中国科学技术大学
University of Science and Technology of China

USTC, CHINA
ADSLAB
先进数据系统实验室

《编译原理与技术》
中间代码生成 II
（与类型相关部分）

计算机科学与技术学院
李 诚
2021-11-01

与类型相关的中间代码生成

中国科学技术大学
University of Science and Technology of China

- 符号表的组织
- 声明语句的翻译
- 数组寻址的翻译
- 类型分析的其他应用

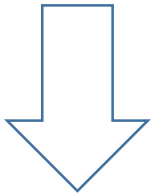
符号表 (Symbol table)

- 符号表的使用和修改伴随编译的全过程
- 存储entity的各种信息
  - 如variable names, function names, objects, classes, interfaces 等
  - 如类型信息、所占用内存空间、作用域
- 用于编译过程中的分析与合成
  - 语义分析：如使用前声明检查、类型检查、确定作用域等
  - 合成：如类型表达式构造、内存空间分配等

符号表 (Symbol table)

代码片段：

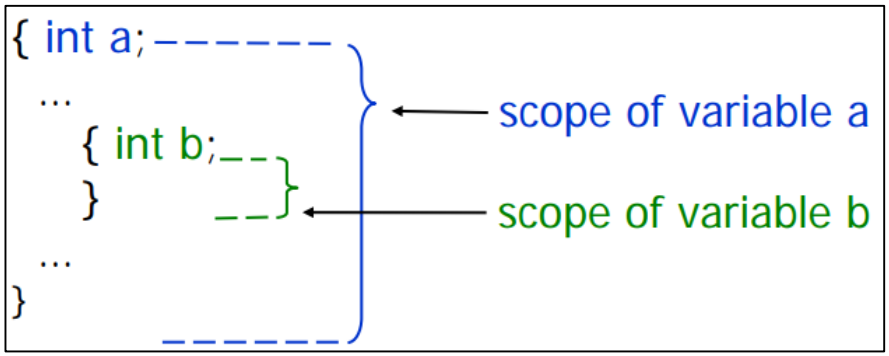
```
extern bool foo(auto int m, const int n);
const bool tmp;
```



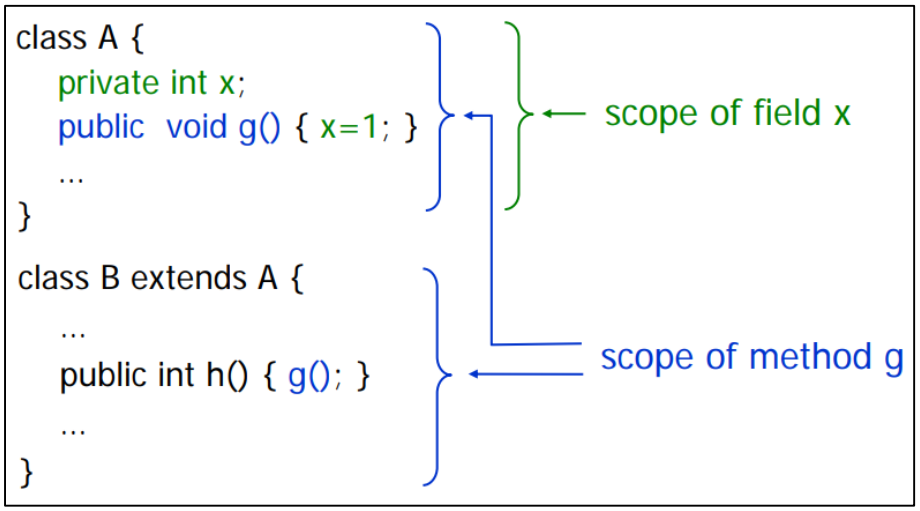
NAME	KIND	TYPE	OTHER
foo	fun	int x int → bool	extern
m	par	int	auto
n	par	int	const
tmp	var	bool	const

符号表

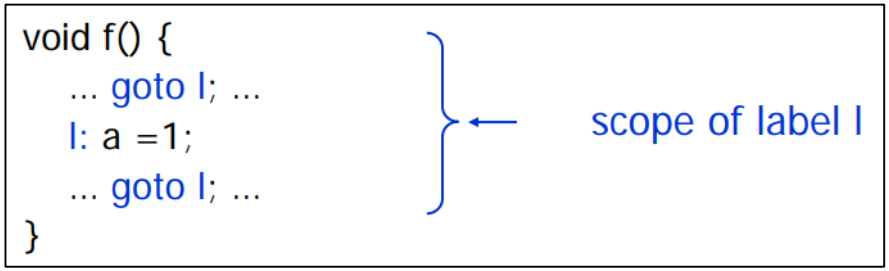
符号表——作用域



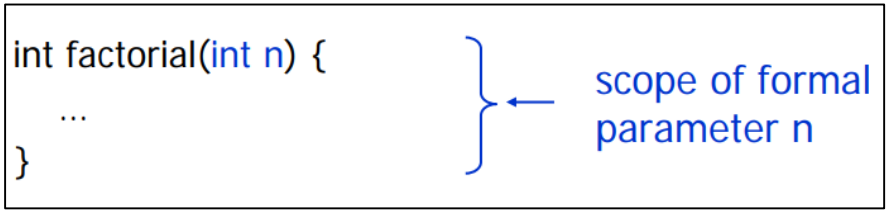
程序块中



对象中的field和methods



语句标号



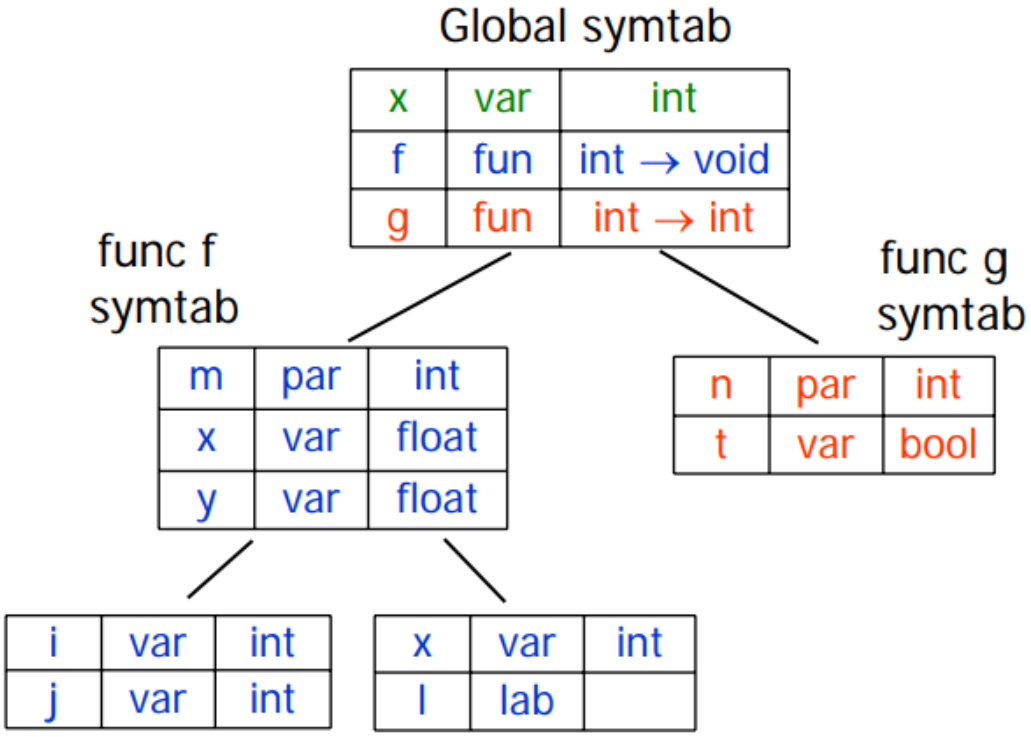
过程或函数定义中的参数

符号表 (Symbol table)


```
int x;

void f(int m) {
    float x, y;
    …
    { int i, j; …; }
    { int x; l: …; }
}

int g(int n) {
    bool t;
    …;
}
```



注：l代表label

符号表 (Symbol table)

- It is built in lexical and syntax analysis phases, used by compiler to achieve compile time efficiency.
  - **Lexical Analysis:** Creates new table entries in the table, example like entries about token.
  - **Syntax Analysis:** Adds information regarding attribute type, scope, dimension, line of reference, use, etc in the table.
  - **Semantic Analysis:** Uses table info to check for semantics i.e. to verify that expressions and assignments are semantically correct(type checking) and update it accordingly.

符号表 (Symbol table)

- It is built in lexical and syntax analysis phases, used by compiler to achieve compile time efficiency.
  - **Intermediate Code generation:** Refers symbol table for knowing how much and what type of run-time is allocated and table helps in adding temporary variable information.
  - **Code Optimization:** For machine dependent optimization.
  - **Target Code generation:** Generates code by using address information of identifier present in the table.

❑具体的操作

Operation	Function
allocate	to allocate a new empty symbol table
free	to remove all entries and free storage of symbol table
lookup	to search for a name and return pointer to its entry
insert	to insert a name in a symbol table and return a pointer to its entry
set_attribute	to associate an attribute with a given entry
get_attribute	to get an attribute associated with a given entry

❑数据结构

- ❖数组、链表、hash表等

与类型相关的中间代码生成

中国科学技术大学
University of Science and Technology of China

- 符号表的组织
- **声明语句的翻译**
  - 类型相关的翻译
- **数组寻址的翻译**
- **类型转换**

声明语句翻译的要点

- 分配存储单元
  - 名字、类型、字宽、偏移
- 作用域的管理
  - 过程调用
- 记录类型的管理
- 不产生中间代码指令，但是要更新符号表

类型化的语言

- 变量的类型
  - 限定了变量在程序执行期间的取值范围和存储空间消耗
- 类型化的语言(typed language)
  - 变量都被给定类型的语言
  - 表达式、语句等程序构造的类型都可以静态确定，运行时不需要额外的操作
- 未类型化的语言(untyped language)
  - 不限制变量值范围的语言,如JavaScript、Perl

类型的两个典型应用

- 静态的语义分析
  - 类型检查

利用**逻辑规则**分析运算分量的类型与运算符预期是否匹配?

