

第八章 符号表

编译程序必须收集，利用出现在源程序中的名字的信息(如表示名字的字符串、名字的类型、种属、存储单元以及其它依赖于语言的特性)

扫描源程序，遇到一个名字就检索表，若它是一个新的名字，就填表；在调用词法语法分析程序时，还要把有关的信息填入（如标号的链，定义否）

语义分析时，用它来检查名字的使用是否与它们原先的说明一致，如GOTO L, L 为标号；

代码生成时，需要了解必须给名字分配多少，以及哪一种存储；

本章内容

- 符号表的内容及作用
- 符号表的组织
- 整理与查找
- 名字的作用范围

8.1 符号表的内容及作用

■ 标识符的属性

- ⊛ 名字
- ⊛ 类型（整、实、双精、布尔、字符、复、标号、指针等）
- ⊛ 种属
- ⊛ 变量的存储位置及长度
- ⊛ 数组的内情向量
- ⊛ 标号的定义标志及位置
- ⊛ 形式参数标志
- ⊛

符号表的内容（续）

■ 过程和函数名

- ⊛ 局部变量（包括形式参数）占用的存储空间
- ⊛ 返回结果类型
- ⊛ 局部过程名及符号表指针
- ⊛ 嵌套外层过程

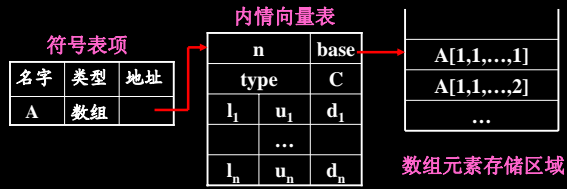
例：符号表中的信息

符号表项

| 名字 | 类型 | 地址 |
|-----|-------|----|
| x | T_INT | 0 |
| y | T_FLO | 4 |
| ... | | |

例：符号表中的信息 - 静态数组

例：符号表中的信息 - 静态数组



符号表的特点

- 符号表中的信息多样化
- 符号表中的信息跟编译实现技术相关
- 符号表中的信息跟语言风范有关

符号表的作用

- 查名字是否存在
- 插入一个新名字
- 访问名字的信息
- 更新名字的信息
- 删除项

符号表的一般格式

| 符号表登记项 (入口) | 名字栏 | 信息栏 |
|-------------|-----|-----|
| 1 | | |
| 2 | | |
| ... | | |
| n | | |

8.2 符号表的组织

- 字典数据结构
- 基本操作
 - ⊗ insert
 - ⊗ lookup
 - ⊗ delete
- 实现
 - ⊗ 线性表
 - ⊗ 搜索树 (二叉树、AVL数、B树)
 - ⊗ 杂凑表

A dictionary data structure is one which is capable of storing objects in sorted order based on key such as a string or an integer.

符号表的组织

- 名字栏一般用等宽表示，处理的办法有两种：
 - ⊗ 语言中限制名字的长度为一个固定值；
 - ⊗ 语言中不限制名字的长度。

| 名字栏 | 信息栏 |
|--------|-----|
| k | |
| AMOUNT | |
| ... | |

实现表的最简单方式是看成记录的线性数组，每一栏目所占存储单元长度不变，每个名字对应一个记录，记录通常由已知个连续的存储字组成。

名字栏的表示

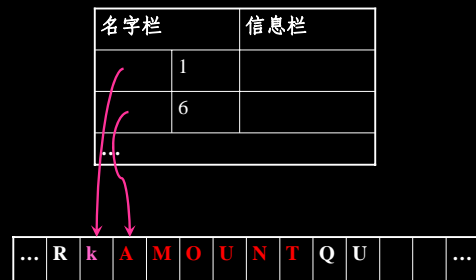


图6.1 (a) 名字存储在分开的数组中

名字栏的表示2

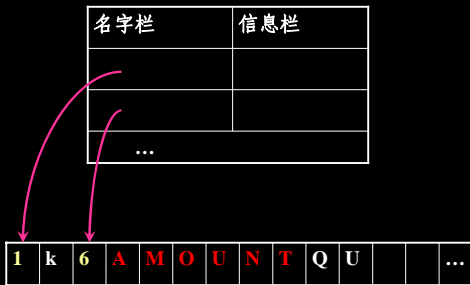


图6.1 (b) 名字存储在分开的数组中

信息栏的表示

- 信息栏用等宽表示，可以包含指针，将额外的部分放到其它地方
- 按种属建立各自的符号表
语言中，名字表示各种类型的对象：变量名，过程名，常数，域名（结构），标号等，由于各种类型的名字因用法不同，栏目及所需的空间相差很大，因此往往独立建表，参见P225图8.4

