第6章 符号表

第五章所讲的语法分析在整个编译器里处于一个承上启下的作用。所谓“承上”的“上”，是指前端的词法分析，将源程序中的单词识别并翻译为Token序列。语法分析作为过渡阶段，基于Token序列，得到句子的语法树结构。所谓“启下”的“下”，则是指语义分析，根据语法的定义进一步生成中间语言，甚至目标代码。

“语义”在语言学界没有统一的概念。从系统开发角度来讲，在计算机编译程序的过程中，语义是指在基于语法分析结果，执行语义动作。比如，假设目标是将程序翻译为目标代码，语义动作就是生成目标代码的动作；假设目标是统计源代码里Token的数量，语义就是统计数量。换句话说，语义是由开发者（编译器设计者）来定义的。接下来需要对语义进行表示、访问，其中一个非常重要的手段就是符号表，即第六章的内容。

第六章的内容包括符号表的地位和作用、符号表的组织与管理、符号表的结构设计、符号表的构造过程，以及运行时刻存储分配。

6.1 符号表的地位和作用

6.1.1 符号表的定义

符号表是**标识符**的**动态语义词典**，属于编译中语义分析的知识库。它的核心目的是组织标识符的查询。其中，符号表面向的对象是标识符，变量、函数名都是标识符。

动态语义词典的定义如下：

- **动态**：指标识符表随着编译过程的执行，会发生变化。（也有一些语言在受限的情况下已经预编译好，可以直接运行，则使用静态标识符表）。

  例如：递归调用。递归的深度不能预知，故递归操作本身是动态的，跟递归相关的标识符内容也是动态的。

- **语义**：指描述标识符所有的信息。

  例如：标识符可能是整型变量或浮点型变量，整型或浮点型就是标识符的一种语义。

- **词典**：指编译器查询的依据，符号表的目的是让编译器能够快速准确地找到想要查询的标准。

6.1.2 标识符的四种语义信息

1. **名字**：标识符源码，用作查询关键字。即一个符号，能用来指示标识符，用于唯一标识标识符的身份，便于查询。
2. **类型**：该标识符的数据类型及其相关信息。
3. **种类**：该标识符在源程序中的语义角色。
4. **地址**：与值单元相关的一些信息。地址保存了标识符所对应变量的内容，“值单元”就是存储的地方。

6.1.3 符号表的基本功能

- **定义和重定义检查**。

  定义：可以修改符号表去定义一个变量，例如定义“int A”，要将 A 填写到符号表里，否则后面不能使用 A。

  重定义检查：例如在 C 语言的一个函数里面，不能有两个“int A”，否则会报错“重定义”，报错的原因是在查符号表时发现 A 已经被定义过了。

- **类型匹配校验**

  脚本语言分为强数据类型和弱数据类型。C 语言、C++ 等语言是强数据类型，变量类型要先定义好，不能把一个字符串类型赋值给一个整型，编译器在查询符号表的过程中，检查到整型和字符串不匹配，会报错。而 Python 是弱数据类型，即一个变量不需要定义类型，可以任意赋值，把字符串类型赋值给整型也没有问题。

- **数据的越界和溢出检查**

  符号表会限制数组的界限，例如生成了一个维度是 10 的静态数组，如果要访问第 11 号元素，编译器将会报错“访问非法”。

- **值单元存储分配信息**

  符号表会定义在哪个位置能找到这个元素。

- **函数、过程的参数传递与校验**

  举个例子，怎么知道两个函数匹不匹配？调用函数的时候写的方式是否合法？以及类似的问题，都可以通过符号表来解决。

符号表实际上是一个逻辑上的概念，并不是物理上制作一个符号表。它贯穿了编译过程中的很多部分，从某种意义上讲，没有符号表，编译器就缺乏了存储的知识，因为编译程序中标识符的定义全部保存在符号表里。

6.2 符号表的组织与管理

6.2.1 符号表的工作原理

符号表的工作有两种操作，一是写，二是读。符号表存储的是标识符对象语义信息，

- 在遇到声明语句，即定义性标识符时，执行写操作，顺着 Token 指针，将语义信息写入表中，如图6.1。

(i ,)⟶ 该标识符符号表项

图 6.1: 符号表操作

可以理解为 (i ,)→ 是一个 Token，前面是语法单元，后面是语义信息。将这个 Token 填到符号表里，从而通过 Token 指针指向的内容，记录标识符的相关语义信息。

- 在遇到操作语句，即应用性标识符时，执行读操作，顺着 Token 指针，读符号表的相应项。例如定义 int A，有 C=A+B，把 A 加 B 的值赋值给 C。目标是查符号表找到 A 符号所在的位置，用 A 所在的位置协助访问 A 的信息。所以同样要把 Token 的指针指向 A 所对应的符号表这一项。

6.2.2 符号表的查询、访问方式

符号表存储的是所有与用户定义相关的语义信息，在整个编译过程中，对符号表的读写非常频繁，符号表采用的数据结构好坏将直接影响最后编译器的性能。我们学过的数据结构线性表、顺序表、索引表和散列表，都可以采用。具体数据结构的使用不是符号表要介绍的主要内容，本章主要介绍的是符号表承载的语义信息，利用什么样的结构承载这些语义信息。因此本小节提出符号表的查询、访问方式，是为了引起大家注意，在具体使用时还需要具体问题具体分析。

6.2.3 符号表的维护、管理方式

不同的程序定义符号表的方式不同，在本书中，符号表可以相对片面的理解为：一个源文件有若干个函数组成，每个函数对应一个符号表，此外还有一个全局的公用符号表。

符号表的组织方式或管理方式一般是针对语言来说的，实际实现时，往往取决于所属语言的程序结构。就 C 语言来说，可以在内存设置一定长度的符号表区，对应内存的一段数据，并建立适当的索引机制，访问相应的符号表，符号表区的形式如图6.2所示：

当前函数的所有符号信息，都保存在“现行函数符号表”。如果这个函数被其他函数调用，它上一层函数的符号信息就保存在“…”里面，以此类推，第一个函数叫 FUNCTION 1，保存在“FUNCTION 1 符号表”。所有符号表的最外层是公用符号表，称为全局符号表。

符号表实际上就是按照这种层次化的方式去组织，把下面部分称为局部符号表区，把上面部分称为全局符号表区。可以采用不同的索引机制去索引，保证能访问到所有符号表。

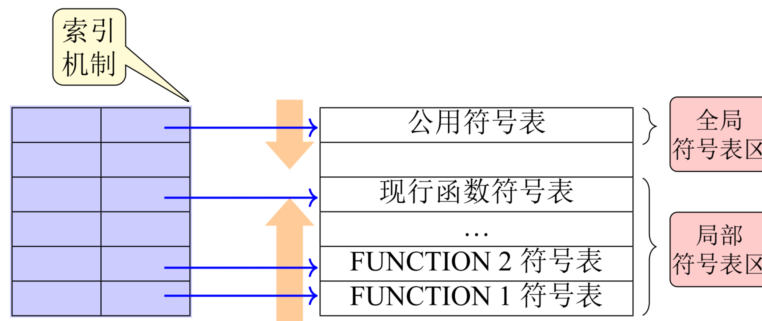

图 6.2: 符号表区形式

6.3 符号表的结构设计

例 6.1 有下列过程函数：

FUNCTION exp (x : REAL; VAR y : INTEGER) : REAL;

CONST pai = 3.14;

TYPE arr = ARRAY[1..5, 1..10] OF INTEGER;

VAR a: arr; b, a: real;

BEGIN …; a[2,5] := 100; b := z + 6; …END;

1. 程序说明：

   这是一段简单的 Pascal 语言程序，定义了一个名为 exp，返回值为实型的函数，包含两个形参 x 和 y，x 是实数（浮点数）类型，是**赋值形参**，y 是整型，是**换名形参**，由关键字 VAR 声明。函数的代码段从 BEGIN 开始，一直到 END。在函数声明和代码段中间的内容，是一系列声明，包括常量标识符 pai=3.14，类型标识符定义整型数组 arr，两个变量标识符 a 和 b，VAR 是定义变量的关键字，后面的是变量名。

