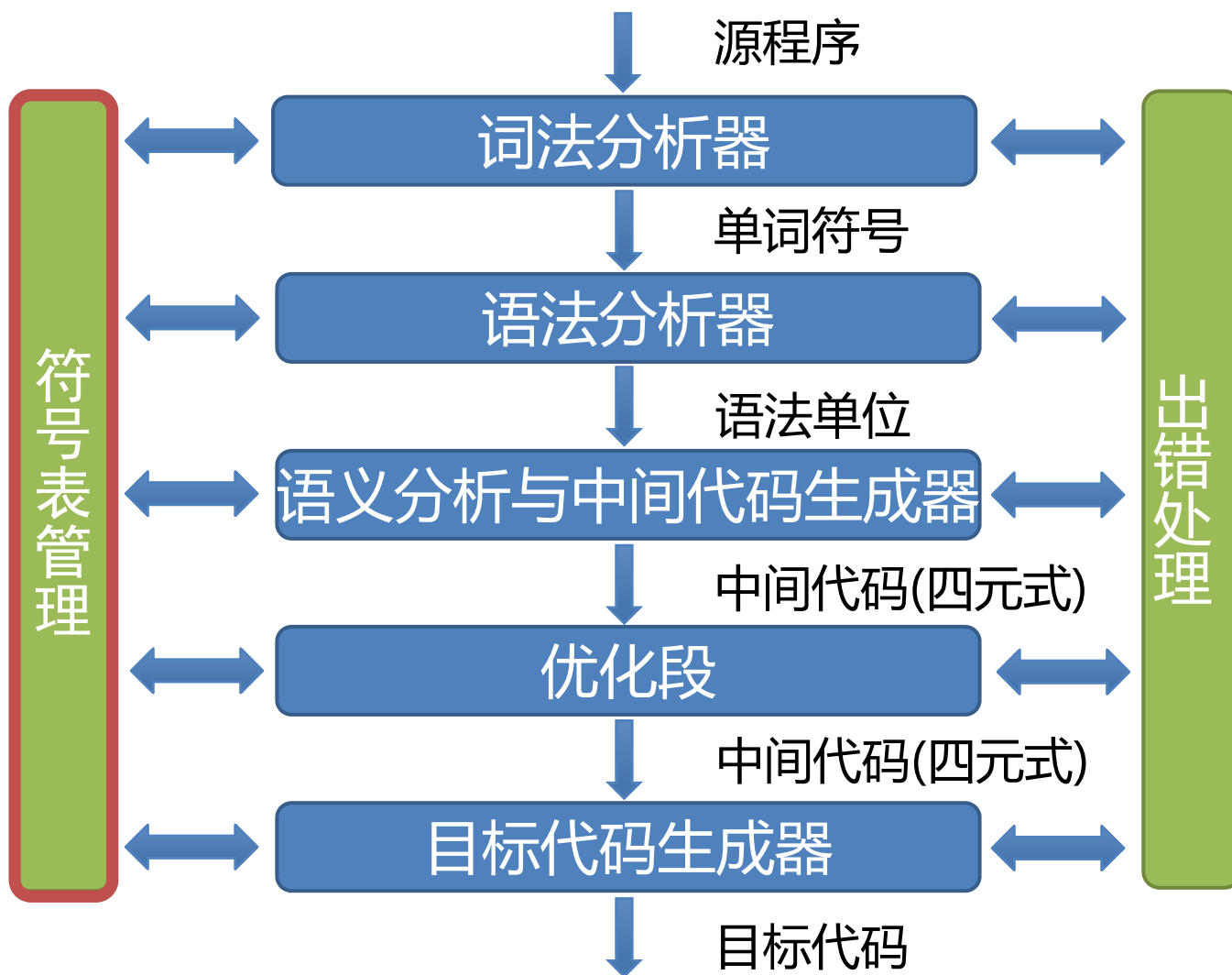


编译原理

符号表

编译程序总框



符号表

- ▶ 符号表的作用与组织
- ▶ 符号表的整理和查找
- ▶ 符号表的内容
- ▶ 利用符号表分析名字的作用域

符号表的作用

- ▶ 登记各类名字的信息
- ▶ 编译各阶段都需要使用符号表
 - ▶ 一致性检查和作用域分析
 - ▶ 辅助代码生成

符号表的组织

- ▶ 符号表的每一项(入口)包含两大栏
 - ▶ 名字栏, 也称主栏, 关键字栏
 - ▶ 信息栏, 记录相应的不同属性, 分为若干子栏
- ▶ 按名字的不同种属建立多张符号表, 如常数表、变量名表、过程名表、...

