第八章 运行时存储组织与符号表

提纲

- 8.1 运行时存储组织概述
  名字、过程、参数传递方式
- 8.2 典型的存储分配方案
- 8.3 栈式存储分配方法
- 8.4 堆式存储分配方法
- 8.5 符号表

8.1 运行时存储组织概述

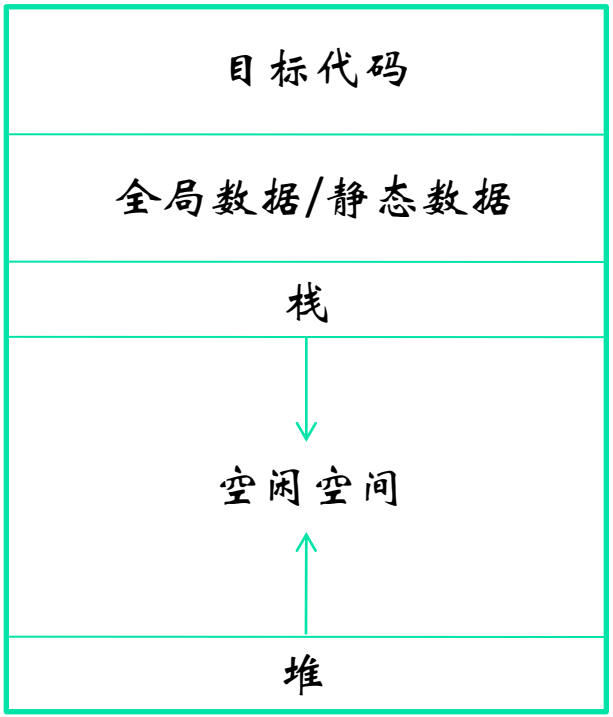
- **存储分配**：在目标程序运行时，应为源程序中的每一个量分配相应的存储单元，这个工作由编译程序完成。
- **存储组织**：存储分配在运行阶段进行，但编译程序要为其设计好存储组织形式，并将这种组织形式通过生成目标代码体现出来。
- **存储组织的实现**：运行阶段，随着目标代码的运行，数据的存储组织形式得以实现。

8.1 运行时存储组织概述

- **存储分配的对象**：编译程序必须为目标程序运行分配的存储空间包括：
  - 用户定义的各种类型的变量和常数所需的存储单元
  - 作为保留中间结果和参数传递用的临时工作单元
  - 调用过程或函数时所需的连接单元、返回地址
  - 组织输入/输出所需要的缓冲区

8.1 运行时存储组织概述

- 运行时刻的存储区域



8.1 运行时存储组织概述-名字

- 名字的概念和作用域
  - 名字代表一个抽象的存储单元，该单元的内容为此名字的值。标识符则是一个字符串，用于指示数据对象、过程、类或对象的入口。所有的标识符都是名字。
  - 名字的作用域：
    - 同一标识符在过程中不同的位置（如不同分程序）可用来代表不同的名字；
    - 程序运行时，同一名字在不同时间可能代表不同的存储单元（如递归情形）。

- 名字的生存期：
  - 全局量：对于嵌套子程序结构，在主程序中说明的量称为全局量；
  - 非局部量：在外层子程序中说明的量称为相对于原子程序的非局部量；
  - 局部量：在内层子程序中说明的量称为相对于内层子程序的局部量。
  - 从存储分配的角度看：
    - 全局量在整个程序运行期间始终占有固定数量的存储单元；
    - 局部量和非局部量只在运行到相应的程序单位时，才被分配数量确定的存储单元。

- **名字的属性：**
  - 名字的属性，包括类型和作用域。
  - 名字的属性可以通过以下方法规定：
    - 通过说明语句明确规定
    - 隐约定
    - 动态确定
  - **静态属性：**如果一个名字的属性是通过说明语句或隐约定规则定义的，则称这种性质是静态属性。
  - **动态属性：**如果名字的属性只有在程序运行时才能知道，则称这种属性是动态属性。

