

## 第六章 中间代码生成

# 类型表达式

哈尔滨工业大学 陈鄞



## 类型表达式 (*Type Expressions*)

- 基本类型是类型表达式
  - *integer*
  - *real*
  - *char*
  - *boolean*
  - *type\_error* (出错类型)
  - *void* (无类型)

## 类型表达式 (*Type Expressions*)

- 基本类型是类型表达式
- 可以为类型表达式命名, 类型名也是类型表达式
- 将类型构造符(*type constructor*)作用于类型表达式可以构成新的类型表达式
  - 数组构造符 *array*
    - 若 *T* 是类型表达式, 则 *array ( I, T )* 是类型表达式 (*I* 是一个整数)

类型	类型表达式
<i>int</i> [3]	<i>array</i> (3, <i>int</i> )
<i>int</i> [2][3]	<i>array</i> (2, <i>array</i> (3, <i>int</i> ) )

## 类型表达式 (*Type Expressions*)

- 基本类型是类型表达式
- 可以为类型表达式命名，**类型名**也是类型表达式
- 将**类型构造符**(*type constructor*)作用于**类型表达式**可以构成新的类型表达式
  - 数组构造符`array`
  - 指针构造符`pointer`
    - 若 $T$ 是类型表达式，则 `pointer ( T )` 是类型表达式，它表示一个指针类型

## 类型表达式 (*Type Expressions*)

- 基本类型是类型表达式
- 可以为类型表达式命名，**类型名**也是类型表达式
- 将**类型构造符**(*type constructor*)作用于**类型表达式**可以构成新的类型表达式
  - 数组构造符 *array*
  - 指针构造符 *pointer*
  - 笛卡尔乘积构造符  $\times$ 
    - 若  $T_1$  和  $T_2$  是类型表达式，则笛卡尔乘积  $T_1 \times T_2$  是类型表达式

## 类型表达式 (*Type Expressions*)

- 基本类型是类型表达式
- 可以为类型表达式命名, 类型名也是类型表达式
- 将类型构造符(*type constructor*)作用于类型表达式可以构成新的类型表达式
  - 数组构造符 *array*
  - 指针构造符 *pointer*
  - 笛卡尔乘积构造符  $\times$
  - 函数构造符  $\rightarrow$ 
    - 若  $T_1$ 、 $T_2$ 、...、 $T_n$  和  $R$  是类型表达式, 则  $T_1 \times T_2 \times \dots \times T_n \rightarrow R$  是类型表达式

## 类型表达式 (*Type Expressions*)

- 基本类型是类型表达式
- 可以为类型表达式命名，**类型名**也是类型表达式
- 将**类型构造符**(*type constructor*)作用于**类型表达式**可以构成新的类型表达式
  - 数组构造符 *array*
  - 指针构造符 *pointer*
  - 笛卡尔乘积构造符  $\times$
  - 函数构造符  $\rightarrow$
  - 记录构造符 *record*
    - 若有标识符  $N_1, N_2, \dots, N_n$  与类型表达式  $T_1, T_2, \dots, T_n$ ，则  $record((N_1 \times T_1) \times (N_2 \times T_2) \times \dots \times (N_n \times T_n))$  是一个类型表达式

## 例

➤ 设有C程序片段：

```
struct stype
{ char[8] name;
  int score;
};
stype[50] table;
stype* p;
```

➤ 和*stype*绑定的类型表达式

➤ *record* ( (*name* × *array*(8, *char*)) × (*score* × *integer*) )


➤ 和*table*绑定的类型表达式

➤ *array* (50, *stype*)

➤ 和*p*绑定的类型表达式

➤ *pointer* (*stype*)






## 第六章 中间代码生成

# 类型表达式

哈尔滨工业大学 陈鄞





## 第六章 中间代码生成

# 声明语句的翻译

哈尔滨工业大学 陈鄞



## 局部变量的存储分配

- 对于声明语句，语义分析的主要任务就是收集标识符的 **类型** 等属性信息，并为每一个名字分配一个 **相对地址**
- 从类型表达式可以知道该类型在运行时刻所需的存储单元数量称为 **类型的宽度**(width)
- 在 **编译时刻**，可以使用类型的宽度为每一个名字分配一个 **相对地址**
- 名字的 **类型** 和 **相对地址** 信息保存在相应的 **符号表** 记录中

## 变量声明语句的SDT

$enter(name, type, offset)$ : 在符号表中为名字 $name$ 创建记录, 将 $name$ 的类型设置为 $type$ , 相对地址设置为 $offset$