- 中间代码生成
  - 声明语句的翻译
  - 数组寻址的翻译
  - 类型转换

- 通过声明语句**收集**变量或函数的类型
- **计算**所占存储空间
- **分配**相对地址
- 类型**转换**适配指令选择

类型的应用

- 静态的语义分析
  - 类型检查
- 中间代码生成
  - 声明语句的翻译
  - 数组寻址的翻译
  - 类型转换

主要内容

- 类型表达式
  - 类型的结构
- 类型等价
  - 结构等价和名字等价
- 类型检查
  - 语法制导翻译方案实现
  - 函数和算符的重载
- 其他知识点

层次一：形式化描述类型结构

层次二：判定两个类型相同的依据

层次三：定义一组逻辑规则检查语句或者表达式中是否存在类型错误

类型表达式 (Type expression)

- 类型可以是语法的一部分，因此也是结构的

考虑以下文法，D代表声明语句，S代表一般语句

$P \rightarrow D\ ;\ S$

$D \rightarrow D\ ;\ D \mid \mathrm{id}\ :\ T$

$T \rightarrow \mathrm{boolean} \mid \mathrm{integer} \mid \mathrm{array}\ [\mathrm{num}\ ]\ \mathrm{of}\ T \mid \uparrow T \mid T\ \text{‘}\rightarrow\text{’}\ T$

类型表达式 (Type expression)

- 类型可以是语法的一部分，因此也是结构的

考虑以下文法，D代表声明语句，S代表一般语句

$P \rightarrow D\ ;\ S$

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

数组　指针　函数

$T \rightarrow \text{boolean} \mid \text{integer} \mid \text{array}\ [\text{num}\ ]\ \text{of}\ T \mid \uparrow T \mid T\ \text{‘}\rightarrow\text{’}\ T$

基本类型　　复杂且可组合的类型

类型表达式 (Type expression)

- 基本类型是类型表达式
  - *integer*
  - *real*
  - *char*
  - *boolean*
  - *type_error //出错类型*
  - *void //无类型*

在类型检查中传递错误

语句的类型

类型表达式 (Type expression)

中国科学技术大学
University of Science and Technology of China

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
    - 如果T是类型表达式，N是一个整数，则*array(N, T)*是类型表达式

类型表达式 (Type expression)

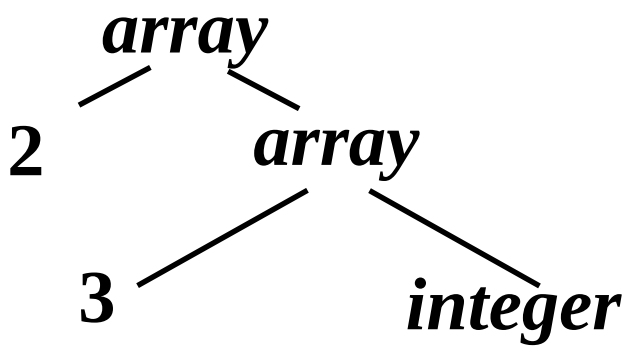
- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
    - 如果T是类型表达式，N是一个整数，则*array(N, T)*是类型表达式

类型	类型表达式
int[3]	*array* (3, *integer*)
int[2][3]	*array* (2, *array* (3, *integer*))

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
    - 如果T是类型表达式，N是一个整数，则*array(N, T)*是类型表达式

类型	类型表达式
int[3]	*array* (3, *integer*)
int[2][3]	*array* (2, *array* (3, *integer*))

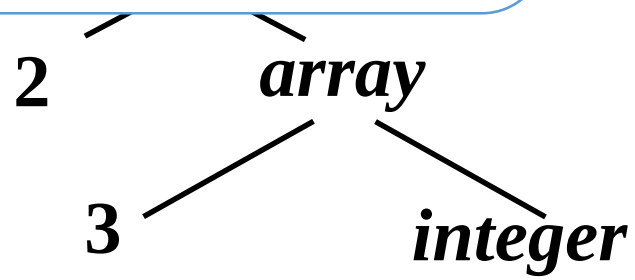


类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
    - 如果T是类型表达式，N是一个

类型	类型表达式
int[3]	*array* (3, *integer*)
int[2][3]	*array* (2, *array* (3, *integer*))

也可以写为
array ({0,…,2}, integer)
其中{0,…,2}代表索引集合
如首元素索引从1开始，则为*array ({1,…,3}, integer)*



类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
  - 指针类型构造算子*pointer*
    - 如果T是类型表达式，则*pointer(T)*是类型表达式

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
  - 指针类型构造算子*pointer*
  - 笛卡尔乘积类型构造算子×
    - 如果T_1和T_2是类型表达式，则$T_1 \times T_2$也是类型表达式
    - 主要用于描述列表和元组，如：表示函数的参数

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
  - 指针类型构造算子*pointer*
  - 笛卡尔乘积类型构造算子×
  - 函数类型构造算子→
    - 若$T_1, T_2, \ldots, T_n$和R是类型表达式，则$T_1 \times T_2 \times \ldots \times T_n \rightarrow R$也是

函数参数　函数返回值

类型表达式 (Type expression)

- 基本类型是类型表达式
- 可为类型表达式命名，类名也是类型表达式
- 将类型构造算子(type constructor)作用于类型表达式可以构成新的类型表达式
  - 数组类型构造算子*array*
  - 指针类型构造算子*pointer*
  - 笛卡尔乘积类型构造算子×
  - 函数类型构造算子→
  - 记录类型构造算子*record*
    - 若有标识符$N_1, N_2, \ldots, N_n$以及对应的类型表达式$T_1, T_2, \ldots, T_n$，则 *record*$((N_1 \times T_1) \times (N_2 \times T_2) \times \ldots \times (N_n \times T_n))$也是类型表达式

记录中的字段

字段对应的类型表达式

类型表达式-例1

- 考虑C语言中数组double a[10][20]，写出a、a[0]、a[0][0]的类型表达式

类型表达式-例1

- 考虑C语言中数组double a[10][20]，写出a、a[0]、a[0][0]的类型表达式

a[0][0]: double

类型表达式-例1

- 考虑C语言中数组double a[10][20]，写出a、a[0]、a[0][0]的类型表达式

a[0][0]: double

a[0]: array(20,double);
pointer(double)

类型表达式-例1

- 考虑C语言中数组double a[10][20]，写出a、a[0]、a[0][0]的类型表达式

a[0][0]: double

a[0]: array(20,double);
pointer(double)

a: array(10,array(20,double));
pointer(array(20,double))

类型表达式-例2

- 为row、table和p分别写出类型表达式：

```
typedef struct{
    int address;
    char lexeme[15];
} row;
row table[101];
row *p;
```

类型表达式-例2

- 为row、table和p分别写出类型表达式：

```
typedef struct{
    int address;
    char lexeme[15];
} row;
row table[101];
row *p;
```

row的类型表达式：

record((address×integer) × (lexeme × (array(15, char))))

table的类型表达式：

array(101, row) //此处row是类型名，因此也是类型表达式

p的类型表达式：

pointer(row)

类型表达式-例3

- 考虑下面的函数f，写出其类型表达式。

```
int *f(char a, char b);
```

f的类型表达式：

(char×char) → pointer(integer)

类型表达式-例3

- 考虑下面的函数f，写出其类型表达式。

```
int *f(char a, char b);
```

f的类型表达式：

(char×char) → pointer(integer)

问题：可否自动化地实现类型表达式的生成?

构造类型表达式的SDD

- 为以下文法制定构造类型表达式的语义规则

产　生　式	语 义 规 则
$T \rightarrow B\ C$	
$B \rightarrow \mathbf{int}$	
$B \rightarrow \mathbf{float}$	
$C \rightarrow \mathbf{[num]}\ C_1$	
$C \rightarrow \varepsilon$	

构造类型表达式的SDD

- 为以下文法制定构造类型表达式的语义规则

产 生 式	语 义 规 则
$T \rightarrow B\ C$	
$B \rightarrow$ int	
$B \rightarrow$ float	
$C \rightarrow$ [num] C_1	
$C \rightarrow \varepsilon$	

- 为每个文法符号设置综合属性t和继承属性b
  - t：该符号对应的类型表达式
  - b：将类型信息从左到右传递

构造类型表达式的SDD

- 为以下文法制定构造类型表达式的语义规则

产 生 式	语 义 规 则
$T \rightarrow B\ C$	$T.t = C.t;\ C.b = B.t$
$B \rightarrow$ int	$B.t =$ *integer*
$B \rightarrow$ float	$B.t =$ *float*
$C \rightarrow$ [num] C_1	$C.t =$ *array*$(num.val, C_1.t);\ C_1.b = C.b$
$C \rightarrow \varepsilon$	$C.t = C.b$

- 为每个文法符号设置**综合属性**t和**继承属性**b
  - t：该符号对应的类型表达式
  - b：将类型信息从左到右传递

构造类型表达式的SDT

- 将SDD改造为SDT

$T \rightarrow B$ {$C.b = B.t$} C {$T.t = C.t$; }

$B \rightarrow$ int {$B.t = integer$}

$B \rightarrow$ float {$B.t = float$}

$C \rightarrow$ [num] {$C_1.b = C.b$} C_1 {$C.t = array(num.val, C_1.t)$; }

$C \rightarrow \varepsilon$ {$C.t = C.b$}

- 但是继承属性的计算与LR分析方法不适配
- 因此，如果要使用LR，就需要改造文法

构造类型表达式的SDT

❑通过改造文法，与LR适配

- ❖引入标记M，C归约时可在栈顶以下位置找到B.t
- ❖引入标记N，把继承属性C.b当做综合属性记录

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow \text{int}\ \{B.t = integer\}$

$B \rightarrow \text{float}\ \{B.t = float\}$

$C \rightarrow [\text{num}]\ N\ C_1\ \{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow \text{int}\ \{B.t = integer\}$

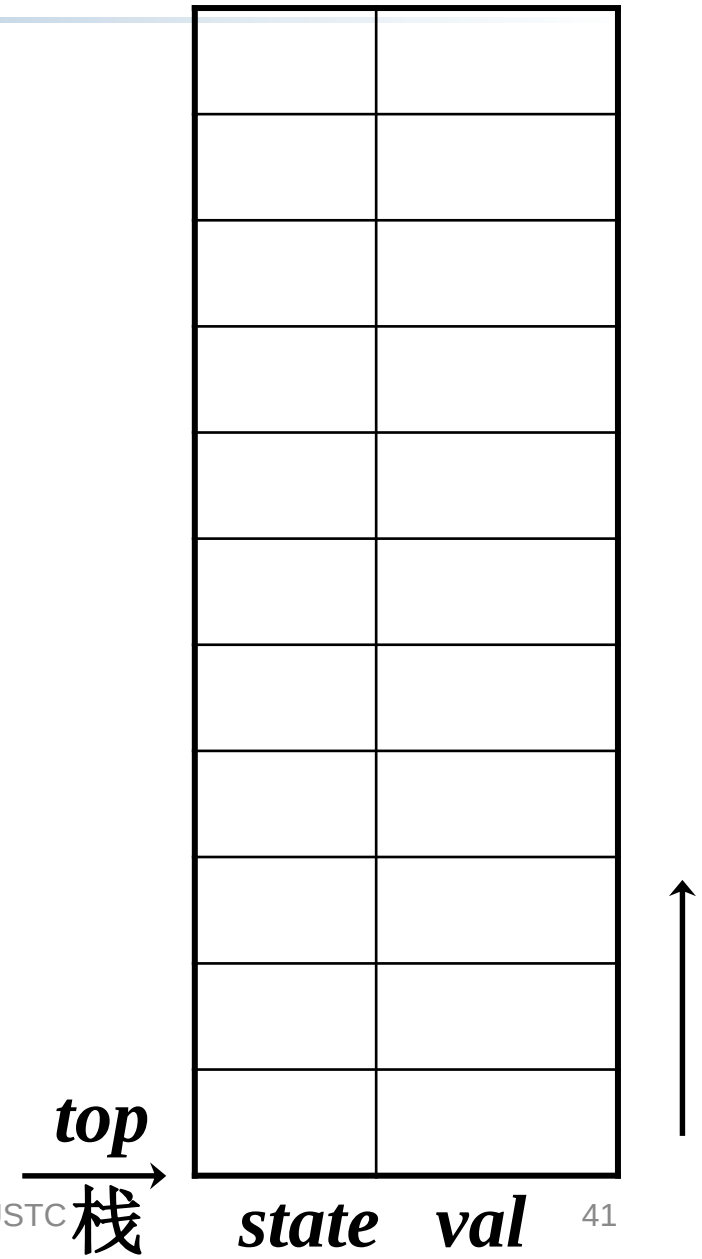
$B \rightarrow \text{float}\ \{B.t = float\}$