名字	信息
----	----

符号表的组织

▶ 对符号表的操作

- ▶ 填入名称
- ▶ 查找名字
- ▶ 访问信息
- ▶ 填写修改信息
- ▶ 删除

	NAME	INFORMATION
(1)	index	整型，变量
(2)	score	实型，变量
(3)	p	整型，形式参数

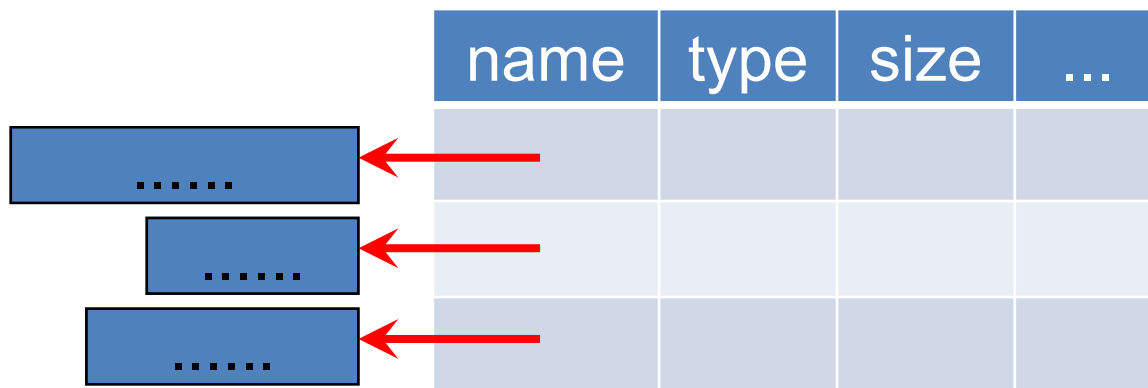
符号表的组织

- ▶ 对符号表进行操作的时机
 - ▶ 定义性出现
 - ▶ `int index`
 - ▶ 使用性出现
 - ▶ `if index < 100 ...`

符号表的组织

▶ 栏目的长度

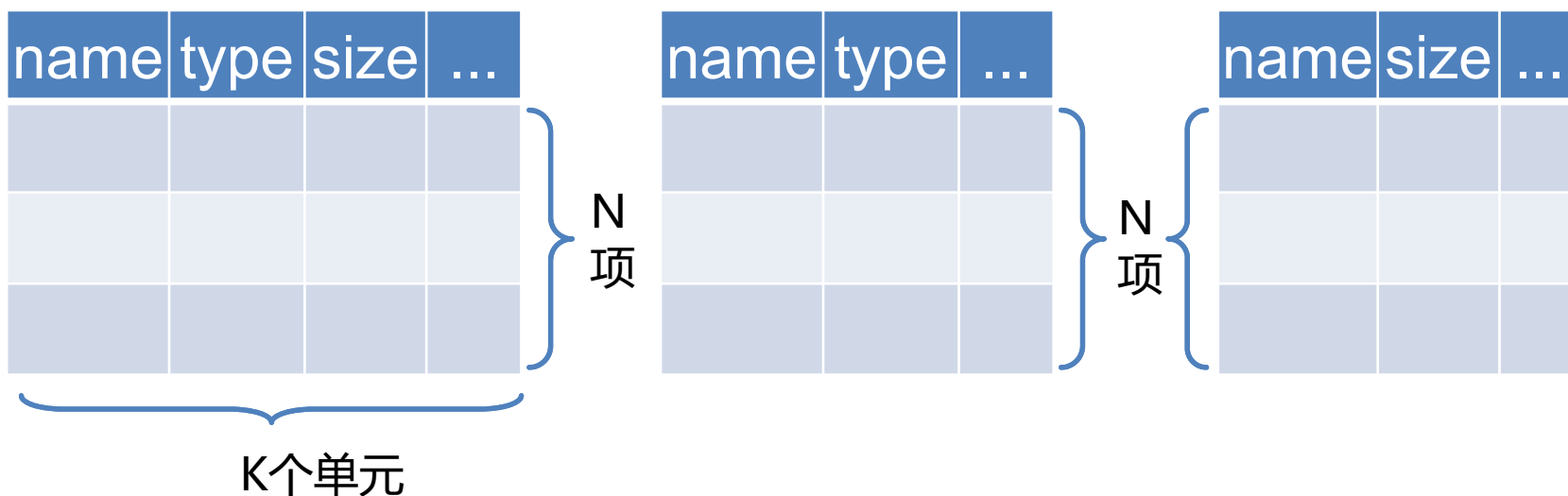
- ▶ 安排各项各栏的存储单元为固定长度
- ▶ 用间接方式安排各栏存储单元



符号表的组织

▶ 符号表的存放

- ▶ 把每一项置于连续K存储单元中，构成一张K*N的表 (N为符号表的项数)
- ▶ 把整个符号表分成M个子表，如 T_1 、 T_2 、...、 T_m ，每个子表含N项



编译原理

符号表的整理和查找

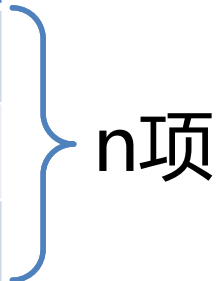
整理和查找

- ▶ 线性查找
- ▶ 二分查找
- ▶ 杂凑查找(HASH技术)

线性查找

- ▶ 按关键字出现的顺序填写各项
 - ▶ 结构简单，节省空间，填表快，
 - ▶ 查找慢，时间复杂度 $O(n)$
 - ▶ 改进：自适应线性表

name	type	size	...




n项

二分查找

- ▶ 表格中的项按名字的“大小”顺序整理排列
- ▶ 填表慢，查找快
- ▶ 时间复杂度: $O(\log_2 n)$
- ▶ 改进：组织成二叉树

name	type	size	...



