



## 第六章 符号表管理技术

- 概述
- 符号表的组织与内容
- 非分程序结构语言的符号表组织
- 分程序结构语言的符号表组织

## 6.1 概述

### (1) 什么是符号表？

在编译过程中，编译程序用来记录源程序中各种名字的特性信息，所以也称为名字特性表。

名字：程序名、过程名、函数名、用户定义类型名、变量名、常量名、枚举值名、标号名等。

特性信息：上述名字的种类、类型、维数、参数个数及目标地址（存储单元地址）等。

## (2) 建表和查表的必要性(符号表在编译过程中的作用)

源程序中变量要先声明，然后才能引用。

用户通过**声明语句**，声明各种名字，并给出它们的类型维数等信息。编译程序在遇到这些声明语句时，应该将声明中的名字以及信息**登录**到符号表中，同时编译程序还要给变量分配存储单元。

存储单元地址也必须登录在符号表中。

当编译程序编译到**引用**所声明的变量时（赋值或引用其值），要进行语法语义正确性检查（类型是否符合要求等）和生成相应的目标程序，这就需要查符号表以取得相关信息。

## 符号表

## 数据区

例: `int x, a, b;`

...

...

...

**L: `x := a + b;`**

...

建表,  
分配存贮

x	简单变量	整型
a	简单变量	整型
b	简单变量	整型
L	标号	


### 1. 语法分析和语义分析

- 说明语句、赋值语句的语法规则;
- 上下文有关分析: 是否声明;
- 类型一致性检查。

### 2. 生成目标代码

**LOAD** a的地址  
**ADD** b的地址  
**STO** x的地址

### (3) 有关符号表的操作：填表和查表

**填表：**当分析到程序中的说明或定义语句时，应将说明或定义的名字，以及与之有关的信息填入符号表中。

例：Procedure P()

**查表：**

- (1) 填表前查表，检查在程序的同一作用域内名字是否重复定义；
- (2) 检查名字的种类是否与说明一致；
- (3) 对于强类型语言，要检查表达式中各变量的类型是否一致；
- (4) 生成目标指令时，要取得所需要的地址。

.....



## 6.2 符号表的组织与内容

### (1) 符号表的结构与内容

符号表的基本结构如下：

名字	特性（信息）

**“名字”域：**存放名字。一般为标识符的符号串，也可  
为指向标识符字符串的指针。



## 名字                      特性（信息）


“特性”域：可包括多个子域，分别表示标识符的有关信息。  
如：

名字（标识符）的种类：简单变量、函数、过程、  
数组、标号、参数等

类型：如整型、浮点型、字符型、指针等

性质：变量形参、值形参等

值：常量名所代表的数值

地址：变量所分配单元的首址或地址位移

大小：所占的字节数

作用域的嵌套层次：

对于数组：维数、上下界值、计算下标量地址所用的信息（数组信息向量）以及数组元素类型等。

对于记录（结构、联合）：域的个数，每个域名、地址位移、类型等。

对于过程或函数：形参个数、所在层次、函数返回值类型、局部变量所占空间大小等。

对于指针：所指对象类型等。



## (2) 符号表的组织方式

1. 统一符号表——不论什么名字都填入统一格式的符号表中

符号表表项应按信息量最大的名字设计。填表、查表比较方便，结构简单，但是浪费大量空间。

2. 对于不同种类的名字分别建立各种符号表

节省空间，但是填表和查表不方便。

- ① 符号名表（用来登记源程序中的常量名、变量名、数组名和过程名等，并记录其属性、引用等）
- ② 常数表（分类型登记各种常量值）
- ③ 标号表（登记标号的定义与应用）
- ④ 分程序入口表（登记过程的层号、分程序符号表的入口等）
- ⑤ 中间代码表

### 3. 折中办法——大部分共同信息组成统一格式的符号表。 特殊信息另设附表，两者用指针连接。

例: begin

real A;

array B[1:100];

:

:

end

A	简变	实型	地址	
B	数组	实型		

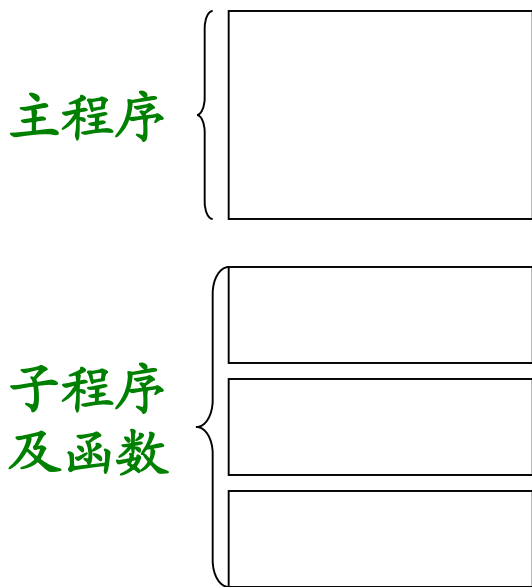
指针连接  
补充

维数	上下界	首地址

## 6.3 非分程序结构语言的符号表组织

(1) 非分程序的结构语言：每个可独立进行编译的程序单元是一个不包含有子模块的单一模块。  
如FORTRAN语言。

### FORTRAN程序构造



主程序和子程序中可定义common语句：

FORTRAN程序中各程序单位之间的数据交换可以通过虚实结合来实现，也可以通过建立公用区的方式来完成。

公用区有两种，一种是无名公用区，任何一个程序中只可能有一个无名公用区；一种是有名公用区，一个程序中可以根据需要由程序员开辟任意多个有名公用区。

无名和有名公用区都通过COMMON语句来进行建立。





## (2) 标识符的作用域及基本处理办法

### 1. 作用域

**全局：**子程序名、函数名和公共区名。

**局部：**程序单元中定义的变量。

### 2. 符号表的组织：

全局符号表
局部符号表



### 3. 基本处理办法:

<1> 子程序、函数名和公共区变量填入全局符号表。

<2> 在子程序(函数)声明部分读到标识符时, 构造局部符号表。

查本程序单元局部符号表, 有无同名

- 有: 重复声明, 报错
- 无: 填表

<3> 在语句部分读到标识符, 查表。

查本程序单元局部符号表, 有无同名

- 有: 已声明过
- 无: 查全局变量表
  - 有: 全局量
  - 无: 无定义标识符



4. 程序单元结束：释放该程序单元的局部符号表。
5. 程序执行完成：释放全部符号表。

