

# 第7章 中间代码生成

---

# 主要内容

---

7.1 中间代码

7.2 说明语句翻译

7.3 组合数据说明的翻译

7.4 赋值语句的翻译

7.5 控制语句的翻译 (if、循环)

## 7.1 中间代码

- 中间代码：源程序的不同表现形式，也称中间表示
- 作用
  - 避免源程序与目标程序之间较大的语义跨度
  - 便于编译系统的实现、移植、代码优化
- 特点
  - 形式简单、语义明确、便于翻译
  - 独立于目标语言
- 种类：
  - 三地址码、四元式；语法(结构)树、三元式
  - 后缀式——逆波兰表示

## 例 7-1 表达式 $(A - 12) * B + 6$ 的中间代码

三地址码

$T_1 = A - 12$

$T_2 = T_1 * B$

$T_3 = T_2 + 6$

逆波兰表示

$A12-B*6+$

(波兰表示

$+*-A12B6)$

四元式

$(-, A, 12, T_1)$

$(*, T_1, B, T_2)$

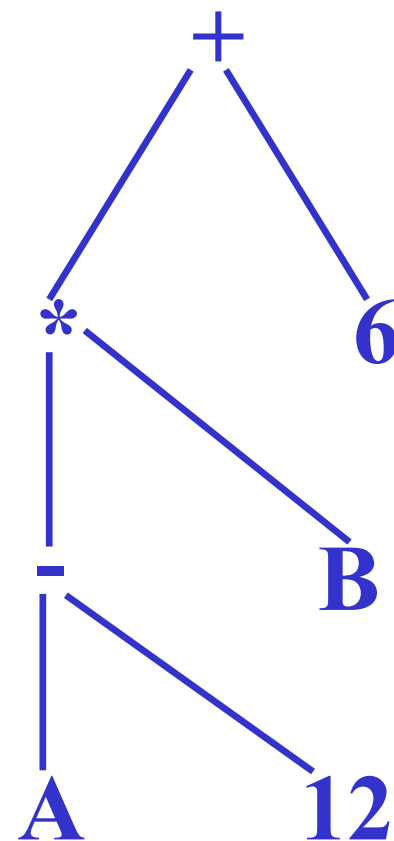
$(+, T_2, 6, T_3)$

三元式

1  $(-, A, 12)$

2  $(*, (1), B)$

3  $(+, (2), 6)$



语法结构树

以  $C := (A - 12) * B + 6$  为例

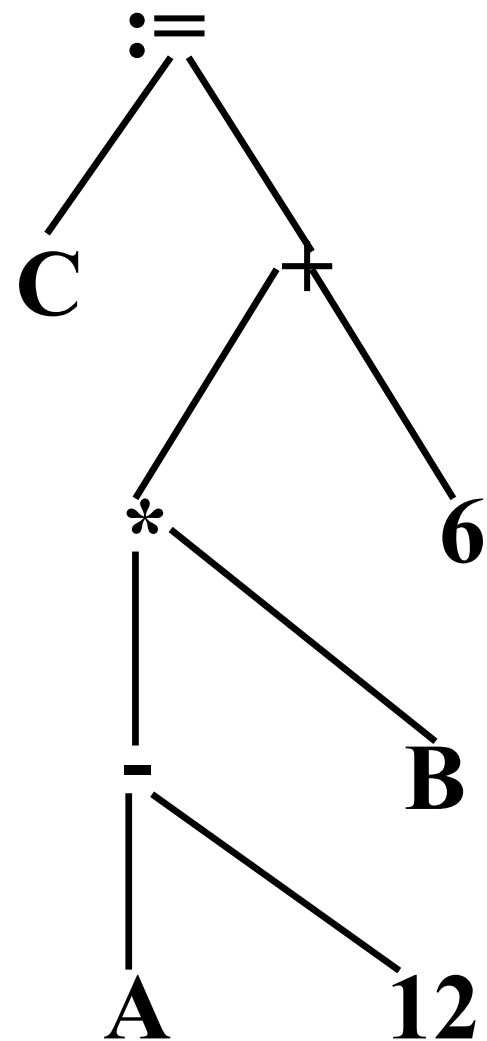
将赋值语句变换为语法结构树

### ■ 属性设置

- E.p 是语法结构树指针
- id.entry 是名字的表项入口
- num.val 是数值

### ■ 基本函数——结点构造

- mknnode 建中间结点
- mkleaf 建叶结点

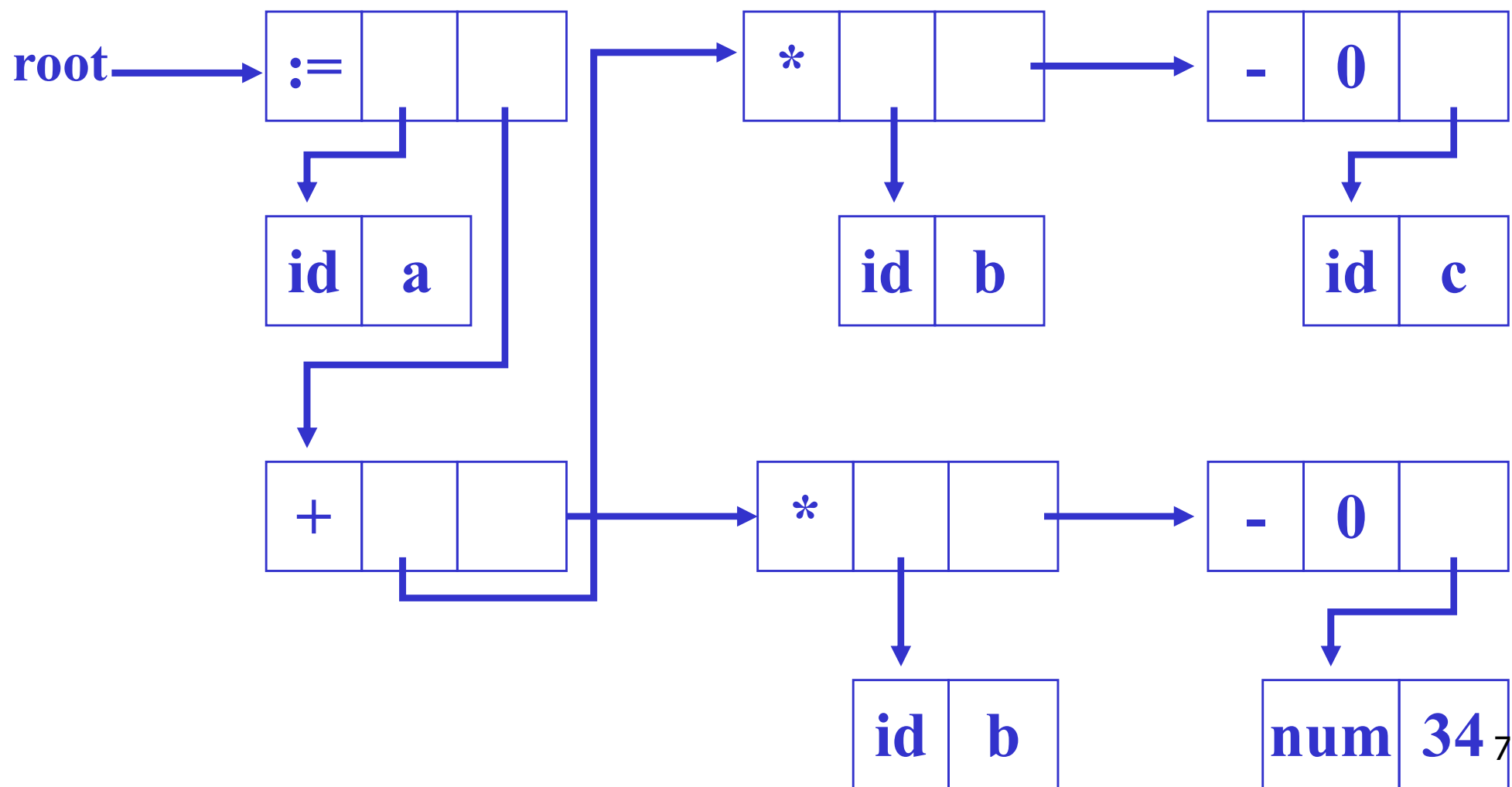


## 生成语法结构树的属性文法

产生式	语义规则
$S \rightarrow id := E$	$S.p := \text{mknode}(':', \text{mkleaf}(id, id.entry), E.p)$
$E \rightarrow E_1 + E_2$	$E.p := \text{mknode}('+', E_1.p, E_2.p)$
$E \rightarrow E_1 * E_2$	$E.p := \text{mknode}('*', E_1.p, E_2.p)$
$E \rightarrow -E_1$	$E.p := \text{mknode}('-', 0, E_1.p)$
$E \rightarrow (E_1)$	$E.p := E_1.p$
$E \rightarrow id$	$E.p := \text{mkleaf}(id, id.entry)$
$E \rightarrow num$	$E.p := \text{mkleaf}(num, num.val)$