杂凑查找(HASH技术)

▶ 杂凑函数 $H(\text{SYM})$: $0 \sim n-1$

▶ n : 符号表的项数

▶ 填表快, 查找快

▶ 要求

▶ 计算简单高效

▶ 函数值分布均匀

name	type	size	...

} n项

编译原理

符号表的内容

符号表的内容

- ▶ 符号表的信息栏中登记了每个名字的有关性质
 - ▶ 类型：整、实或布尔等
 - ▶ 种属：简单变量、数组、过程、函数等
 - ▶ 大小：长度，即所需的存储单元字数
 - ▶ 相对数：指分配给该名字的存储单元的相对地址

PL语言简介

- ▶ PL语言是PASCAL语言的一个子集
 - ▶ 常量、类型、变量和过程等说明
 - ▶ 过程说明允许嵌套，过程调用允许递归，过程参数可以是值参数和变量参数
 - ▶ 3种标准类型：整型、字符型、布尔型；自定义数组类型
 - ▶ 赋值语句、条件语句、while语句、过程调用语句、读语句和写语句

PL 语言编译程序的符号表

► 表格的定义

- 名字表(nametab)
- 程序体表(btab)
- 层次显示表(display)
- 数组信息表(atab)
- 中间代码表(code)

名字表(nametab)

► 登记程序中出现的各种名字及其属性

	name	kind	lev	typ	normal	ref	adr/val/size	link
0								
1								
⋮								
tx →								

指向同一程序体中定义的上一个名字在 nametab 中的位置，每个程序体在 nametab 中登记的第一个名字的 link 为 0

▪ **val**, 当名字为常量名时，填入他们的相应值

▪ **size**, 当名字为类型名时，填入该类型数据所需存贮单元数目

在相
真入

程序体

	name	kind	lev	typ	normal	ref	adr/val/size	link
0								
1								
⋮								
tx →								

