第四章：语义分析

抽象地址
符号表

1. 抽象地址

- 1.1 地址分配
- 1.2 抽象地址结构
- 1.3 层数定义
- 1.4 过程活动记录
- 1.5 分配原则

1.1 地址分配原则

- 静态分配

  在编译时间即为所有数据对象分配固定的地址单元，且这些地址在运行时间始终保持不变。是一种直观的方法，但不适用于动态申请空间。

- 动态分配

  程序中变量分配的地址不是具体的地址，而是一个抽象地址，当程序运行时根据抽象地址分配具体的物理地址。

1.2 抽象地址的结构

层数	偏移

- 抽象地址的形式是一个二元组，由层数和偏移组成。
- 层数主要是针对嵌套式语言，表示的是某个函数所处的嵌套定义层数。
- 偏移是针对过程活动记录的一个相对偏移量

1.3 层数的定义

- 嵌套式语言中：
  - 主程序设为0层
  - 主程序直接定义的函数和过程定义为1层
  - 若某函数为L层，则该函数直接定义的函数为L+1层

- 并列式语言中：
  - 全局变量定义为0层，函数中的变量定义为1层

1.4 过程活动记录

- 每次函数被调用时都会给函数分配一片空间，用以存储如图信息，我们把这片存储空间称作过程活动记录
- 偏移是针对过程活动记录的，通过起始位置+偏移量就可以找到对应的物理位置。
- 过程活动记录中存储的顺序实际上是处理的先后顺序。

1.5 空间分配原则

- 临时变量分配一个单元(这个只是假设，具体情况根据目标机和操作数的类型决定)
- 局部变量按类型大小分
- 形参
  - 地址引用型形参：分一个单元
  - 值引用型形参：按类型大小分
  - 过/函形参：分两个单元（入口地址，display表信息）

2. 符号表概述

- 符号表在编译的不同阶段都要用到
  - 在语义分析中，符号表所登记的内容将用于语义检查（如检查一个名字的使用和原先的声明是否一致）；
  - 在产生中间代码中，用于类型检查及转换；
  - 在目标代码生成阶段，当对名字进行地址分配时，符号表是地址分配的依据；

- 合理地组织符号表的意义：

在编译程序工作的过程中，需要对符号表进行频繁的访问（查表和填表等操作），因此，合理地合理地**组织符号表**，并相应地选择好查表和填表的方法，是提高编译程序效率的重要一环。

组织符号表主要解决的问题：

符号表的总体组织

符号表表项的排列

符号表的局部化

2.1 符号表的总体组织

- 符号表存储的是标识符的语义信息，包含两部分：**标识符**的名字；**语义字**(种类、类型、抽象地址. . .)
- 不同类别的标识符所包含的信息是不同的。
- 符号表的总体组织既可以采用多表结构，也可以采用单表结构, 也可以二者折中。究竟采用哪种结构并没有统一的规定，编译程序可根据实际处理语言的需要进行选择。

1、多表结构

- 是将属性完全相同的那些符号存放在一张符号表中，从而有常量表、类型表、变量表等。
- 多表结构的优点：每个符号表的属性个数和结构完全相同，各个表项是等长的，并且表项中的每个属性都是有效的，管理起来方便一致，空间效率高。
- 多表结构的缺点：编译程序将同时管理若干个符号表，增加了总体管理的工作量和复杂度。并且对各类标识符的共同属性如类型的管理必须设置重复的运行机制。