$C \rightarrow [\text{num}]\ N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow \text{int}\ \{B.t = integer\}$

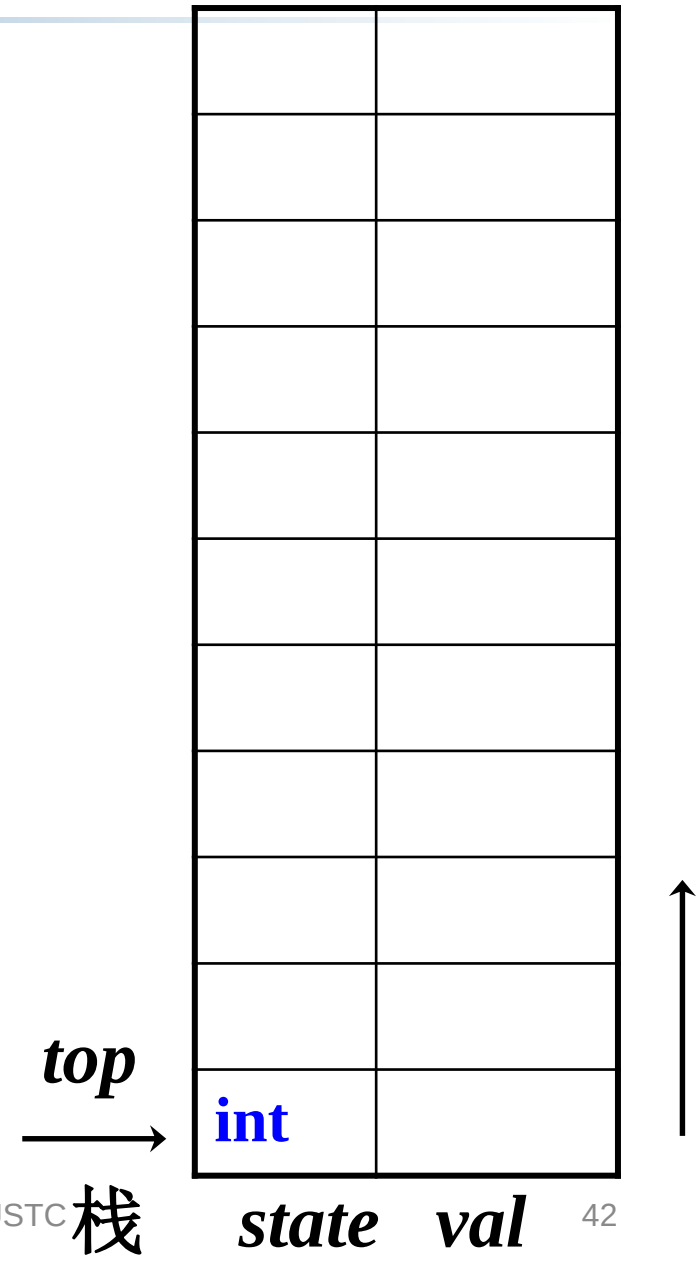
$B \rightarrow \text{float}\ \{B.t = float\}$

$C \rightarrow [\text{num}]\ N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow \text{int}\ \{B.t = integer\}$

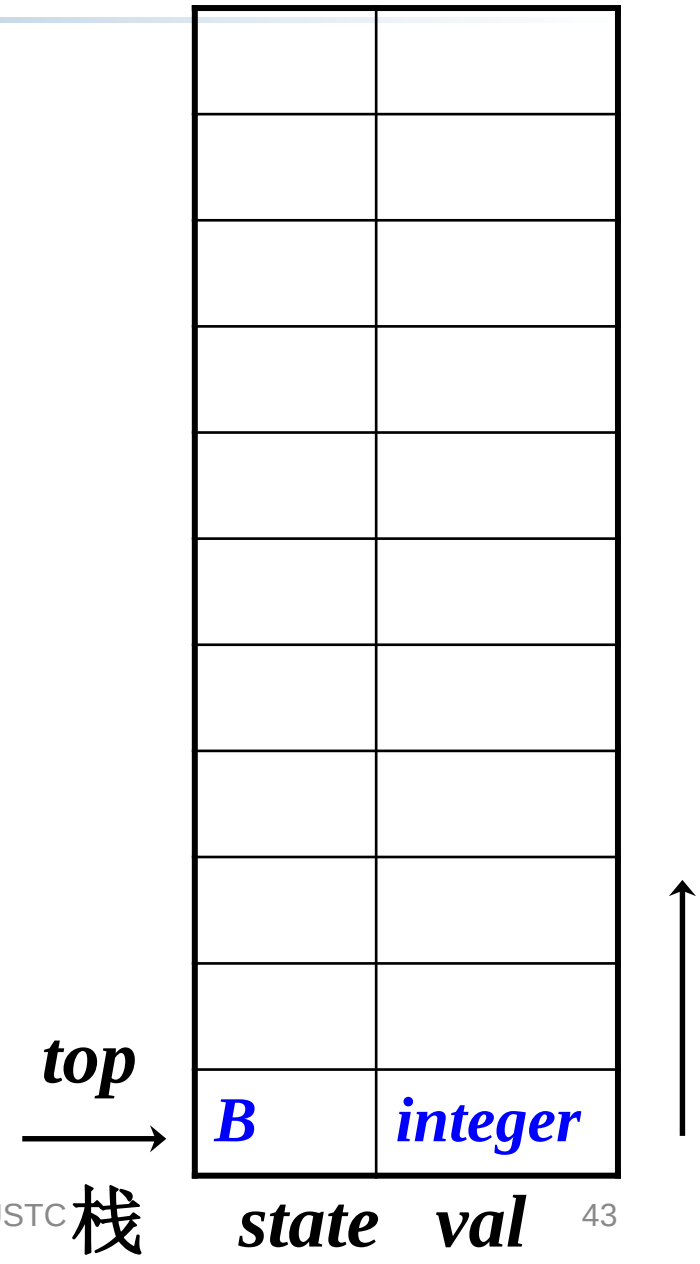
$B \rightarrow \text{float}\ \{B.t = float\}$

$C \rightarrow [\text{num}]\ N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

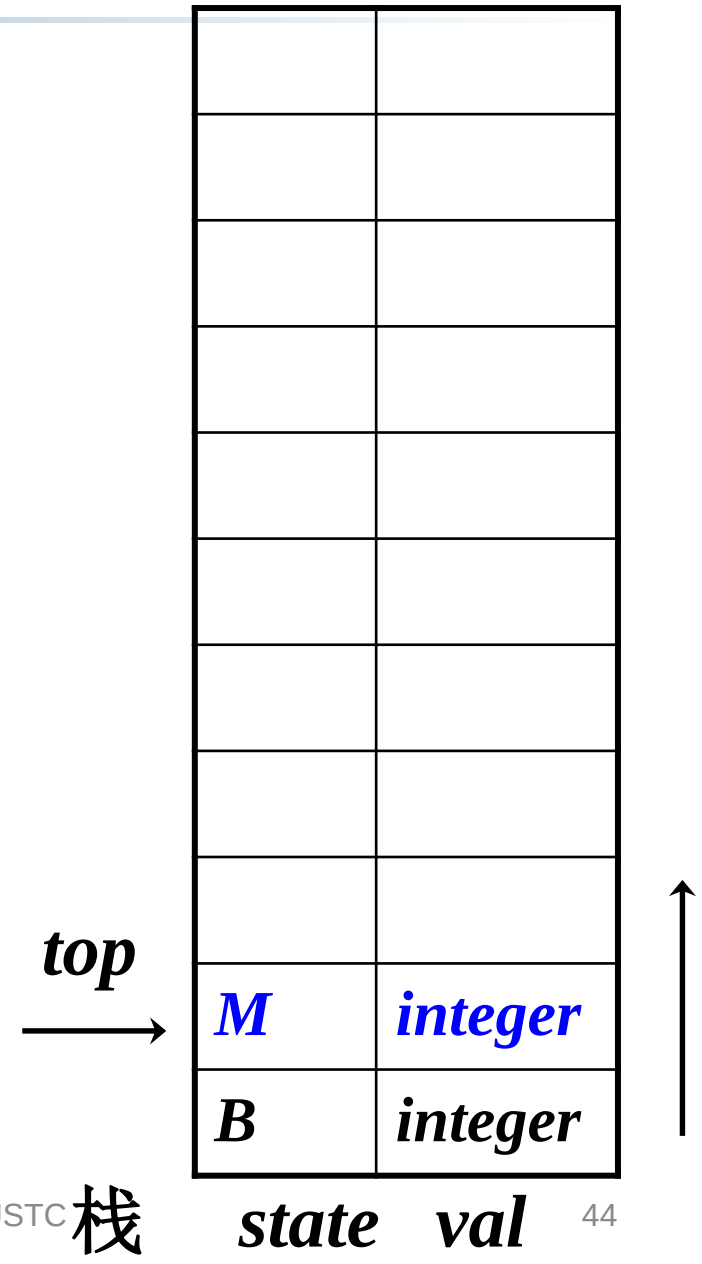
$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

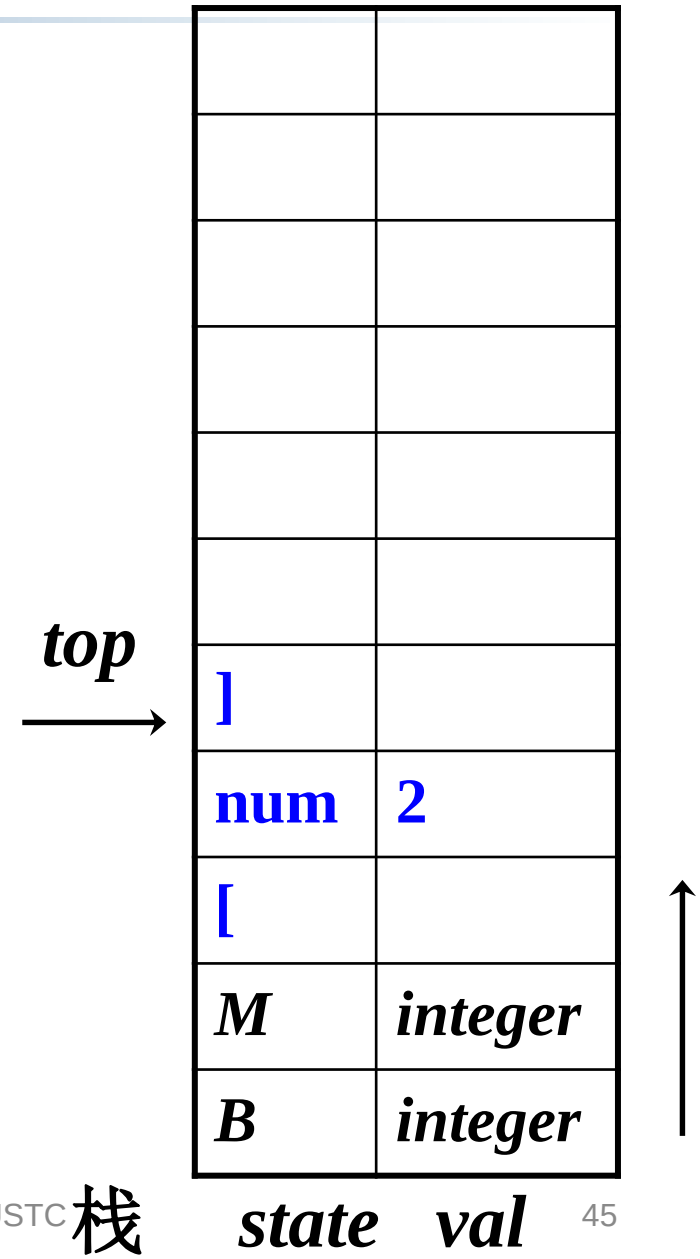
$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

