



School of Computer Science & Technology
Harbin Institute of Technology



第二章 高级语言及其文法

重点：文法的定义与分类，程序设计语言的定义及设计基础。

难点：程序设计语言的语义定义及设计基础。





第2章 高级语言及其文法

2.1 语言概述

2.2 程序语言的定义

2.3 程序设计语言的设计基础

2.4 本章小结



2.1 语言概述

语言是一定的群体用来进行
信息交流的工具。



2.1 语言概述

- 信息交流的基础是什么？
 - 按照共同约定的生成规则和理解规则去生成“句子”和理解“句子”
 - 例：
 - “今节日上课始开译第一编”
 - **I am a せんせい.**
 - “今日开始上第一节编译课”
 - **I am a teacher.**



2.1 语言概述

■ 语言的特征

■ 自然语言(Natural Language)

- 是人与人的通讯工具

- 语义(semantics):环境、背景知识、语气、二义性——难以形式化

■ 计算机语言(Computer Language)

- 计算机系统间、人机间通讯工具

- 严格的语法(Grammar)、语义(semantics)——易于形式化：严格



2.1 语言概述

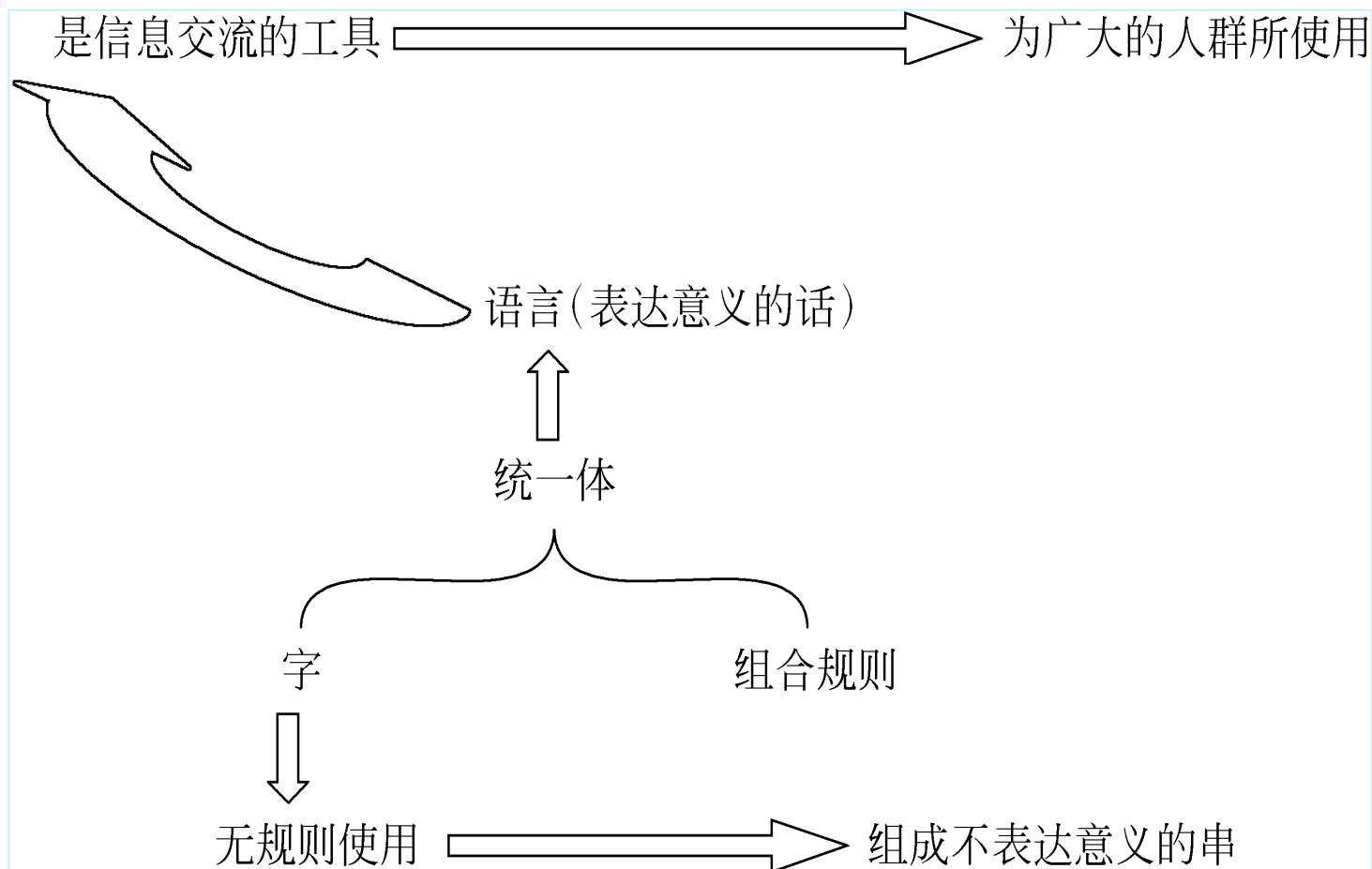
- 语言的描述方法——现状
 - 自然语言：自然、方便-非形式化
 - 数学语言（符号）：严格、准确-形式化
 - 形式化描述
 - 高度的抽象，严格的理论基础和方便的计算机表示。