2、单表结构

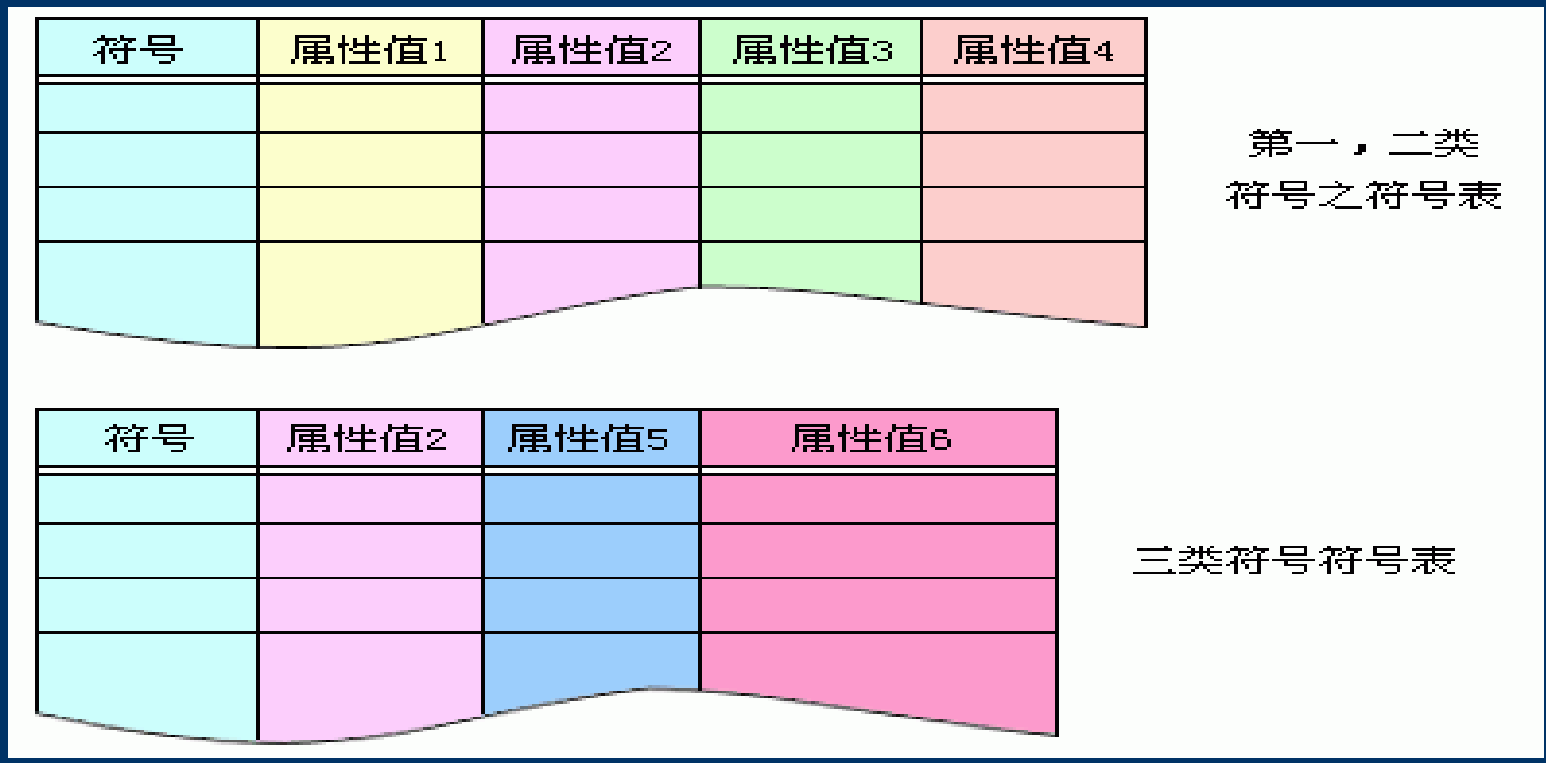
是将所有符号都组织在一张符号表中。

- 单表结构的优点：总体管理集中单一，且不同种类符号的共同属性可一致地管理和处理。
- 单表结构的缺点：由于符号属性的不同，为完整表达各类符号的全部属性必将出现不等长的表项，以及表项中属性位置交错重叠的复杂情况，极大地增加了符号表管理的复杂度。
- 可行的解决方法：把所有符号的可能属性作为符号表表项属性。
  - 忧点：有助于降低符号表管理的复杂度；
  - 缺点：对于某些类具体符号可能增加无用的属性空间，从而增加了空间开销。