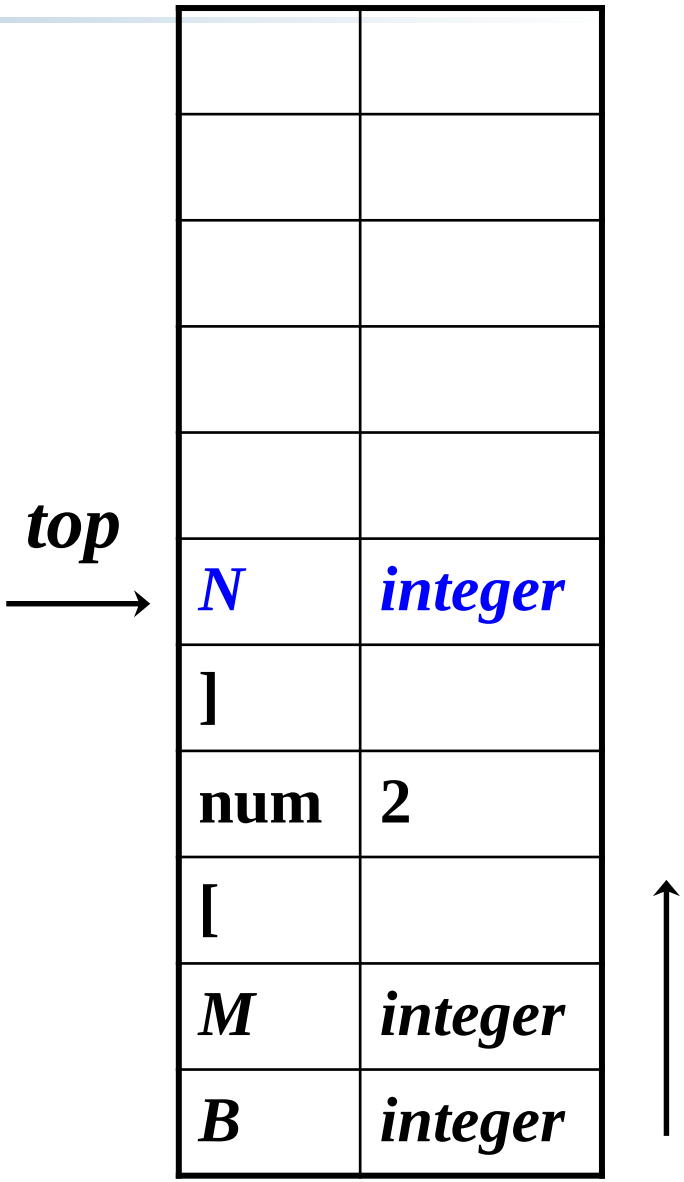
$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$

top ⟶

]	
num	3
[	
N	*integer*
]	
num	2
[	
M	*integer*
B	*integer*

栈 *state* *val*

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

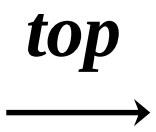
$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



N	*integer*
]	
num	3
[	
N	*integer*
]	
num	2
[	
M	*integer*
B	*integer*

栈 *state* *val*

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C\ \{T.t = C.t;\ \}$

$M \rightarrow \varepsilon\ \{M.t = B.t\}$

$B \rightarrow$ int $\{B.t = integer\}$

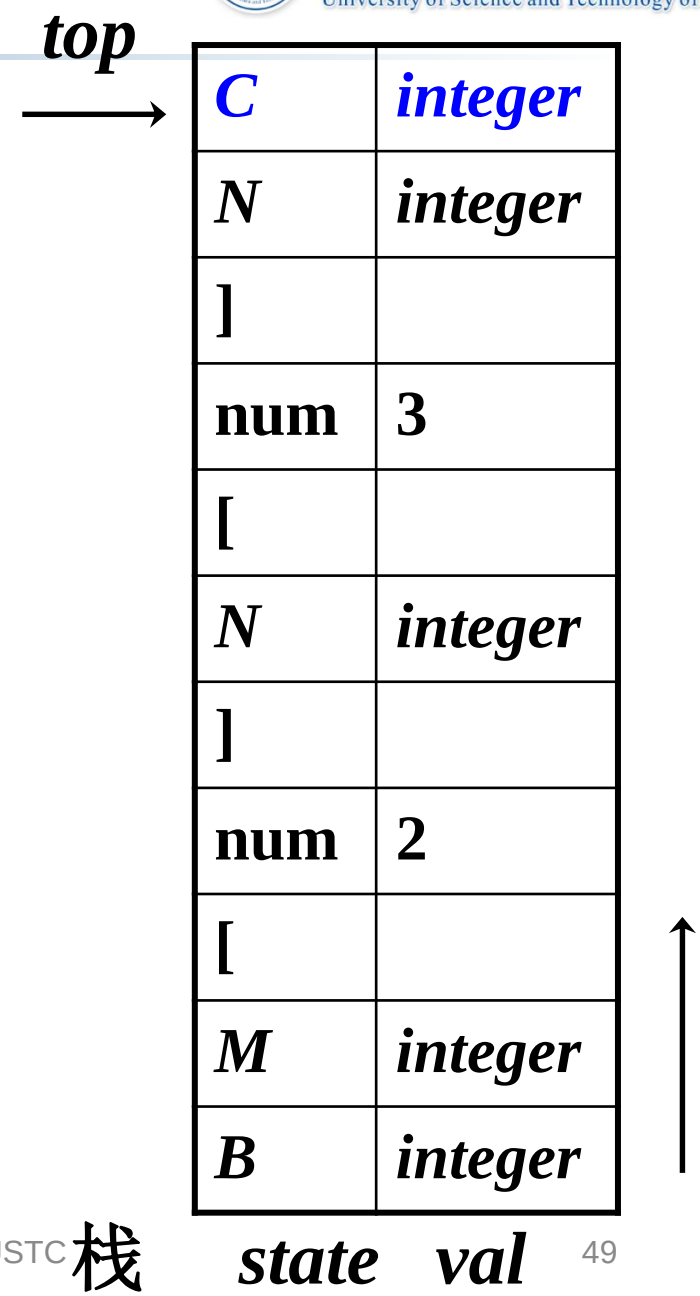
$B \rightarrow$ float $\{B.t = float\}$

$C \rightarrow$ [num] $N\ C_1$

$\{C.t = array(num.val, C_1.t);\ \}$

$N \rightarrow \varepsilon\ \{N.t = C.b\}$

$C \rightarrow \varepsilon\ \{C.t = C.b\}$



构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C$ {$T.t = C.t$; }

$M \rightarrow \varepsilon$ {$M.t = B.t$}

$B \rightarrow$ int {$B.t = integer$}

$B \rightarrow$ float {$B.t = float$}

$C \rightarrow$ [num] $N\ C_1$

{$C.t = array(num.val, C_1.t)$; }

$N \rightarrow \varepsilon$ {$N.t = C.b$}

$C \rightarrow \varepsilon$ {$C.t = C.b$}

top →

C	*array(3, integer)*
N	*integer*
]	
num	2
[	
M	*integer*
B	*integer*

栈 ↑

state *val*

完成第一次归约
$C \rightarrow$ [num] $N\ C_1$

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C$ {$T.t = C.t$; }

$M \rightarrow \varepsilon$ {$M.t = B.t$}

$B \rightarrow$ int {$B.t = integer$}

$B \rightarrow$ float {$B.t = float$}

$C \rightarrow$ [num] $N\ C_1$

{$C.t = array(num.val, C_1.t)$; }

$N \rightarrow \varepsilon$ {$N.t = C.b$}

$C \rightarrow \varepsilon$ {$C.t = C.b$}

top →

栈	state	val
C	array(2, array(3,integer))	
M	integer	
B	integer	

完成第二次归约
$C \rightarrow$ [num] $N\ C_1$

构造类型表达式的SDT

❑分析int[2][3]的LR栈操作

$T \rightarrow B\ M\ C$ {$T.t = C.t$; }

$M \rightarrow \varepsilon$ {$M.t = B.t$}

$B \rightarrow$ int {$B.t = integer$}

$B \rightarrow$ float {$B.t = float$}

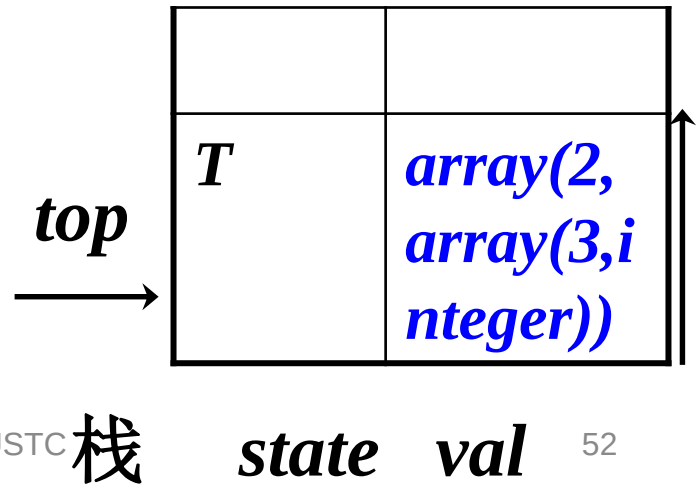
$C \rightarrow$ [num] $N\ C_1$

{$C.t = array(num.val, C_1.t)$; }

$N \rightarrow \varepsilon$ {$N.t = C.b$}

$C \rightarrow \varepsilon$ {$C.t = C.b$}

完成第三次归约
$T \rightarrow B\ M\ C$



声明语句的翻译

- 例：文法G_1如下：

$P \rightarrow D ; S$

$D \rightarrow D ; D$

$D \rightarrow id : T$

$T \rightarrow integer \mid real \mid array [num] of T_1 \mid \uparrow T_1$

声明语句的翻译

- 有关符号的属性

T.type - 变量所具有的类型，如

整型 INT

实型 REAL

数组类型 array（元素个数，元素类型）

指针类型 pointer（所指对象类型）

T.width - 该类型数据所占的字节数

offset - 变量的存储偏移地址

声明语句的翻译

T.type		T.width
整型	INT	4
实型	REAL	8
数组	array（num, T_1）	num.val * T_1.width
指针	pointer（T_1）	4
enter(name, type, offset)—将类型type和偏移offset填入符号表中name所在的表项。		