## 语法结构树的直观描述

例 7-2:  $a := b * (-c) + b * (-34)$



# 语法结构树的数组存储形式

$a := b * (-c) + b * (-34)$

以语法分析为中心

$S \rightarrow id := E$

地址	算符	操作数1	操作数2
0	id	b	
1	id	c	
2	-	0	1
3	*	0	2
4	id	b	
5	num	34	
6	-	0	5
7	*	4	6
8	+	3	7
9	id	a	
10	:=	9	8



## 后缀式（逆波兰表示）

- 操作数 1, 操作数 2, 运算符
- 操作数, 运算符
- 例 8-7:

求  $a := b * (-c) + b * (-34)$  的后缀式

后缀式:  $a b c - * b 34 - * + :=$

## 生成后缀式的属性文法

产生式	语义规则
$S \rightarrow id := E$	<b>Print(id.name    E.code    “:=”)</b>
$E \rightarrow E_1 + E_2$	<b>E.code := E<sub>1</sub>.code    E<sub>2</sub>.code    “+”</b>
$E \rightarrow E_1 * E_2$	<b>E.code := E<sub>1</sub>.code    E<sub>2</sub>.code    “*”</b>
$E \rightarrow -E_1$	<b>E.code := E<sub>1</sub>.code    “-”</b>
$E \rightarrow (E_1)$	<b>E.code := E<sub>1</sub>.code</b>
$E \rightarrow id$	<b>E.code := id.name</b>
$E \rightarrow num$	<b>E.code := num.val;</b>

注释： || 表示代码序列的连接

## 三地址码

一般形式  $x := y \text{ op } z$

- 其中  $x, y, z$  为变量名、常数或编译产生的临时变量
- 四元式  $(\text{op}, y, z, x)$

种类:  $x := y \text{ op } z$

$x := \text{op } y$

$x := y$

$\text{if } x \text{ relop } y \text{ goto } l$

$(\text{relop}, x, y, l)$

双目运算

单目运算

赋值语句

条件转移语句

$\text{relop}: >, <, =, = < \text{等}$

## 其它语句的三地址码

goto l	无条件转移
param x	实在参数
call p, n	过程调用
return x	过程返回
x := y[i]	数组运算
x[i] := y	
x := &y	指针运算
x := *y	
*x = y	

## 生成三地址码的属性文法

产生式	语义规则
$S \rightarrow id := E$	$S.code := E.code    gen(id.place := E.place)$
$E \rightarrow E_1 + E_2$	$E.place := newtemp; E.code := E_1.code    E_2.code   $ $gen(E.place := E_1.place + E_2.place)$
$E \rightarrow E_1 * E_2$	$E.place := newtemp; E.code := E_1.code    E_2.code   $ $gen(E.place := E_1.place * E_2.place)$
$E \rightarrow -E_1$	$E.place := newtemp; E.code := E_1.code   $ $gen(E.place := 'uminus' E_1.place)$
$E \rightarrow (E_1)$	$E.place := E_1.place; E.code := E_1.code$
$E \rightarrow id$	$E.place := id.place; E.code := ''$
$E \rightarrow num$	$E.place := num.val; E.code := ''$

注释： || 表示代码序列的连接

## 7.2 说明语句的翻译

---

- 作用

- 说明语句 (Declarations) 用于对程序中规定范围内使用的各类变量、常数、过程进行说明

- 编译要完成的工作

- 在符号表中记录被说明对象的属性，为执行做准备

# 要关心的问题

## ■ 类型

- 基本类型/内部类型 (built-in): 整型、实型、双精度型、逻辑型、字符型
- 用户定义类型——结构描述

## ■ 作用域——有效范围

- 一般: 说明所在的分程序、过程范围中

# 要关心的问题

## ■ 类型的作用

- 引入数据抽象、隐蔽数据的基本表示
  - 用户无需注明字节数
- 规定可用的运算
  - 类型检查
- 控制数据精度
  - 规定存储单元的字节数，优化空间管理



## 变量说明的翻译

- 在符号表中填写变量的属性
  - 种别、类型、相对地址、作用域.....等
- 相对地址
  - 全局变量表示为静态数据区的偏移值(offset)
  - 局部变量表示为局部数据区(活动记录部分)的偏移值
  - 两种数据区：静态数据区、动态数据区

## 例 7-3：相对地址举例

Begin

real x[8];

integer i, j;

.....

end

实型8字节，整型4字节

■ 名字 相对地址

■ x

■ i

■ j

0

64

68

0

8

63

64

68

X[1]

X[2]


.....

X[8]