3、折中方式

根据符号属性相似程度分类组织成若干张表，每张表中记录的符号都有比较多的相同属性。

按折中方式重新组织上例中的3类符号，可构成2张符号表，如图所示：



2.2 符号表表项的排列

- 研究符号表项排列的意义：编译过程中，每当遇到一个名字都要查找符号表。如果发现一个新名字，或者发现已有名字的新信息，则要修改符号表，填入新名字和新信息。符号表被频繁地用来建表、查表、填充和引用表的属性，因此表项的排列组织对系统运行的效率起着十分重要的作用。
- 由于符号表的建立是一次性的，符号表的查找是重复性的，因此查表的方法更为重要，对查表效率的要求决定了建表的方法。

1、线性组织

- 录入方式　规定符号表中表项按符号被扫描到的先后顺序录入；
- 线性组织的优点　没有空白项，存储空间效率高。
- 线性组织的缺点　运行效率低，特别当表项数目较大后效率就非常低。对于符号个数无法确定的情况下，无法确定符号表的总长度。但当事先能确定符号个数且个数不大（公认小于20）的情况下，使用线性组织是非常合适的。

2、排序组织

排序组织的符号表，就是在符号表中的表项按其符号的字符代码串（可以看成一个整数值）的值的大小从大到小（或从小到大）排列的。

关于排序表的表项建立及符号查找，通常采用“二分法”

3、散列组织

❖一个符号在散列表中的位置，由对该符号代码值进行某种函数操作（杂凑函数）所得到的函数值来确定的

❖假设选定杂凑函数fhash，对符号代码值杂凑运算之后得到杂凑值是Vhash，可表示为：

Vhash=fhash(<符号代码值>)

❖设符号的散列位置Lhash：

Lhash=mod (Vhash, N)

其中N为散列表的表长

优缺点：效率最高，但实现上较复杂而且要消耗一些额外的存储空间。

2.3 符号表查表技术

- 顺序查表法
- 折半查表法（二分法）
- 散列查表法（哈希表）

3. 符号表的局部化处理

- 源程序的局部化单位
- 标识符的作用域
- 局部化区的语义错误检查
- 标识符的处理原则
- 符号表的组织

3.1 源程序的局部化单位

程序中允许有声明的部分称为一个局部化单位,通常是一个子程序(函数、过程)或分程序：

Procedure P（……）……end

Function F（……）…… end

int G（……）{…… }

{……}

注：主程序也可以被看作为一个局部化单位。

Pascal过程声明嵌套的例子

```
1.procedure P( );
2.    var x,y:integer;
3.    procedure Q( );
4.          var x,z:real;
5.          begin
6.          .....x..z..y..
7.          end
8.  begin
9.    ......x.....y..
10.  end
```

局部化单位$_2$

局部化单位$_1$

带有分程序的C程序的例子

```
void main( )
{
    int a = 1;                                   B1
    int b = 1;
    {
        int b = 2;                               B2
        {
            int a = 3;                           B3
            printf("a = %d, b = %d\n",a, b);
        }
        {
            int b = 4;                           B4
            printf("a = %d, b = %d\n",a, b);
        }
    }
    printf("a = %d, b = %d\n", a, b);
}
```

a = 3, b = 2

a = 1, b = 4

a = 1, b = 1

3.2 标识符的作用域

- 一个局部化单位所建立的所有符号表表项称为该局部化单位的符号表。（局部化单位符号表的表项的存储位置不一定相邻）
- 一个局部化单位的符号表的有效范围是该局部化单位。
- 在程序的点P处，对它有效的符号表是包含P的那些嵌套外层的局部化单位的符号表。

- 在点P处遇使用性出现的标识符查符号表时，只能按着由内层到外层的次序（体现标识符的可视性作用域）查在此处有效的那些局部化单位的符号表，而不访问在此无效的那些符号表，这就要求编译程序合理组织符号表，使其能够在程序的每点判断出哪些局部化单位的符号表是有效的，这就是符号表局部化处理的本质。