2. 符号表要回答的问题：

   - 需要进符号表的标识符

     exp (函数，附带信息：类型、参数情况和入口地址⋯), pai (常量), arr (类型), a (下标变量), b (简单变量), …

   - 怎样检查出：a 重定义、z 无定义以及下标变量？a[2,5] 的值地址在何处？…

   ※ 符号表的体系结构设计

   由于标识符的种类不同，导致语义属性也不尽相同。下面提供一个符号表的体系结构如图6.3，观察符号表是怎样组织的：

   符号表 (SYNBL) 由词法分析里学到的 Token，留有一个指针指向它。符号表有四个属性，可以看成是一个表格。

   - 名字 (NAME)：符号表的名字。

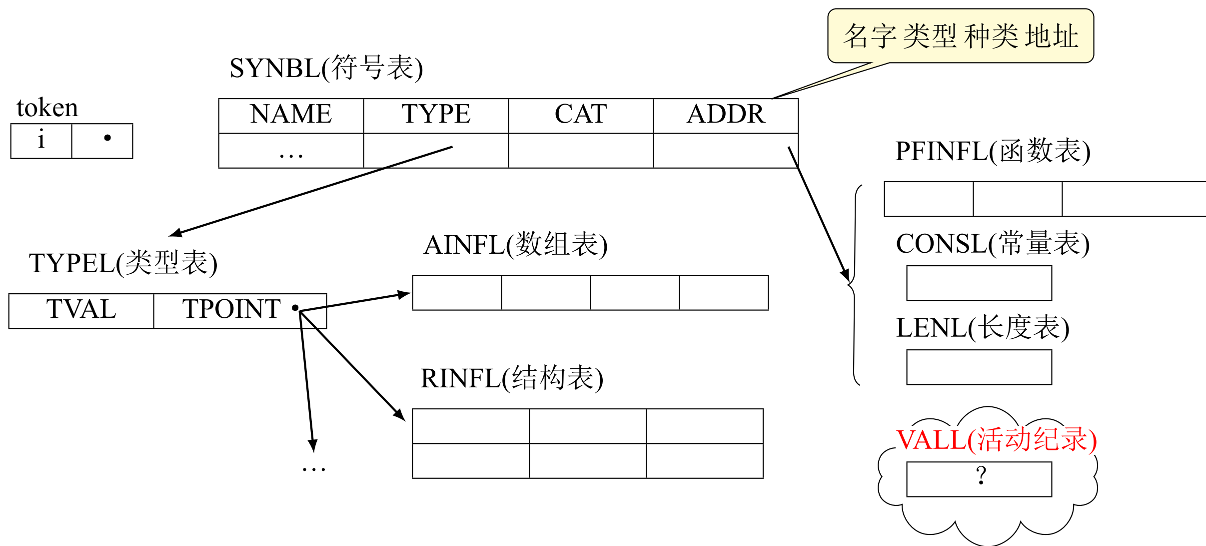

图 6.3: 符号表体系结构

- 类型 (TYPE)：符号的数学类型。

  int、float、char 都是数学类型，类型仍然是一个指针，指向一个叫“类型表”的结构。也就是说，符号表是使用一个额外的表来描述类型的。类型表既能表示常见的数据类型，还能记录其它的类型（通过指针指向其他表）。例如“数组表 (ALNFL)”，用来描述数组，在 Pascal 语言里，数组是一个常见的类型。数组大小，上界和下界，数组的每个元素是什么，都可以通过数组表来定义。此外，类型表还可以指向“结构表 (RINFL)”，C 语言里的 structure 类型即结构体类型，也可以用类型表来描述。类型表可以指向丰富的类型，所有的数据类型都可以通过类型表来定义。

- 种类 (CAT)：变量的种类，按值传递（变量）或按地址传递（函数名）。

- 地址 (ADDR)：值单元的描述，可简单理解为标识符所存在的物理地址。

  地址指向的是内容，例如假设种类是个函数，这时地址指向的是函数表 (PFINFL)，来描述函数的信息；假设定义的是一个常量 pai，指向的就是常量表 (CONSL)；还可以指向长度表 (LENL)，记录这个类型多大，占几个字节；最重要的，可能会指向**活动记录 (VALL)**，活动记录和函数的执行是同时进行的，函数执行过程中会生成相应的活动记录。所有的变量，真正保存的地址就是活动记录里关于这个变量描述的内容，换句话说，变量的物理存储保存在活动记录里。

下面分别具体介绍符号表各个部分的内容。

6.3.1 符号表总表 (SYNBL)

总表结构包括四项内容：

NAME	TYPE	CAT	ADDR

图 6.4: 符号表总表 (SYNBL) 结构

- NAME (名字)：标识符源码（或内部码）。

- TYPE (类型)：指针，指向类型表相应项。
- CAT (种类)：种类编码。

  f (函数)，c (常量)，t (类型)，d (域名)，v (常规变量)，vn (换名形参，即地址传递，只需拷贝存储地址)，vf (赋值形参，即值传递，需要将实参拷贝一份到形参存储位置)。
- ADDR (地址)：指针，根据标识符的种类不同，分别指向函数表 PFINFL，常数表 CONSL，长度表 LENL，活动记录 VALL，…

下面分别具体介绍符号表各个部分的内容。

6.3.2 类型表 (TYPEL)

类型表结构包括两项内容：

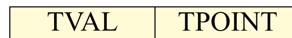

图 6.5: 符号表类型表 (TYPEL) 结构

- TVAL (类码)：表示类型的编码。

  1. 静态数据类型：包括 i (整型)、r (实数型/浮点型)、c (字符型)、b (布尔型)，以及编译器简单预定义好的其他静态数据类型。

  2. 复杂数据类型，例如 a (数组型) 或 d (结构体型)，结构体有几个域，每个域是什么，是程序员自己写的，编译器不知道，只能在看到源程序的时候去分析它。从编译的角度来看，它是一个不确定的动态结构，但从执行的角度来看，它又是确定的（这里不需要用静态和动态来区分）。
- TPOINT (指针)：根据数据类型不同，指向不同的信息表项。指针进一步描述数据的类型。

  1. 基本数据类型 (i, r, c, b) ——nul (空指针)。数据类型是预定义好的，不需要进一步描述，指针部分不需要指向任何单元。

  2. 数组类型 (a) ——指向数组表。编译器不知道具体地址，需要根据用户输入的源程序才知道。如果定义了一个数组，指针可能要指向的是数组表。

  3. 结构类型 (d) ——指向结构表。同理数组类型。

6.3.3 数组表 (AINFL)

数组表结构包括四项内容：

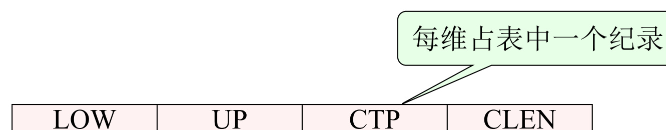

图 6.6: 符号表数组表 (AINFL) 结构

- LOW (数组的下界)：C 语言自动设置为 0。
- UP (数组的上界)：用户定义的最大访问范围。
- CTP (成分类型指针)：指针，指向该维数组成分类型（在类型表中的信息）。
- CLEN (成分类型长度)：成分类型的数据所占值单元的个数（假定：值单元个数依字长为单位计算）。

6.3.4 结构表 (RINFL)

一个结构体会包括若干项，称之为域，每个域占表中的一个记录，指示结构体的每一项都是什么类型。结构表结构包括三项内容：

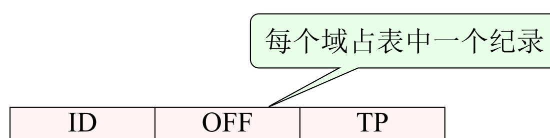

图 6.7: 符号表结构表 (RINFL) 结构

- ID (结构的域名)：每个域的名字。
- OFF (区距)：是 id_k 的值单元首地址相对于所在记录值区区头位置。

  计算公式如下：

$$\begin{aligned} \text{约定:}\quad & off_1 = 0, \\ & off_2 = off_1 + \text{LEN}(tp_1), \ldots\ldots \\ & off_n = off_{n-1} + \text{LEN}(tp_{n-1}) \end{aligned}$$

  公式说明：第 1 个域的偏移是 0，即区距是 0。第 2 个域的区距将第 1 个域所对应的区距加上第 1 个域所对应变量的长度。以此类推，第 n 个域的区距就是域 n-1 的起始地址再加上第 n-1 个域对应变量的长度，称为第 n 个域的偏移。

- TP (域成分类型指针)：指针，指向 id_k 域成分类型（在类型表中的信息）。

  思考：为什么类型要用一个复杂的类型表去表示？

  1. 重用：定义数组类型之后，这个类型可以定义很多变量，不需要每定义一次变量就把类型重新定义一遍。

  2. 复杂数据类型，需要对类型本身进行复杂地描述。如果全是预定义类型，一个编码就足够，但是复杂的数据类型用简单的类型无法描述，在设计数据结构时，就不能统一用一个表来做，要分出一个数据类型来单独描述。这也是一种编程的常规的习惯，由于描述的对象很复杂，不是一个静态的东西，因此需要一个动态的内容去描述。

6.3.5 函数表 (PFINFL)——过程或函数语义信息

- LEVEL (层次号)：该过函静态层次嵌套号，用来表示函数的位置（并非递归的层次）。

LEVEL	OFF	FN	ENTRY	PARAM	…

图 6.8: 符号表函数表 (PFINFL) 结构

- OFF (区距)：该过函自身数据区起始单元相对该过函值区区头位置。
- FN (参数个数)：该过函的形式参数的个数（可以没有）。
- PARAM (参数表)：指针，指向形参表（描述每个参数的内容）。形参是函数非常重要的语义信息，且数量可能较多，因此构建形参表，并以指针形式由 PARAM 指向形参表。
- ENTRY (入口地址)：该函数目标程序首地址（运行时填写）。