### (3) 符号表的组织方式

1. 无序符号表——按扫描顺序建表，查表要逐项查找。

查表操作的平均长度为  $(n + 1) / 2$ 。



## 2. 有序符号表：符号表按变量名进行字典式排序。

线性查表：  $(n + 1) / 2$

折半查表：  $\log_2 n - 1$

## 3. 散列符号表（Hash表）：

符号表地址 = Hash（标识符）

解决：冲突





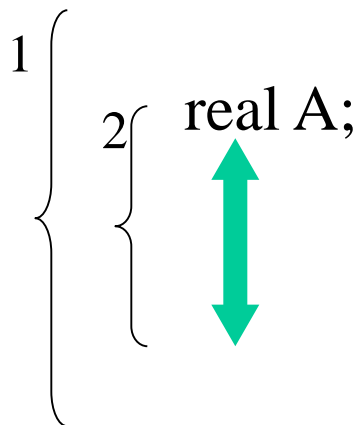
## 6.4 分程序结构语言的符号表组织

(1) 分程序的结构语言：模块内可嵌入子模块。

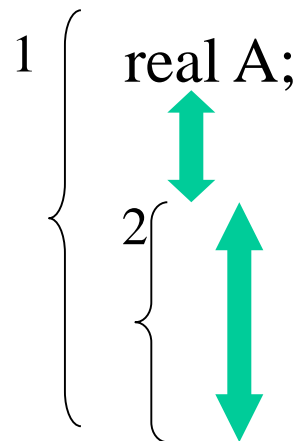
(2) 标识符的作用域和基本处理方法

作用域：标识符局部于所定义的模块（最小模块）

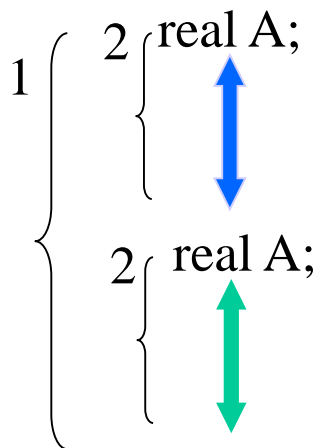
- ① **模块**中所定义标识符的作用域是定义该标识符的子程序。



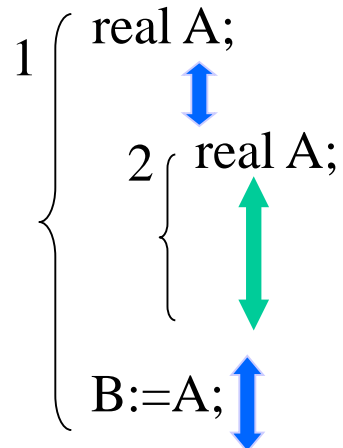
A为内分程序局部变量



A为内分程序全局变量

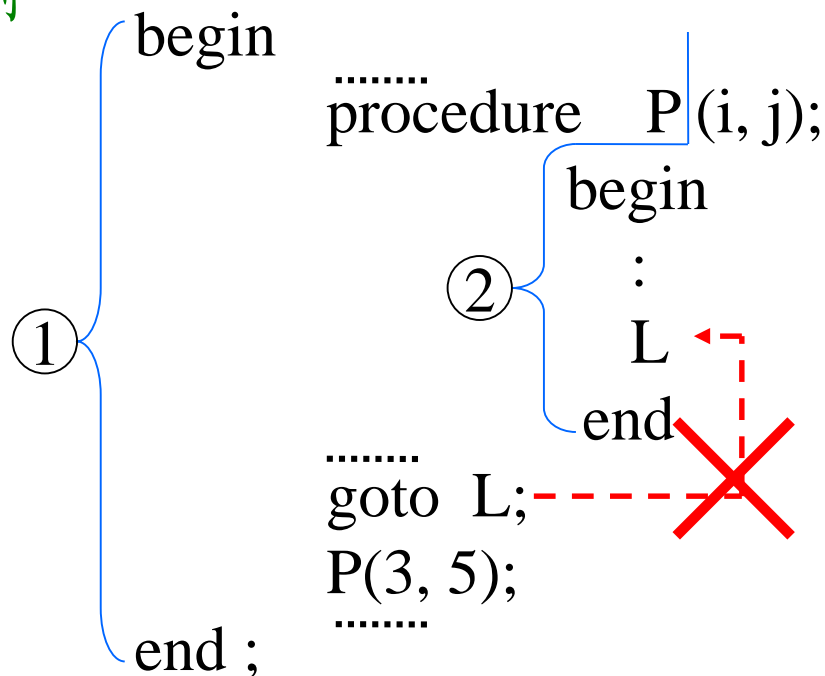


都是局部变量



② **过程或函数**说明中定义的标识符（包括形参）  
其作用域为本过程体。

例



③ 循环语句中定义的标识符，其作用域为该循环语句。

```
for ... .. do
  begin
    :
    L ← - - - - -
  end
goto L; - - - - -
      :
      X
```