符号表登记项的建立

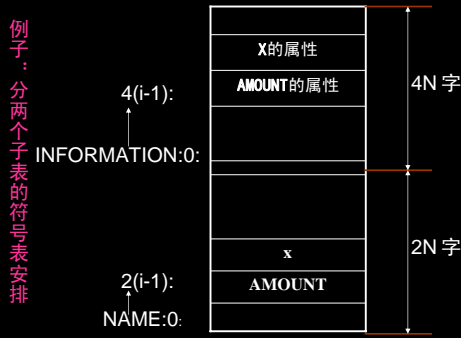
I. 符号表名字何时建立:

1. 词法分析。当识别了一个标识符时，它可以建立，不能填充种属，类型等；insert() 操作返回 (\$ID, 符号表指针)；
2. 但是当名字在同一个程序块或过程中，可用作标号、实型变量、域名时，词法分析器只能返回 (\$ID, TOKEN)，由语法分析程序为标识符建表

II. 这些信息在不同的时刻插入符号表:

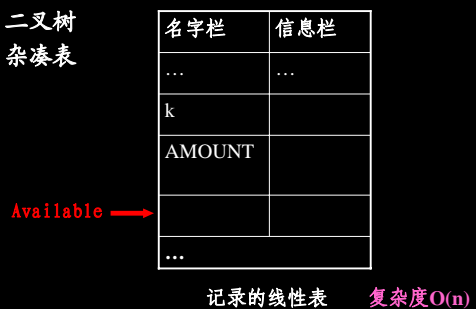
1. 遇到显式说明时，把属性插入；
2. 语法可以隐含的说明变量的某种用途，如标号后的 ':'，所以和产生式
语句 → id: 语句 相关的语义动作是把 id 为标号的事实插入符号表，填入标号链，定义否；
3. 地址分配；
4. 类似过程说明的语法告诉我们某些名字是形参。

符号表的两种存储方式：各项依次连续存储；按某些列依次存放。



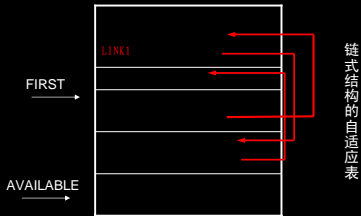
8.3 整理与查找

- 线性表
- 二叉树
- 杂凑表



自适应表 (Self-organizing List)

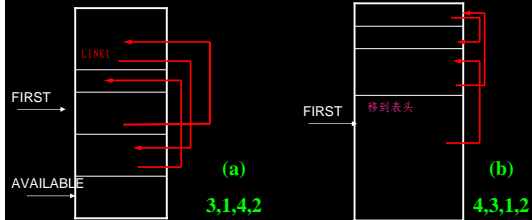
以少量额外空间为代价，在每一个记录中增加一个Link域，并按Link域指示的顺序检索表
 设定First指出第一个记录的位置，Link指出下一个记录



Link域把所有的项按“最新最近”访问原则连成一条链使得在任何时候，这条链的第一个元素所指的项是最近被查询过的项，第二个元素是次新查询过的项……依据是刚刚被访问过的项以大概率被继续访问

做法

当引用一个名字或者当第一次建立该名字的纪录时，我们通过移动指针把名字纪录移到表的前部，下图说明把NAMB4纪录移到头上（NAMB4刚刚被引用），原来的键是3142，现在改为4312



折半查找

- 采用线性表
 - ⊗ 表始终保持有序，即插入时先要找到位置
 - ⊗ 按照折半查找算法进行，复杂度 $O(\log n)$
 - (缺点：插入元素慢)
- 采用二叉排序树
 - ⊗ 建立二叉排序树
 - ⊗ 二叉排序树的查找算法
 - (缺点：有额外开销，平衡树)

杂凑法

$h(\text{KEY})=k$

高效、均匀

$h(x)=m$

$h(\text{AMOUNT})=n$

| KEY | INFO |
|-----|--------|
| 0 | |
| ... | |
| m | X |
| ... | |
| n | AMOUNT |
| ... | |

l

w

A. 开放的散列表

- 给定关键字 KEY，在长度为 N 的表中加入或者找到该记录的算法：
 - ⊗ 计算 $k = h(\text{KEY})$;
 - ⊗ 若表中 k 的位置为空，或在该元名字域中有这个关键字 KEY，则返回；
 - ⊗ 否则冲突，插入到邻近位置，即置 $k = (k+1) \bmod M$ （其中 M 为表长），转第二步。
- 只要该表未填满，名字的查找总会终止。

例

表长 $M = 8$ ，名字的始符为 A ~ H， $h(K)$ 的位置为 KEY 的第一个字母在字母表的位置

| |
|------|
| HAY |
| CAT |
| DOG |
| COLD |
| DAY |
| HOT |

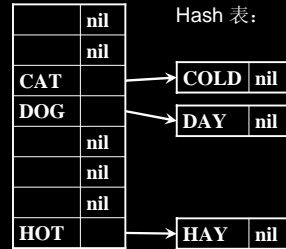
$h(\text{CAT})=3$
 $h(\text{DOG})=4$
 $h(\text{COLD})=3$
 $h(\text{DAY})=4$
 $h(\text{HOT})=8$
 $h(\text{HAY})=8$