2.1 语言概述

- 语言——形式化的内容提取
 - 语言(Language): 满足一定条件的句子集合
 - 句子(Sentence): 满足一定规则的单词序列
 - 单词(Token): 满足一定规则的字符(Character)串
- 语言是字和组合字的规则
 - 例（自然语言：第译始二天课今开编上节）
 - 今天开始上第二节编译课

2.1 语言概述





2.1 语言概述

- 程序设计语言——形式化的内容提取
 - 程序设计语言(**Programming Language**): 组成程序的所有语句的集合。
 - 程序(**Program**): 满足语法规则的语句序列。
 - 语句(**Sentence**): 满足语法规则的单词序列。
 - 单词(**Token**): 满足词法规则的字符串。
- 例: 变量:=表达式
 - if 条件表达式 then 语句
 - while 条件表达式 do 语句
 - call 过程名(参数表)



2.1 语言概述

- 描述形式——文法
 - 语法——语句
 - 语句的组成规则
 - 描述方法：巴科斯BNF范式、语法(描述)图
 - 词法——单词
 - 单词的组成规则
 - 描述方法：巴科斯BNF范式、正规式



形式语言与自动机理论的产生与作用

- 语言学家Chomsky最初从产生语言的角度研究语言。
 - 1956年，通过抽象，他将语言形式地定义为是由一个字母表中的字母组成的一些串的集合。可以在字母表上按照一定的规则定义一个文法（Grammar），该文法所能产生的所有句子组成的集合就是该文法产生的语言。



形式语言与自动机理论的产生与作用

- 克林（Kleene）在1951年到1956年间，从识别语言的角度研究语言，给出了语言的另一种描述。
 - 克林是在研究神经细胞中，建立了自动机，他用这种自动机来识别语言：对于按照一定的规则构造的任一个自动机，该自动机就定义了一个语言，这个语言由该自动机所能识别的所有句子组成。



形式语言与自动机理论的产生与作用

- 1959年，Chomsky通过深入研究，将他本人的研究成果与克林的研究成果结合了起来，不仅确定了文法和自动机分别从生成和识别的角度去表达语言，而且**证明了文法与自动机的等价性**。
- 这就是我们编译的基础
- 生成-----编写程序的过程
- 识别-----编译程序的过程



形式语言与自动机理论的产生与作用

- 20世纪50年代，人们用**巴科斯范式**（Backus Nour Form 或 Backus Normal Form，简记为BNF）成功地对高级语言ALGOL-60进行了描述。实际上，巴科斯范式就是上下文无关文法（Context Free Grammar）的一种表示形式。这一成功，使得形式语言在20世纪60年代得到了大力的发展。



形式语言与自动机理论的产生与作用

- 形式语言与自动机理论除了在计算机科学领域中的直接应用外，更在计算学科人才的**计算思维的培养**中占有极其重要的地位
- 计算思维能力的培养，主要是由基础理论系列课程实现的，该系列主要由从数学分析开始到形式语言结束的一些数学和抽象程度比较高的内容的课程组成。
 - 它们构成的是一个梯级训练系统。在此系统中，连续数学、离散数学、计算模型等三部分内容要按阶段分开，三个阶段对应与本学科的学生在大学学习期间的思维方式和能力的变化与提高过程的三个步骤。