声明语句的翻译

计算被声明名字的类型和相对地址

P→ {*offset = 0*} D ; S

相对地址初始化为0

D→ D ; D

D→ id : T {*enter(* id.*lexeme, T.type, offset) ;*

offset = offset + T.width }

更新符号表信息

T→integer {*T.type = integer; T.width = 4*}

T→real {*T.type = real; T.width = 8*}

类型=>字宽

T→array [number] of T_1

{*T.type = array(num.val,* T_1*.type);*

*T.width = num.val * *T_1*.width*}

T→ ↑ T_1 {*T.type = pointer(*T_1*.type); T.width = 4*}

声明语句翻译的要点

- 分配存储单元
  - 名字、类型、宽、偏移
- 作用域的管理
  - 过程调用
- 记录类型的管理
- 不产生中间代码指令，但是要更新符号表

允许自定义过程时的翻译

- 所讨论语言的文法

$P \rightarrow D; S$

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

proc id ; D ; S

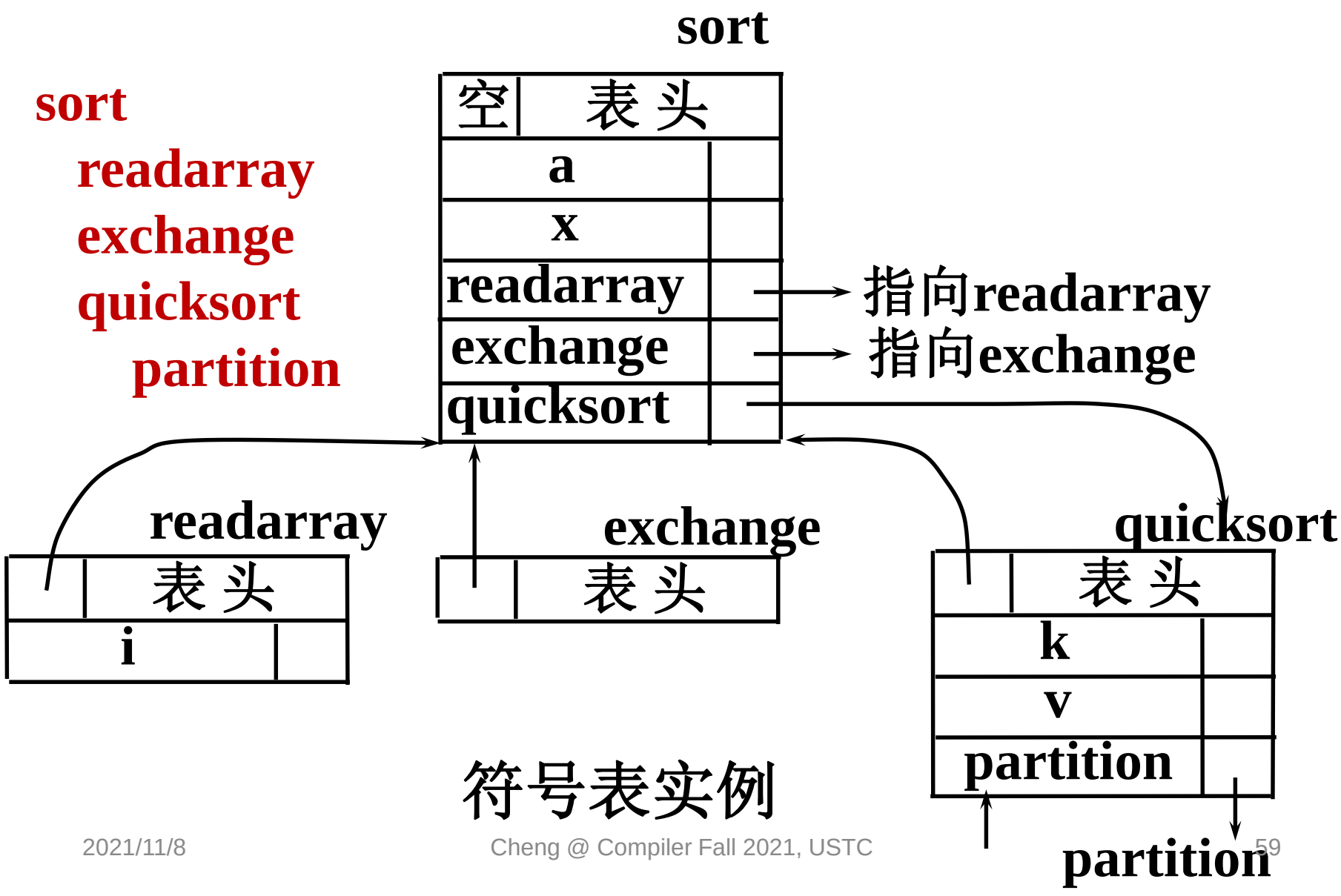
- 管理作用域(过程嵌套声明)
  - 每个过程内声明的符号要置于该过程的符号表中
  - 方便地找到子过程和父过程对应的符号

```
sort
    var a:…; x:…;
    readarray
        var i:…;
    exchange
    quicksort
        var k, v:…;
        partition
            var i, j:…;
```

教科书186页图6.14

过程参数被略去

各过程的符号表



符号表实例

❑符号表的特点及数据结构

- ❖各过程有各自的符号表：哈希表
- ❖符号表之间有双向链
  - ➢父→子：过程中包含哪些子过程定义
  - ➢子→父：分析完子过程后继续分析父过程
- ❖维护符号表栈(*tblptr*)和地址偏移量栈(*offset*)
  - ➢保存尚未完成的过程的符号表指针和相对地址

❑语义动作用到的函数

/* 建立新的符号表，其表头指针指向父过程符号表*/

1. mkTable(parent-table)

/* 将所声明变量的类型、偏移填入当前符号表*/

2. enter(current-table, name, type, current-offset)

/* 在父过程符号表中建立子过程名的条目*/

3. enterProc(parent-table, sub-proc-name, sub-table)

/*在符号表首部添加变量累加宽度，可利用符号表栈tblptr和偏移栈offset（栈顶值分别表示当前分析的过程的符号表及可用变量偏移位置）*/

4. addWidth(table, width)

声明语句的处理

$P \rightarrow M\ D;\ S$

$M \rightarrow \varepsilon$

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow \text{proc id}\ ;\ N\ D_1;\ S$

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

$N \rightarrow \varepsilon$

声明语句的处理

$P \rightarrow M\ D;\ S$

tblptr：符号表栈
offset：偏移量栈

$M \rightarrow \varepsilon$ {*t = mkTable (nil);*
push(t, tblptr); push (0, offset) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$

$D \rightarrow$ id : T

$N \rightarrow \varepsilon$

建立主程序（最外围）的符号表偏移从0开始

声明语句的处理

$P \rightarrow M\ D;\ S$

$M \rightarrow \varepsilon$ {*t* = *mkTable* (*nil*);
push(*t*, *tblptr*); *push* (0, *offset*) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$

$D \rightarrow$ id : T {*enter*(*top*(*tblptr*), id.*lexeme*, *T.type*, *top*(*offset*));
top(*offset*) = *top*(*offset*) + *T.width* }

$N \rightarrow \varepsilon$

将变量name的有关属性填入当前符号表

声明语句的处理

$P \rightarrow M\ D;\ S$

$M \rightarrow \varepsilon$ {*t* = *mkTable* (*nil*);
push(*t*, *tblptr*); *push* (0, *offset*) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$

$D \rightarrow$ id :T {*enter*(*top*(*tblptr*), id.*lexeme*, *T.type*, *top*(*offset*));
top(*offset*) = *top*(*offset*) + *T.width* }

$N \rightarrow \varepsilon$ {*t* = *mkTable*(*top*(*tblptr*));
push(*t*, *tblptr*); *push*(0, *offset*) }

建立子过程的符号表和偏移从0开始

声明语句的处理

$P \rightarrow M\ D;\ S$

$M \rightarrow \varepsilon$ {*t* = *mkTable* (*nil*);
push(*t*, *tblptr*); *push* (0, *offset*) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$ {*t* = *top*(*tblptr*);
addWidth(*t*, *top*(*offset*)); *pop*(*tblptr*); *pop*(*offset*);
enterProc(*top*(*tblptr*), id.*lexeme*, *t*) }

$D \rightarrow$ id :T {*enter*(*top*(*tblptr*), id.*lexeme*, *T*.*type*, *top*(*offset*));
top(*offset*) = *top*(*offset*) + *T*.*width* }

$N \rightarrow \varepsilon$ {*t* = *mkTable*(*top*(*tblptr*));
push(*t*, *tblptr*); *push*(0, *offset*) }

保留当前过程声明的总空间；弹出符号表和偏移栈顶（露出父过程的符号表和偏移；在父过程符号表中填写子过程名有关条目

声明语句的处理

$P \rightarrow M\ D;\ S$ {*addWidth (top (tblptr), top (offset));*
pop(tblptr); pop (offset) }

$M \rightarrow \varepsilon$ {*t = mkTable (nil);*
push(t, tblptr); push (0, offset) }

$D \rightarrow D_1\ ;\ D_2$

$D \rightarrow$ proc id ; $N\ D_1;\ S$ {*t = top(tblptr);*
addWidth(t, top(offset)); pop(tblptr); pop(offset);
enterProc(top(tblptr), id.*lexeme, t)* }

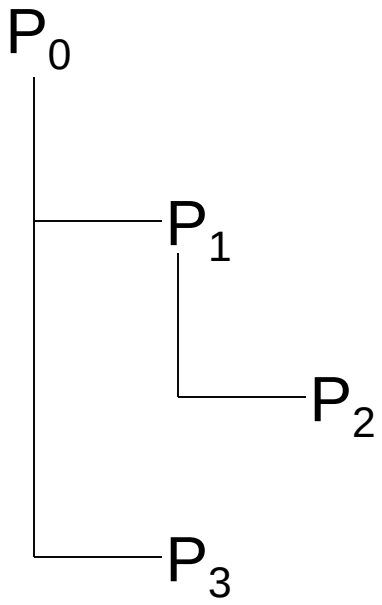
$D \rightarrow$ id :T {*enter(top(tblptr),* id.*lexeme, T.type, top(offset));*
top(offset) = top(offset) + T.width }

$N \rightarrow \varepsilon$ {*t = mkTable(top(tblptr));*
push(t, tblptr); push(0, offset) }

修改变量分配空间大小并清空符号表和偏移栈

举例：过程嵌套声明


```
i : int;  j : int ;
PROC P1 ;
   k : int;  f : real ;
   PROC P2;
          l : int ;
         a1 ;
   a2;
PROC P3;
   temp : int ;  max : int ;
   a3;
```