对 LEVEL、OFF 进行说明：函数包括数据对象以及对对象进行的操作，这两部分内容都需要载入内存，操作部分不属于本章讨论范围。下面针对数据对象部分，进一步讨论。

编译器处理一个函数的所有数据对象时，载入内存一定是连续存储的。设计的函数是静态的，函数被载入内存一次就产生一次活动，在这次活动中处理的数据对象所在区域，被存储在活动记录中。活动记录会存储形参变量、局部变量、临时变量。

以 $x = x + 10$ 为例，编译器将该语句分解为计算和赋值两步，开始计算得到 $x + 10$ 的结果，即为临时变量，存放中间结果。除变量数据之外，活动记录还需要记录管理数据，如函数嵌套调用时的返回地址存储，断点保存等。管理数据从区头开始存放，占据一定空间，变量数据区起始单元相对区头的位置，记为区距 OFF。层次号 LEVEL 用于处理变量作用域问题，以 C 语言程序段为例，主函数中定义了 x, y，中间某一段过程中调用的 x 来自该段程序中定义的 x，调用的 y 来自前面定义的 y，我们可以通过层次号区分不同作用域的变量。如图所示，将前面的 int x, y; 所在作用域定为 L 层，则中间段就属于 L+1 层，从逻辑上来说，可以将 x 进行区分，开始定义的 x 是第 L 层的，中间段的 x 是 L+1 层的。在 C 语言中，函数不可嵌套定义，而在 pascal 语言中，可以在函数内部定义子函数，在这种情况下，LEVEL 的标识尤为重要。LEVEL 表示该过程或函数静态层次嵌套号，根据写好的程序进行判断，不随运行时刻而改变。

6.3.6 其他表 (…)

- 常量表 (CONSL)：存放相应常量的初值，仅有一个域。对于字符常量、字符串常量、实型常量、整型常量等这些不同类型的常量，分别列表。
- 长度表 (LENL)：存放相应数据类型所占值单元个数，仅有一个域。
- 活动记录表 (VALL)：一个函数（或过程）虚拟的值单元存储分配表；是函数（或过程）所定义的数据，在运行时刻的内存存储映像。

6.4 符号表的构造过程示例

例 6.2 根据函数构造符号表:

FUNCTION exp (x : REAL; VAR y : INTEGER) : REAL;

CONST pai = 3.14;

TYPE arr = ARRAY[1..5, 1..10] OF INTEGER;

VAR a: arr; b, a: real;

BEGIN …; a[2,5] := 100; b := z + 6; …END;

填表过程如图6.9：

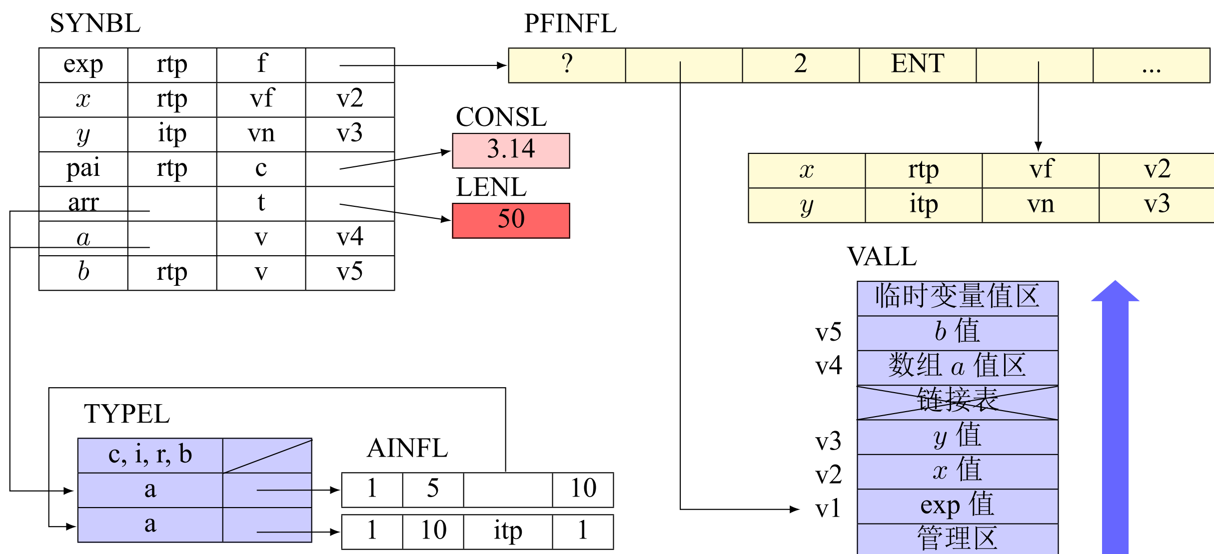

图 6.9: 符号表填写过程

1. exp (函数)

- 填符号表 SYNBL：NAME 域填函数名字 exp，TYP 域填函数的返回值类型 rtp（实数型），CAT 域填写函数的种类 f（函数），ADDR 域指向函数表。

  说明：rtp 是指针，指向 TYPEL 中的 r 项，此处是为了节省画图空间，简化为这样。

- 填函数表 PFINFL：没有完整的程序不知道函数表的层次号，编译时才知道，所以暂时不填 LEVEL 域，OFF 域指向 exp 返回值所存的内容，即 v1 的地址，FN 域填函数的变量个数 2，ENTRY 域填函数入口地址 ENT，PARAM 域指向形参表，暂时不填。

2. x (变量)

- 填符号表 SYNBL：函数有两个变量，在符号表中继续填 x 的相关内容，NAME 域填变量名字 x，TYP 域填变量类型 rtp（实数型，指向类型表的 r 项），CAT 域填写变量种类 vf（赋值形参，按值传递的参数）。x 值存在活动记录 VALL 中 exp 值的下一个位置（VALL 中的地址从下至上依次增大），x 的起始地址记为 v2，将其填在 ADDR 域内（实际上 v2 是一个指针，指向 VALL 中 v2 的内容，此处为了记录简洁，没有画出指针）。

- 填形参表 PARAM：根据 x 的名字、类型、种类、地址，填写 NAME、TYP、CAT、ADDR 四项内容。

思考：为什么在符号表中填完 x，还要在参数表中再填一遍？两项内容是完全相同的。

3. y (变量)

- 填符号表 SYNBL：在符号表中继续填 y 的相关内容，NAME 域填变量名字 y，TYP 域填变量类型 itp（整型，指向类型表的 i 项），CAT 域填写变量种类 vn（换名形参，按地址传递的参数）。y 值存在活动记录 VALL 中 x 值的下一个位置，起始地址记为 v3，将其填在 ADDR 域内。

  思考：y 是按地址传递的参数，那么 VALL 中变量 y 中存的内容是什么？是 y 所指向的值还是 y 的地址？

  答：保存的是 y 的地址，这里有一个重要的概念——解引用。按地址传递的好处是可以直接修改原始变量的内容；坏处是多了一次解引用，解释地址指向的内容，多了一次跳转，导致速度变慢。
- 填形参表 PARAM：根据 y 的名字、类型、种类、地址，填写 NAME、TYP、CAT、ADDR 四项内容。

4. pai (常量)

- 填符号表 SYNBL：在符号表中继续填 pai 的相关内容，NAME 域填常量名字 pai，TYP 域填常量类型 rtp（实数型，指向类型表的 r 项），CAT 域填写常量种类 c。ADDR 域指向常量表 CONSL。
- 填常量表 CONSL：填入 3.14。

5. arr (数组类型)

- 填符号表 SYNBL：在符号表中继续填 arr 的相关内容，NAME 域填名字 arr，TYP 域指向数组的定义——类型表。
- 填类型表 TYPEL：没有数组的定义，要新增。TVAL 域填类码 a，TPOINT 域指向数组表。
- 填数组表 ALNFL：LOW 域填数组的下界 1（Pascal 语言从 1 开始），UP 域填数组的上界 5，CTP 域指向类型表，表示每个单元的类型，仍然是数组类型 a（嵌套）。由于该数组类型没有被定义过，所以类型表再次新增一行。CLEN 域填值单元的长度 10（填完 ALNFL 之后得到，根据数组范围可计算整个数组长度为 50）。
- 填类型表 TYPEL：TVAL 域填类码 a（与上一个数组类型的定义不同），TPOINT 域指向数组表。
- 填数组表 ALNFL：LOW 域填数组的下界 1，UP 域填数组的上界 10，CTP 域指向类型表，此处填 itp（整型，指向类型表的 i 项），CLEN 域填值单元的长度 1。此时，反推上一个数组单元的长度为 10。
- 填符号表 SYNBL：CAT 域填写数组种类 t（类型，可以用作定义其他变量的数据类型）。ADDR 域指向长度表。

- 填长度表 CONSL：填入 50。

6. a (变量)