i

j

## 文法描述

**P**  **D**

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

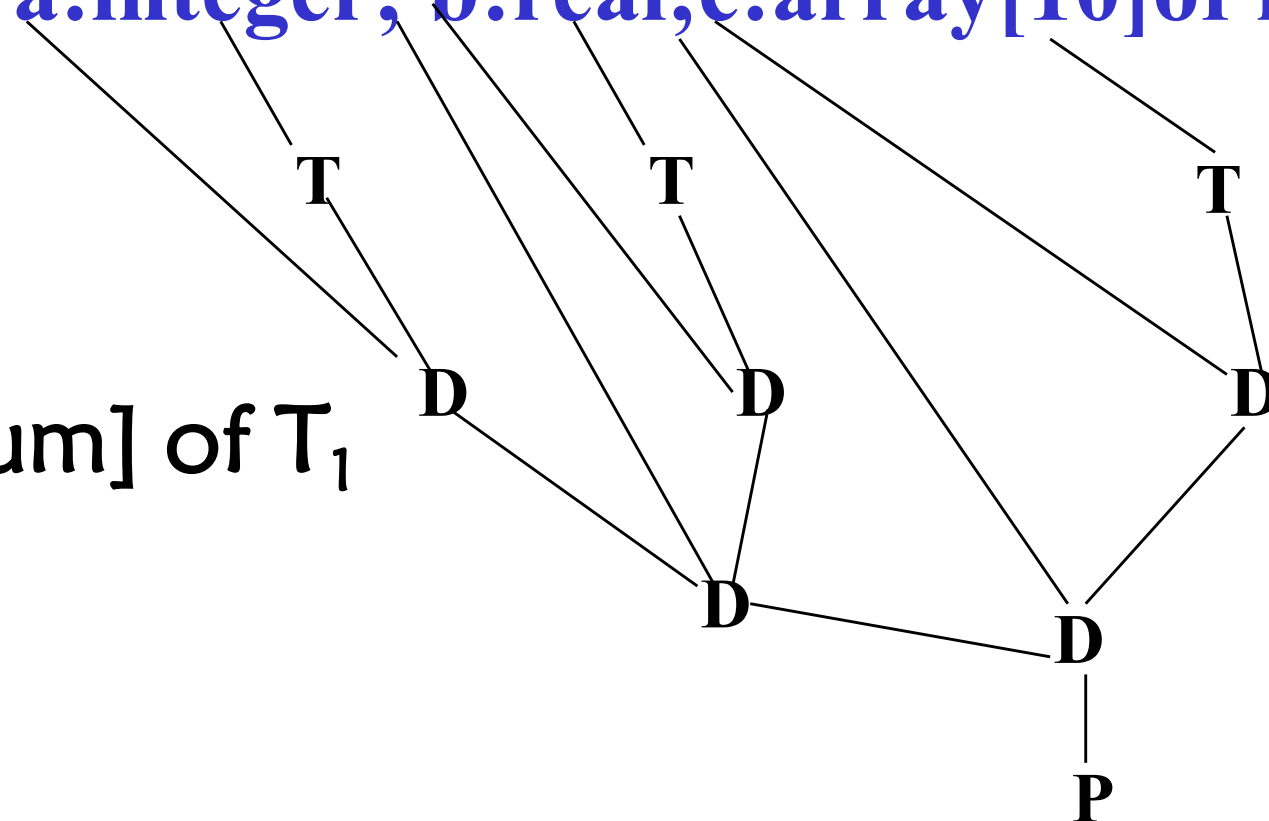
**T → integer**

**T → real**

$$T \rightarrow \text{array}[\text{num}] \text{ of } T_1$$
$$T \rightarrow \uparrow T_1$$

例如：

**a:integer; b:real;c:array[10]of real**



# 用到的属性、过程、与全局量

- 类型T的属性
  - type 类型
  - width 占用的字节数
- 基本子程序
  - enter: 设置变量的类型和地址
  - array: 数组类型处理
- 全局量
  - offset: 已分配空间字节数

## 说明语句的翻译模式

$P \rightarrow \{ \text{offset} := 0 \} D$

$D \rightarrow D ; D$

$D \rightarrow \text{id} : T \{ \text{enter}(\text{id.name}, T.\text{type}, \text{offset});$   
 $\quad \text{offset} := \text{offset} + T.\text{width} \}$

$T \rightarrow \text{integer} \{ T.\text{type} := \text{integer}; T.\text{width} := 4 \}$

$T \rightarrow \text{real} \{ T.\text{type} := \text{real}; T.\text{width} := 8 \}$

$T \rightarrow \text{array}[\text{num}] \text{ of } T_1$

$\quad \{ T.\text{type} := \text{array}(\text{num.val}, T_1.\text{type});$

$\quad T.\text{width} := \text{num.val} * T_1.\text{width} \}$

$T \rightarrow \uparrow T_1 \{ T.\text{type} := \text{pointer}(T_1.\text{type}); T.\text{width} := 4 \}$

## 例 7-4 x:real; i:integer 的翻译

$$\mathbf{D \rightarrow id : T \{enter( id.name, T.type, offset );}$$
$$\mathbf{offset := offset + T.width\}}$$

$P \Rightarrow \{ offset := 0 \} D$

$\Rightarrow \{ offset := 0 \} D ; D$

$\Rightarrow \{ offset := 0 \} x: T \{enter( x, T.type, offset ); offset := offset + T.width\}; D$

$\Rightarrow \{ offset := 0 \} x : real \{T.type := real; T.width := 8\} \{enter( x, T.type, offset ); offset := offset + T.width\}; D$

$\Rightarrow x : real \{enter( x, real, 0 ); offset := 8\}; D$

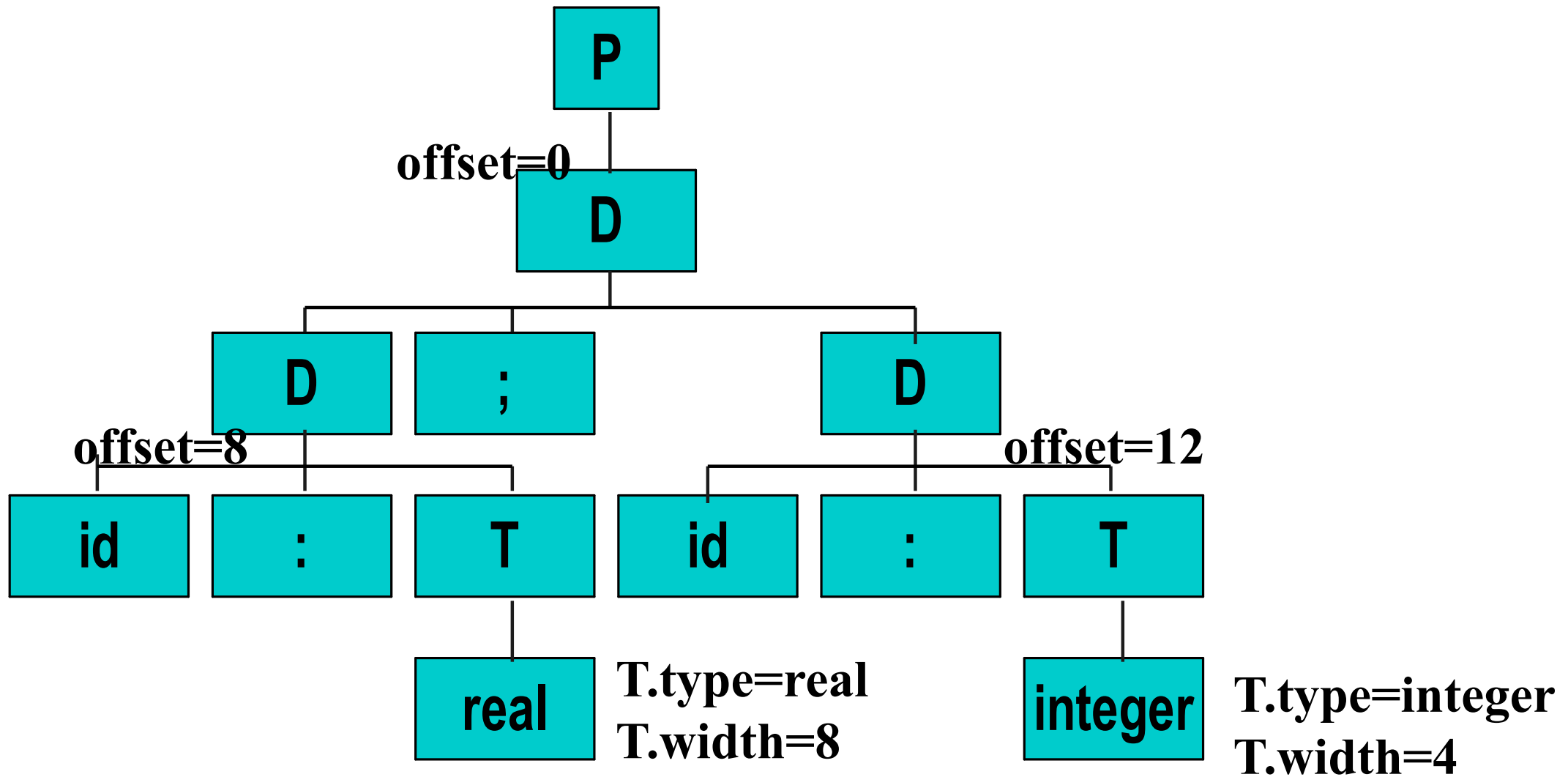
$\Rightarrow x : real \{enter( x, real, 0 ); offset := 8\}; i : T \{enter( id.name, T.type, offset ); offset := offset + T.width\}$

$\Rightarrow x : real \{enter ( x, real, 0 ); offset := 8\}; i : integer \{T.type := integer; T.width := 4\} \{enter( i, T.type, offset ); offset := offset + T.width\}$

$\Rightarrow x : real \{enter( x, real, 0 )\}; i : integer \{enter( i, integer, 8 ) ; offset 12\}$

## 例 7-4 x:real; i:integer 的翻译

$D \rightarrow id : T \{ \text{enter}(id.name, T.type, offset); \quad offset := offset + T.width \}$



`enter(x,real,0)`

`enter(i,integer,8)`

## 7.3 组合数据说明的翻译

- 分类

- 同结构的组合数据(同质数据结构)