- 常见数据类型的存储分配：
  - 基本数据类型的存储分配
    - 基本数据类型包括整型、实型、布尔型和指针类型
    - 按类型宽度分配连续存储空间
  - 记录结构的存储分配
    - 记录结构由不同类型的数据组合而成，将其分量依次连续存储在一个数据区中

- 常见数据类型的存储分配：
  - 数组的存储分配
    - 单块存储方式：把数据区中的一片相连单元分配给数组元素，数组的所有元素按次序连续存储在这片数据区中（按行的次序和按列的次序）
    - 多块存储方式：对每一行都分配一个单块数据区，每行元素按递增次序存放在数据区中。再设立指针表，用以指示单块数据区的开始位置
    - 信息向量：通常在编译时只安排数组的信息向量的存储，而把实际元素的存储放到运行阶段，安排在数据区栈顶，用多少安排多少。信息向量的长度是固定的，因此在编译阶段在数据区中可以给它分配存储空间

8.1 运行时存储组织概述-过程

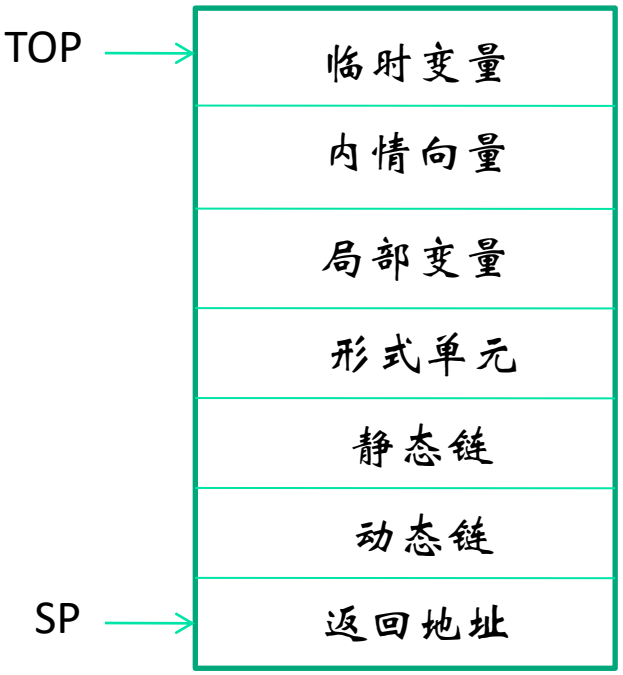
- 过程活动与过程的活动记录
  - 过程（procedure）,就是一个子程序，C语言中称为函数。
  - 每当过程被执行，就产生该过程的一个活动。如果过程允许递归，则在某个时刻该过程可能存在几个活动。

- 活动记录：为管理过程活动所需的信息，当过程调用发生时，使用一个连续的存储块记录该活动的相关信息，称为该过程的活动记录。
  - 活动记录不属于过程本身，而属于过程活动，即过程的执行
  - 活动记录是一片连续存储单元
  - 存储单元的表述：在编译时，任何变量运行时存储单元都可由一个对偶（数据区编号，位移）表示。需要引用某个变量时，采用（基址，位移）的形式

- 当过程返回时，过程的活动记录一般包含如下信息：
  - 连接数据
    - 返回地址
    - 动态链
    - 静态链
  - 形式单元
  - 局部数据