   填符号表 SYNBL：在符号表中继续填 a 的相关内容，NAME 域填变量名字 a，TYP 域指向类型表中 arr 定义的数组类型 a，CAT 域填写变量种类 v。a 值存在活动记录 VALL 中，起始地址记为 v4，将其填在 ADDR 域内（占 50 个单元）。

   VALL 中链接表的作用：静态数据类型直接放在底部，复杂数据类型编译器不知道内容，需要指令分析所以放在链接表上面。

7. b (变量)

   填符号表 SYNBL：在符号表中继续填 b 的相关内容，NAME 域填变量名字 b，TYP 域填变量类型 rtp（实数型，指向类型表的 r 项），CAT 域填写变量种类 v。b 值存在活动记录 VALL 中，起始地址记为 v5，将其填在 ADDR 域内。

※ **强调**：如果种类是类型，指向的是长度表，因为要知道这个类型占多少空间；如果种类是变量，只需要知道它的物理地址，指向地址。

例 6.3 根据类型说明填写符号表

TYPE arr = ARRAY [1..10] 0F ARRAY [1..5] 0F INTEGER;

设：实型占 8 个存储单元，整型占 4 个单元，布尔型和字符型占 1 个单元。

i，r，c，b 是不同的项，它们的指针都为空。该例主要介绍二维数组如何存储，填表过程如图6.10:

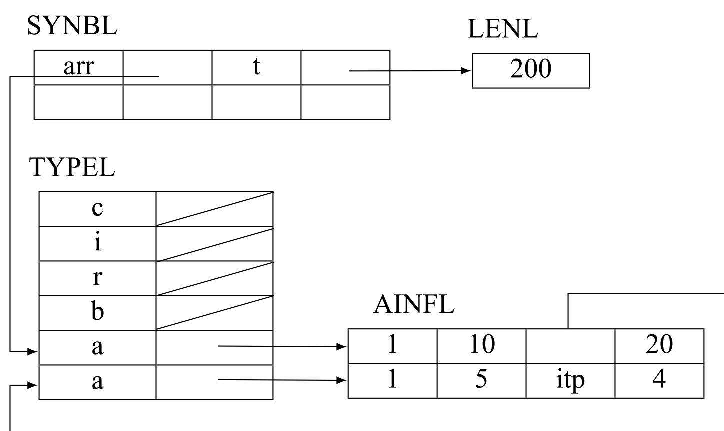

图 6.10: 符号表填写过程

- 填符号表 SYNBL：NAME 域填数组名字 arr，TYP 域指向类型表，arr 是一个二维数组（数组嵌套数组）。
- 填类型表 TYPEL：TVAL 域填类码 a，TPOINT 域指向数组表。
- 填数组表 ALNFL：LOW 域填数组的下界 1，UP 域填数组的上界 10，CTP 域指向类型表，仍然是数组类型 a（嵌套），类型表再次新增一行。CLEN 域填值单元的长度 20（填完 ALNFL 之后得到，根据数组范围可计算整个数组长度为 200）。

- 填类型表 TYPEL：TVAL 域填新的类码 a，TPOINT 域指向数组表。
- 填数组表 ALNFL：LOW 域填数组的下界 1，UP 域填数组的上界 5，CTP 域填 itp（整型，指向类型表的 i 项），CLEN 域填值单元的长度 4。此时，反推上一个数组单元的长度为 20。
- 填符号表 SYNBL：CAT 域填写数组种类 t（类型，可以用作定义其他变量的数据类型）。ADDR 域指向长度表。
- 填长度表 CONSL：填入 200。

例 6.4 根据类型说明填写符号表

TYPE rec = RECORD

u: INTEGER;

v: ARRAY[1..10] OF BOOLEAN;

r: RECORD x, y : REAL END

END;

设：实型占 8 个存储单元，整型占 4 个单元，布尔型和字符型占 1 个单元。

一个记录可以理解为结构体，由 3 部分组成：第一个域 u ——整型；第二个域 v ——10 维的布尔类型数组；第三个域 r ——结构体，由两个域 x 和 y 构成，都是实数类型。

复杂的结构在符号表里的表示过程如图6.11：

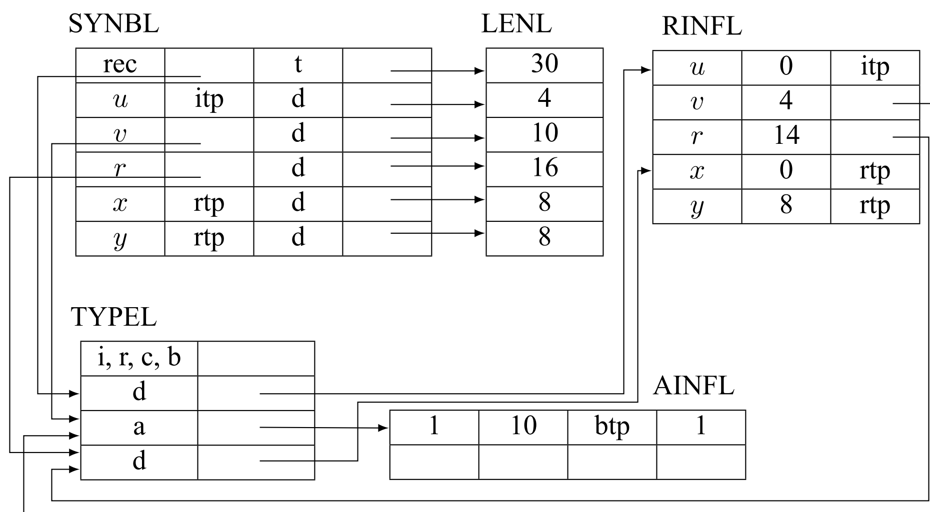

图 6.11: 符号表填写过程

- 填符号表 SYNBL：NAME 域填记录名字 rec，TYP 域指向类型表，新增一个结构体类型 d（跳转到 TYPEL），CAT 域填写结构体种类 t，ADDR 域指向长度表。
- 填类型表 TYPEL：若 TVAL 域填类码 d，TPOINT 域指向结构表（跳转到 RLNFL）；若 TVAL 域填类码 a，TPOINT 域指向数组表（跳转到 ALNFL）。
- 填结构表 RLNFL：

1. 第一条记录：ID 域填第一个结构体的第一个域名 u，OFF 域填区距 0（因为是第一个域），TP 域填 itp（整型，指向类型表的 i 项）。同时在符号表中描述 u，NAME 域填 u，TYP 域填 itp，CAT 域填结构类型 d，ADDR 域指向的长度表填 4。

2. 第二条记录：ID 域填第一个结构体的第二个域名 v，OFF 域填区距 4（第一个域的区距 + 第一个域的长度），TP 域指向类型表，新增一个数组类型 a（跳转到 TYPEL）。同时在符号表中描述 v，NAME 域填 v，TYP 域指向类型表中新添加的数组类型 a，CAT 域填结构类型 d，ADDR 域指向的长度表填 10。

3. 第三条记录：ID 域填第一个结构体的第三个域名 r（仍是一个结构体），OFF 域填区距 14（第二个域的区距 + 第二个域的长度），TP 域指向类型表，新增一个结构类型 d（跳转到 TYPEL）。同时在符号表中描述 r，NAME 域填 r，TYP 域指向类型表中新添加的数组类型 d，CAT 域填结构类型 d，ADDR 域指向长度表。

4. 第四条记录：ID 域填第二个结构体的第一个域名 x，OFF 域填区距 0（因为 x 是新纪录的第一个域），TP 域填 rtp（实数型，指向类型表的 r 项）。同时在符号表中描述 x，NAME 域填 x，TYP 域填 rtp，CAT 域填结构类型 d，ADDR 域指向的长度表填 8。

5. 第五条记录：ID 域填第二个结构体的第一个域名 y，OFF 域填区距 8（第一个域的区距 + 第一个域的长度），TP 域填 rtp（实数型，指向类型表的 r 项）。同时在符号表中描述 y，NAME 域填 y，TYP 域填 rtp，CAT 域填结构类型 d，ADDR 域指向的长度表填 8。

- 填数组表 ALNFL：LOW 域填数组的下界 1，UP 域填数组的上界 10，CTP 域填 btp（布尔型，指向类型表的 b 项），CLEN 域填值单元的长度 1。此时，反推数组的长度为 10。
- 填长度表 CONSL：完善 r 的长度 $= 16(x + y)$，rec 的长度 $= 30(u + v + r)$。

6.5 运行时刻存储分配

本节介绍**标识符**标识符变量的**地址分配**与对它们的访问，主要讲解类码为 v 的标识符地址分配问题。

6.5.1 标识符值单元分配

一个变量保存的位置，在 C++ 语言里用 new 就可以得到，但是需要注意，程序运行是有环境的，不能单独写一个 new 语句来运行，要放在一段程序里，具体来说是放在一个函数里去执行。这个问题衍生出一个思考，分配 new 的时候，是在一个什么样的条件下去分配？在函数内部可以做 new 操作，外部也行，因为 main 函数也是一个函数。

对于值单元的分配，一般有两种策略。

1. 静态分配:

   在**编译阶段**即可完成真实的地址分配。在编译时对所有数据对象分配固定的存储单元，且在运行时始终保持不变。

- 优点：程序编译之后、执行之前，就知道每个变量所存的位置和大小。静态分配不需要计算变量的存储位置，可以提高程序执行的效率。
- 缺点：如程序递归调用时，不能做到静态分配。递归的层数由用户输入的变量决定，递归函数每个变量的存储位置不能确定，这种情况下采用动态分配。

2. 动态分配:

   在**运行时刻**进行的值单元分配，即动态地决定变量所存储的位置和大小，在编译时只能进行相对地址分配。

   - 栈式动态分配：栈常用于先进后出的操作。
   - 堆式动态分配：堆常用于维护一个有序的表，排序、top-k 等可以用堆加速操作过程。

   注：值单元分配是以过程函数为单位的，每个过程函数有其自身的活动记录。“过程”在 Pascal 语言里可以简单地理解为没有返回值的函数。

6.5.2 活动记录

首先介绍三个基本概念：

1. 过程：一个可执行模块，过程或函数，通常完成特定的功能。一个过程在编译器里指一段有独立功能的程序，可以完成一个函数或者一个 Pascal 过程，也可以称为一个函数。
2. 活动：过函的一次执行。每执行一次过程函数体，则产生该过函的一个活动。
3. 活动记录：一个有结构的连续存储块。用来存储过函一次执行中所需要的的信息。所谓结构，是指定义了管理数据、形参、局部变量等的位置结构，可视作一种数据结构。

三者关系：过程是一个抽象的概念，不对应函数具体的执行；活动是这个抽象概念的一次具体实现过程；活动记录是伴随着活动被定义的概念，一个活动记录用来记录一个活动所需要或产生的信息。

注意：

- 活动记录并不是针对某一个函数，而是针对这个函数的一次执行。更准确地说，是运行时所产生的一个记录，而不运行就没有活动或活动记录，只有过程。
- 活动记录仅是一种存储映像，编译程序所进行的运行时刻存储分配是在符号表中进行的，符号表中分配的变量就存储在活动记录里。
- 如果不支持可变数据结构（如动态数组，需要动态存储），活动记录的体积是可以在编译时确定的。

活动记录结构如图6.12所示：

1. 连接数据区

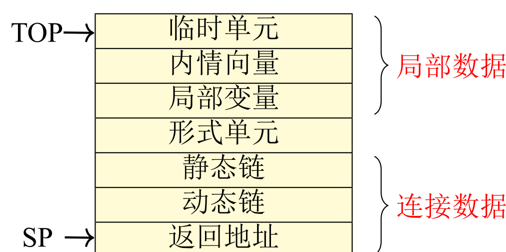

图 6.12: 活动记录结构

- 返回地址：保存断点地址，返回主控程序时继续执行的位置。(断点地址：函数被调用时的返回地址，是汇编语言执行的地址，而不是活动记录的返回地址。)
- 动态链：指向调用该过程的主调程序的活动记录的指针。（直接外层：与层次有关，如果是在第 3 层调用，直接外层就是第 2 层。）
- 静态链：指向静态直接外层活动记录的指针。

2. 形式单元

   用来存放实参的值或地址。

3. 局部数据区

   用来存放局部变量、内情向量、临时单元。内情向量存放计算过程中的一些相关参数；临时单元存放运算过程中的中间结果（在生成中间代码时生成的非用户定义的临时变量）。

4. 栈指针

   - SP：指向现行过程活动记录的起点，即第一个单元。
   - TOP：指向（已占用）栈顶单元，即活动记录的最后一个单元。

   用 SP 和 TOP 就可以确定活动记录所在的物理存储的区间。

例 6.5 动态链、静态链

动态是指运行过程中，而静态是指编写程序时。

- 动态链与程序调用有关。

  M 程序调用 N 子程序的活动记录栈结构如下图，当 M 程序载入内存时，两个指针指向 M 活动记录的起始位置和终止位置，定义了当前可操作的数据区。当 N 子程序载入内存时，N 活动记录进入活动记录栈，这两个指针移动到指向 N 活动记录的起始位置和终止位置，表示当前运行函数为 n。这两个指针限定了当前可操作的数据区，即当前运行函数的数据区。

  当 N 运行结束返回 M 时，N 活动记录弹栈，两个指针又重新指向 M 活动记录的起始位置和终止位置。为了获知 M 活动记录的位置，在 n 活动记录中开辟一个域，当 N 活动记录压栈时，M 程序将自身活动记录首地址放入该域中，指针指向 M 活动记录首地址，该指针就是动态链。

- 静态链与程序设计相关。

Pascal 语言允许嵌套定义函数，例如在函数 P 中嵌套定义了函数 M 和函数 N，在函数 M 中又嵌套定义了函数 Q。换言之，Q 子程序可以访问 M 的数据，也可以访问 P 的数据，但是不能访问 N 的数据，而 M 可以调用 N，N 也可以调用 M。此处涉及函数作用域的问题，内层的函数可以访问外层的、嵌套的外层的数据。例如在函数 P 中定义了变量 x，在函数 Q 中存在 $x = 10$; 语句，在活动记录中结构如下图所示。函数 P 的活动记录压栈，包含 x 的信息，不断调用，将 Q 压栈，此时动态链指向 Q 活动记录的起始位置和终止位置，但是 x 不在 Q 的活动记录内，无法根据动态链找到 x 的位置。Q 访问 x 需要获知 P 活动记录的首地址，再根据变量 x 的区距找到 x。函数 Q 保存静态定义的外层 M，再外层 P 等所有外层在内存中最新活动记录的首地址信息，就可以进行上述访问操作，这就是静态链。

此类变量作用域的问题，与静态设计相关，需要通过静态链解决。静态链可以指向静态直接外层活动记录的首地址，也可以通过多步跳转访问各个外层函数的数据。

6.5.3 简单的栈式存储分配

以 C 语言为例：没有分程序结构，过程定义不允许嵌套，但允许过程的递归调用。

例 6.6 C 语言过程调用关系：Main() -> Q() -> R()

1. C 语言程序的存储组织

   活动记录栈状态如图6.13，从下往上，地址由小到大。由于是栈式存储分配，后调用的函数存储在栈顶。当 R() 执行结束后弹栈，再执行 Q()，以此类推。下图中当前运行函数为 R，可操作数据区由指针 SP 和 TOP 进行限定。SP 和 TOP 分别指向当前活动记录的首尾地址，即 R 的第一个单元和最后一个单元。

图 6.13: C 语言调用过程 VALL-1

2. C 的活动记录

   如图6.14所示，将参数个数加入活动记录中，该形式支持被调函数完成实参到形参的传递。Old SP 指向上一个调用函数的过程所对应活动记录的首地址，由动态链完成。C 语言中函数不可嵌套定义，因此活动记录中只有动态链，没有静态链。

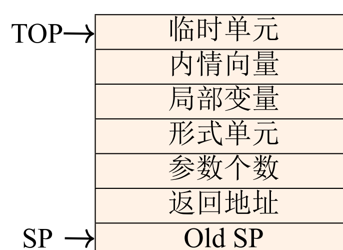

图 6.14: C 语言调用过程 VALL-2

3. C 语言的过程调用与返回

(1) 过程调用

- 过程调用的四元式序列

  (param,entry(t_1), __, __)

  ……

  (param,entry(t_n), __, __)

  (call, entry(P), n, __)

  四元式的第一个元素是操作的运算符或函数，第二、三个元素是操作对象，第四个元素是结果；(param, entry(t_1), __, __) ……(param, entry(t_n), __, __) 表示对变量 t_1 到 t_n，用 param 进行操作，得到的结果保存在对应四元式的最后一个位置 __；(call, entry(P), n, __) 表示调用 P 函数。

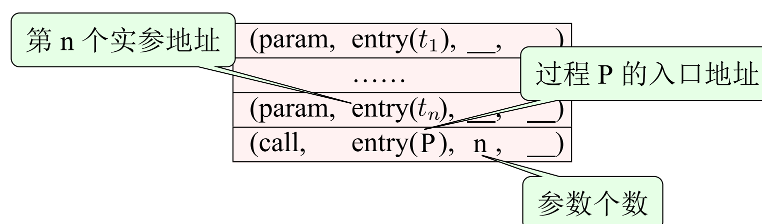

图 6.15: 调用函数过程四元式序列

图6.15 四元式序列描述调用函数的过程。t_1 到 t_n 表示函数的实参地址，entry(P) 表示调用函数的入口地址，参数个数是 n。

- 对应的目标指令

  四元式：(param, entry(t_i), __, __)

  作用：现有主调过程的活动记录，此时还未执行到函数 P 对应的活动记录。构建一个过程 P 所对应的活动记录，(param, entry(t_i), __, __) 将 t_i 写入子程序 P 活动记录中的形参区对应的位置。