2. 2程序语言的定义

如何实现语言结构的 形式化描述？

考虑赋值语句的形式：

左部量 = 右部表达式

$a = a + a$

$b = m[3] + b$

$m[1] = a + m[2]$

句子的组成规则

- $\langle \text{赋值语句} \rangle \rightarrow \langle \text{左部量} \rangle = \langle \text{右部表达式} \rangle$
- $\langle \text{左部量} \rangle \rightarrow \langle \text{简单变量} \rangle$
- $\langle \text{左部量} \rangle \rightarrow \langle \text{下标变量} \rangle$
- $\langle \text{简单变量} \rangle \rightarrow a$
- $\langle \text{简单变量} \rangle \rightarrow b$
- $\langle \text{简单变量} \rangle \rightarrow c$
- $\langle \text{下标变量} \rangle \rightarrow m[1]$
- $\langle \text{下标变量} \rangle \rightarrow m[2]$
- $\langle \text{下标变量} \rangle \rightarrow m[3]$
- $\langle \text{右部表达式} \rangle \rightarrow \langle \text{简单变量} \rangle \langle \text{运算符} \rangle \langle \text{简单变量} \rangle$
- $\langle \text{右部表达式} \rangle \rightarrow \langle \text{简单变量} \rangle \langle \text{运算符} \rangle \langle \text{下标变量} \rangle$
- $\langle \text{右部表达式} \rangle \rightarrow \langle \text{下标变量} \rangle \langle \text{运算符} \rangle \langle \text{简单变量} \rangle$
- $\langle \text{右部表达式} \rangle \rightarrow \langle \text{下标变量} \rangle \langle \text{运算符} \rangle \langle \text{下标变量} \rangle$
- $\langle \text{运算符} \rangle \rightarrow +$
- $\langle \text{运算符} \rangle \rightarrow -$