- ▶ 记录各程序体的总信息
- ▶ 用于对源程序中定义的名字的作用域进行分析
- ▶ 对名字表进行管理

btab

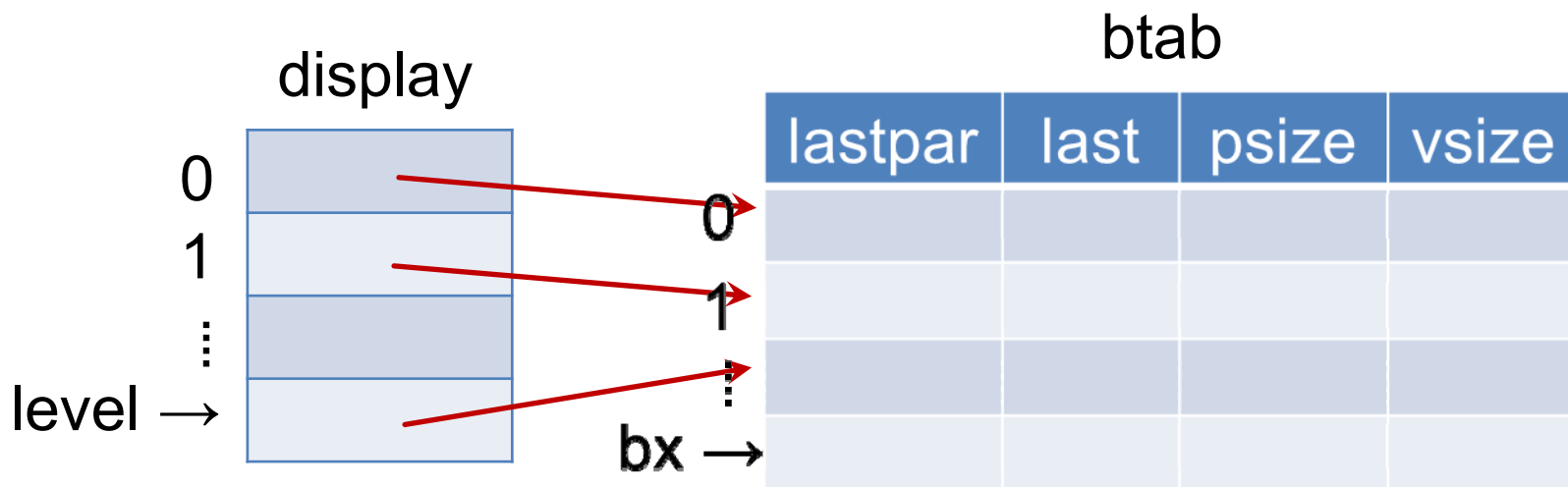
	lastpar	last	psize	vsize
0				
1				
⋮				

指
n 本程序体所有
括连接数据所

本程序体所有局部数据所
需空间大小

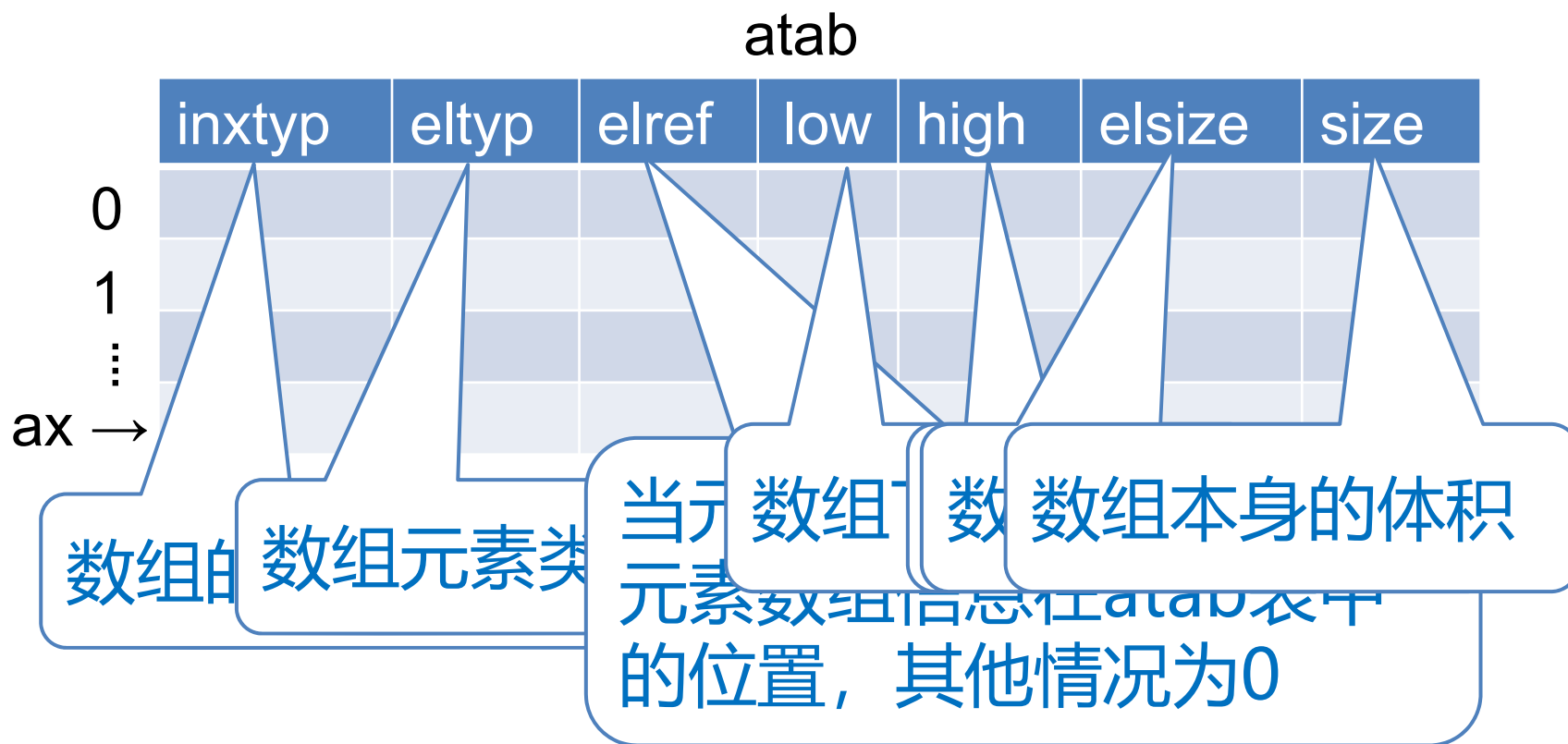
层次显示表display

- ▶ 描述正在处理的各嵌套层
- ▶ 对程序体表进行管理



数组信息表atab

- ▶ 用于记录每个数组的详细信息



数组声明示例

► type a=array[1..10, 1..20] of integer;