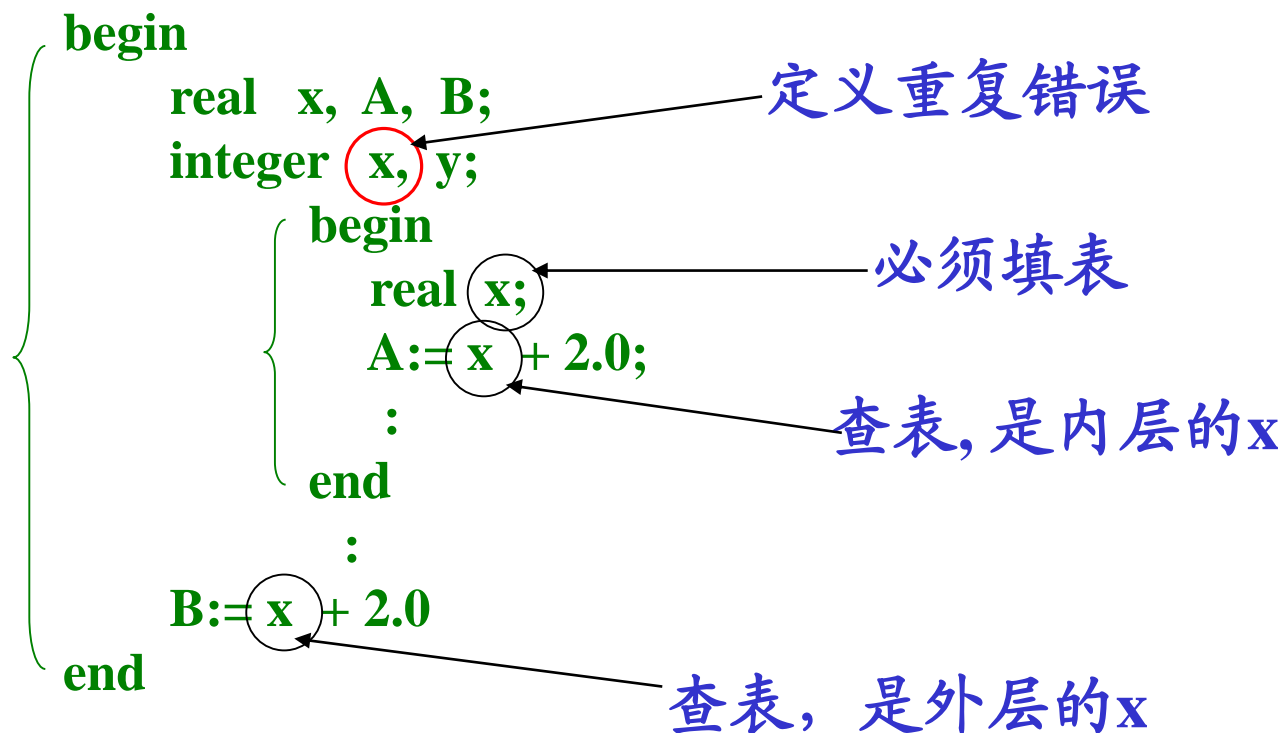
不能从循环体外转到循环体内。循环语句应看作一层！

## 基本处理办法:

建查符号表均要遵循标识符作用域规定进行。

建表: 不能重复, 不能遗漏。

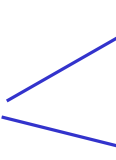
查表: 按标识符作用域查找。



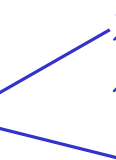
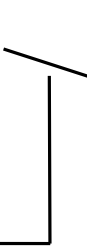
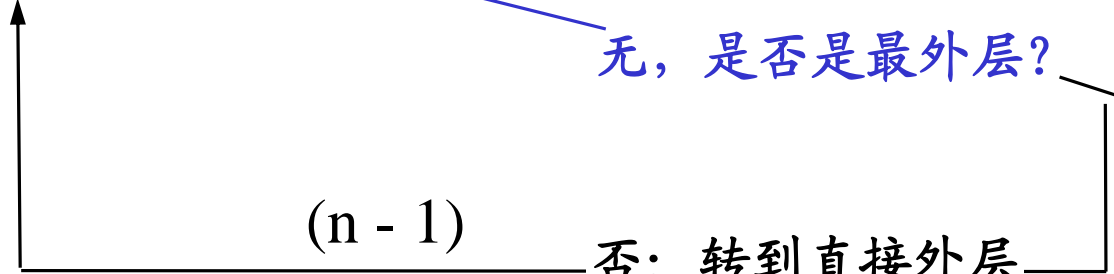
## 处理方法:

假设标识符是先声明后引用（标号例外，要特殊处理）。

a. 在程序声明部分读到标识符时（声明性出现）建表：

查本层符号表，有无同名    
有：重复声明，报错  
无：填入符号表

b. 在语句中读到标识符（引用性出现），查表：

查本层符号表，有无同名    
有：即已声明。则取该名字信息（局部量）。  
无，是否是最外层？    
是：未声明标识符。报错  
否：转到直接外层    
(n - 1)

## c. 标准标识符的处理

主要是语言定义的一些标准过程和函数的名字。它们是标识符的子集。如 `sin` `con` `abs`.... (注意它们不是语言的保留字)

特点：1) 用户不必声明就可全程使用。

2) 设计编译程序时，标准名字及其数目已知。

处理方法：1) 单独建表：使用不便，费时。

2) 预先将标准标识符填入名字表中。因为它们是全程量，所以应填入最外层。

## 分程序符号表的组织方式:

### 1、分层组织符号表的登记项

各分程序符号表登记项按照语法识别顺序连续排列在一起，不为其内层分程序的符号表登记项所割裂。

### 2、用“分程序表”索引各分程序符号表的信息

分程序表中的各登记项是自左至右扫描源程序的过程中，按分程序出现的顺序依次填入的，且对每一个分程序填写一个登记项。

分程序表登记项序号隐含地表征各分程序的编号。



## 分程序表结构:

OUTERN	ECOUNT	POINTER
--------	--------	---------

其中:

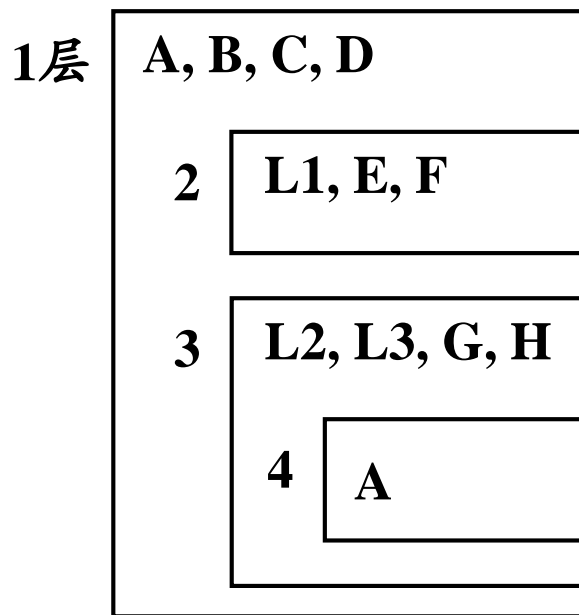
**OUTERN**——指明该分程序的直接外层分程序的编号

**ECOUNT**——记录该分程序符号表登记项的个数

**POINTER**——指向该分程序符号表的起始位置

例：设有如下分程序：

嵌套结构为：



**PROCEDURE ...**

**VAR A, B, C, D: REAL;**

**PROCEDURE ...**

**LABEL L1;**

**VAR E, F: REAL;**

**BEGIN**

**...**

**END;**

**PROCEDURE ...**

**LABEL L2, L3;**

**VAR G, H: REAL;**

**FUNCTION ...**

**VAR A: INTERGER;**

**BEGIN**

**...**

**END;**

**BEGIN**

**...**

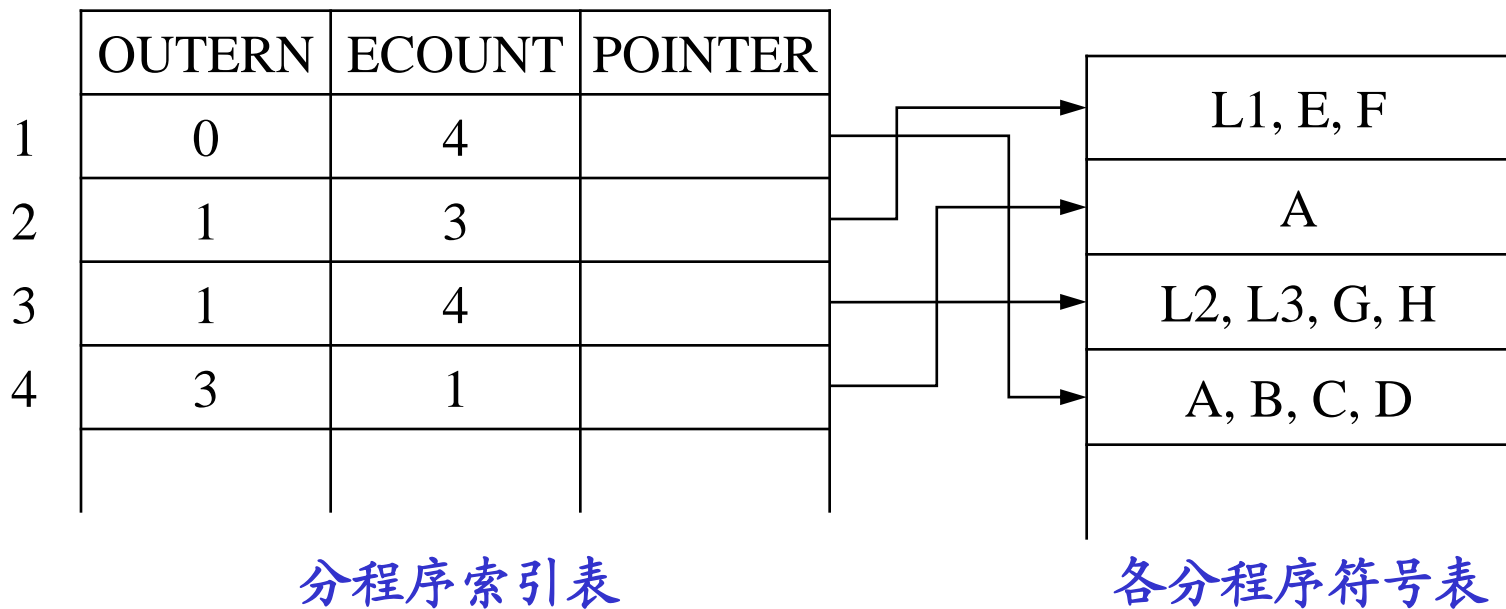
**END;**

**BEGIN**

**...**

**END;**

## 分程序表和符号表:



分程序索引表的形成顺序——每个模块头的出现顺序

分程序符号表的形成顺序——每个模块(语法分析时的  
语法单位)识别顺序

## 分程序符号表构造方法:

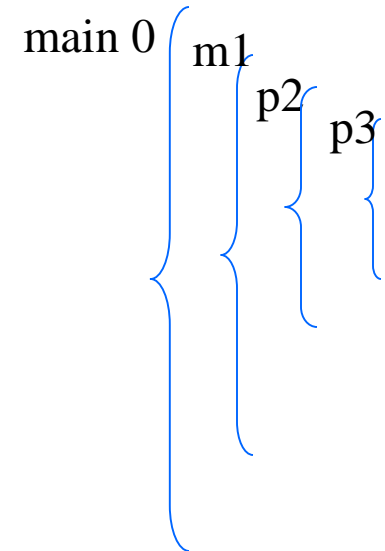
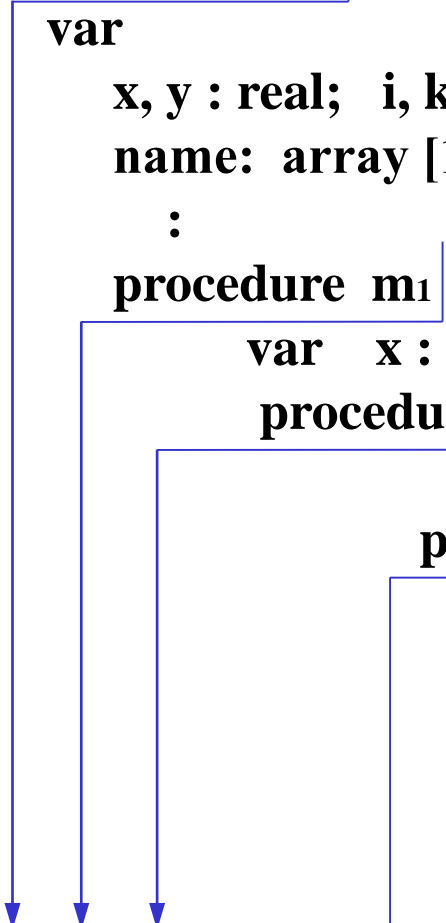
本例中，分程序符号表的形成顺序为2、4、3、1，这个次序是闭分程序的次序(分程序中**END**出现的次序)。

为使各分程序的符号表连续地邻接在一起，并在扫描具有嵌套分程序结构的源程序时，总是按先进后出的顺序来扫描其中各个分程序，可设一个临时工作栈。

每当进入一层分程序时，就在栈顶预造该分程序的符号表，而当遇到该层分程序的结束符(**END**)时，此时该分程序的全部登记项已位于栈顶，再将该分程序的全部登记项移至正式符号表中。

例：Pascal程序的分程序结构示例如下

```
program main0(...);  
  var  
    x, y : real;  i, k: integer;  
    name: array [1...16] of char;  
    :  
    procedure m11(ind: integer);  
      var x : integer;  
      procedure p22(j : real);  
        :  
        procedure p33;  
          var  
            f : array [1...5] of intrger  
            test1: boolean;  
            begin  
              :  
            end;  
          {p3}  
      end;  
    end;  
  end;  
end;
```





0 1 2

```
begin
:
end; {p2}
procedure q2;
var r1, r2 : real;
begin
:
p2(r1+r2);
:
end; {q2}
begin
:
p2(x/y);
:
end; {m1}
begin
:
m1(i+k);
:
end {main}
```

program main (...);

```
var
x, y : real; i, k : integer;
name: array [1...16] of char;
:
procedure m1 (ind:integer);
var x : integer;
procedure p2 (j : real);
:
procedure p3;
var
f : array [1...5] of intrger;
test1: boolean;
begin
:
end; {p3}
begin
:
end; {p2}
procedure q2;
var r1,r2 : real;
begin
:
p2(r1+r2);
:
end; {q2}
begin
:
p2(x/y);
:
end; {m1}
begin
:
m1(i+k);
:
end {main}
```

