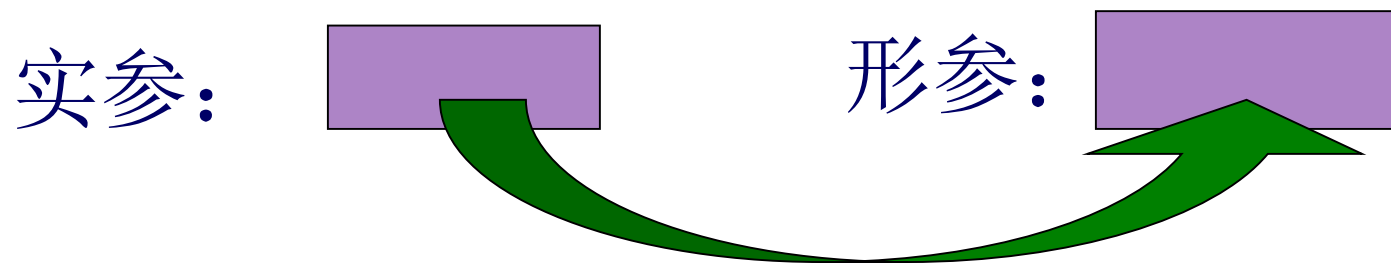
- ① 传值调用 (call-by-value)
- ② 引用调用 (call-by-reference)
- ③ 复制恢复 (copy-restore)
- ④ 传名调用 (call-by-name)

1. 传值调用

主调过程计算实在参数，并把它们的右- 值放入到形式参数的存储空间中

子程序为每一个形参开辟一个存贮单元，用于存放 相应实参的值

子程序执行时，每当访问形参时，就直接访问形参单元



1. 传值调用

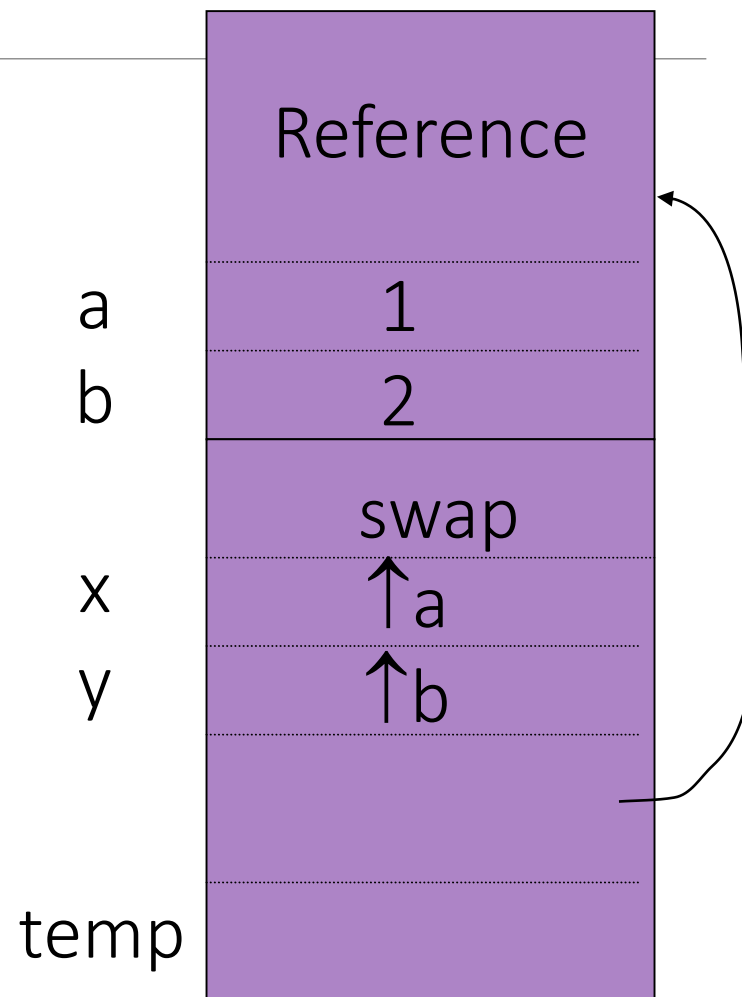
使用传值的方法，调用`swap(a,b)`等价于下面几步：

```
x: = a
y: = b
temp: = x
x: = y
y: = temp
```