- 数组、集合

- 异结构的组合数据(异质数据结构)

- 记录、结构、联合

- 抽象数据类型

- 类、模块



# 数组的引用与分配策略

## ■ 操作

### ■ 元素的引用、修改:

- 数组:  $A[i,j,\dots,k]$

- 记录、结构、联合:  $B.h.j$

### ■ 结构的引用、修改: $A$ , $B$ , $A.c$

## ■ 分配策略

- 静态: 直接完成相应的分配工作

- 动态: 构造代码, 以在运行时调用分配过程

# 数组说明的翻译

- 符号表及有关表格(内情向量表)处理

- 维数、下标上界、下标下界

- 空间分配

- 首地址、需用空间计算

- 存放方式

- 按行存放、按列存放——影响具体元素地址的计算

c: 计算元素偏移地址时的不变量

a: 数组首元素的地址

n: 数组维数

<b>low<sub>1</sub></b>	<b>up<sub>1</sub></b>	<b>d<sub>1</sub></b>
<b>low<sub>2</sub></b>	<b>Up<sub>2</sub></b>	<b>d<sub>2</sub></b>
.....		
<b>low<sub>n</sub></b>	<b>up<sub>n</sub></b>	<b>d<sub>n</sub></b>
<b>c</b>	<b>a</b>	<b>n</b>

内情向量表

## 静态数组分配要完成的工作

---

- 数组存放在一个连续的存储区中
- 知道起始地址
- 要计算该数组的大小
- 按照与简单变量类似的方式进行分配

# 动态数组分配要完成的工作

- 准备

- 上下界的计算
- 体积的计算

- 动态分配子程序

- 将计算的结果告诉动态分配子程序
- 进行分配

# 动态分配方案下数组说明的代码结构

$D \rightarrow \text{id:array } [\text{low}_1:\text{up}_1, \dots, \text{low}_n:\text{up}_n] \text{ of } T$

$\text{low}_1.\text{code}$

送工作单元  $W_1$

$\text{up}_1.\text{code}$

送工作单元  $W_2$

.....

$\text{low}_n.\text{code}$

送工作单元  $W_{2n-1}$

$\text{up}_n.\text{code}$

送工作单元  $W_{2n}$

其它参数 (n, type)

转：动态分配子程序入口

# 数组元素的引用

## ■ 数组元素的翻译

- 完成上下界检查
- 生成完成相对地址的计算代码

## ■ 目标

- $x := y[i]$  和  $y[i] := x$
- $y$  为数组地址的固定部分——相当于第1个元素的地址， $i$  是相对地址，不是数组下标

## 数组元素地址计算-按行存放

- 一维数组  $A[\text{low}_1:\text{up}_1]$  ( $n_1 = \text{up}_1 - \text{low}_1 + 1$ )

$$\begin{aligned}\text{Addr}(A[i]) &= \text{base} + (i - \text{low}_1) * w \quad /* \text{注: } w \text{ 为类型的宽度} */ \\ &= (\text{base} - \text{low}_1 * w) + i * w = c + i * w\end{aligned}$$

- 二维数组  $A[\text{low}_1:\text{up}_1, \text{low}_2:\text{up}_2] \sim A[i_1, i_2]$

$$\begin{aligned}\text{Addr}(A[i_1, i_2]) &= \text{base} + ((i_1 - \text{low}_1) * n_2 * w + (i_2 - \text{low}_2)) * w \\ &= \text{base} + (i_1 - \text{low}_1) * n_2 * w + (i_2 - \text{low}_2) * w \\ &= \text{base} - \text{low}_1 * n_2 * w - \text{low}_2 * w + i_1 * n_2 * w + i_2 * w \\ &= \text{base} - (\text{low}_1 * n_2 - \text{low}_2) * w + (i_1 * n_2 + i_2) * w \\ &= c + (i_1 * n_2 + i_2) * w\end{aligned}$$

- n维数组:

$$\text{Addr}(A[E_1, E_2, \dots, E_k]) = c + (((E_1 * n_2 + E_2) * n_3 + \dots + E_{k-1}) * n_k + E_k) * w$$

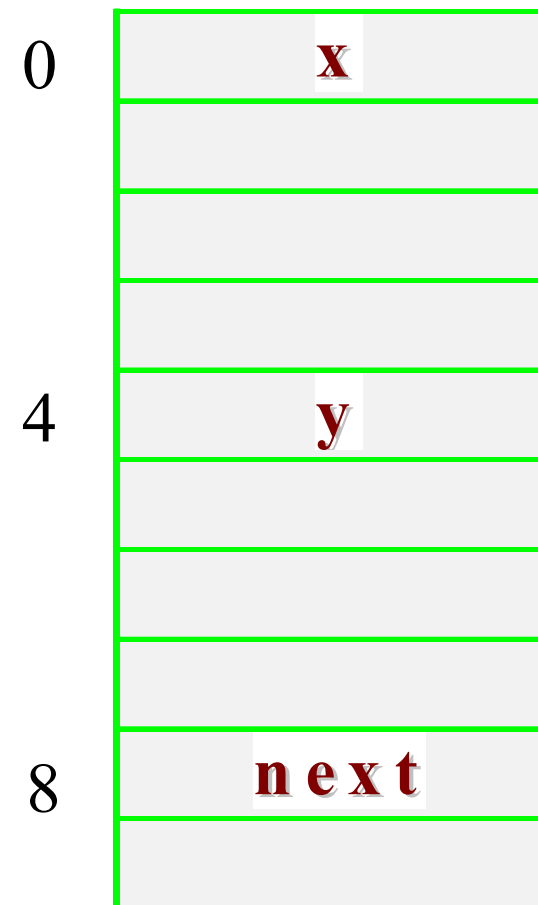
# 记录说明的翻译

- 空间分配

- 设置首地址和各元素的相对地址
- 对齐: 大于所需的空间

- 例:

```
struct node {  
    float x, y;  
    struct node *next;  
} node;
```





# 记录说明的翻译

- 符号表及有关表格处理
  - 各元素的名字、类型、字节数
- 生成代码，完成元素的相对地址的计算
- 目标
  - $y := x[i]$  和  $x[i] := y$
  - $x$  是记录名， $i$  是相对地址
  - 间接地址问题：  $*x = y$

## 7.4 赋值语句的翻译

### 翻译的一般需求

- 充分了解各种语法成分的语义
  - 包括：控制结构、数据结构、单词
  - 了解它们的实现方法
- 了解目标语言的语义
  - 了解中间代码的语义
  - 了解运行环境
  - 了解目标代码的语义

## 7.4 赋值语句的翻译

- 基本子程序

- 产生一条中间代码 `gen(code)`, `emit(code)`
- 产生新的临时变量 `newtemp`

- 属性设置

- 继续用属性 `code` 表示中间代码序列
- 继续用属性 `place` 表示存储位置

## 7.4 赋值语句的翻译

$S \rightarrow id := E$	$S.code := E.code \parallel gen( id.place := 'E.place )$
$E \rightarrow E_1 + E_2$	$E.place := newtemp;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E.place := 'E_1.place + 'E_2.place)$
$E \rightarrow E_1 * E_2$	$E.place := newtemp;$ $E.code := E_1.code \parallel E_2.code \parallel$ $gen(E.place := 'E_1.place * 'E_2.place)$

## 7.4 赋值语句的翻译

$E \rightarrow - E_1$

$E.place := newtemp;$

$E.code := E_1.code \parallel gen(E.place' := 0 - E_1.place)$

$E \rightarrow ( E_1 )$

$E.place := E_1.place; \quad E.code := E_1.code$

$E \rightarrow id$

$E.place := id.place; \quad E.code := ''$

$E \rightarrow num$

$E.place := num.val; \quad E.code := ''$

## 例 7-5 $a := -c + b * 34$ 的中间代码生成过程

S.code

$\Rightarrow$  E.code || gen( 'a:='E.place )  
/\* 'a' 为 id.place \*/

$\Rightarrow$  E<sub>1</sub>.code || E<sub>2</sub>.code  
|| gen( 't1:='E<sub>1</sub>.place+'E<sub>2</sub>.place )  
|| gen( 'a:=t1' )