过程声明层次图

举例：过程嵌套声明

❑初始：M→ε

null	总偏移：

P_0

top → | P_0 | 0 |

符号栈　偏移栈

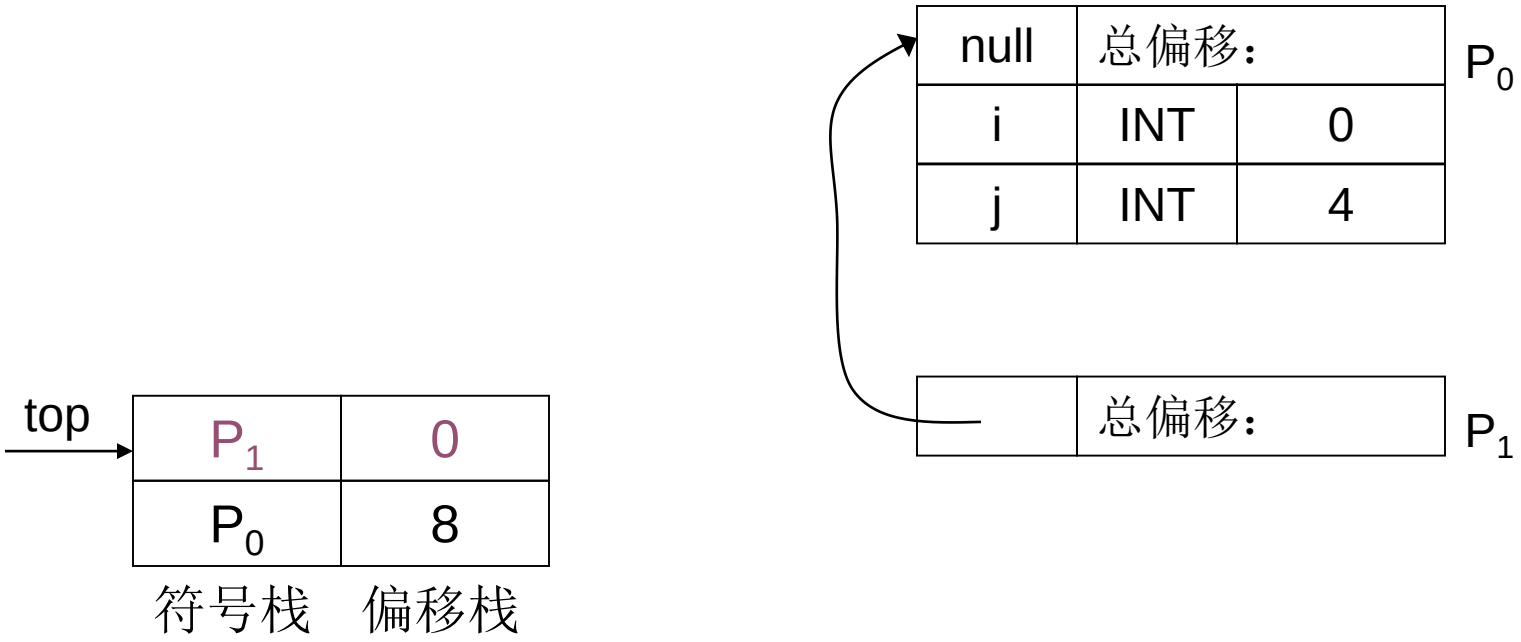
举例：过程嵌套声明

- i : int ; j : int ;

null	总偏移：		P_0
i	INT	0	
j	INT	4	

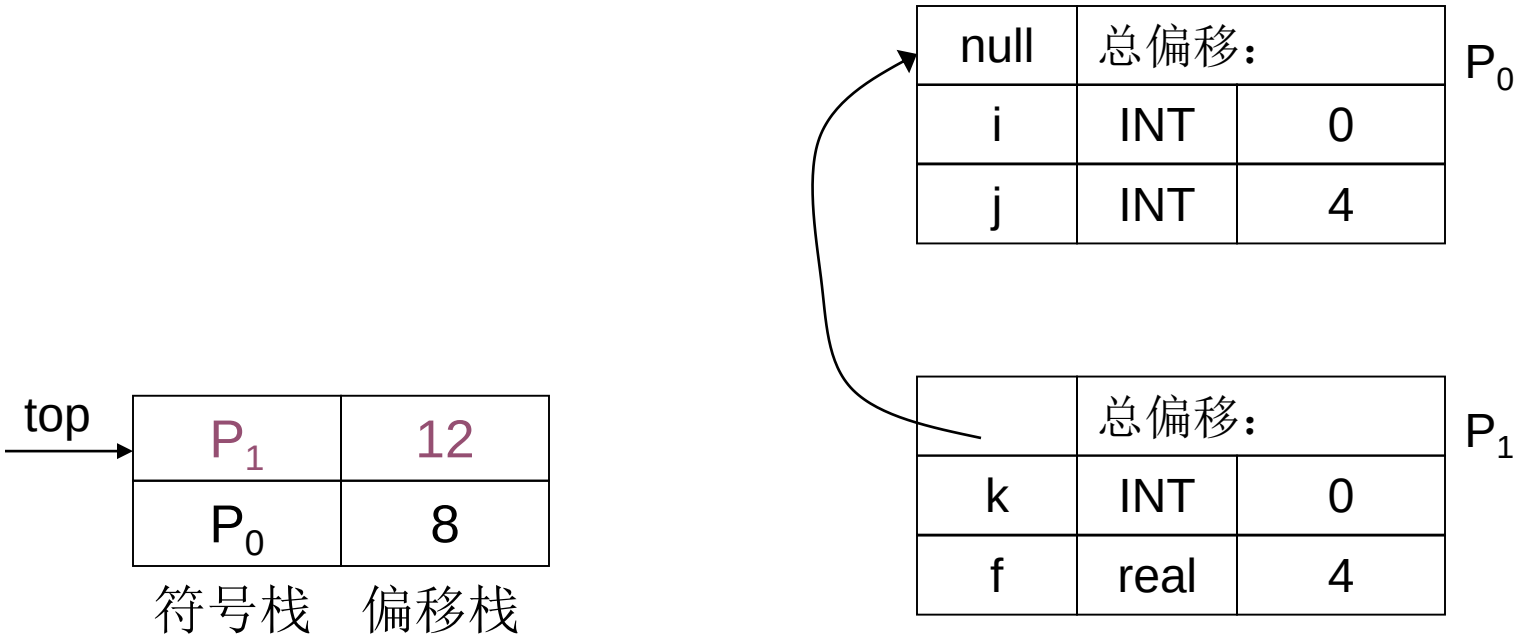
举例：过程嵌套声明

- PROC P_1；（N→ε）



举例：过程嵌套声明

- k : int ; f : real ;



举例：过程嵌套声明

- PROC P_2；（N→ε）

top

P_2	0
P_1	12
P_0	8

符号栈　偏移栈

null	总偏移：		P_0
i	INT	0	
j	INT	4	

	总偏移：		P_1
k	INT	0	
f	real	4	

	总偏移：	P_2

举例：过程嵌套声明

- l : int ;