3.3 局部化区的语义错误检查

- 变量的重复声明

  在一个程序的局部化区里，同一个标识符不能被声明两次。

- 标识符的使用有无声明

  强类型语言规定标识符必须先声明后使用，弱类型语言无要求。

3.4 标识符处理原则

- 遇到声明性标识符时，首先查找当前局部化单位的符号表中有无与此同名者，若有，则报告重复定义的语义错误；若无，则将该标识符的名字及其属性填入符号表 。
- 遇到使用性出现，按由内到外的次序依次查其嵌套外层的局部化单位的符号表，若有，则根据查得的属性判断对其使用是否合法；若无，则报告有使用而无声明的语义错误。

例1.(注:将fun1和fun2作为无参函数)

```
1.   int x=10;
2.   fun1( )
3.   {int x=20, z;
4.      ….
5.   z=x+10;
6.      ….
7.   }
8.   fun2( )
9.   {int y;
10.   y=x+30;
11.  }
```

x	itPtnr	varKind	dir	0	m	10
fun1		routKind	…	….		
fun2		routKind	…	..	….	….

x	itPtnr	varKind	dir	1	m	20
z	itPtnr	varKind	dir	1	m+1	

y	itPtnr	varKind	dir	1	m	

例2：(建立符号表表项时检查有无重复定义)

……

```
1.   int x=10;
2.   fun1( )
3.   {
4.     int x=20, z;
5.     int x=30;
6.       ….
7.   z=x+10;
8.       ….
9.   }
10.  fun2( )
11.  {int x=30;

12.   y=x+100;
13.  }
```

x	itPtnr	varKind	dir	0	m	10
fun1		routKind	…	….		
fun2		routKind	…	..	….	….

x	itPtnr	varKind	dir	1	m	20
z	itPtnr	varKind	dir	1	m+1	

x	itPtnr	varKind	dir	1	m	30

3.5 符号表的组织

从符号表的组织方式上看，可分为两大类：一类是局部式，一类是全局式。

- 全局式符号表则把整个程序的符号表作为一个表，即建表和查表单位是整个符号表；
- 局部式符号表是指把每个局部化单位的符号表作为一个独立的表来处理，即把每个局部符号表作为建表和查表单位。

一、全局式线性组织的符号表的局部化处理思想

- 每当进入一个局部化区，记住本层符号表的始地址；
- 每当遇到定义性标识符时，构造其语义信息并查本层符号表，若查到同名标识符，则表示有错，否则往符号表里填写标识符及其语义信息。
- 每当遇到使用性标识符时，查整个符号表（从后往前），若查不到同名标识符，则表示有错（因为我们假定声明在前），否则找出相应语义信息并将它传给有关部分。
- 每当结束一个局部化区时，“删除” 本层符号表。

- 用一个Scope栈来实现标识符的嵌套作用域，Scope[ℓ]指向ℓ层符号表始地址；
- “删除”本层符号表的方法一般有两种：

1. 真删除法:当退出一个局部化单位时，删除掉该局部化单位的符号表；

ℓ表示层数计数器，s表示符号表的最后一项的地址，则删除式的局部化算法如下：

s := Scope[ℓ]-1

ℓ := ℓ -1 ;