nametab

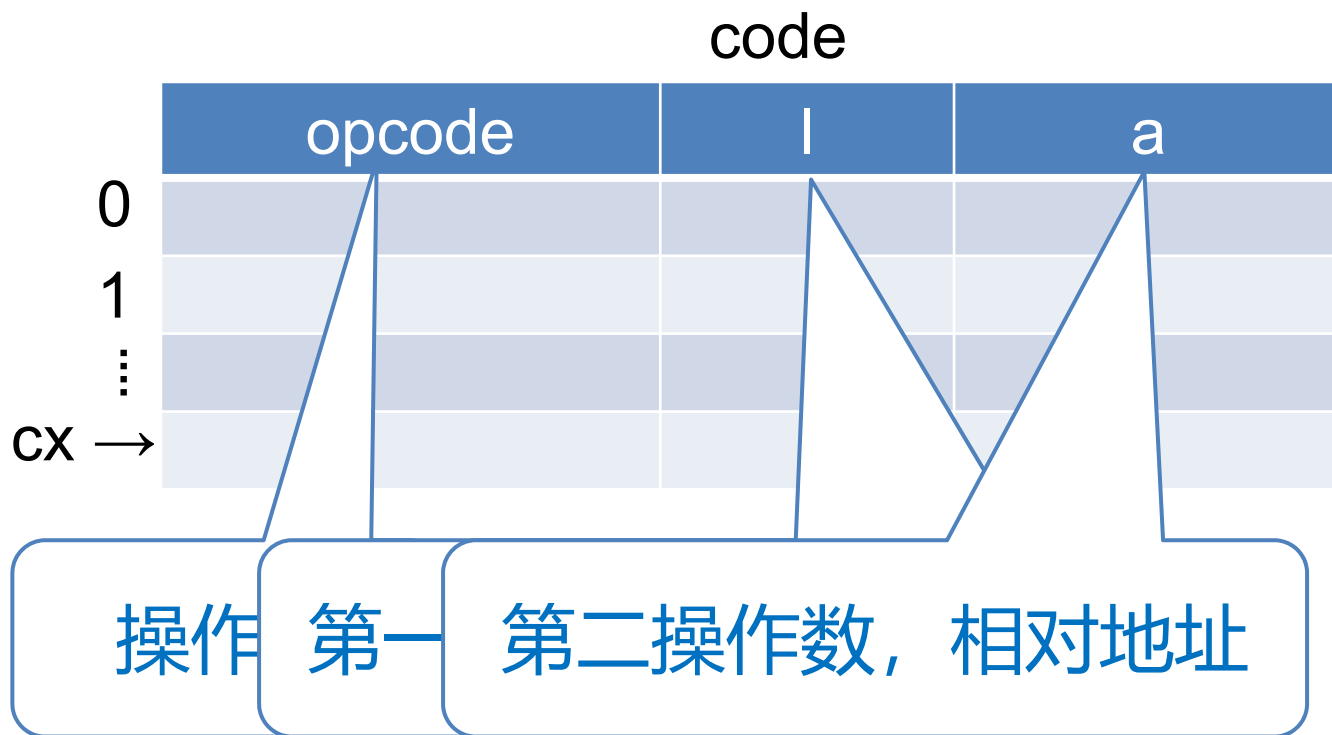
	name	kind	typ	ref	...
⋮					
k	a	type	arrays	n	
⋮					
tx →					

atab

	inxtyp	eltyp	elref	low	high	elsize	size
⋮							
n	ints	arrays	m	1	10	20	200
⋮							
m	ints	ints	0	1	20	1	20
ax →							

中间代码表code

- 用于存放编译程序所产生的每条中间代码



编译原理

利用符号表分析名字的作用域

名字的作用范围

- ▶ 在许多程序语言中，允许同一个标识符在不同过程中代表不同的名字
- ▶ 名字都有一个确定的作用范围
- ▶ **作用域**
 - ▶ 一个名字能被使用的区域范围称作这个名字的作用域
- ▶ 两种程序体结构
 - ▶ 并列结构，如FORTRAN
 - ▶ 嵌套结构，如PASCAL，PL

FORTRAN程序结构

- ▶ 一个程序由一个主程序段和若干辅程序段组成
- ▶ 辅程序段可以是子程序、函数段或数据块
- ▶ 每个程序段由一系列的说明语句和执行语句组成，各段可以独立编译
- ▶ 模块结构，没有嵌套和递归
- ▶ 各程序段中的名字相互独立，同一个标识符在不同的程序段中代表不同的名字
- ▶ 把局部名和全局名分别存在不同的地方