top

P_2	4
P_1	12
P_0	8

符号栈　偏移栈

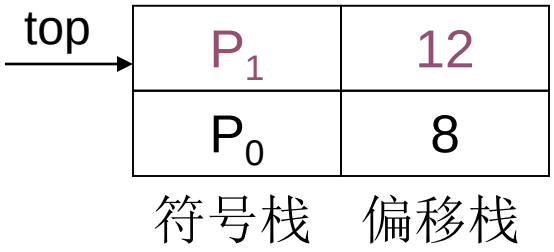
null	总偏移：		P_0
i	INT	0	
j	INT	4	

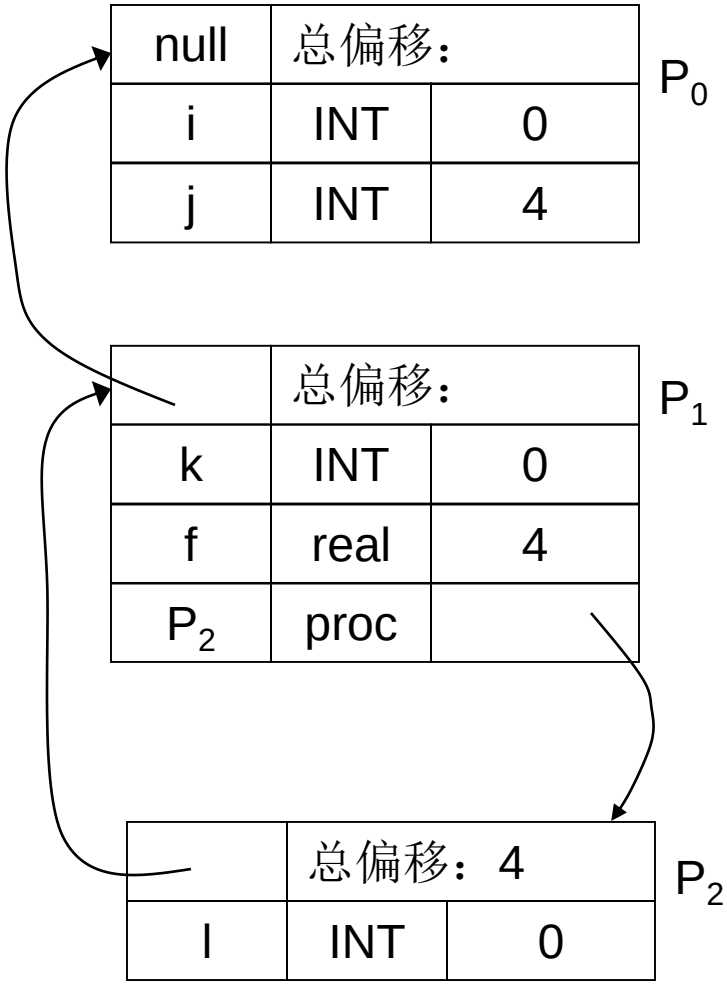
	总偏移：		P_1
k	INT	0	
f	real	4	

	总偏移：		P_2
l	INT	0	

举例：过程嵌套声明

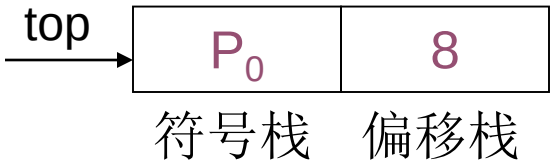
- a_1；

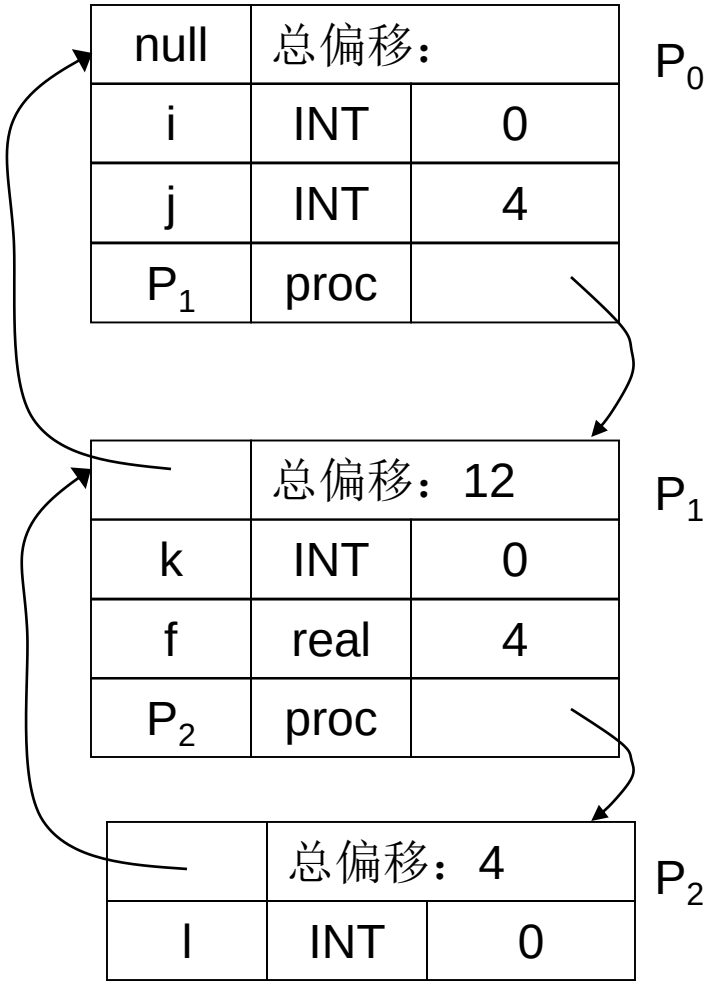

top
P1
12
P0
8
符号栈
偏移栈


null
总偏移：
i
INT
0
j
INT
4
P0
总偏移：
k
INT
0
f
real
4
P2
proc
P1
总偏移：4
l
INT
0
P2

举例：过程嵌套声明

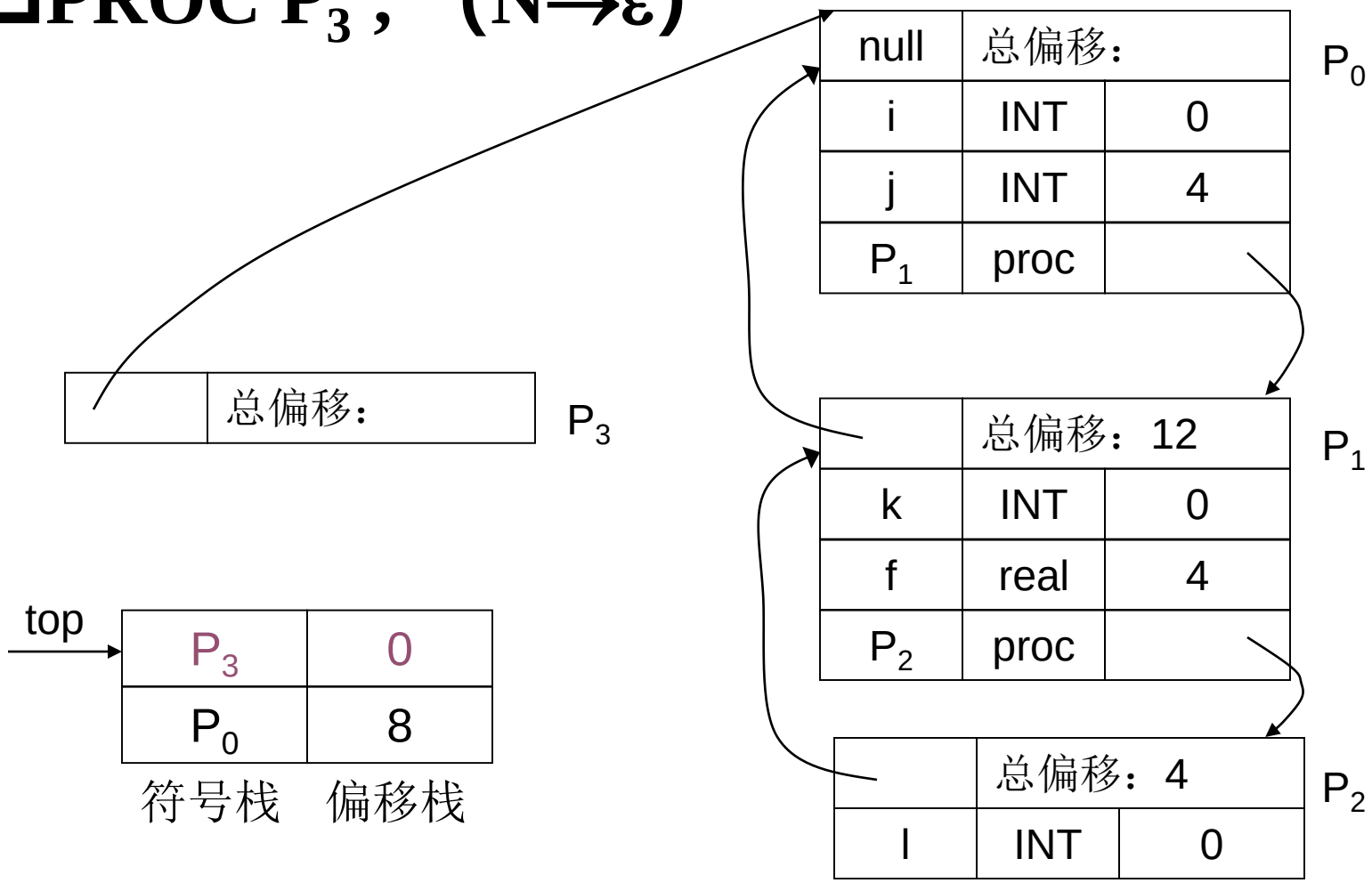
- a_2 ;


top
P0
8
符号栈
偏移栈


null
总偏移：
i
INT
0
j
INT
4
P1
proc
P0
总偏移：12
k
INT
0
f
real
4
P2
proc
P1
总偏移：4
l
INT
0
P2

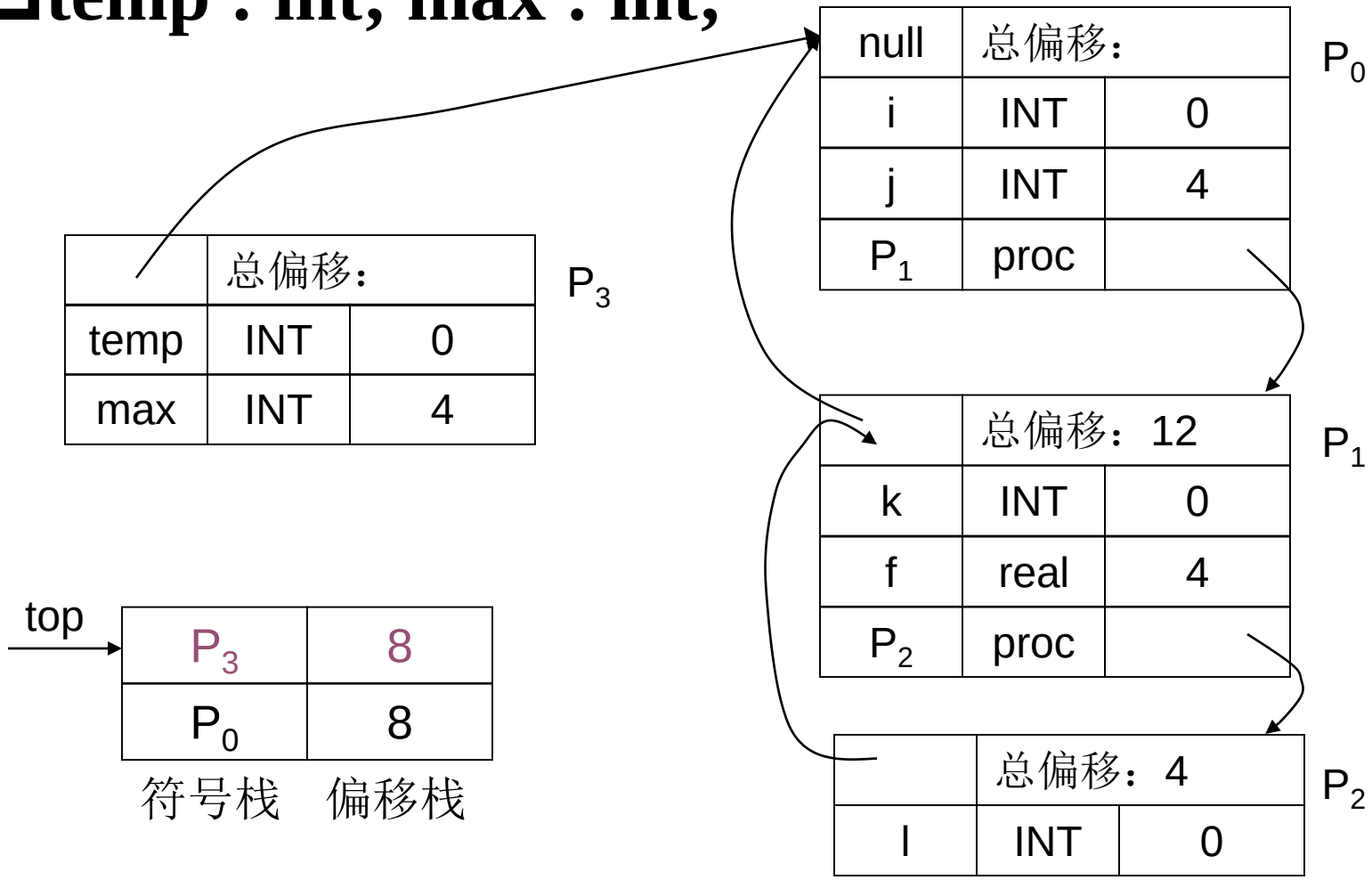
举例：过程嵌套声明

- PROC P_3；（N→ε）



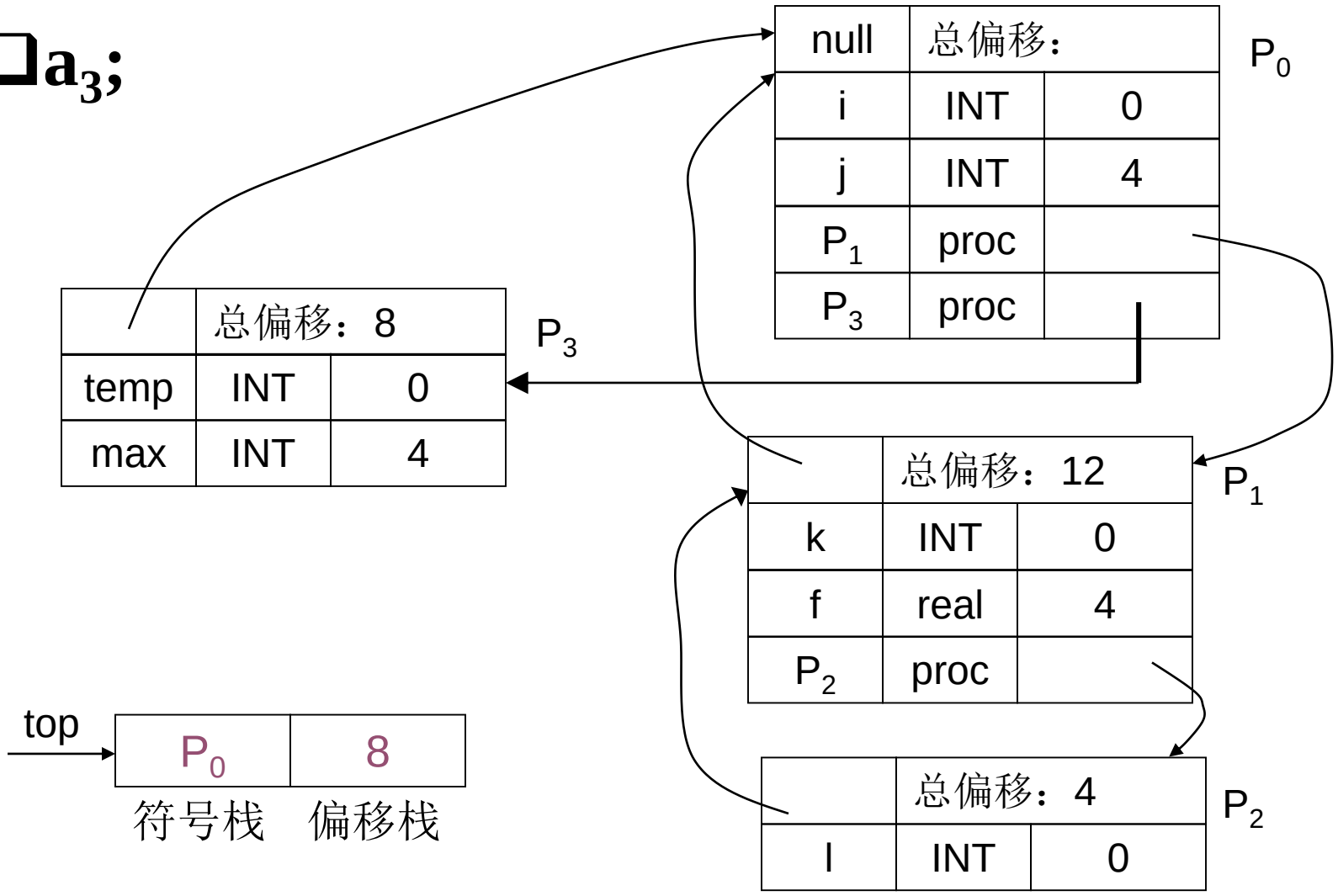
举例：过程嵌套声明

- temp : int; max : int;