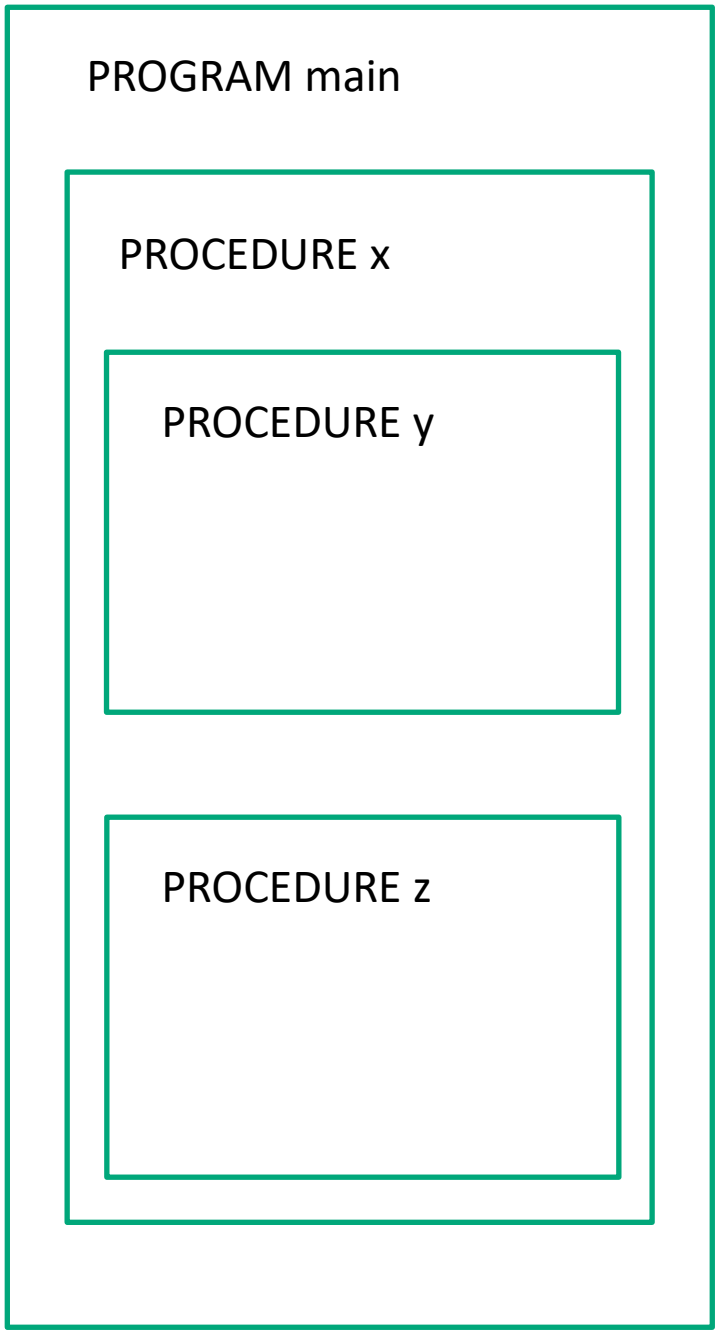
- 活动记录通常使用两个指针：SP指向现行活动记录的起始地址；TOP指向已被占用的数据区栈顶单元

- 活动记录中每个域的大小不是固定不变的，但几乎所有域的长度都可以在编译时确定，除非含有那种大小须由实在参数决定的形式参数



- **静态层次、静态外层和动态外层**
  - 静态层次：过程嵌套结构的程序设计语言，源程序编写完后，过程间形成一种嵌套层次关系，过程的这种嵌套结构中的层次称为该过程的静态层次。
  - 静态外层：直接包含某个过程的外层称为该过程的静态外层
  - 动态外层：在运行时刻，调用某过程的子程序称为该过程的动态外层

```
PROGRAM main;
VAR a, b, c: real;
   PROCEDURE x;
   VAR d, e: real;
      PROCEDURE y;
      VAR f, g: real;
      BEGIN
        …
      END; {y}
      PROCEDURE z;
      VAR h, i, j: real
      BEGIN
        …
      END; {z}
   BEGIN
     …
   END; {x}
BEGIN
  …
END; {main}
```



8.1 运行时存储组织概述-参数传递方式

- 程序运行时，调用者与被调用者之间的信息交流是通过全局量或参数传递方式进行的。
  - 传地址：调用者将实在参数的地址计算出来传递给被调用者。每个形式参数在被调用者的数据区中需要一个存放实参地址的单元。被调用者对形参的所有引用都作用在实参上