/\* newtemp t1 为 E.place \*/

$\Rightarrow$  E<sub>11</sub>.code || gen( 't2:= 0 -' E<sub>11</sub>.place )  
/\* newtemp t2 为 E<sub>1</sub>.place \*/

```

|| E21.code || E22.code
|| gen( 't3:= ' E21.place '*' E22.place )
    /* newtemp t3 为 E2.place */
|| gen( 't1:=t2+t3' ) || gen( 'a:=t1' )
⇒ gen( 't2:= 0 - c' ) || gen( 't3:=b*34' )
    /* 'c' 为 E11.place */
    /* 'b' 为 E21.place */
    /* '34' 为 E22.place */
|| gen( 't1:=t2+t3' ) || gen( 'a:=t1' )
    /* E21.code 和 E22.code 为空 */

```

## 7.4 赋值语句的翻译

---

- 1) 找出分析树中使用的产生式
- 2) 根据产生式的语义规则，计算式子中的各属性
- 3) 反复使用 1) 和 2) 改写式子，最后得到代码生成语句组成的式子

组成语句的翻译结果（中间代码序列）

结果：开始符号的属性 S.code



## 表达式翻译中的其它问题

- 临时变量空间的统计
  - 了解需求、及时释放
- 运算合法性检查
  - 利用符号表保存的名字类型
- 类型自动转换
  - 填加专用指令

## 类型转换: itr, rti

```
S → id := E   S.code := E.code || {if id.type=E.type then
    gen( id.place:='E.place ) else if id.type=real then
    gen(id.place':=' itr( E.place) else gen(id.place':='rti ( E.place)}
E → E1 + E2   E.place := newtemp; E.code := E1.code || E2.code ||
    {E.type:=E1.type;
    if E1.type=E2.type then gen(E.place':='E1.place'+E2.place)
        else{ if E1.type=real then gen(E.place':='E1.place'+itr(E2.place))
            else gen(E.place':='itr(E1.place)'+E2.place);
            E.type=real}}
```

# 带有数组引用的赋值语句的翻译

- n维数组元素地址的计算公式:

$$\text{Addr}(A[E_1, E_2, \dots, E_k]) = c + (((E_1 * n_2 + E_2) * n_3 + \dots + E_{k-1}) * n_k + E_k) * w$$

- 文法:

- $S \rightarrow L := E, \quad E \rightarrow L, \quad L \rightarrow \text{id} \mid \text{Elist}, \quad \text{Elist} \rightarrow \text{id}[E \mid \text{Elist1}, E]$

- 约定:

- array: 指向符号表中指向数组名表项的指针, Elist 的继承属性
  - Elist.ndim: 记录Elist中的维数 (下标表达式的个数)
  - limit (array, j) 返回nj, 即由array指示的数组的第j维的维长

$\text{id}[E_1, E_2, \dots, E_k]$

$S \rightarrow L := E$       {if  $L.\text{offset} = \text{null}$  then  $\text{gen}(L.\text{place}' := 'E.\text{place})$   
                              else  $\text{gen}(L.\text{place}[L.\text{offset}]' := 'E.\text{place})$ }

$E \rightarrow L$             {if  $L.\text{offset} = \text{null}$  then  $E.\text{place} := L.\text{place}$  else  
                              { $E.\text{place} := \text{newtemp}; \text{gen}(E.\text{place}' := 'L.\text{place}[L.\text{offset}])$ }}

$L \rightarrow \text{id}$             { $L.\text{place} := \text{id.place}; L.\text{offset} := \text{null}$ }

$L \rightarrow \text{Elist}$         { $L.\text{place} := \text{newtemp}; L.\text{offset} := \text{newtemp};$   
                               $\text{gen}(L.\text{place}' := 'c(\text{Elist.array}))$ ;  
                               $\text{gen}(L.\text{offset}' := 'Elist.place * \text{width}(\text{Elist.array}))$ }

$\text{Elist} \rightarrow \text{id}[E$     { $\text{Elist.place} := E.\text{place}; \text{Elist.ndim} := 1$ ;  
                               $\text{Elist.array} := \text{id.place}$ }

$\text{Elist} \rightarrow \text{Elist}_1, E$  { $t := \text{newtemp}; m := \text{Elist}_1.\text{ndim} + 1$ ;  
                               $\text{gen}(t' := 'Elist_1.place' * 'limit(\text{Elist.array}, m)$ );  
                               $\text{gen}(t' := 't' + 'E.place)$ ;  $\text{Elist.array} := \text{Elist}_1.\text{array}$ ;  
                               $\text{Elist.place} := t$ ;  $\text{Elist.ndim} := m$ }

## 7.5 控制语句的翻译

- 高级语言的控制结构
  - 顺序 begin 语句; ... ; 语句end
  - 条件 if\_then\_else、if\_then  
switch、case
  - 循环 while\_do、do\_while  
for、repeat\_until
  - 转移语句及其三地址码
    - goto n (j,\_,\_,n)
    - if x relop y goto n (jrelop,x,y,n)

# 布尔表达式的翻译

## ■ 基本文法

$E \rightarrow E_1 \text{ or } E_2 \mid E_1 \text{ and } E_2 \mid \text{not } E_1 \mid (E_1) \mid \text{id}_1 \text{ relop id}_2 \mid \text{id}$

## ■ 处理方式

### ■ 数值表示法（与算术表达式的处理类似）

- 真:  $E.\text{place}=1$
- 假:  $E.\text{place}=0$
- **newtemp**——申请临时工作单元

### ■ 真假出口表示法（作为条件控制）

- 真出口:  $E.\text{true}$
- 假出口:  $E.\text{false}$
- **newlab**——申请新标号

## 数值表示法——使用综合属性

$E \rightarrow E_1 \text{ or } E_2$      $E.\text{place} := \text{newtemp};$

$E.\text{code} := E_1.\text{code} \parallel E_2.\text{code} \parallel$

$\text{gen}(E.\text{place} := 'E_1.\text{place}' \text{ or } 'E_2.\text{place}')$

$E \rightarrow E_1 \text{ and } E_2$      $E.\text{place} := \text{newtemp};$

$E.\text{code} := E_1.\text{code} \parallel E_2.\text{code} \parallel$

$\text{gen}(E.\text{place} := 'E_1.\text{place}' \text{ and } 'E_2.\text{place}')$

$E \rightarrow \text{not } E_1$      $E.\text{place} := \text{newtemp};$

$E.\text{code} := E_1.\text{code} \parallel \text{gen}(E.\text{place} := ' ' \text{ not } 'E_1.\text{place}')$

## 数值表示法——使用综合属性

$E \rightarrow (E_1)$      $E.place := E_1.place; E.code := E_1.code$

$E \rightarrow id_1 \text{ relop } id_2$      $E.place := \text{newtemp};$

$E.code := \text{gen}(' \text{if } id_1.place \text{ relop } id_2.place \text{ goto}'$   
 $\text{nextstat}+3) ||$

$\text{gen}(E.place := " 0") ||$

$\text{gen}(' \text{goto } \text{nextstat}+2) ||$

$\text{gen}(E.place := " 1")$

$E \rightarrow id$      $E.place := id.place; E.code := "$

注:nextstat 是下一条三地址码指令在输出序列中的序号



## 真假出口表示法——使用继承属性

$E \rightarrow E_1 \text{ or } E_2$      $E_1.\text{true} := E.\text{true};$   $E_1.\text{false} := \text{newlab};$   
 $E_2.\text{true} := E.\text{true};$   $E_2.\text{false} := E.\text{false};$   
 $E.\text{code} := E_1.\text{code} \parallel \text{gen}(E_1.\text{false}':')$