程序语言的定义

- 语言的定义决定了程序设计语言具有
- 什么样的语言功能
- 什么样的数据结构
- 什么样的程序结构
- 以及具体的使用形式等细节问题。



程序语言的定义

对于语言用户来说：程序语言定义就是一本用户手册。

- 对于编译程序设计者来说：程序语言定义就是具体实现的理论依据,通常是以文法形式来进行描述。



2.2.1 程序语言定义的规则

程序语言的定义是指这样一组规则，利用它就可以产生一个程序。

规则：

词法规则：标识符、常数等

■ 语法规则：程序结构、语句构成等



2.2.1 程序语言定义的规则

一 . 词法规则

字母表就是一个有穷字符集。

■ C语言的字母表为：

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



一 . 词法规则

词法规则是指单词符号的形成规则

在程序语言中，单词符号一般包括：

- 各类型的常数、标识符、关键字、算符和界符等

正规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

一. 词法规则

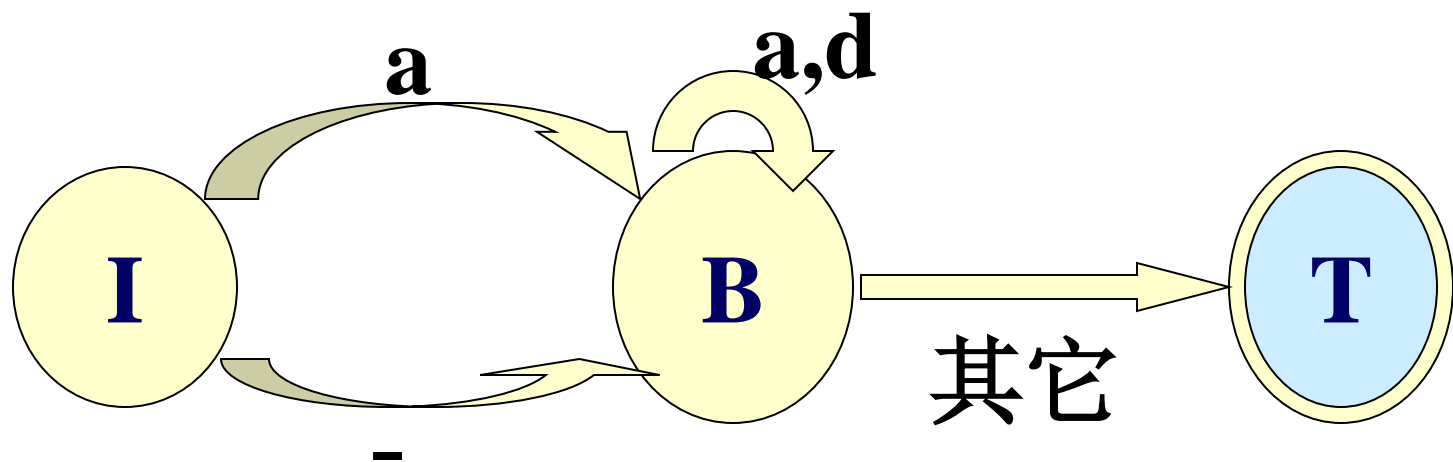
例：C语言标识符的文法描述

$L(G) = \{w / w \text{ 为字母或 } \text{'-'} \text{ 打头的字母数字串} \}$

解：P: $I \rightarrow aB$ $I \rightarrow -B$ $I \rightarrow a$

$B \rightarrow aB$ $B \rightarrow dB$ $B \rightarrow a$ $B \rightarrow d$

识别 $L(G)$ 的自动机

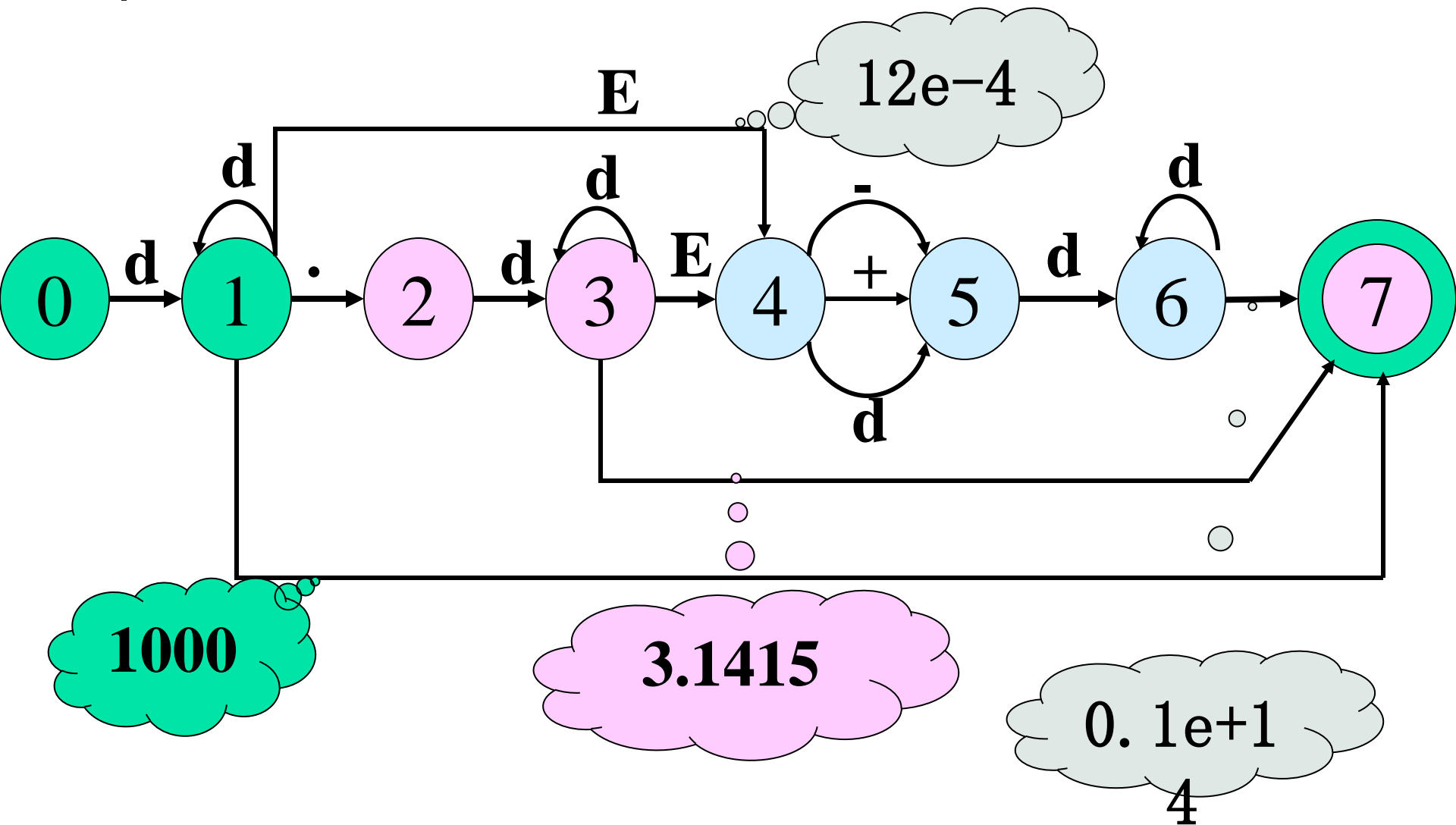




一. 词法规则

- 例：C语言实常数的文法描述
- 文法：
- $S \rightarrow dA$ $A \rightarrow dA$ $A \rightarrow eD$
- $A \rightarrow .B$ $B \rightarrow dC$ $C \rightarrow dC$
- $C \rightarrow eD$ $D \rightarrow -E$ $D \rightarrow +E$
- $D \rightarrow dF$ $E \rightarrow dF$ $F \rightarrow dF$
- $F \rightarrow d$

例：C语言实常数的自动机描述





二. 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我们讨论语法分析的理论基础

三、主要语言成分的文法描述

■ 1、表达式

■ $G[E]:$

■ $E \rightarrow E + E$

■ $E \rightarrow E - E$

■ $E \rightarrow E * E$

■ $E \rightarrow E / E$

$E \rightarrow (E)$

■ $E \rightarrow id$

$G[E]:$

$E \rightarrow E + T$

$E \rightarrow E - T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow T / F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow id$



2、布尔表达式G (B)

- **$B \rightarrow B \text{ or } B$**
- **$B \rightarrow B \text{ and } B$**
- **$B \rightarrow \text{not } B$**
- **$B \rightarrow (E)$**
- **$B \rightarrow \text{id relop id}$**
- **$B \rightarrow \text{true}$**
- **$B \rightarrow \text{false}$**