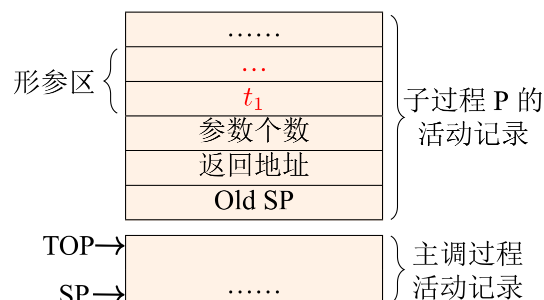

图 6.16: C 语言调用过程 VALL-3

对应指令：

$(i+3)$[TOP] := entry(t_i).Addr //将 t_i 地址填到活动记录的形参区去

以上语句表示：在 TOP 地址之上第 $i+3$ 个单元，保存变量 t_i 的地址。$(i+3)$ 表示在 TOP 地址之上增加 $i+3$ 个偏移量，因为形参区的第一个单元与 TOP 之间相差 3 个单元，所以 $i+3$ 可以索引到当前所要访问的形参编号。

四元式：(call, entry(P), n, __)

作用：表示要执行过程 P，但此时 P 活动记录里的一些必要信息还没有填写。(call, entry(P), n, __) 填写 Old SP 和参数个数 n。

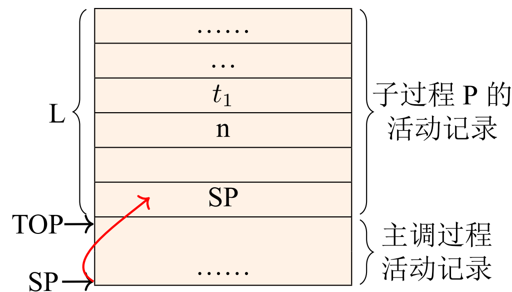

图 6.17: C 语言调用过程 VALL-4

对应指令：

1[TOP] := SP　　//保护现行 SP

3[TOP] := n　　//传递参数个数

JSP　P

以上语句表示：Old SP 将上一个过程即主调过程的 SP（指向主调过程的首地址），填写在 TOP 地址之上 1 个偏移量的位置。参数个数 n 填写在 TOP 地址之上 3 个偏移量的位置。最后跳转到 P，指的是汇编语言里真正要执行的函数过程的首地址，不是活动记录的首地址。

- 子过程 P 需完成自己的工作：定义自己的活动记录

此时，SP 和 TOP 还未指向当前要执行的子过程 P 的活动记录，需要分别指向子过程 P 的起始地址和终止地址，还需要填写 P 执行之后的返回地址。（假设 P 活动记录长度为 L）

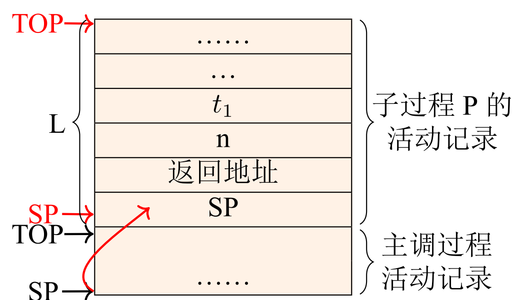

图 6.18: C 语言调用过程 VALL-5

对应指令：

SP := TOP + 1　　//定义过程 P 的 SP

1[SP] := 返回地址　　//保护返回地址

TOP := TOP + L　　//定义新 TOP

(2) 过程返回

- 过程返回的四元式：(ret, __, __, __)

ret 指令表示返回。

- 对应的目标指令：

  此时，子过程 P 执行结束，返回主调过程，需要将 TOP 和 SP 重新指向主调过程的起始和终止位置：TOP 移动到主调过程的最后一个单元，SP 的前一个单元；SP 移动到 P 活动记录中 Old SP 记录的地址，即主调过程活动记录的 SP。最后根据子过程 P 的返回地址，跳转回调用位置的下一条语句。（注：返回地址表示函数真实执行的位置，活动记录中记录函数里所存储的量的位置。）

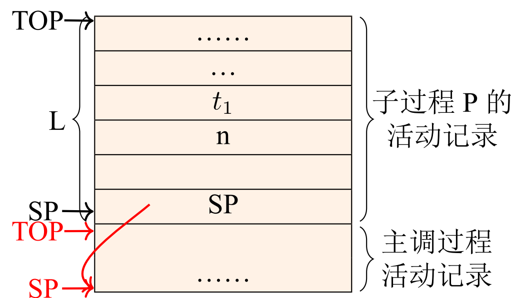

图 6.19: C 语言调用过程 VALL-6

对应指令：

TOP := SP − 1	//恢复 TOP
SP := 0[SP]	//恢复 SP
X := 2[TOP]	//取返回地址，X 为某一变址器
UJ 0[X]	//按 X 中的返回地址实行变址转移

6.5.4 嵌套过程语言的栈式存储分配

1. 标识符的作用域

   (1) 过程嵌套的一个关键问题：**标识符的作用域问题**

   标识符的作用范围往往与它所处的过程相关，也就是说，同一个标识符，在不同的程序段里，代表不同的对象，具有不同的性质，因此要分配不同的存储空间。

   (2) 标识符的有效范围：**服从最小作用域原理**

   - 在外层未定义，而在内层定义的，服从内层定义；
   - 在外层已定义，而在内层未定义的，服从外层定义；
   - 在外层已定义，而在内层也定义的，在外层服从外层定义，在内层服从内层定义（就近原则）。