$E \rightarrow E_1 \text{ and } E_2$      $E_1.\text{true} := \text{newlab};$   $E_1.\text{false} := E.\text{false};$   
 $E_2.\text{true} := E.\text{true};$   $E_2.\text{false} := E.\text{false};$   
 $E.\text{code} := E_1.\text{code} \parallel \text{gen}(E_1.\text{true}':') \parallel E_2.\text{code}$

$E \rightarrow \text{not } E_1$      $E_1.\text{true} := E.\text{false};$   $E_1.\text{false} := E.\text{true}$   
 $E.\text{code} := E_1.\text{code}$

## 真假出口表示法——使用继承属性

$E \rightarrow (E_1)$        $E_1.true := E.true; E_1.false := E.false$   
                     $E.code := E_1.code$

$E \rightarrow id_1 \text{ relop } id_2$   
                     $E.code := \text{gen}('if' \ id_1.place \ \text{relop} \ id_2.place \ 'goto' \ E.true) \ ||$   
                     $\text{gen}('goto' \ E.false)$

$E \rightarrow id$              $E.code := \text{gen}('if' \ id_1.place \ 'goto' \ E.true) \ ||$   
                     $\text{gen}('goto' \ E.false)$

## 真假出口表示法——使用继承属性

---

$E \rightarrow E_1 \text{ relop } E_2$

$E.\text{code} := E_1.\text{code} \ || \ E_2.\text{code}$

$\ || \ \text{gen}(\text{'if' } E_1.\text{place relop } E_2.\text{place 'goto' } E.\text{true})$

$\ || \ \text{gen}(\text{'goto' } E.\text{false})$

例如：  $a < b$  or  $c < d$  and  $e < f$

---

if  $a < b$  goto Etrue

goto L1

L1:if  $c < d$  goto L2

goto Efalse

L2:if  $e < f$  goto Etrue

goto Efalse

Efalse、Etrue在此或者在此之后：需要返回去  
填写

例如：  $4+a > b-c$  and  $d$

---

$t1 = 4 + a$

$t2 = b - c$

if  $t1 > t2$  goto L1

goto Efalse

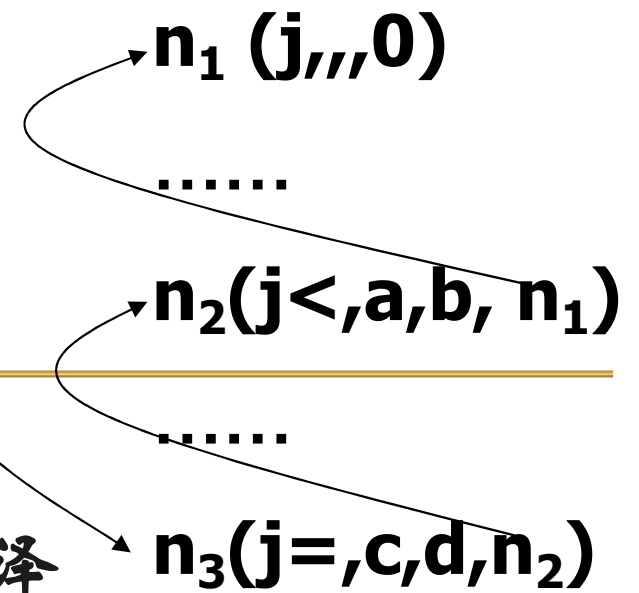
L1: if  $d$  goto Etrue

goto Efalse

Efalse、Etrue在此或者在此之后：需要返回去填写

E.Turelist  $n_3$

## 拉链回填



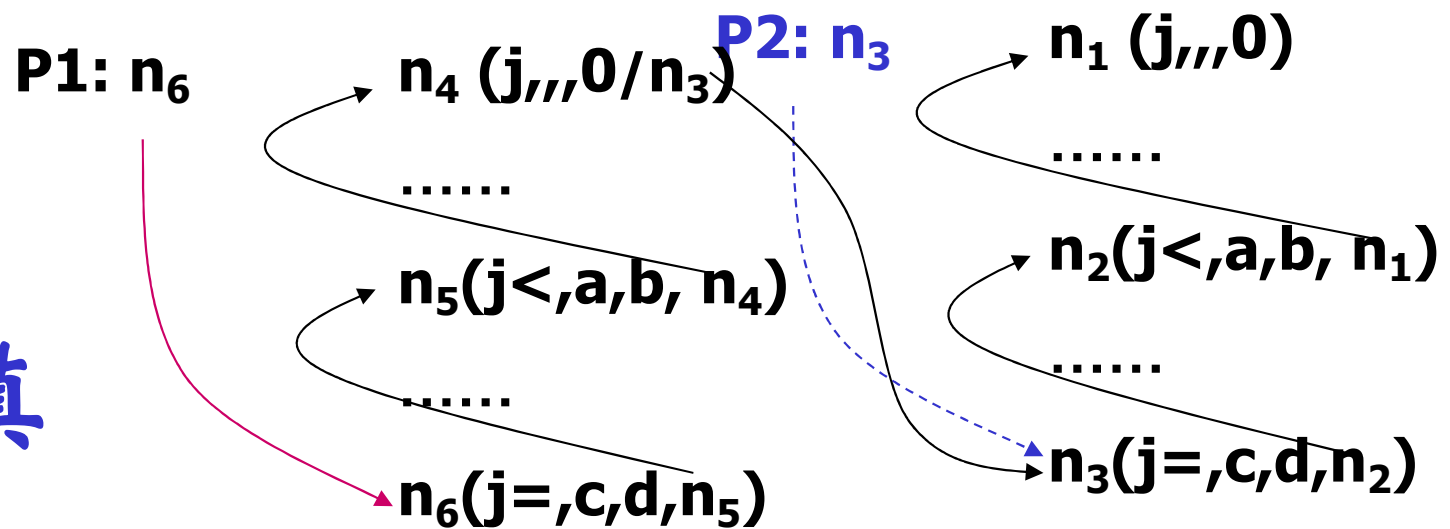
- 两遍扫描:

- 构造语法树，遍历语法树并翻译
- 效率低

- 一遍扫描

- 先产生暂时没有填写目标标号的转移指令;
- 对于每一条这样的指令作适当的记录;
- 一旦目标标号被确定下来，再将它“回填”到相应的指令中
- 布尔表达式E设属性E.turelist和E.falselist
  - E.truelist——对应真出口 Etrue
  - E.falselist——对应假出口 Efalse