8.1 运行时存储组织概述-参数传递方式

- 传值：调用者将实在参数的值计算出来并传递给被调用者。每个形参在被调用者的数据区中需要一个存放实参值的单元。被调用者对形参的所有引用都是对存放值的形参单元的引用
- 传结果：调用者将实在参数的地址和值都传递给被调用者。每个形参在被调用者数据区中需要两个单元，分别存放实参值和实参地址。被调用者通过值单元存取值，调用结束后，返回前，按地址将结果写入实参

```
procedure p(x);
   begin
      …
      x := x+5;
      writeln(x,a);
   end;

… …
   a := 3;
   p(a);
L: write(a);
… …
```

传址：
a地址→x
8,8
8

传值：
a值→x
8,3
3

传结果：
a地址→x1
a值→x
x →x1
8,3
8

8.2 典型的存储分配方案

- 大部分现有编译中采用的方案主要有两种：
  - 静态存储分配方案
    - 在编译阶段对源程序中的量分配以固定存储单元，运行时始终不变
  - 动态存储分配方案
    - 在运行阶段动态的为源程序中的量进行存储单元分配

- 静态存储分配要求源程序满足以下要求：
  - 不允许有可变体积的数据，如可变数组等
  - 不允许有递归
  - 不允许有需在运行时动态确定性质的名字
- 采用静态存储分配，每次过程活动，一个过程中的同名局部量总是结合到相同的存储单元。即使过程活动结束，过程中局部量的值仍能保留，如果程序结束前再次进入该过程，局部量的值将和上次该过程结束时一样

- 动态存储分配方案将分配存储单元的工作放在运行阶段。
  - 在编译阶段为运行阶段设计好存储组织形式，为每一个数据项安排好它在数据区中的相对位置，并将这些安排以代码的形式插入到生成的目标代码中。
  - 在运行阶段，随着目标代码的执行，就会按编译程序安排的存储方式进行存储分配
- 两种分配方案
  - 栈式存储分配：在运行时把存储器作为一个栈进行管理，每当调用一个过程，它所需要的存储空间就分配在栈顶，一旦退出，它所占用的空间就被释放
  - 堆式存储分配：运行时把存储区组织成堆，以便用户对存储空间进行申请与归还

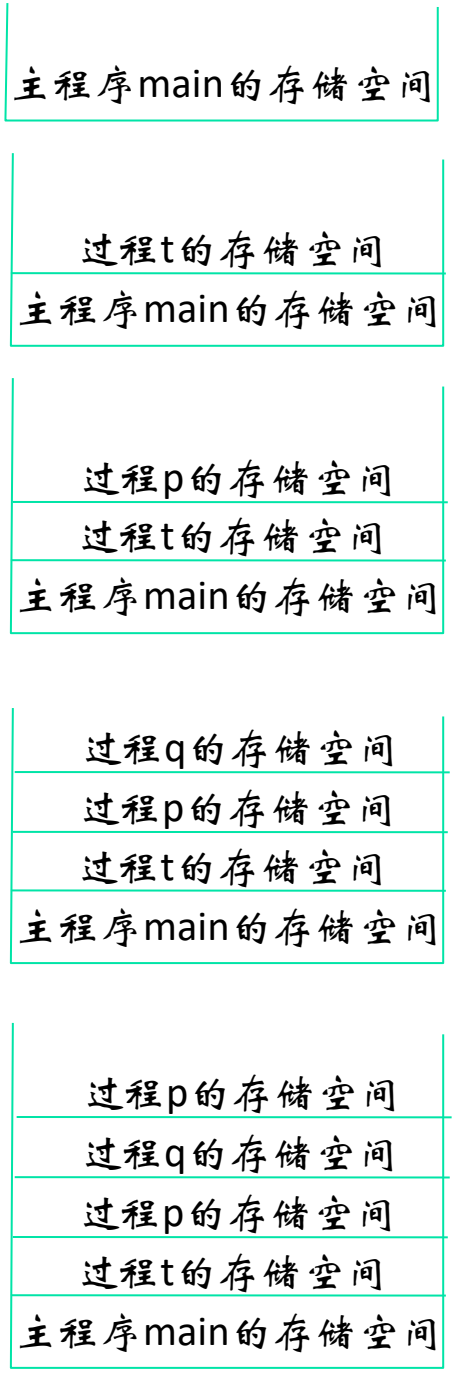
- 存储分配时需要考虑的问题
  - 过程是否递归
  - 控制从过程的活动返回时，局部量的值是否变化
  - 过程如何访问非局部量
  - 过程调用时参数如何传递
  - 过程是否可作为参数传递
  - 过程是否作为结果被返回
  - 存储区能否在程序控制下动态地被分配
  - 存储区是否必须显示地被分配
  - 临时变量如何分配
  - …… ……