主程序

```
PROGRAM ...  
...  
end
```

辅程序1

```
SUBROUTINE ...  
...  
end
```

辅程序2

```
FUNCTION ...  
...  
end
```

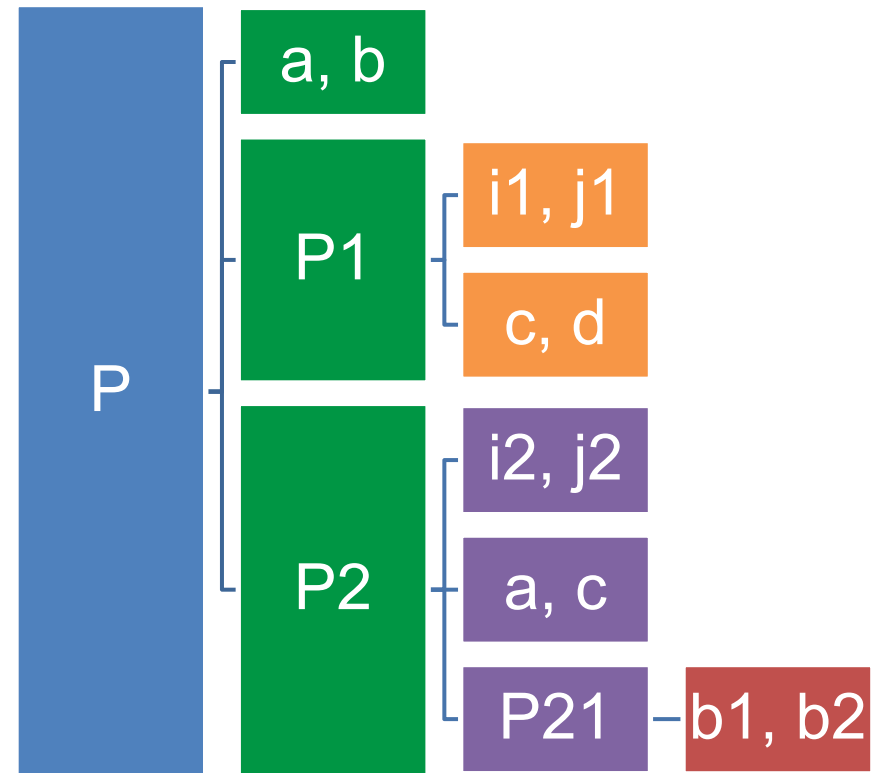
PASCAL/PL程序结构

- ▶ PASCAL/PL程序本身可以看成是一个操作系统调用的过程，过程可以嵌套和递归
- ▶ 一个PASCAL/PL过程

```
过程头;  
    说明段（由一系列的说明语句组成）;  
begin  
    执行体（由一系列的执行语句组成）;  
end
```

PASCAL/PL程序结构

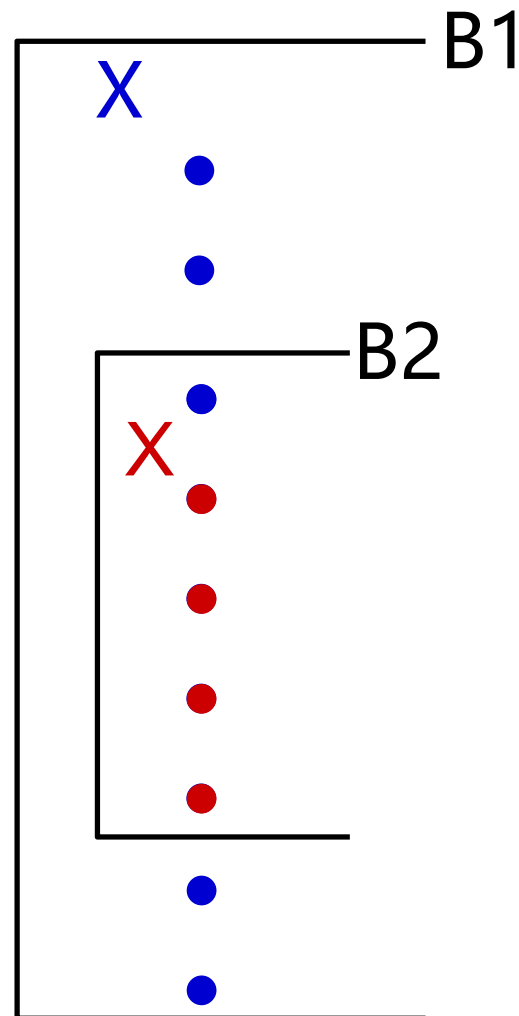
```
program P;  
  var a,b : integer;  
  procedure P1 (i1, j1:integer);  
    var c,d : integer  
  ..  
end;  
  procedure P2 (i2, j2:integer);  
    var a,c : integer;  
    procedure P21;  
      var b1,b2 : boolean;  
    ..  
end;  
..  
end.  
end.
```



PASCAL/PL名字作用域分析

▶ 最近嵌套原则

- ▶ 一个在子程序B1中说明的名字X只在B1中有效（局部于B1）
- ▶ 如果B2是B1的一个内层子程序且B2中对标识符X没有新的说明，则原来的名字X在B2中仍然有效
- ▶ 如果B2对X重新作了说明，那么，B2对X的任何引用都是指重新说明过的这个X