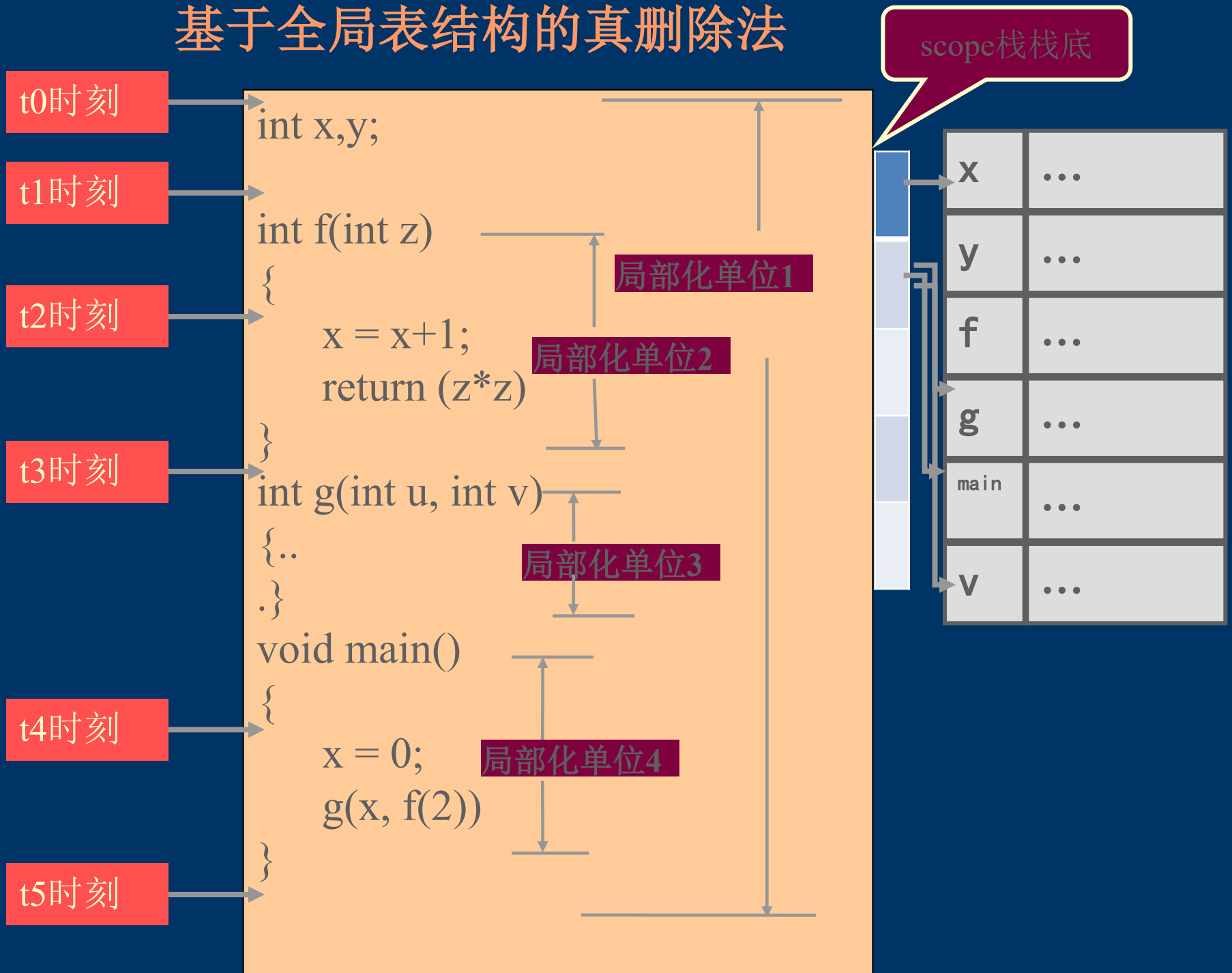
2. 驻留法:不删除表，但采取一定措施不去查那些已无效的符号表部分。算法如下：

每当退出一个局部化区时，将一个跳转项添加到符号表中：