- 涉及编译程序具体实现时应解决的四个问题
  - 临时变量、数组等特殊量的存储分配
  - 实在参数与形式参数结合的问题
  - 调用结束的控制返回问题
  - 嵌套过程之间对标识符引用的寻找问题

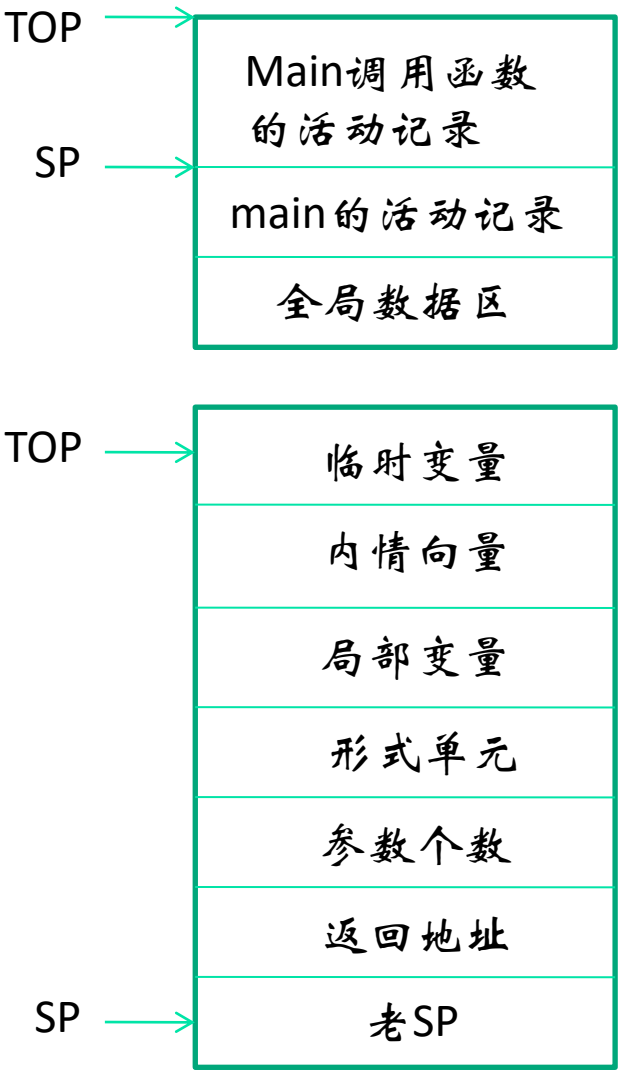
8.3 栈式存储分配

- 以过程为单位的栈式存储分配：是指把整个程序的数据空间设计成一个栈，以过程调用为单位来设置数据区。
- 在程序运行时：每当调用一个过程，就根据该过程的说明为该过程在数据区栈的顶部分配一定数据量的存储单元，作为该过程的数据区；
- 过程运行结束时：从栈中释放它所占用的存储空间。
- 如果是递归调用：则根据调用深度分别为每次调用设置不同的数据区。
- 栈式存储分配进一步分为：
  - 简单栈式存储分配
  - 嵌套结构语言的栈式存储分配

```
program main;
   const a = 10;
   var b, c: integer;
       d, e: real;
   procedure p(x: real);
      var f: real
      procedure q(y: real);
         const g = 5;
         var n: boolean;
         begin
            if e<0 then p(f);
         end
      begin {p}
      q(e)
      end; {p}
   procedure t;
      var j: real;
      begin {t}
         p(e);
      end; {t}
   begin {main}
      while c>0 do t;
      p(d);
   end; {main}
```