- ①  $P \rightarrow \{ offset = 0 \} D$
- ②  $D \rightarrow T id; \{ enter(id.lexeme, T.type, offset);$   
 $offset = offset + T.width; \} D$
- ③  $D \rightarrow \varepsilon$
- ④  $T \rightarrow B \{ t = B.type; w = B.width; \}$   
 $C \{ T.type = C.type; T.width = C.width; \}$
- ⑤  $T \rightarrow \uparrow T_1 \{ T.type = pointer(T_1.type); T.width = 4; \}$
- ⑥  $B \rightarrow int \{ B.type = int; B.width = 4; \}$
- ⑦  $B \rightarrow real \{ B.type = real; B.width = 8; \}$
- ⑧  $C \rightarrow \varepsilon \{ C.type = t; C.width = w; \}$
- ⑨  $C \rightarrow [num] C_1 \{ C.type = array(num.val, C_1.type);$   
 $C.width = num.val * C_1.width; \}$

符号	综合属性
$B$	$type, width$
$C$	$type, width$
$T$	$type, width$

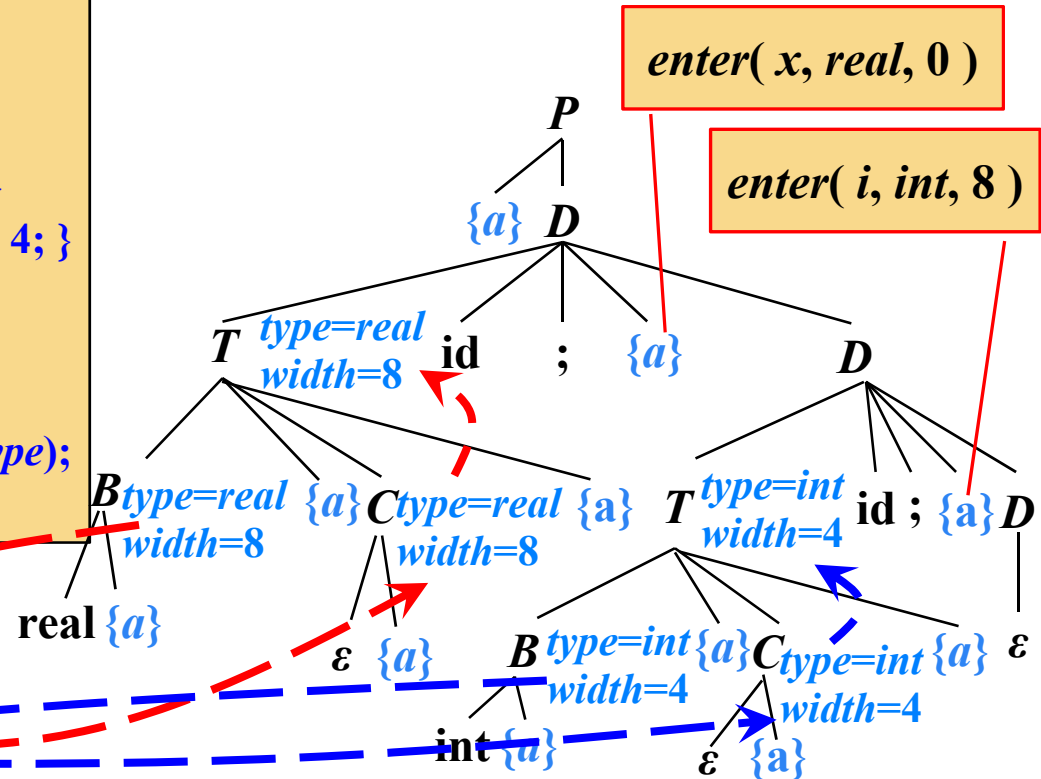
变量	作用
$offset$	下一个可用的相对地址
$t, w$	将类型和宽度信息从语法分析树中的 $B$ 结点传递到对应于产生式 $C \rightarrow \varepsilon$ 的结点