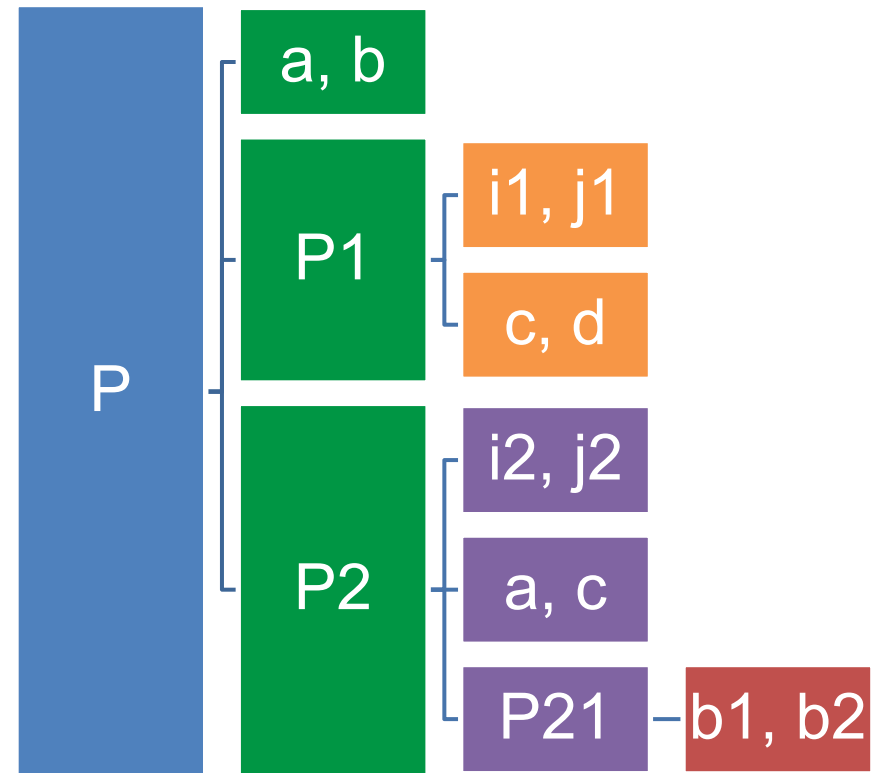


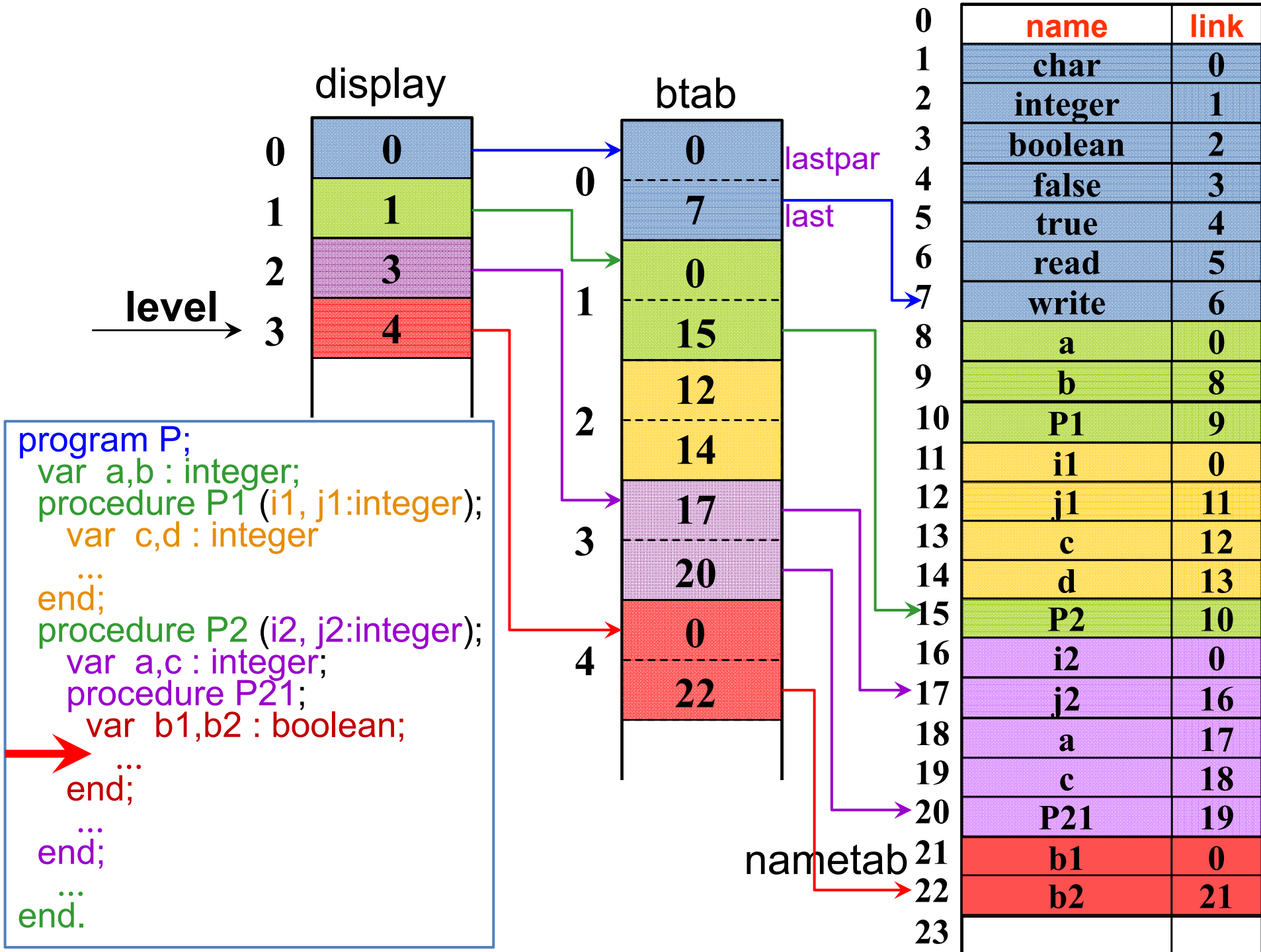
名字作用域分析

- ▶ PL语言的中间代码: `opcod l a`
 - ▶ `opcod`: 操作码
 - ▶ `l`: 第一操作数, 程序体层数
 - ▶ `a`: 第二操作数, 相对地址
- ▶ 编译时如何确定变量的层数(地址)?