2. 引用调用（传地址）

把实在参数的地址传递给相应的形式参数

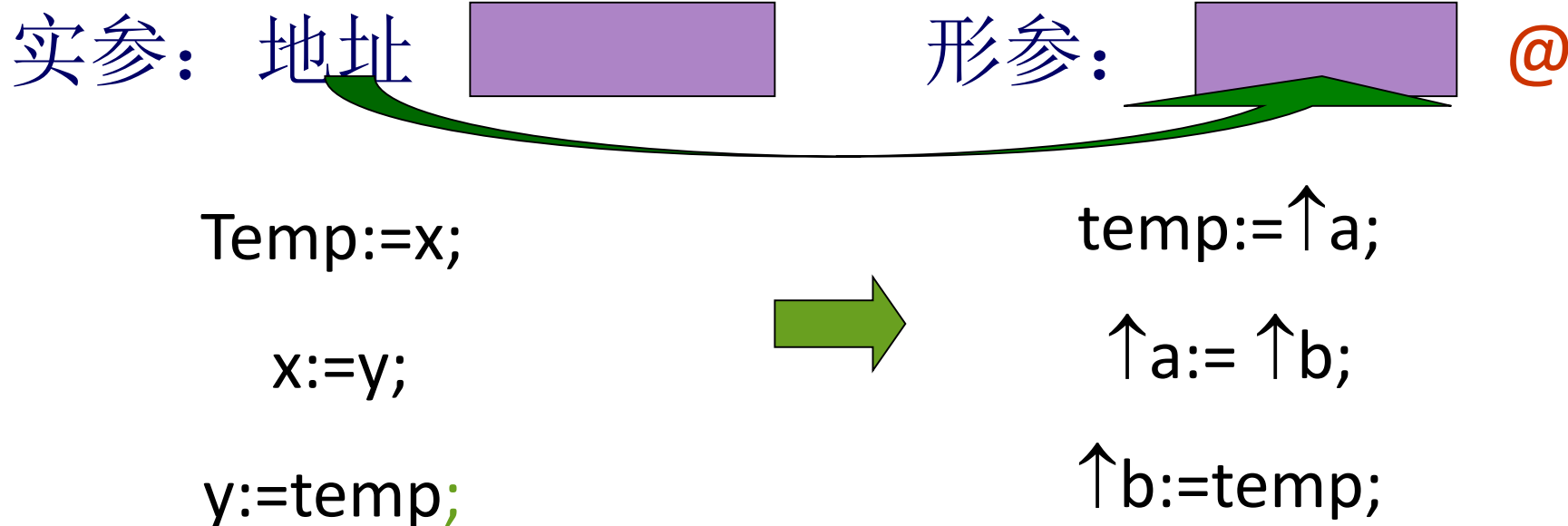
在目标代码中，在被调用过程中对形式参数的一次引用就成为对传递给被调用过程的指针的一个间接引用。