## 例：“*real x; int i;*”的语法制导翻译

- ①  $P \rightarrow \{ \text{offset} = 0 \} D$
- ②  $D \rightarrow T \text{ id}; \{ \text{enter}(\text{id.lexeme}, T.\text{type}, \text{offset}); \text{offset} = \text{offset} + T.\text{width}; \} D$
- ③  $D \rightarrow \varepsilon$
- ④  $T \rightarrow B \quad \{ t = B.\text{type}; w = B.\text{width}; \}$   
 $\quad C \quad \{ T.\text{type} = C.\text{type}; T.\text{width} = C.\text{width}; \}$
- ⑤  $T \rightarrow \uparrow T_1 \{ T.\text{type} = \text{pointer}(T_1.\text{type}); T.\text{width} = 4; \}$
- ⑥  $B \rightarrow \text{int} \{ B.\text{type} = \text{int}; B.\text{width} = 4; \}$
- ⑦  $B \rightarrow \text{real} \{ B.\text{type} = \text{real}; B.\text{width} = 8; \}$
- ⑧  $C \rightarrow \varepsilon \{ C.\text{type} = t; C.\text{width} = w; \}$
- ⑨  $C \rightarrow [\text{num}] C_1 \{ C.\text{type} = \text{array}(\text{num.val}, C_1.\text{type}); C.\text{width} = \text{num.val} * C_1.\text{width}; \}$

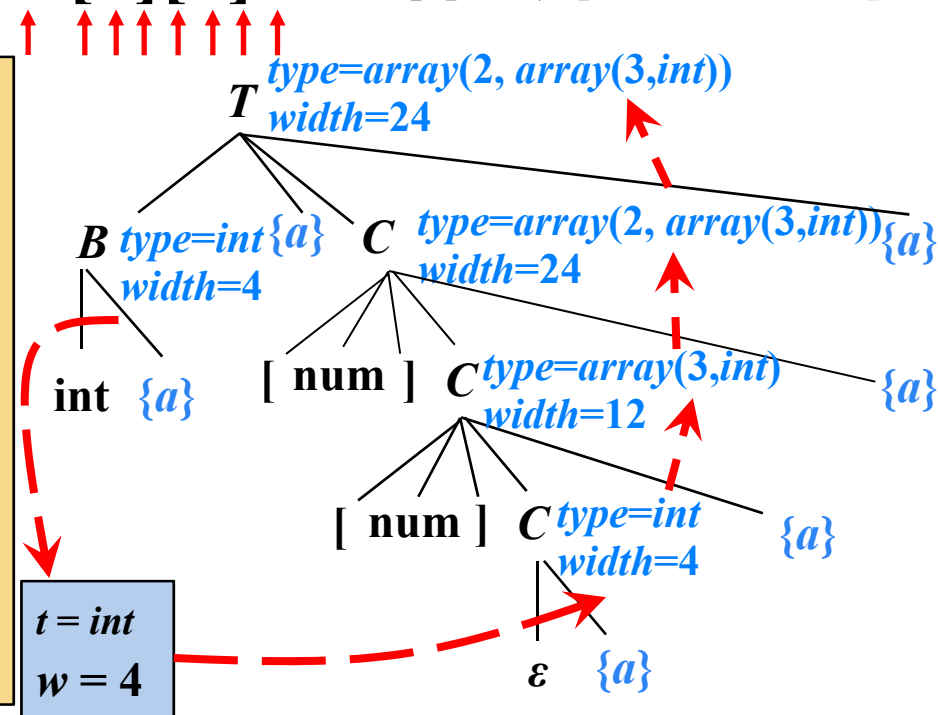
offset = 12

$t = \text{int}$   
 $w = 4$




## 例：数组类型表达式“ $int[2][3]$ ”的语法制导翻译

- ①  $P \rightarrow \{ offset = 0 \} D$
- ②  $D \rightarrow T id; \{ enter( id.lexeme, T.type, offset );$   
 $offset = offset + T.width; \} D$
- ③  $D \rightarrow \varepsilon$
- ④  $T \rightarrow B \quad \{ t = B.type; w = B.width; \}$   
 $C \quad \{ T.type = C.type; T.width = C.width; \}$
- ⑤  $T \rightarrow \uparrow T_1 \{ T.type = pointer( T_1.type ); T.width = 4; \}$
- ⑥  $B \rightarrow int \{ B.type = int; B.width = 4; \}$
- ⑦  $B \rightarrow real \{ B.type = real; B.width = 8; \}$
- ⑧  $C \rightarrow \varepsilon \{ C.type = t; C.width = w; \}$
- ⑨  $C \rightarrow [num] C_1 \{ C.type = array( num.val, C_1.type );$   
 $C.width = num.val * C_1.width; \}$







## 第六章 中间代码生成

# 声明语句的翻译

哈尔滨工业大学 陈鄞