3、描述语句

- $G[S]: S \rightarrow id=E$
- $S \rightarrow \text{if } B \text{ then } S$
- $S \rightarrow \text{if } B \text{ then } S \text{ else } S$
- $S \rightarrow \text{while } B \text{ do } S$
- $S \rightarrow \{ L \}$
- $L \rightarrow L ; S$
- $L \rightarrow S$



4、调用语句G (S)

- $S \rightarrow \text{call id}(\text{Elist})$
- $\text{Elist} \rightarrow \text{Elist}, \text{E}$
- $\text{Elist} \rightarrow \text{E} \mid \varepsilon$

5、类型说明和过程说明语句G (P)

$P \rightarrow D$

$D \rightarrow D; D$

$D \rightarrow \text{id}:T$

$D \rightarrow \text{id} (\text{Elist}) \quad D ; S$

$T \rightarrow \text{int}$

$T \rightarrow \text{float}$

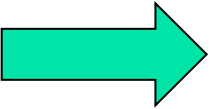


6、数组说明语句G (L) :

$$L \rightarrow \text{id}[\text{Elist}]$$
$$\text{Elist} \rightarrow \text{Elist}, \text{E}$$
$$\text{Elist} \rightarrow \text{E}$$

2.2.2 语义

对于一个语言来说，不仅要给出它的词法、语法规则，而且要定义它的单词符号和语法单位的意义，这就是语义问题

例： $a = b + c * d / 2$  $temp1 = c * d$
 $Temp2 = temp1 / 2$
 $temp3 = b + temp2$
 $a = temp3$

例do 999 I=1,10



2.2.2 语义

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

所谓语言的语义是指这样一组规则，使用它可以定义一个程序的意义。

$E \rightarrow E1 + E2$

{ $E.place = E1.place + E2.place$ }

$S \rightarrow id := E$

```
{  p:=lookup(id.name);  
  if p<>nil then emit(p= E.place)  
  else  error      }
```



2.3设计基础

2.3.1 典型程序结构简介

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

一. 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

.Int x

X=100

X

100

END

SUBROUTINE SUB1

char x

X='y'

X

y

.END

一个名字对应多个对象

但是不同程序段里的同名公用块 却代表同一个存储区域

PROGRAM MAIN

Int a, b

common a, b

A=100

B=50

END

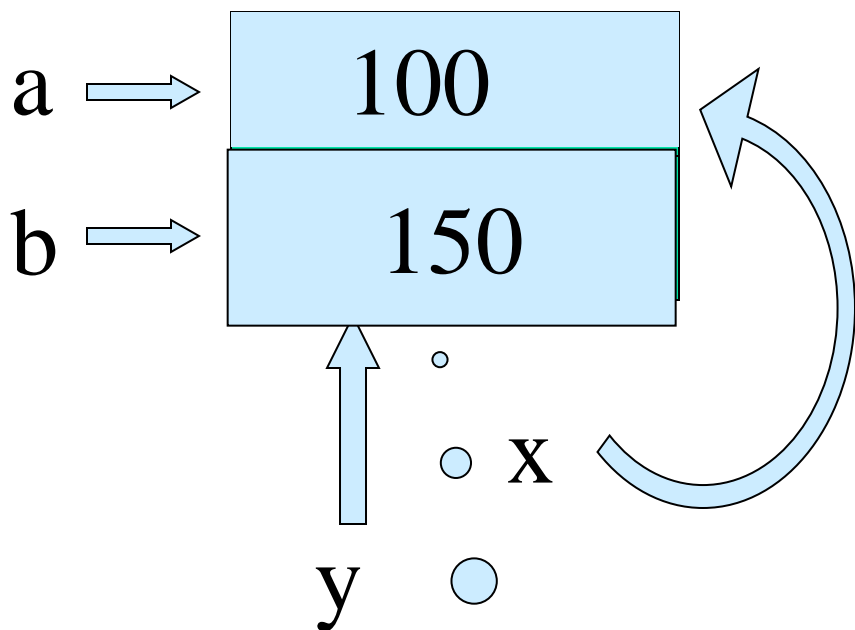
SUBROUTINE SUB1

Int x, y

common x, y

X=x+y

END



多个名字对
应一个对象

共享存储
单元

二、Pascal

Program main

Procedure P1

说明部分P11

Begin

Begin
end

Begin可执行部分
end

end
Procedure P2

Begin

end

Begin

end

Pascal的

程序结构

Pascal 允许
子程序嵌套
定义

也允许并列
定义

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

Procedure P1

int X;

引用变量和说明这个变量的语句应该共处一个最小程序单位中：

Begin

X=100;

P1中说明的X只在P1中有效

end



Procedure P1

对于允许嵌套结构的程序设计语言, 还要解决内部过程引用外部过程中定义的量的问题

Int x;