2. 引用调用（传地址）

子程序为每个形参开辟一个单元，用于存放相应实参的地址，
当实参为表达式或常数时，则存放它们值的临时单元。

执行时，子程序间址方式访问这些形参单元



3. 复制恢复（传值结果）

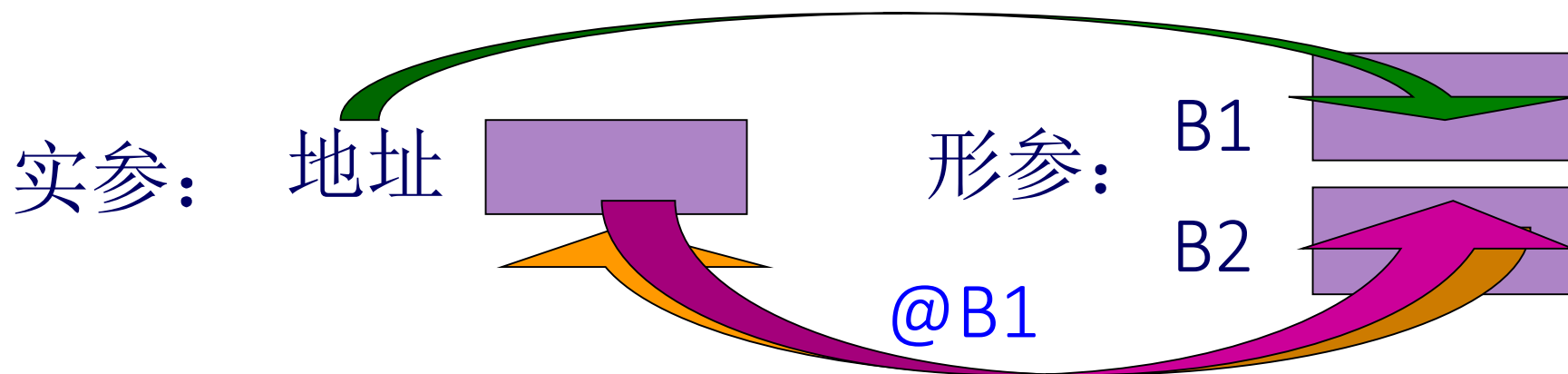
实现：

1. 当控制流入到被调用过程之前，把实在参数的右-值和左-值传递到被调用过程中；
2. 当控制返回时，把形式参数的现行右-值复制回到相应的实在参数的左-值中。

3. 复制恢复（传值结果）

子程序为每个形参分配两个存储单元B1和B2，B1用于存放**实参地址**，B2用于存放**实参值**。

执行时，对B2单元使用直接访问形式；返回前，按B1中的地址把B2中的内容存入主调程序的实参单元中。



4. 传名调用

在主调程序中设置计算实参地址和右值的形参替换子程序THUNK

子程序中为相应实参开辟一个形式单元，用于存放该实参的THUNK子程序的入口地址。

执行时，每当要对形参进行访问时，就调用THUNK子程序，以获得相应实参地址或值

对形参的访问是发生在实参单元上的

例：

Begin

procedure p(x,y,z)

begin

y=y+1;

z=z+x;

end

a=2;

b=3;

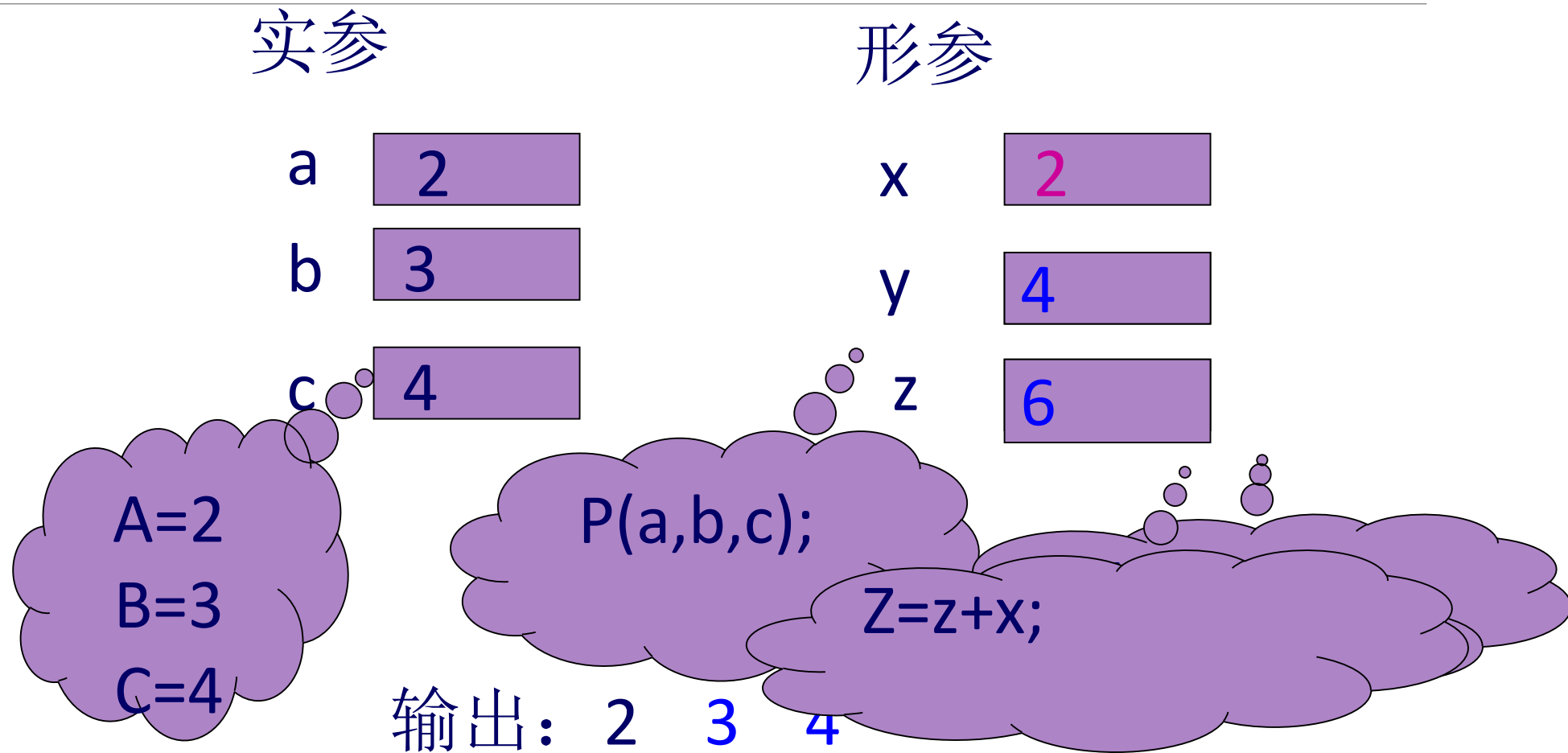
c=4;