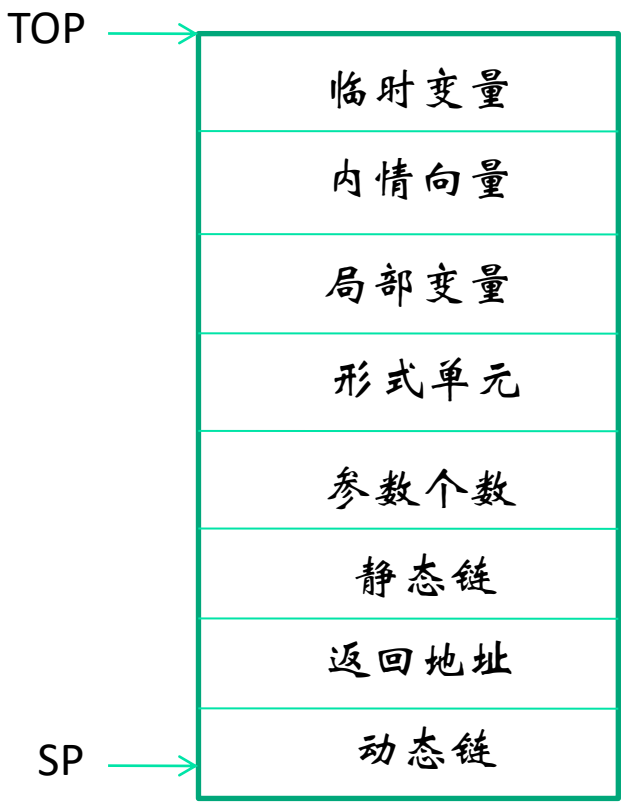


- 简单栈式存储分配（C语言）
  - 把存储区组织成栈，程序运行时，每当调用一个子程序，就在栈顶为它分配一块空间作为其工作时的临时数据区，即该子程序的活动记录；当该子程序结束时，将其活动记录弹出栈。一个子程序活动记录的体积在编译时可以静态确定



- 嵌套结构语言的栈式存储分配
  - 对于允许过程嵌套的语言，在过程调用时存在对非局部变量的访问问题。
  - 为访问非局部名字，被调用者必须知道该名字所在的活动记录。这就需要被调用者能够获得其所有外层过程的最新活动记录的起始地址。
    - 静态链
    - Display表

- **静态链（指向直接静态外层）**
  - 在栈式数据区中增加一个静态链，它从一个过程的当前活动记录指向其直接静态外层的最新活动记录的起始地址。
  - 当调用者访问非局部变量时，可以利用层差信息，跳过若干活动记录，找到名字被说明层的活动记录，再结合名字的相对地址信息，即可访问该名字了
- **动态链（指向直接动态外层）**
  - 动态链记录着调用者活动记录的起始位置。当被调用者结束运行时，利用动态链可以得到调用者的活动记录的起始地址，从而使调用结束后能够从调用者的活动记录中访问数据。