Procedure P11

Begin

x=x+1;

end

Begin

end

三.java

Java语言是一种面向对象的高级语言，它很重要的方面是类和继承的概念，同时支持多态性和动态绑定等特性

```
Class car{
Int color_num;
Int door_num;
Int speed;
Double price;
• Void ABS( )
Void push_break()
{ {
}
}
Void add_oil()
{
}
}
```



一个类把有关的数据及其操作封装在一起构成一个抽象数据类型

一个子类继承其父类的所有数据和方法，并且可以加入自己新的定义

在java中，变量和方法的定义之前可以加上public、private、protected等修饰词，其它类的对象对于这些变量和方法的使用



2.3.2 设计基础

一. 名字

程序设计语言的对象：存贮单元

常用能反映其本质的、有助于记忆的名字
来表示一符号空间

特性：一个名字对应一个对象

多个名字对应一个对象

一个名字对应多个对象



一.名字

- 名字具有属性，通常由说明语句给出
- 一个名字的属性，包括：类型和作用域等

类型：决定了它有什么样的值，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



二. 数据类型

1. 初等数据类型

①数值数据：整形、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：

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 $\text{int } A[l_1 \cdots u_1][l_2 \cdots u_2] \dots [l_n \cdots u_n]$
为 n 维数组

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

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

$$D = a + (i_1 - l_1) d_1 + (i_2 - l_2) d_2 + \dots + (i_{n-1} - l_{n-1}) d_{n-1} + (i_n - l_n) d_n$$

整理后 $C = (\dots (l_1 d_2 + l_2) d_3 + l_3) d_4 + \dots$

C 是数组计算
中不变的部分



1. 数组

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：

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

这样我们就解决了数组元素的寻址问题

在编译具体实现时,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

l₁	u_n	d₁	
l₂	u_n	d₂	
.....	
l_n	u_n	d_n	
N维数	C常数	T类型	A首地址



1. 数组

对于确定数组来说，内情向量可
登记在符号表中；

例：Int a[10], b;

名字	记号	类型	种属	addr
a		↑	数组		0
b		int	简变		40



1. 数组

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



2. 记录（结构）

从逻辑上讲，记录是由已知的数据组合起来的一种结构

Struct student

{char name[20];

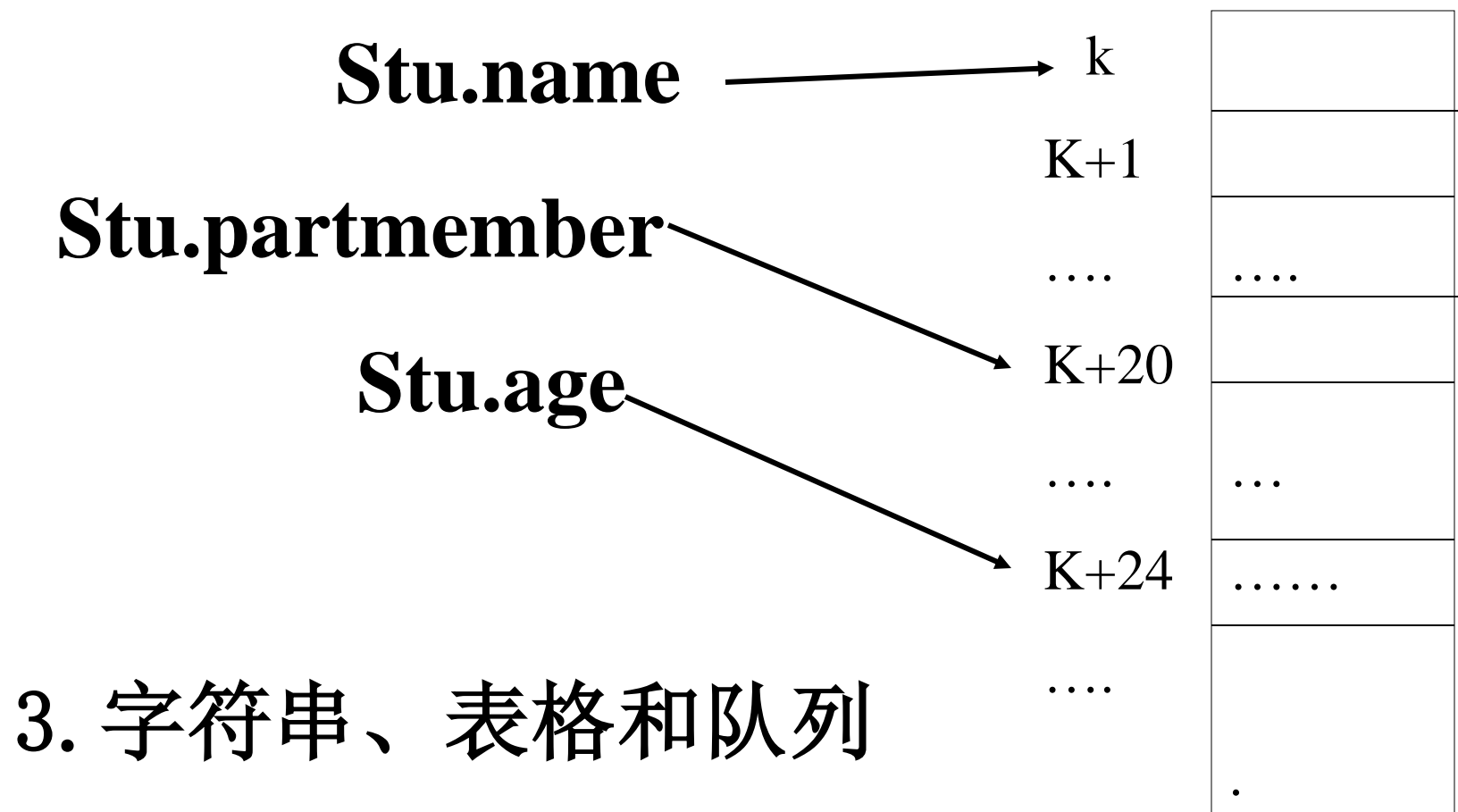
boolean partmember;