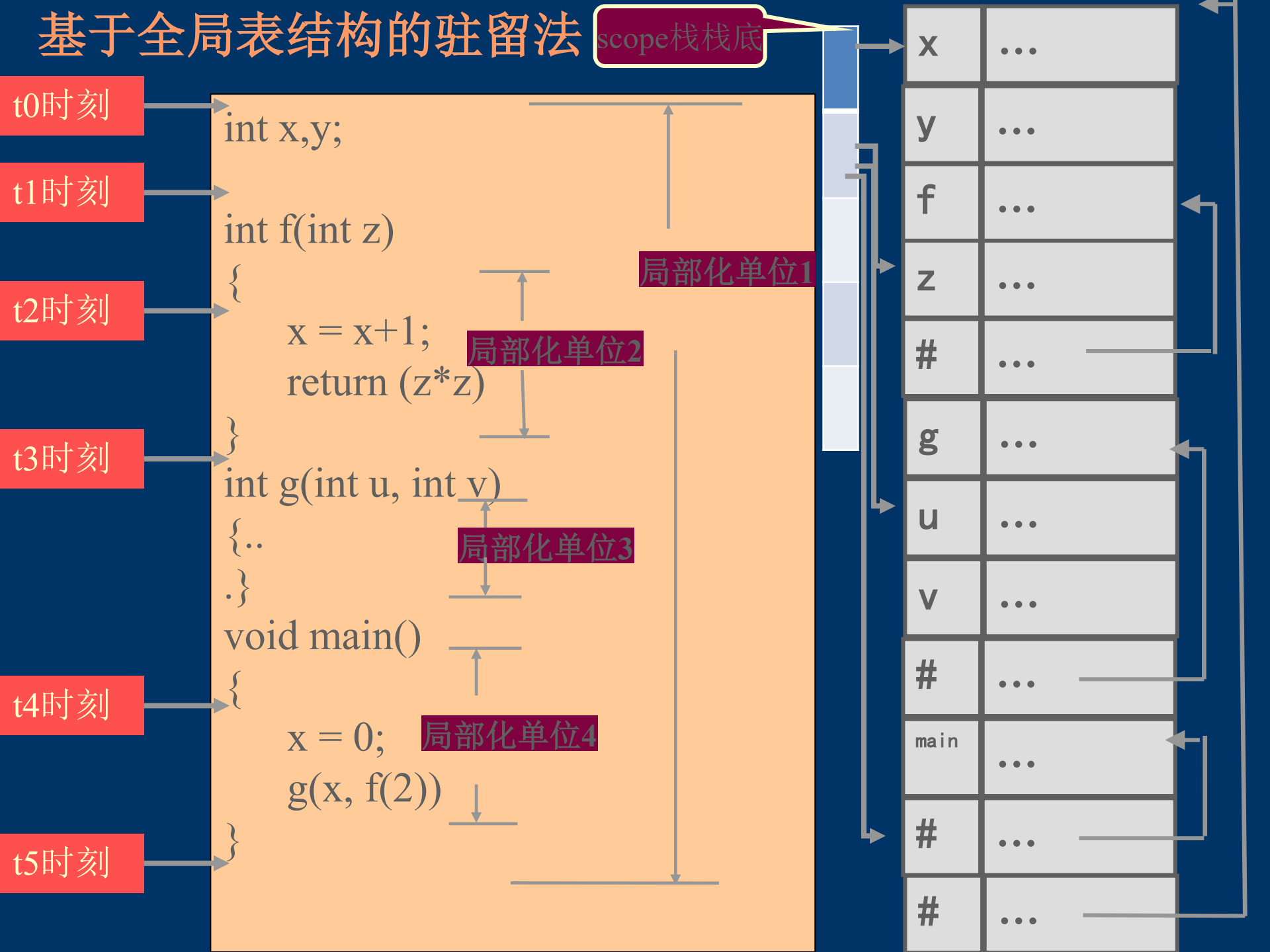
s := s +1 ;

SymbTab[s]:=(# , Scope(ℓ)-1)