## 拉链回填

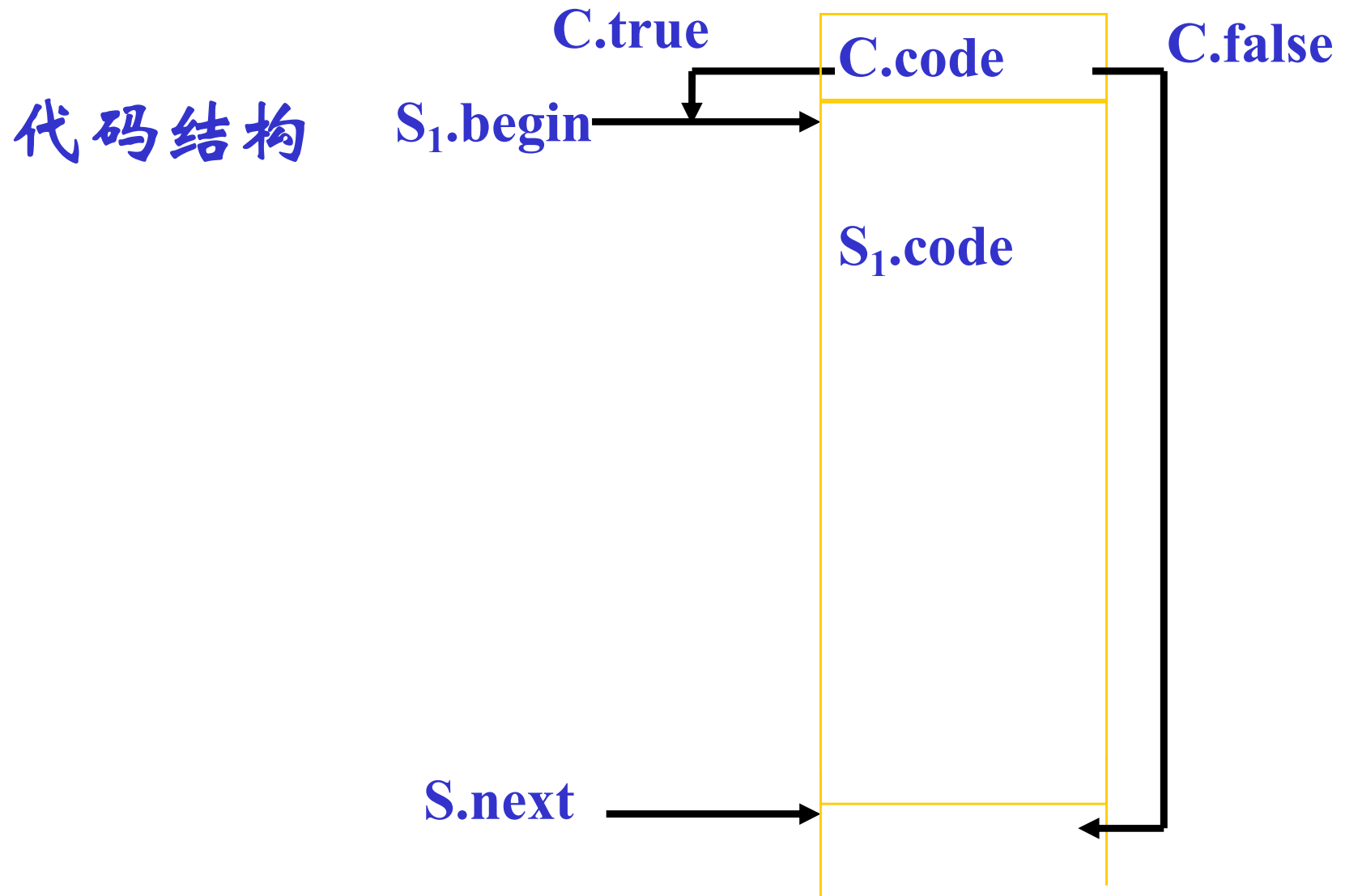


翻译模式用到如下三个函数：

1. **makelist( $i$ )**: 创建一个仅包含 $i$ 的新表,  $i$  是四元式数组的一个索引 (地址), 或说 $i$ 是四元式代码序列的一个标号
2. **merge( $p_1, p_2$ )**: 将由 $p_2$ 指向的表接在 $p_1$ 所指向的表后, 返回 $p_1$
3. **backpatch ( $p, i$ )**: 把 $i$ 作为目标标号回填到 $p$ 所指向的表中的每一个转移指令中去。

此处的“表”都是为“回填”所拉的链

$S \rightarrow \text{if } C \text{ then } S_1 \text{ 的翻译}$





例：if  $a > b$  then  $a = a + a$

---

if  $a > b$  goto Ctrue      ( *$S_1.begin$* )

goto Cfalse      ( *$S.next$* )

$S_1.begin$ :  $t1 := a + a$

$a := t1$

$S.next$ :

## if C then $S_1$ 的属性文法

$S \rightarrow$  if C then  $S_1$    C.true := newlabel;  
                                   $S_1$ .next := C.false := S.next;  
                                  S.code := C.code ||  
                                  gen( C.true ':' ) ||  $S_1$ .code

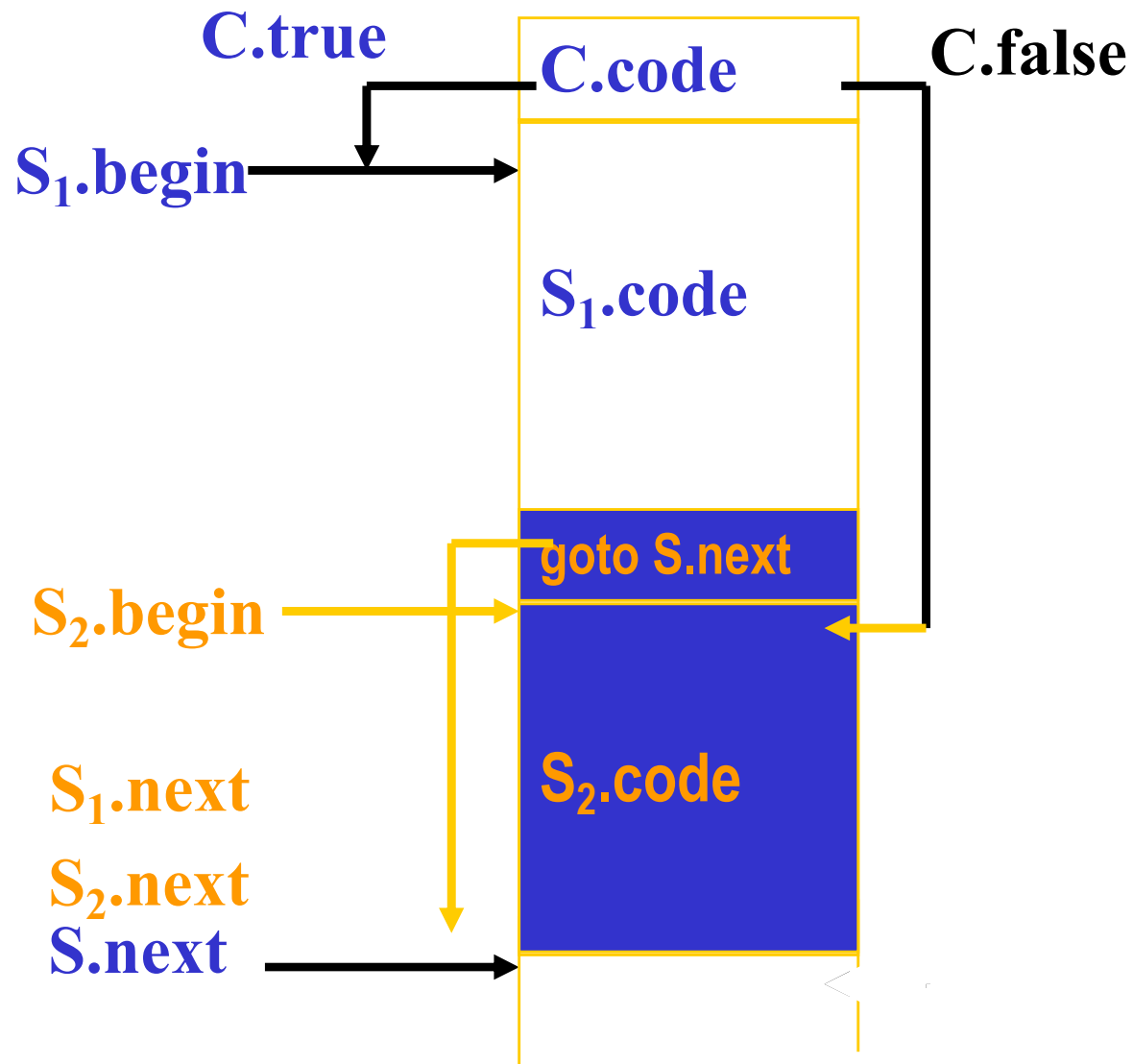
语义过程 newlabel 产生新的标号

例： if a>b then if b>c then a=b+c

例： if a>b and b>c then a=b+c

$S \rightarrow \text{if } C \text{ then } S_1 \text{ else } S_2 \text{ 的翻译}$

## 代码结构



例： if  $a > b$  then  $a = a + c$  else  $a = a + b$

---

if  $a > b$  goto L1      (*Ctrue, S<sub>1</sub>.begin*)

goto L2      (*Cfalse, S<sub>2</sub>.begin*)

L1:     $t1 := a + c$       *Ctrue, S<sub>1</sub>.begin*

$a := t1$

goto S.next

L2:     $t2 := a + b$       *Cfalse, S<sub>2</sub>.begin*

$a := t2$

S.next:

## if-then-else 条件语句的属性文法

$S \rightarrow \text{if } C \text{ then } S_1$   
           $\text{else } S_2$

$C.\text{true} := \text{newlabel};$   
 $C.\text{false} := \text{newlabel};$   
 $S_1.\text{next} := S_2.\text{next} := S.\text{next};$   
 $S.\text{code} := C.\text{code} || \text{gen}(C.\text{true}':') ||$   
           $S_1.\text{code} || \text{gen}(\text{'goto' } S.\text{next}) ||$   
           $\text{gen}(C.\text{false}':') || S_2.\text{code}$

**S<sub>1</sub>.begin**

**S<sub>2</sub>.begin**

# $S \rightarrow \text{while } C \text{ do } S_1$ 翻译

## 代码结构

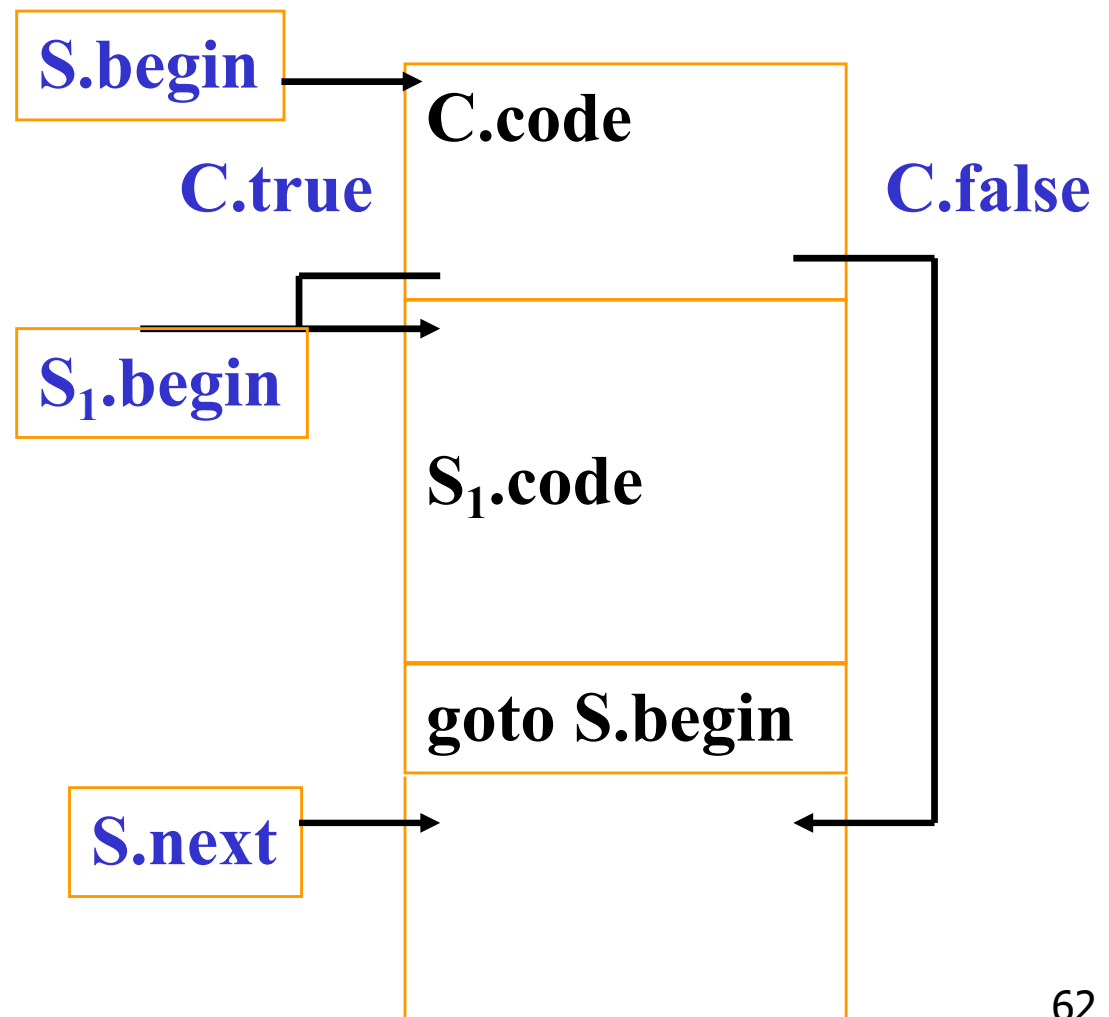
### ■ 属性设置 (继承)

#### ■ 布尔式 $C$

- 代码段真出口 true
- 代码段假出口 false

#### ■ 语句 $S$

- 代码段的入口 begin
- 后续段入口 next



例:  $\text{while } x > y \text{ do } z := x + 1$

---

L1:if  $x > y$  L2

*Ctrue,  $S_1.begin$*

goto L3

*Cfalse,  $S.next$*

L2:t1=x+1

*$S_1.begin$*

z=t1

goto L1

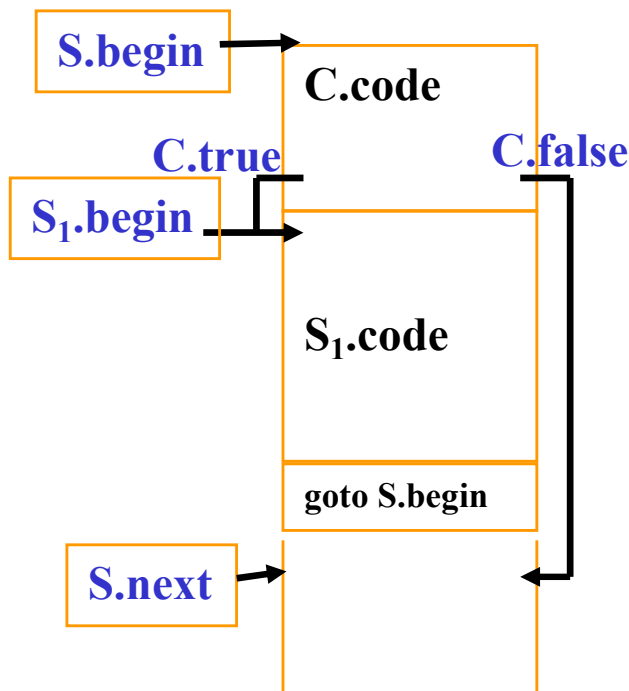
*$S.begin$*

L3:

## 循环语句的属性文法

$S \rightarrow \text{while } C \text{ do } S_1$      $S.\text{begin} := S_1.\text{next} := \text{newlabel};$   
    $C.\text{true} := S_1.\text{begin} := \text{newlabel};$   
    $C.\text{false} := S.\text{next};$

$S.\text{code} := \text{gen}(S.\text{begin}':') \mid\mid C.\text{code} \mid\mid$   
                                  $\text{gen}(C.\text{true}':') \mid\mid S_1.\text{code} \mid\mid$   
                                  $\text{gen}('goto' S.\text{begin})$





## 例 7-6：翻译下列语句

**while a < b do**

**E<sub>1</sub>**

**if c < 5 then**

**E<sub>2</sub>**

**S<sub>3</sub>** { **S<sub>1</sub>** **while x > y do z := x + 1;**

**else**

**S<sub>2</sub>** **x := y**

while a<b do if c<5 then while x>y do z:=x+1 else x:=y

生  
成  
的  
三  
地  
址  
码  
序  
列

```
L1: if a < b goto L2
      goto Lnext
L2: if c < 5 goto L3
      goto L4
L3: if x > y goto L5
      goto L1
L5: t1 := x + 1
      z := t1
      goto L3
L4: x := y
      goto L1
Lnext:
```

} E<sub>1</sub>.code

} E<sub>2</sub>.code

} S<sub>1</sub>.code

} S<sub>2</sub>.code

} S<sub>3</sub>.code