0 1 2



```
program main0(...);
```

```
var
```

```
  x, y : real; i, k: integer;  
  name: array [1...16] of char;  
  :
```

```
  procedure m11(ind:integer);
```

```
    var x : integer2
```

```
    procedure p2(j : real);
```

```
      :
```

```
      procedure p3;3
```

```
        var
```

```
        f : array [1...5] of integer
```

```
        test1: boolean;
```

```
        begin
```

```
          :
```

```
        end; {p3}
```

```
      begin
```

```
        :
```

```
      end;{p2}
```

```
    procedure q2;2
```

```
      var r1,r2 : real;
```

```
      begin
```

```
        :
```

```
        p2(r1+r2);
```

```
        :
```

```
      end; {q2}
```

```
    begin
```

```
      :
```

```
      p2(x/y);
```

```
      :
```

```
    end;{m1}
```

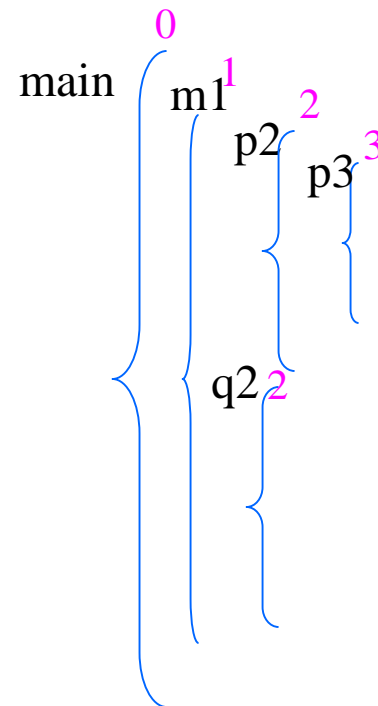
```
begin
```

```
  :
```

```
  m1(i+k);
```