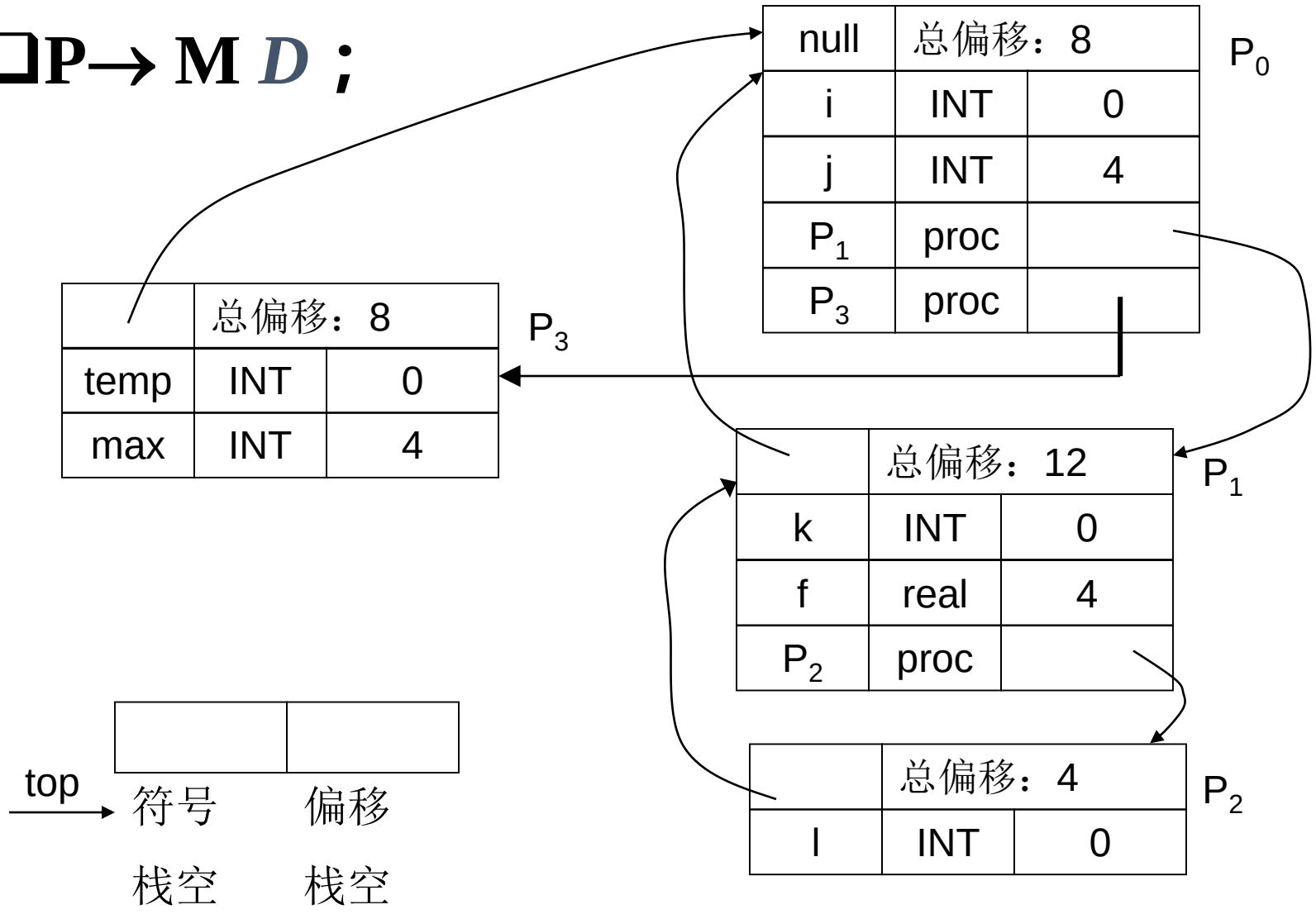
举例：过程嵌套声明

- a_3;


null
总偏移：
i
INT
0
j
INT
4
P1
proc
P3
proc
P0
总偏移：8
temp
INT
0
max
INT
4
P3
总偏移：12
k
INT
0
f
real
4
P2
proc
P1
top
P0
8
符号栈
偏移栈
总偏移：4
l
INT
0
P2

举例：过程嵌套声明

❑P→ M *D* ;



声明语句翻译的要点

- 分配存储单元
  - 名字、类型、字宽、偏移
- 作用域的管理
  - 过程调用
- **记录类型的管理**
- **不产生中间代码指令，但是要更新符号表**

记录的域名管理

❑描述记录的文法

$T \rightarrow$ record D end

记录类型单独建符号表，域的相对地址从0开始

$T \rightarrow$ record L D end

$L \rightarrow \varepsilon$

```
record
    a :…;
    r : record
            i :…;
            . . .
        end;
     k : …;
end
```

记录的域名管理

❑描述记录的文法

$T \rightarrow$ record D end

记录类型单独建符号表，域的相对地址从0开始

$T \rightarrow$ record L D end

```
record
    a :…;
    r : record
            i :…;
            . . .
        end;
    k : …;
end
```

$L \rightarrow \varepsilon$ { *t = mkTable(nil);*

push(t, tblptr); push(0, offset) }

建立符号表，进入作用域

记录的域名管理

❑描述记录的文法

$T \rightarrow$ record D end

记录类型单独建符号表，域的相对地址从0开始

$T \rightarrow$ record L D end

{*T.type = record (top(tblptr));*

T.width = top(offset);

pop(tblptr); pop(offset) }

$L \rightarrow \varepsilon$ { *t = mkTable(nil);*

push(t, tblptr); push(0, offset) }

```
record
    a :…;
    r : record
            i :…;
            . . .
        end;
    k : …;
end
```

设置记录的类型表达式和宽度，退出作用域

记录的域名管理

❑描述记录的文法

$T \rightarrow$ record D end

记录类型单独建符号表，域的相对地址从0开始

$T \rightarrow$ record L D end

{*T.type* = *record* (*top*(*tblptr*));

T.width = *top*(*offset*);

pop(*tblptr*); *pop*(*offset*) }

$L \rightarrow \varepsilon$ { *t* = *mkTable*(*nil*);

push(*t*, *tblptr*); *push*(0, *offset*) }

D的翻译同前

```
record
    a :…;
    r : record
            i :…;
            . . .
        end;
    k : …;
end
```

举例：记录域的偏移

❑有2个C语言的结构定义如下：

```
struct A {
    char c1;
    char c2;
    long l;
    double d;
} S1;
```

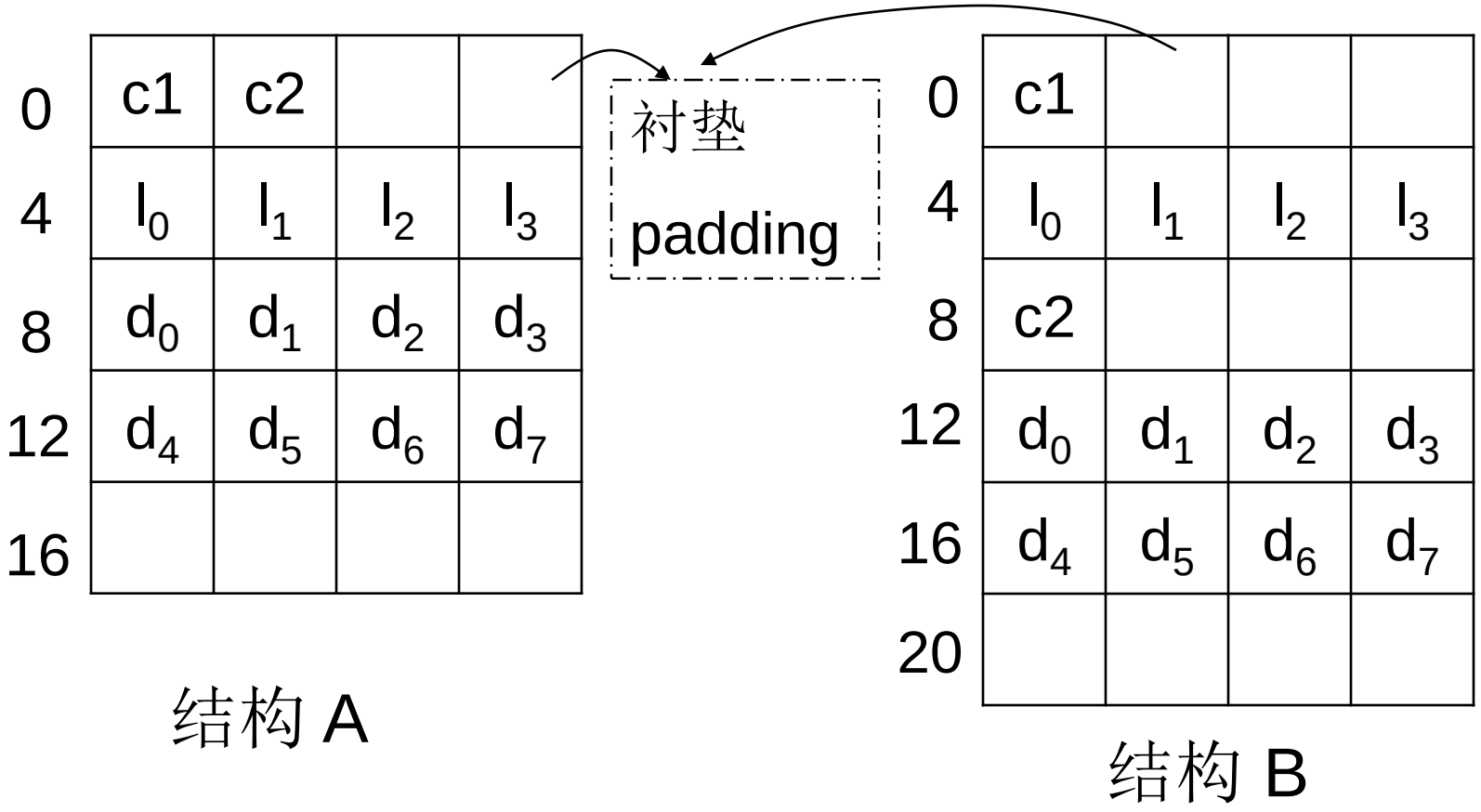
```
struct B {
    char c1;
    long l;
    char c2;
    double d;
} S2;
```

举例：记录域的偏移

- 数据（类型）的对齐 - alignment
- 在 X86-Linux下：
  - char：对齐1，起始地址可分配在任意地址
  - int，long，double：对齐4，即从被4整除的地址开始分配
- 注*：其它类型机器，double可能对齐到8
  - 如sun－SPARC

举例：记录域的偏移

❑结构A 和 B的大小分别为16和20字节(Linux)



举例：记录域的偏移

❑2个结构中域变量的偏移如下：

```
struct A {
  char c1;   0
  char c2;   1
  long l;    4
  double d;  8
} S1;
```

```
struct B {
    char c1;    0
    long l;     4
    char c2;    8
    double d;  12
} S2;
```

中国科学技术大学
University of Science and Technology of China
USTC, CHINA
ADSLAB
先进数据系统实验室

《编译原理与技术》
中间代码生成 II

TBA