P(a,b,c);

print a,b,c;

end

1. 传值



2. 传地址

实参		形参	
a	2	x	&a
b	4	y	&b
c	6	z	&c

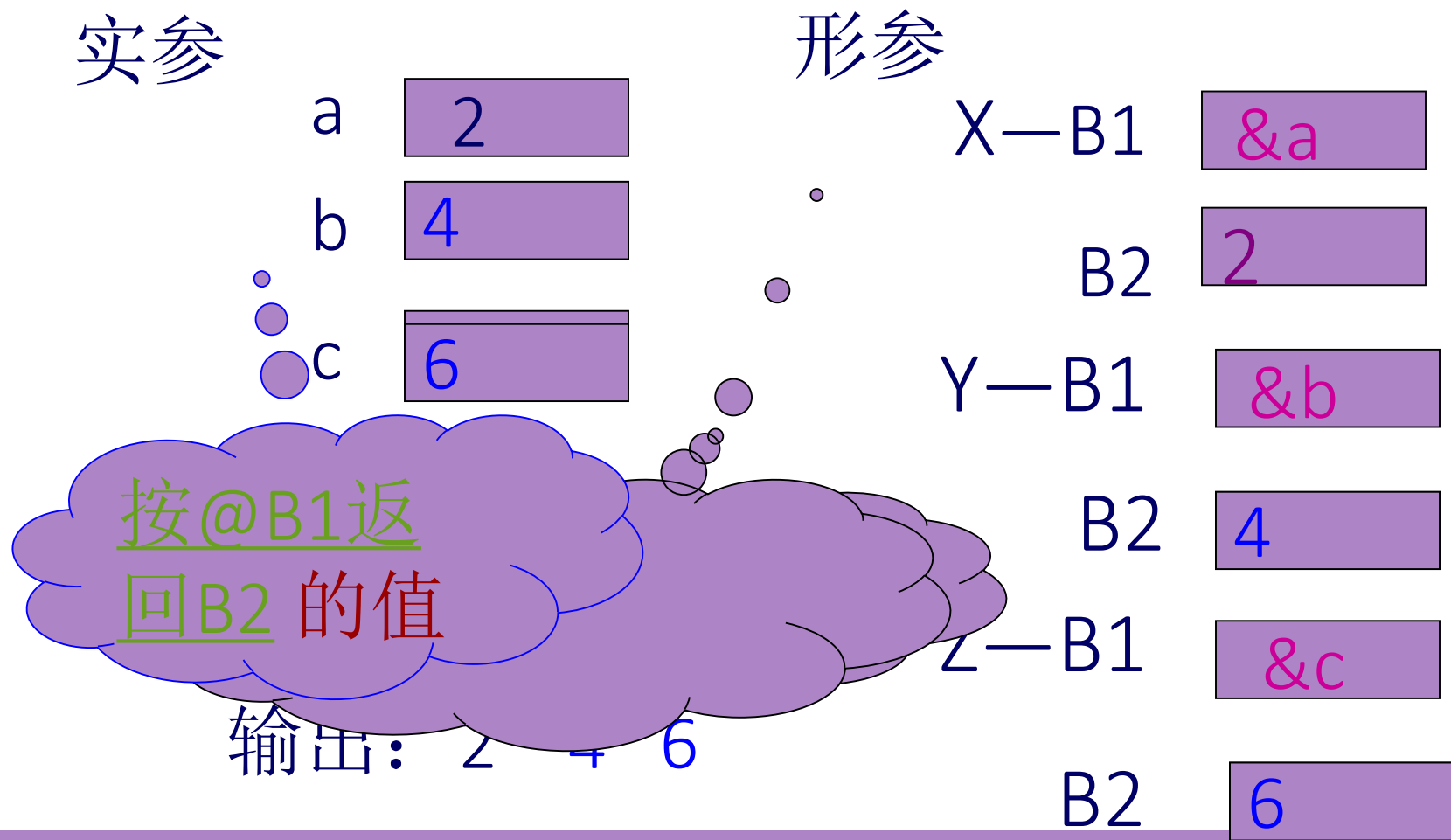
P(a,b,c);