2. 活动记录

   (1) 问题的提出：

   过程 Q 可能会引用到它的**任意外层**过程的最新活动记录中的某些数据，该如何存储？

   (2) 解决问题的思想：

   为了在活动记录中查找这些非局部名字所对应的存储空间，过程 Q 运行时必须设法跟踪它的**所有外层**过程的最新活动记录的地址。

(3) 解决方案：

活动记录中增加**静态链**如图6.20的最新活动记录的首地址。

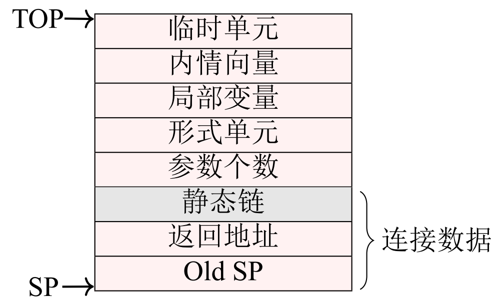

图 6.20: 添加静态链的活动记录

3. 嵌套层次显示表（display）和活动记录结构

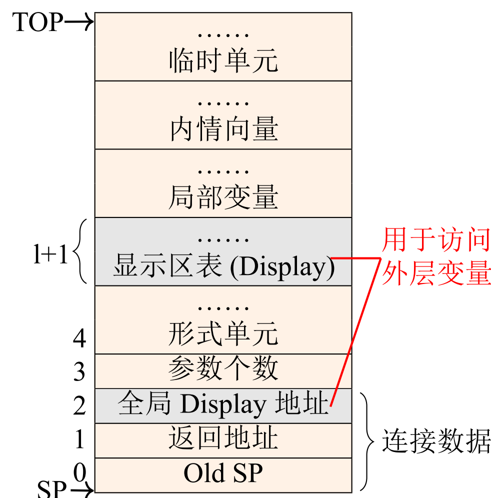

图 6.21: 添加嵌套层次显示表的活动记录

(1) 连接数据区：0 2

- Old SP——主调过程的活动记录首地址
- 全局 display 地址——主调过程的显示区表首址，用于访问当前活动记录所有外层的活动记录信息

(2) 参数个数：3

(3) 形参值单元区：入口为 4

- 换名形参 (vn)——分配 2 个单元（地址传递）
- 赋值形参 (vf)——按相应类型长度分配

(4) 显示区表（display）：指向外层活动记录的指针，占 I+1 个单元

I 为层次号，包含直接外层嵌套的 I 个过程的活动记录的首地址，再加上本过程的活动记录首地址

(5) 局部变量区：入口为 off + I + 2

- off 为形参区最后一个值单元地址
- 局部变量值单元按相应类型长度分配地址
- 类型标识符、常量标识符等不分配值单元；常量放在常数表，跟函数表用没有关系

(6) 临时变量区：

编译系统定义的变量，按局部变量值单元分配原则分配地址

4. Display 表的建立

设过程调用关系为 Q() -> R()，且 R() 的层次号为 I，则 Q 与 R 的 display 表的关系如图6.22：

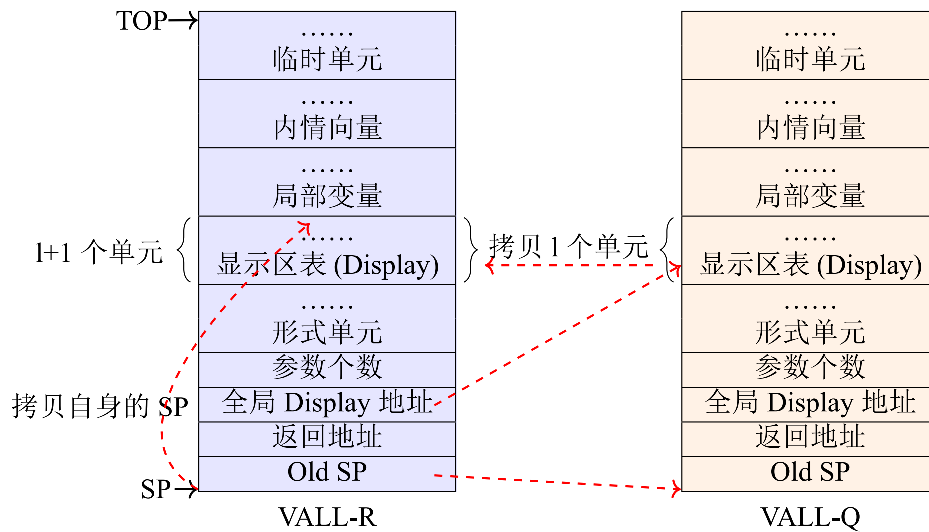

图 6.22: Q 与 R 的 display 表关系

R 的活动记录：

- Old SP：指向 Q 活动记录的首地址。
- 全局 Display 地址：指向 Q 的显示区表首地址。让当前活动记录能访问到所有外层的活动记录，外层的活动记录存储在 Q 的活动记录的显示区表里。
- 显示区表 (Display)：长度为 I+1 个单元。从 Q 的显示区表拷贝 I 个单元（第 I 层拷贝 I 个），并将自身活动记录首地址写入显示区表的第 I+1 个单元。

例 6.7 设有 Pascal 程序片段如图6.23：

这是 Pascal 中一个完整程序形式，函数（或过程）是用 program（或 procedure）定义的。P 是主程序，除了定义两个变量外，还定义了两个子程序，分别是 Q 和 S，Q 又定义了自己的内部函数为 R。因此该程序片段中，P 是 0 层，Q 和 S 是 1 层，R 是 2 层。

主控程序的代码在最后的 begin 到 end，调用了函数 S，S 代码段中，调用了函数 Q，Q 代码段中调用了函数 R。因此，整个过程的调用关系为 P->S->Q->R。

根据调用关系，我们可以判断活动记录栈的大致形式如下。下面对 P、S、Q、R 的活动记录进一步说明，模拟运行时的活动记录。图6.24是整个调用过程的内存映像。

- P 的活动记录：

P 为 0 层，由于 P 是系统调用的，Old SP 为 0，返回地址的值运行时存放断点地址，全局 Display 地址为 0，参数个数为 0，Display 表长度为 1，存放自身活动记录首地址 0。根据函数定义填写局部变量为 a 和 x，用“a-(0,5)”表示变量 a 的层次号为 0，偏移量为 5。

- S 的活动记录：

  P 调用 S，S 的活动记录载入内存。S 的 Old SP 指向 P 的活动记录首地址 0，返回地址内容运行时进行填写，全局 Display 指向 P 的 Display 表首地址 4，参数个数为 0。S 是 1 层，Display 表长度为 2，先拷贝 P 的 Display 表 0，再写入自身活动记录首地址 13，根据函数定义填写局部变量 c 和 i。

- Q 的活动记录：

  S 调用 Q，Q 的活动记录载入内存。Q 的 Old SP 指向 S 的活动记录首地址 13，返回地址内容运行时进行填写，全局 Display 指向 S 的 Display 表首地址 17，参数个数为 1，接着存放形参 b。Q 也是 1 层，Display 表长度为 2，拷贝 S 的 Display 表中前 1 个单元内容 0，再写入自身活动记录首地址 27，根据函数定义填写局部变量 i。

- R 的活动记录：

  Q 调用 R，R 的活动记录载入内存。R 的 Old SP 指向 Q 的活动记录首地址 27，返回地址内容运行时进行填写，全局 Display 指向 Q 的 Display 表首地址 35，参数个数为 2，依据参数类型进行存放形参 u 和 v。R 是 2 层，Display 表长度为 3，拷贝 Q 的 Display 表中前 2 个单元 0 和 27，再放入自身活动记录首地址 41，然后根据函数定义填写局部变量 c 和 d。

5. 值单元的地址分配值单元分配是依据活动记录的结构，在符号表中进行的。

例 6.8 Pascal 程序片段如下，P1 所在层 level=2，试给出符号表组织及值单元分配情况。图6.25左侧是符号表的内容，右侧紫色框是活动记录。

设:(1) 实型占 8 个存储单元，整型占 4 个单元，布尔型和字符型占 1 个单元

(2) 换名形参 vn 分配 2 个单元，赋值形参 vf 按相应类型长度分配

PROCEDURE P1(x: REAL; VAR y: BOOLEAN);

CONST pai = 3.14;

TYPE arr = ARRAY [1..10] OF INTEGER;

VAR m: INTEGER;

a : arr;

l : REAL;

FUNCTION F1(z : REAL; k : INTEGER): REAL;

BEGIN ……END;

……;

BEGIN

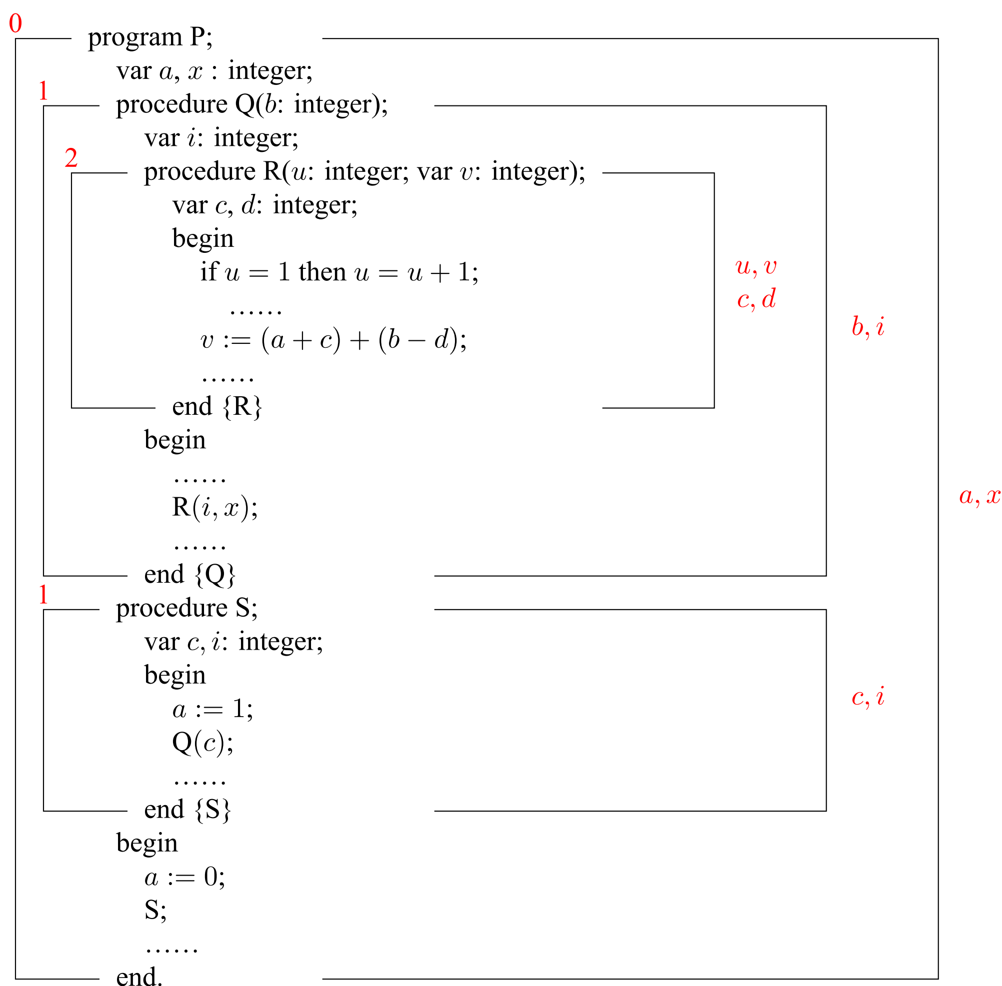

图 6.23: Pascal 程序片段

……;

END;

该程序片段定义了一个过程 P1，层次号为 3。包括 1 个常量标识符，1 个类型标识符，3 个局部变量，1 个内部函数。P1 定义了 F1，F1 的层次号为 4。

符号表组织及值单元分配情况过程：

(1) P1 (过程)

- 填符号表 SYNBL：NAME 域填过程名字 P1，TYP 域没有返回值不填，CAT 域填写函数的种类 p (过程)，ADDR 域指向函数表。
- 填函数表 PFINFL：LEVEL 域填函数的层次号 3，OFF 域填区距 3，FN 域填函数的变量个数 2，ENTRY 域填函数入口物理地址 Entry，PARAM 域指向形参表。
- 填形参表 PARAM：根据 x 和 y 的名字、类型、种类、地址，填写 NAME、TYP、CAT、ADDR 四项内容。其中 ADDR 内容在填完 VALL 之后，填入变量的层次号和偏移，x 对应 (3,4)，y 对应 (3,12)。填好形参表之后，在符号表中填入变量 x 和 y 的相关信息（与形参表中内容一致）。