B. 溢出混列

- 溢出项插入到一个完全独立的表中
- 如果溢出项不多，可用线性查表法作溢出表的工作
- 当溢出表很大时此法有局限性

C. 使用链的溢出混列

- 为了解决溢出表的查找，在溢出表中使用链



杂凑函数的构造

- 总时间 = 计算 h + 查找长度
- 取舍策略: 分布均匀、查找长度短

例: $M = 128$, 设计 h , 该表为 FORTRAN 的符号表, 以字母开头, 后跟 1~5 个字母或者数字, 这样一共有 $26 * 36^5$ 个标识符, 他们映像到 0~127 表项中 (用 8 进制表示):

A~Z → 41~72
 0~9 → 20~29
 空白 → 01

如标识符 CAT2 表示为

43 41 64 22 01 01
 C A T 2

由于表项 128 项, 要产生 7 位二进制数字作为结果的映像

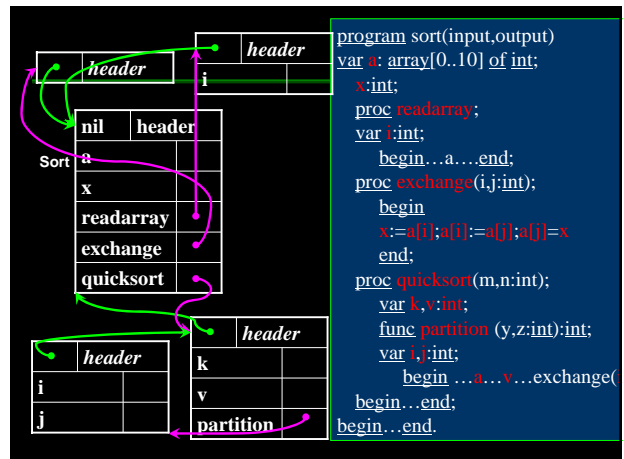
方法

- 最简单法, 截取头 7 位二进制位作为 h 的结果得到 1000011
- 取 2~8 位
- 取 6~12 位
- 选和
- 除法
- 平方取中

8.4 名字的作用范围

变量的作用域问题

- 符号表中如何表示不同作用域中的同名变量?
- 查找变量时能够遵守作用域嵌套原则?



8.4.1 FORTRAN 的符号表组织

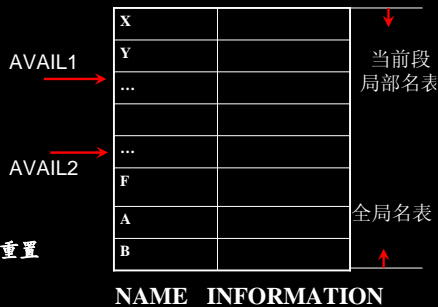
■ 模块结构：程序由1个主程序块和多个子程序块组成

1. 对于一遍扫描编译程序：
 - 分块编译，局部量在本块编译完后，就可以消失，全局量不能消失
2. 对于多遍扫描编译程序：
 - 处理完一个块后应该把它的局部名表保存在外存中以便后续遍处理这块时使用

```
PROGRAM MAIN
INTEGER A
REAL F, B, X, Y
COMMON B, A, F
A = -14
B = 99.9
F = 0.2
CALL SUB(X, Y)
...
END
SUBROUTINE SUB(P, Q)
INTEGER I
REAL A, B, P, Q
COMMON A, I, B
...
END
```

| Main Program | Common Memory Storage | Subroutine |
|--------------|-----------------------|------------|
| B | 99.9 | A |
| A | -14 | I |
| F | 0.2 | B |

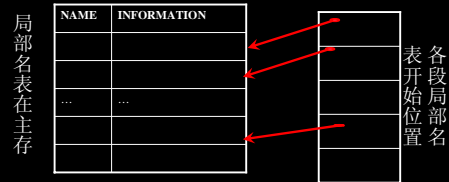
一遍扫描的处理办法



- 一块处理完了，重置 AVAIL1；
- 当 AVAIL1 == AVAIL2 的时候，表区已满；

多遍扫描的处理办法

- 处理完一块后，它的局部名表保存到外存，当后续遍使用时装入
- 采用一维数组纪录各块的局部符号表开始位置



8.4.2 Pascal符号表组织

- 最近作用域嵌套原则：
 - ④ 一个名字的作用域是那个包含了这个名字说明的最小过程（或函数）
 - ④ 进入时建立一张子符号表；退出时删除（释放）相应的子符号表
 - ④ 过程是嵌套的，用层来表示当前处理到的过程