PASCAL/PL程序示例

```
program P;  
  var a,b : integer;  
  procedure P1 (i1, j1:integer);  
    var c,d : integer  
  ..  
end;  
  procedure P2 (i2, j2:integer);  
    var a,c : integer;  
    procedure P21;  
      var b1,b2 : boolean;  
    ..  
end;  
..  
end.  
end.
```





display	
0	0
1	1
2	3
3	4

btab	
0	0
1	0
2	0
3	0
4	0

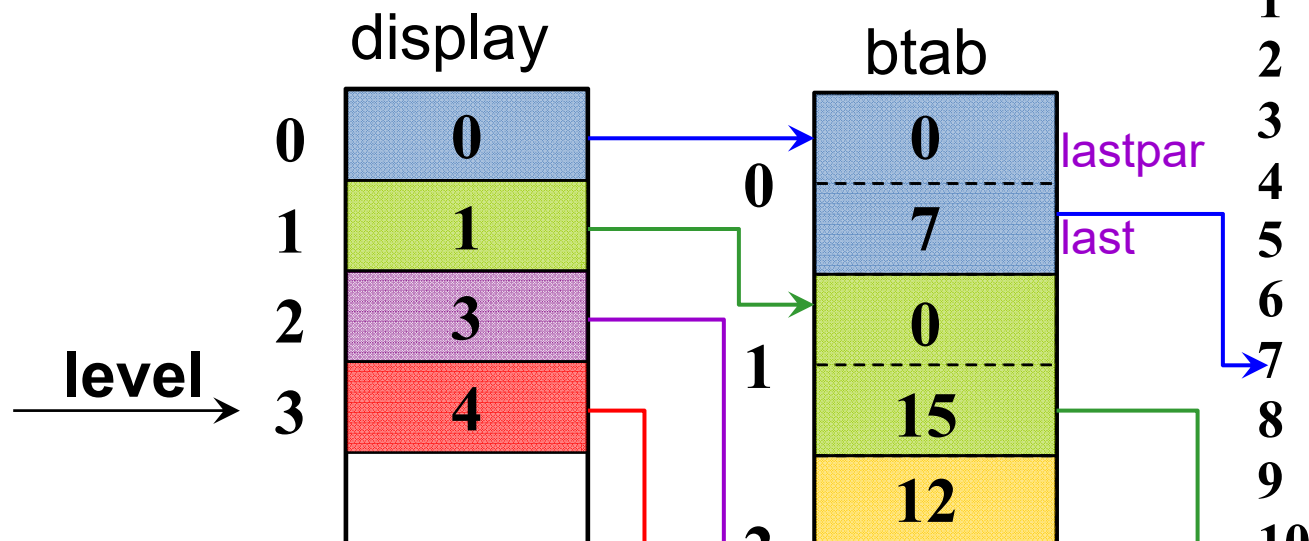
nametab

0	name	link
1	char	0
2	integer	1
3	boolean	2
4	false	3
5	true	4
6	read	5
7	write	6
8	a	0
9	b	8
10	P1	9
11	i1	0
12	j1	11
13	c	12
14	d	13
15	P2	10
16	i2	0
17	j2	16
18	a	17
19	c	18
20	P21	19
21	b1	0
22	b2	21
23		

```

program P;
var a,b : integer;
procedure P1 (i1, j1:integer);
  var c,d : integer
  ...
end;
procedure P2 (i2, j2:integer);
  var a,c : integer;
  procedure P21;
    var b1,b2 : boolean;
    ...
  end;
end;
end.

```

0	name	link
1	char	0
2	integer	1
3	boolean	2
4	false	3
5	true	4
6	read	5
7	write	6
8	a	0
9	b	8
10	P1	9
11	i1	0
12	j1	11
13	c	12
14	d	13
15	P2	10
16	i2	0
17	j2	16
18	a	17
19	c	18
20	P21	19
21	b1	0
22	b2	21
23		

```

program P;
var a,b : integer;
procedure P1 (i1, j1:integer);
  var c,d : integer
  ...
end;
procedure P2 (i2, j2:integer);
  var a,c : integer;
  procedure P21;
    var b1,b2 : boolean;
    ...
  end;
end;
end;
end.

```

b1 := **a** + **b**

nametab

名字作用域分析

- ▶ 指令格式: `opcod l a`
 - ▶ `opcod`: 操作码
 - ▶ `l`: 第一操作数, 程序体层数
 - ▶ `a`: 第二操作数, 相对地址
- ▶ 编译是如何确定变量的层数(地址)?
- ▶ 运行时如何根据指令中变量的层数和相对地址确定变量的存储单元?

小结

- ▶ 符号表的作用与组织
- ▶ 整理和查找
- ▶ 符号表的内容
- ▶ 名字的作用域分析