基于全局表结构的真删除法



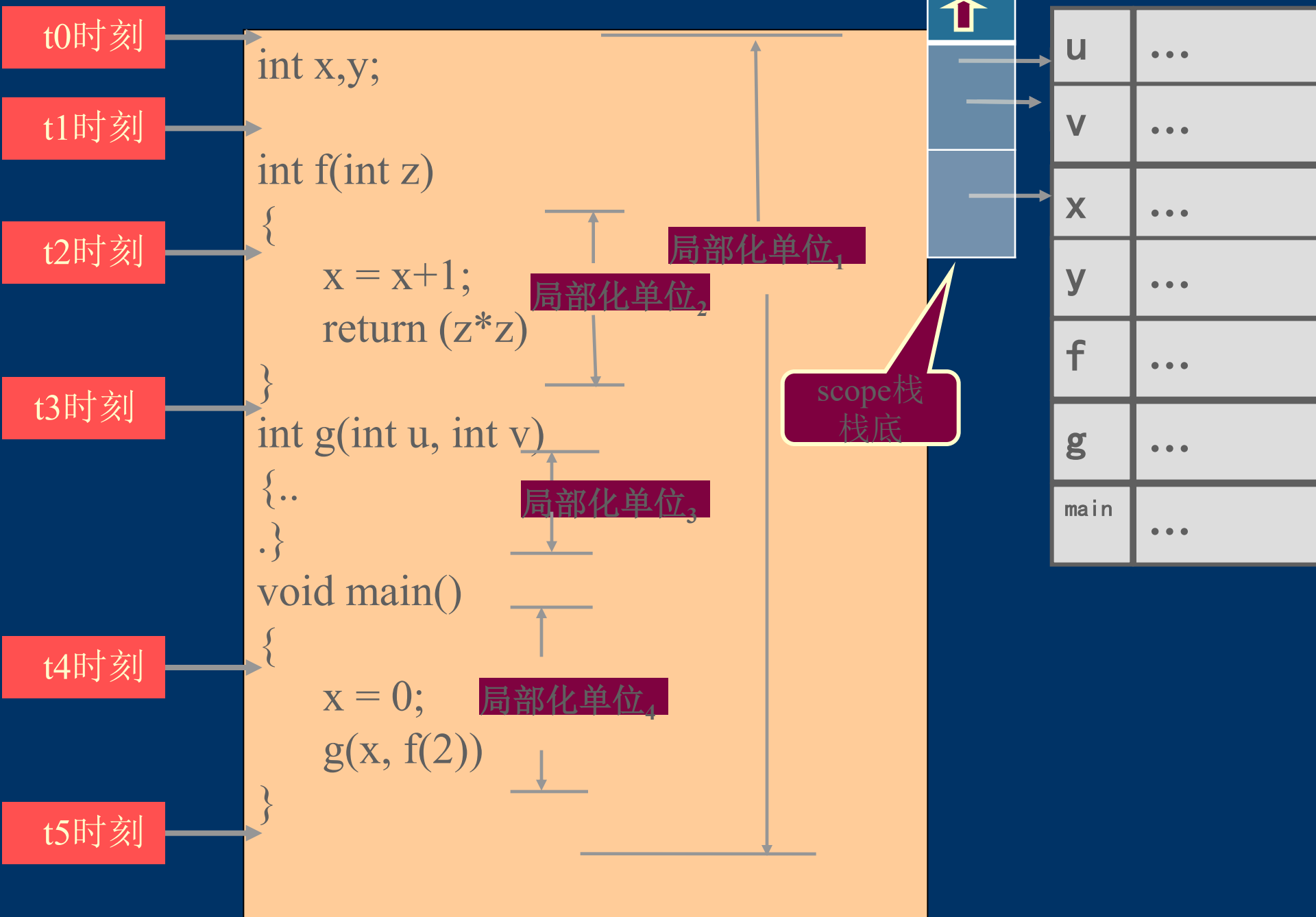
基于全局表结构的驻留法



二、局部式线性组织的符号表的局部化处理思想

- 在不同时刻，不是所有的局部符号表都有效。为了标记在某一时刻，哪些符号表是有效的，用到一个辅助数据结构Scope栈(栈便于体现标识符的嵌套作用域原则)。
- Scope栈用于存放当前有效的局部化单位的局部符号表的始地址。每当进入一个新的局部化区，就将它的符号表始地址存入Scope栈，当退出一个局部化区时，再将Scope栈的栈顶元素弹出。
- 删除法、驻留法之分都是基于全局表的结构，而对局部表只有删除法。

局部表结构的管理


t0时刻
t1时刻
t2时刻
t3时刻
t4时刻
t5时刻
int x,y;
int f(int z)
{
x = x+1;
return (z*z)
}
int g(int u, int v)
{..
.}
void main()
{
x = 0;
g(x, f(2))
}
局部化单位1
局部化单位2
局部化单位3
局部化单位4
scope栈
栈底
u
v
x
y
f
g
main
…