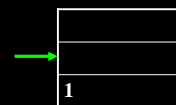
```
program sort(input,output)
var a: array[0..10] of int;
x:int;
proc readarray;
var i:int;
begin...a...end;
proc exchange (i,j:int);
begin
x:=a[i];a[i]:=a[j];a[j]=x
end;
proc quicksort (m,n:int);
var k,v:int;
func partition (y,z:int):int;
var i,j:int;
begin ... a... v... exchange(
begin...end;
begin...end.
```

- 栈符号表：从栈顶插入符号；查找也是从栈顶开始；

- 过程嵌套层次表 DISPLAY 过程的子符号表在栈符号表中的起始位置；DISPLAY 本身是个栈

| name | info | previous |
|------|------|----------|
| | | |
| x | | 0 |
| a | | 2 |

- Previous 指针 同层内的栈符号表项通过 Previous 指针相连 每层最后一个的 Previous 指针为空



| | name | info | previous |
|---|-------|------|----------|
| 4 | | | |
| 3 | reada | | 0 |
| 2 | x | | 3 |
| 1 | a | | 2 |

| |
|---|
| 1 |
|---|

```

program sort(input,output)
var a: array[0..10] of int;
x:int;
proc readarray;
var i:int;
begin...a...end;
proc exchange (i,j:int);
begin
x:=a[i];a[i]:=a[j];a[j]=x
end;
proc quicksort (m,n:int);
var k,v:int;
func partition (y,z:int):int;
var i,j:int;
begin ... a...v...exchange(
begin...end;
begin...end.
    
```

| | name | info | previous |
|---|-------|------|----------|
| 5 | | | |
| 4 | i | | 0 |
| 3 | reada | | 0 |
| 2 | x | | 3 |
| 1 | a | | 2 |

| |
|---|
| 4 |
| 1 |

```

program sort(input,output)
var a: array[0..10] of int;
x:int;
proc readarray;
var i:int;
begin...a...end;
proc exchange (i,j:int);
begin
x:=a[i];a[i]:=a[j];a[j]=x
end;
proc quicksort (m,n:int);
var k,v:int;
func partition (y,z:int):int;
var i,j:int;
begin ... a...v...exchange(
begin...end;
begin...end.
    
```

| | name | info | previous |
|---|------|------|----------|
| | j | | 0 |
| | i | | 5 |
| | exch | | 0 |
| 2 | x | | 3 |
| 1 | a | | 2 |

| |
|---|
| 4 |
| 1 |

```

program sort(input,output)
var a: array[0..10] of int;
x:int;
proc readarray;
var i:int;
begin...a...end;
proc exchange (i,j:int);
begin
x:=a[i];a[i]:=a[j];a[j]=x
end;
proc quicksort (m,n:int);
var k,v:int;
func partition (y,z:int):int;
var i,j:int;
begin ... a...v...exchange(
begin...end;
begin...end.
    
```

| | name | info | previous |
|---|-------|------|----------|
| | v | | 0 |
| | k | | 7 |
| | n | | 6 |
| 4 | m | | 5 |
| | quick | | 0 |
| 2 | x | | 3 |
| 1 | a | | 2 |

| |
|---|
| 4 |
| 1 |

```

program sort(input,output)
var a: array[0..10] of int;
x:int;
proc readarray;
var i:int;
begin...a...end;
proc exchange (i,j:int);
begin
x:=a[i];a[i]:=a[j];a[j]=x
end;
proc quicksort (m,n:int);
var k,v:int;
func partition (y,z:int):int;
var i,j:int;
begin ... a...v...exchange(
begin...end;
begin...end.
    
```

| | name | info | previous |
|----|-------|------|----------|
| 12 | j | | 0 |
| 11 | i | | 12 |
| 10 | z | | 11 |
| 9 | y | | 10 |
| 8 | parti | | 0 |
| 7 | v | | 8 |
| 4 | m..k | ... | ... |
| 3 | quick | | 0 |
| 2 | x | | 3 |
| 1 | a | | 2 |

| |
|---|
| 9 |
| 4 |
| 1 |

```

program sort(input,output)
var a: array[0..10] of int;
x:int;
proc readarray;
var i:int;
begin...a...end;
proc exchange (i,j:int);
begin
x:=a[i];a[i]:=a[j];a[j]=x
end;
proc quicksort (m,n:int);
var k,v:int;
func partition (y,z:int):int;
var i,j:int;
begin ... a...v...exchange(
begin...end;
begin...end.
    
```

参考文献及作业

1. Kenneth C.Louden,冯博琴译,《编译原理与实践》,机械工业出版社, P227-P264;
2. 严蔚敏,《数据结构与算法》,清华大学出版社, Hash表部分与 Binary Tree 部分。

本章作业: P236 1, 2, 3, 6