int age;

} stu;

记录结构最简单的存贮方式是连续存放

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



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

五.语句与控制结构

1.表达式

要解决的问题： ①优先级
②结合率

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



(1)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

(2)赋值句

A=B。

左值

名字的左值指它所代表的存贮单元地址

右值

名字的右值指该单元的内容



(3)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)**



(4)简单句和复合句

简单句是指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;}

六. 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?



参数的传递方式:

传值调用 (`call-by-value`)

引用调用 (`call-by-reference`)

复制恢复 (`copy-restore`)

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。

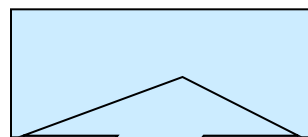
子程序为每一个形参开辟一个存储单元，用于存放 相应实参的值。

子程序执行时，每当访问形参时，就直接访问形参单元。

实参：



形参：





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
```




1. 传值调用

主调程序把实在参数的地址传递给相应的形式参数。

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地址的一次间接引用。

也就是对形参的访问，实际上是发生在实参的存贮单元上



2. 引用调用（传地址）

- 主调程序把实在参数的地址传递给相应的形式参数。
- 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地址的一次间接引用。
- 也就是对形参的访问,实际上是发生在实参的存贮单元上

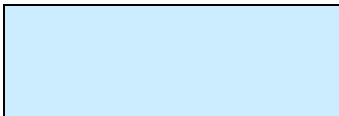




2. 引用调用（传地址）

- 具体实现时：
- 被调过程为每个形参开辟一个存贮单元，用于存放相应实参的地址。
- 当实参为表达式或常数时，先将表达式和常数的值计算出来后，存放临时单元，然后将临时单元的地址传给形参单元
- 被调过程执行时，间址方式访问这些形参单元



图示如下:

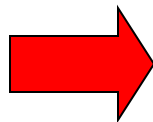
实参: 地址  形参:  @



Temp:=x;

x:=y;

y:=temp;



temp:=↑a;

↑a:= ↑b;

↑b:=temp;

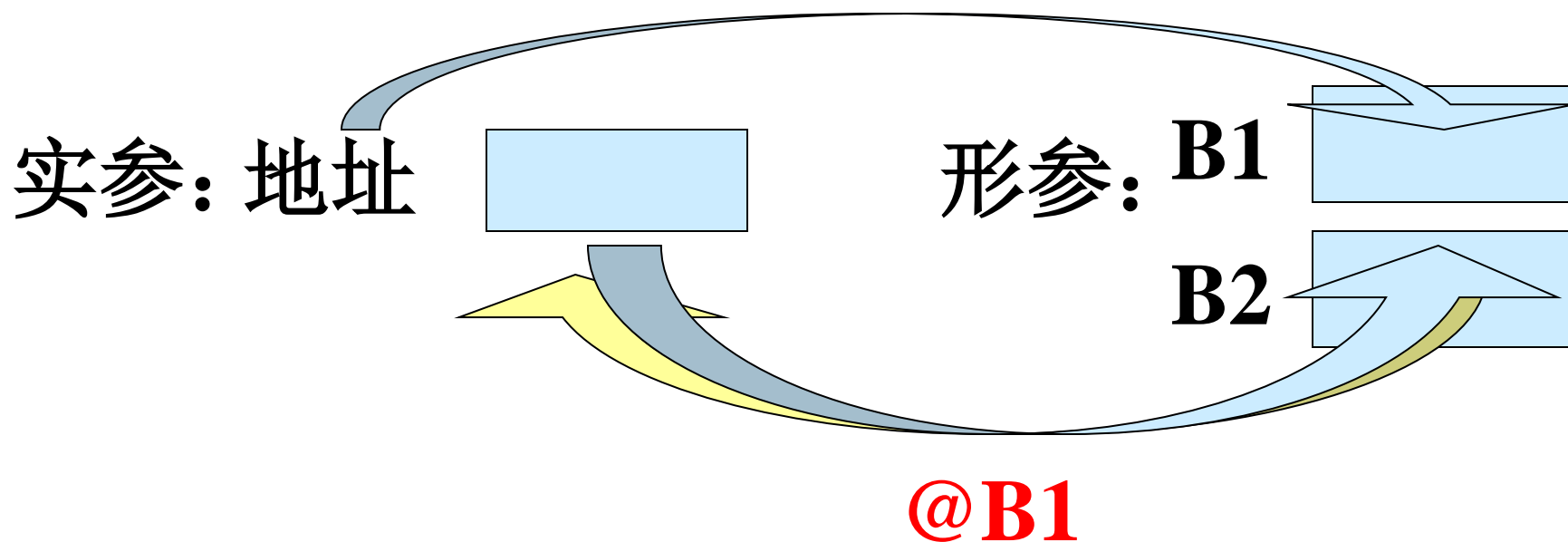


3. 复制恢复（传值结果）

- 实现：
- **1.**在控制进入被调过程之前,主调程序将实参的值和地址传递给形参
- **2.**当控制从被调过程返回时, 被调过程把形参的值传递给相应的实参
- 具体实现时,被调过程为每个形参分配两个存贮单元**B1**和**B2**, **B1**用于存放实参地址, **B2**用于存放实参值。

3. 复制恢复（传值结果）

执行被调过程时, 对B2单元使用直接访问形式; 返回前, 按B1中的地址把B2中的值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实参值的形实替换子程序**THUNK**
- 子程序中为每个形参开辟一个存贮单元，用于存放**THUNK**子程序的入口地址。

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



例：有程序段：

Begin

procedure p(x,y,z)

a=2;

begin

b=3;

y=y+1;

c=4;

z=z+x;

P(a,b,c);

end

print a,b,c;

end



传值:

实参

形参

a

2

x

2

b

3

y

4

c

4

z

6

P(a,b,c);

Z=z+x;

输出: 2 3 4



传地址:

实参

形参

a

2

x

&a

b

4

y

&b

c

6

z

&c

P(a,b,c);

Z=z+x;

@z=@z+@x

输出: 2 4

传值结果:

实参

形参

a

2

b

4

c

6

X—B1

&a

B2

2

Y—B1

&b

B2

4

Z—B1

&c

B2

6

按@B1地址
返回B2 的值

输出: 2 4 6



传名:

实参

形参

a

2

x

&thunk

b

4

y

&thunk

c

6

z

&thunk

Prologue

Z=z+x;
JSR Thunk
Z → c, x → a

y=y+1;
JSR 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:

实参

a

2

b

3

a+b

5

形参

x

5

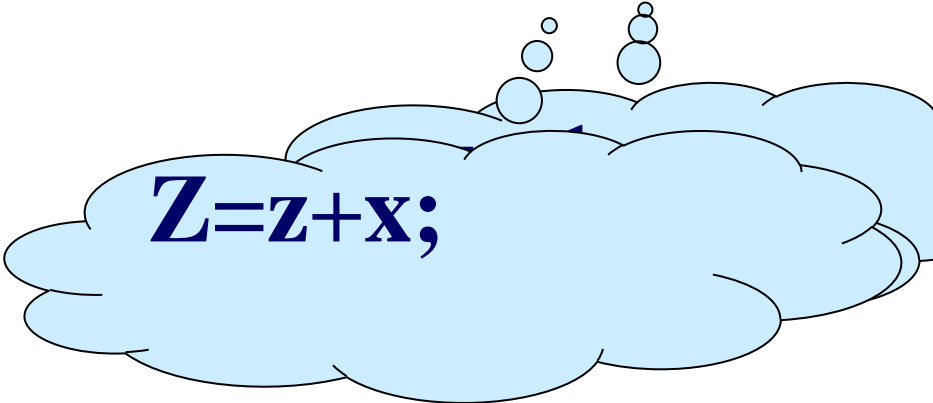
y

3

z

7

输出: 2



Z=z+x;



传地址:

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:

实参

形参

a

9

b

3

x

&thunk

y

&thunk

z

&thunk

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

$Z = z + x;$

JSR Thunk

$Z \rightarrow a, x \rightarrow a+b$

输出:

nk



本章小结

- 语言及其描述
- 常见程序结构
- 参数传递方式