输出: 2 4

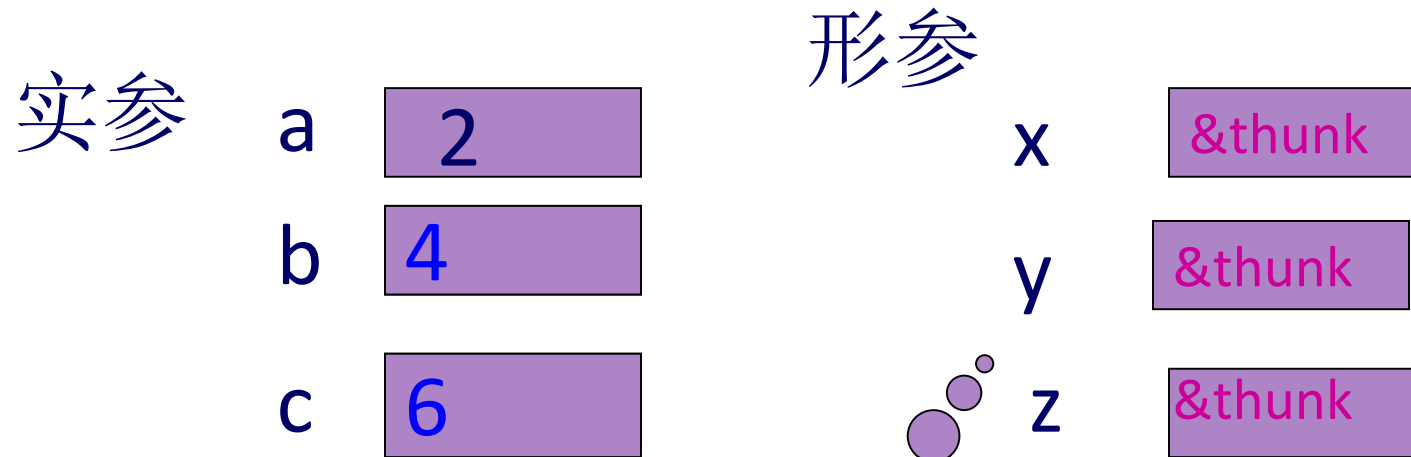
Z=z+x;

@z=@z+@x

3. 传值结果



4. 传名



P(a,b,c);

Z=z+x;

jsr Thunk

Z → c, x → a

y=y+1;

ISR Thunk

Y → b

例：有程序段：

```
procedure p(x,y,z)
```

```
begin
```

```
    y=y+1;
```

```
    z=z+x;
```

```
end
```

```
Begin
```

```
    a=2;
```

```
    b=3;
```

```
    P(a+b,a,a);
```

```
    print a
```

```
end
```

传值:

```
procedure p(x,y,z)
```

```
begin
```

```
  y=y+1;
```

```
  z=z+x;
```

```
end
```

```
Begin
```

```
  a=2;
```

```
  b=3;
```

```
  P(a+b,a,a);
```

```
  print a
```

```
end
```

实参

形参

a

2

x

5

b

3

y

3

a+b

5

z

7

A=2

B=3

P(a+b,a,a);

Z=z+x;

输出: 2

传地址:

```
procedure p(x,y,z)
```

```
begin
```

```
  y=y+1;
```

```
  z=z+x;
```

```
end
```

```
Begin
```

```
  a=2;
```

```
  b=3;
```

```
  P(a+b,a,a);
```

```
  print a
```

```
end
```

实参

形参

a

8

b

3

a+b

5

x

&a+b

y

&a

z

&a

P(a+b,a,a);

Z=z+x;

@z=@z+@x

输出: 8

传名:

```
procedure p(x,y,z)
```

```
Begin
```

```
begin
```

```
a=2;
```

```
y=y+1;
```

```
b=3;
```

```
z=z+x;
```

```
P(a+b,a,a);
```

```
end
```

```
print a
```

```
end
```

实参

形参

a

9

.

x

&thunk

b

3

.

y

&thunk

.

z

&thunk

P(a+b,a,a);

输出:

Z=z+x;

jsr Thunk

Z → a, x → a+b

学习内容

2.1 语言及其文法

2.2 程序语言定义

2.3 程序语言的构造基础

学习内容

2.1 语言及其文法

- 字符集
- 串
- 语言
- 文法

学习内容

2.1 语言及其文法

2.2 程序语言定义

- 词法规则
- 语法规则
- 形式语言

学习内容

2.1 语言及其文法

2.2 程序语言定义

2.3 程序语言的构造基础

- 程序结构（一个高级语言程序如何组织）
- 构造基础（存储与访问，语句与控制结构）
- 参数传递（4种不同的参数传递方式）