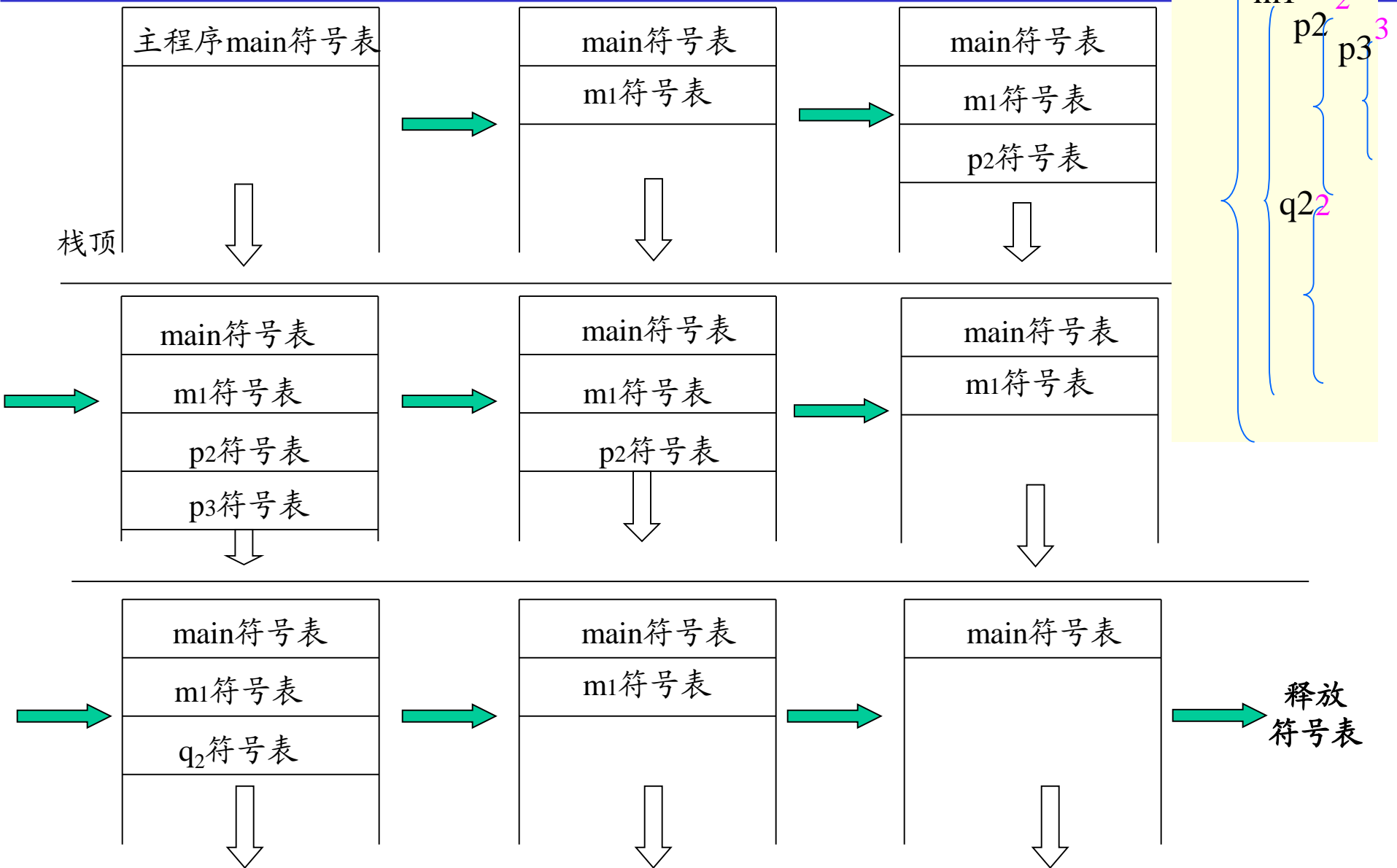
```
  :
```

```
end {main}
```





# 栈式符号表结构





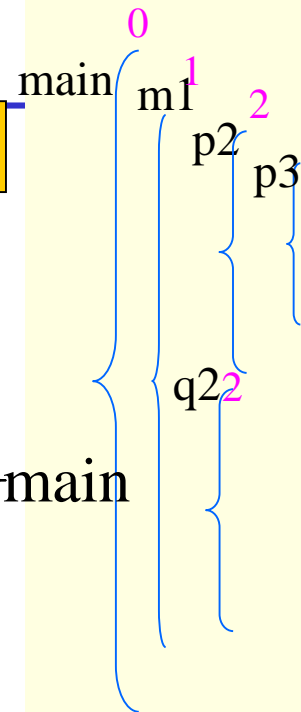


编译P<sub>3</sub>说明  
部分后的符  
号栈状态:

分程序索引表

主
m <sub>1</sub>
p <sub>2</sub>
p <sub>3</sub>

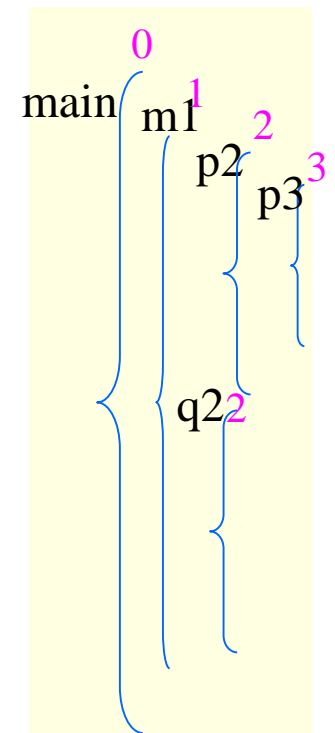
符号表序号	名字	kind	type	level	other
1	x	var	real	0	
2	y	var	real	0	
3	i	var	int	0	
4	k	var	int	0	
5	name	var	array	0	
6	m <sub>1</sub>	proc		0	
7	ind	para	int	1	
8	x	var	int	1	
9	p <sub>2</sub>	proc		1	
10	j	para	real	2	
11	p <sub>3</sub>	proc		2	
12	f	var	array	3	
13	test1	var	bool	3	





## 编译 $q_2$ 说明部分后:

1	x	var	real	0	main
2	y	var	real	0	
3	i	var	int	0	
4	k	var	int	0	
5	name	var	array	0	
6	$m_1$	proc			$m_1$
7	ind	para	int	1	
8	x	var	int	1	
9	$p_2$	proc		1	
10	j	para	real	1	
11	$q_2$	proc		1	$q_2$
12	$r_1$	var	real	2	
13	$r_2$	var	real	2	





当过程和函数体编译完成后，应将与之相应的参数名和局部变量名以及后者的特性信息从符号表中删去。

要求：给你一段程序，会画出其栈式符号表！

作业： P155: 3、5