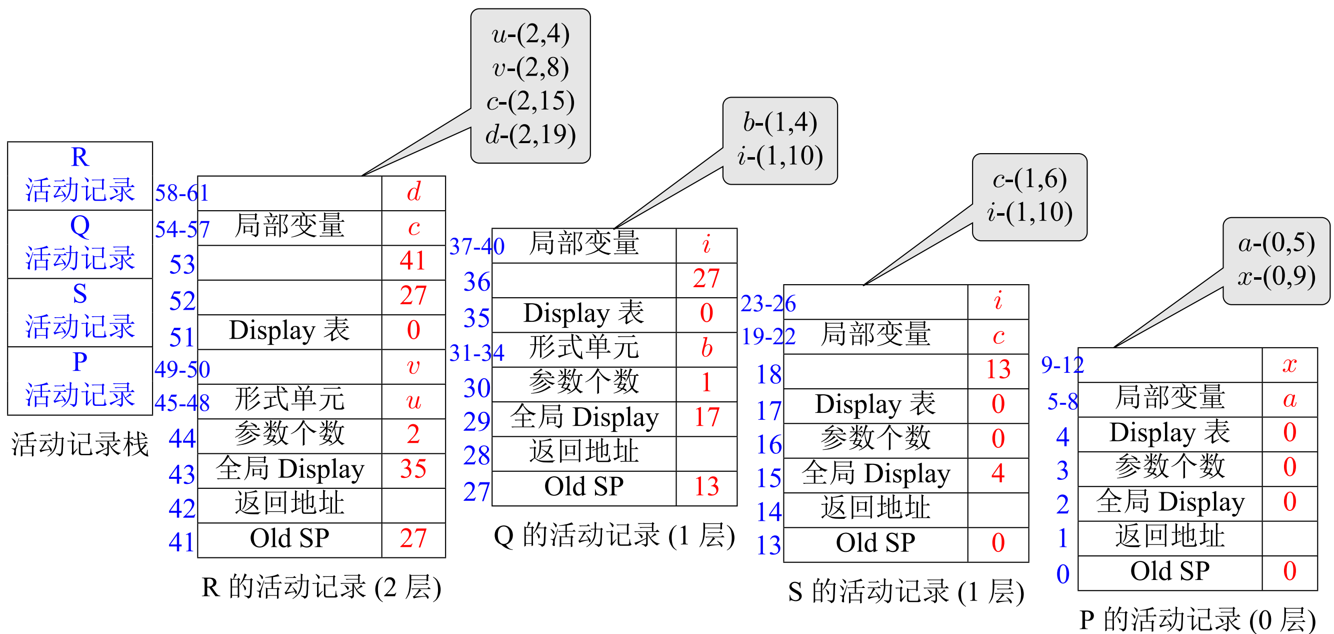

图 6.24: 活动记录栈调用过程

- 填活动记录 VALL：参数个数为 2；形式单元填 x，实型占 8 个单元，偏移量为 4-11，填 y，布尔型占 1 个单元，偏移量为 12-13。Display 表长度为 4。

(2) pai (常量)

- 填符号表 SYNBL：在符号表中继续填 pai 的相关内容，NAME 域填常量名字 pai，TYP 域填常量类型 rtp（实数型，指向类型表的 r 项），CAT 域填写常量种类 c。ADDR 域指向常量表 CONSL。
- 填常量表 CONSL：填入 3.14。

(3) arr (数组类型)

- 填符号表 SYNBL：在符号表中继续填 arr 的相关内容，NAME 域填名字 arr，TYP 域指向数组的定义——类型表。
- 填类型表 TYPEL：没有数组的定义，要新增。TVAL 域填类码 a，TPOINT 域指向数组表。
- 填数组表 ALNFL：LOW 域填数组的下界 1，UP 域填数组的上界 10，CTP 域填数组元素类型 itp（整数型，指向类型表的 i 项）。CLEN 域填值单元的长度 4。
- 填符号表 SYNBL：CAT 域填写数组种类 t（类型，可以用作定义其他变量的数据类型）。ADDR 域指向长度表。
- 填长度表 CONSL：填入 40。不填活动记录 VALL，因为 arr 是类型不是变量。

(4) m (局部变量)

- 填符号表 SYNBL：在符号表中继续填 m 的相关内容，NAME 域填变量名字 m，TYP 域填 itp，CAT 域填写变量种类 v，ADDR 域填活动记录中变量的层次号（偏移量）即 (3,18)。

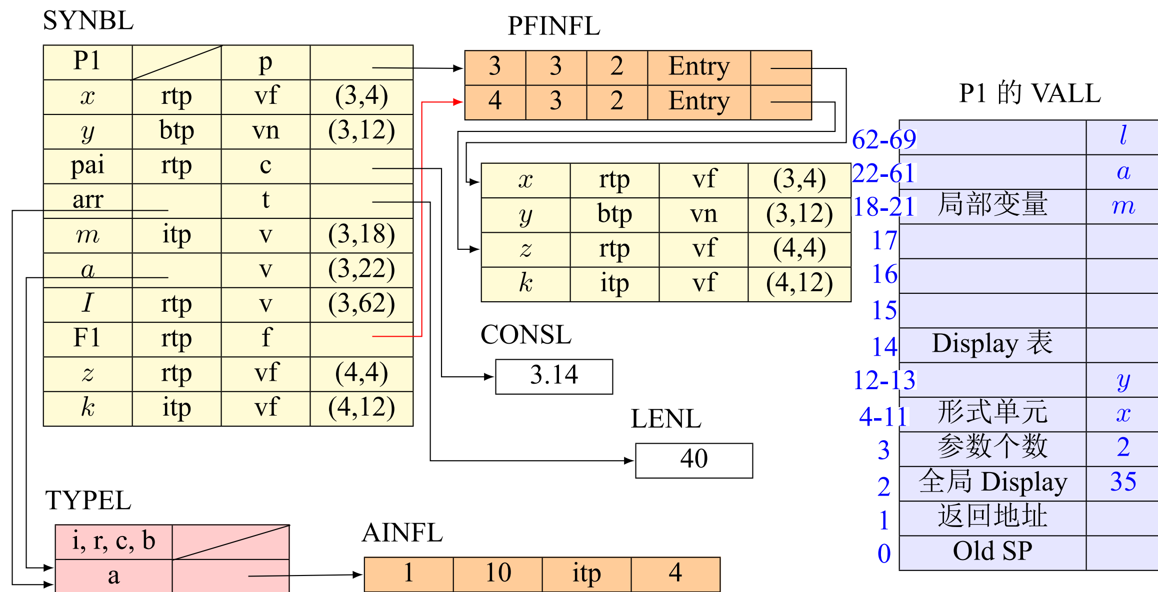

图 6.25: 符号表 + 活动记录填写过程

- 填活动记录 VALL：在 Display 表上方填局部变量 m，整型占 4 个单元，偏移量为 18-21。

(5) a (局部变量)

- 填符号表 SYNBL：在符号表中继续填 a 的相关内容，NAME 域填变量名字 a，TYP 域指向类型表中 arr 定义的数组类型 a，CAT 域填写变量种类 v，ADDR 域填活动记录中变量的层次号（偏移量）即 (3,22)。
- 填活动记录 VALL：局部变量 m 上填 a，数组类型占 40 个单元，偏移量为 22-61。

(6) I (局部变量)

- 填符号表 SYNBL：在符号表中继续填 I 的相关内容，NAME 域填变量名字 I，TYP 域填 rtp，CAT 域填写变量种类 v，ADDR 域填活动记录中变量的层次号（偏移量）即 (3,62)。
- 填活动记录 VALL：局部变量 a 上填 I，实型占 8 个单元，偏移量为 62-69。

(7) F1 (函数)

- 填符号表 SYNBL：NAME 域填过程名字 F1，TYP 域填函数返回值类型 rtp，CAT 域填写函数的种类 f (函数)，ADDR 域指向函数表。
- 填函数表 PFINFL：LEVEL 域填函数的层次号 4，OFF 域填区距 3，FN 域填函数的变量个数 2，ENTRY 域填函数入口物理地址 Entry，PARAM 域指向形参表。
- 填形参表 PARAM：根据 z 和 k 的名字、类型、种类、地址，填写 NAME、TYP、CAT、ADDR 四项内容。其中 ADDR 内容在填完 VALL 之后，填入变量的层次号和偏移，z 对应 (4,4)，y 对应 (4,12)。填好形参表之后，在符号表中填入变量 z 和 k 的相关信息（与形参表中内容一致）。

- 填活动记录 VALL：参数个数为 2；形式单元填 z，实型占 8 个单元，偏移量为 4-11，填 k，整型占 4 个单元，偏移量为 12-15。Display 表长度为 5。