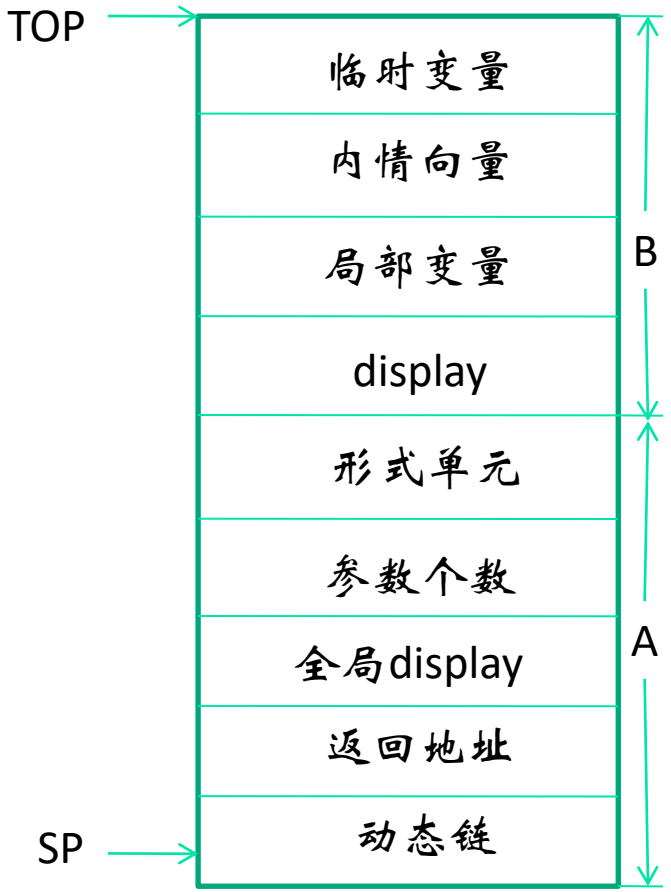


```
program main;
  const a = 10;
  var b, c: integer;
      d, e: real;
  procedure p(x: real);
    var f: real
    procedure q(y: real);
      const g = 5;
      var n: boolean;
      begin
        if e<0 then p(f);
      end
    begin {p}
    q(e)
    end; {p}
  procedure t;
    var j: real;
    begin {t}
      p(e);
    end; {t}
  begin {main}
    while c>0 do t;
    p(d);
  end; {main}
```

栈顶TOP值	运行栈	调用语句及当前TOP值	活动记录起始地址
0	动态链DL		SP=0
1	返回地址RA		
2	静态链SL		
3	a		
4	b		
5	c		
6	d		
7	e	TOP=7; t;	
8	动态链DL=0		SP=8
9	返回地址RA		
10	静态链SL=0	参数个数	
11	o		
12	j	TOP=12; p(e);	
13	动态链DL=8		SP=13
14	返回地址RA		
15	静态链SL=0		
16	l		
17	x		
18	f	TOP=18; q(e);	
19	动态链DL=13		SP=19
20	返回地址RA		
21	静态链SL=13		
22	l		
23	y		
24	g		
25	n	TOP=25; p(f);	

- Display表
  - 采用静态链，每次访问非局部量需要沿着静态链跳过若干个活动记录，这样速度就受到影响。为了提高访问速度，引进一个指针数组——嵌套层次显示表（display），依次存放着现行层、直接外层、……直至主程序层等每一层过程的最新活动记录的起始地址。
  - 每个过程的形式单元数目在编译时可以确定，因此display表的相对地址在编译时可以确定。过程的层数可以静态确定，因此每个过程的display表的大小也可在编译时确定。即整个活动记录可在编译时确定。
  - 全局display表是调用者的display表

- 过程调用时的存储管理
  （设：A调用B）
  - 过程调用发生时完成以下工作：
    - 调用者建立参数表、连接数据等
    - 调用者进行形、实参数的代换
    - 被调用者保存状态、初始化局部数据，并转过程起始地址
  - 过程调用结束时，需要完成以下工作：
    - 被调用者参数返回
    - 被调用者根据返回地址恢复调用者的数据区并将返回地址送PSW（状态寄存器）
    - 调用者恢复执行——从断点处继续执行



8.4 堆式存储分配方法

- 若一个程序设计语言允许用户动态地申请和释放存储空间，而且申请和释放之间不一定遵循“后申请先归还”的原则，那么栈式动态分配方案就不适用了。
- 对于这类语法成分，可以采用堆式存储分配方案。在存储空间中专门保留一片连续的存储块（通常叫做堆），在运行程序的过程中，每遇到这种语法成分，就由运行时刻存储管理程序从堆中分配一块区域，不需要时可由此管理程序释放该区域。

8.5 符号表

- 符号表：用于记录源程序中各种名字的类型和特征等信息的表格。
- 符号表的作用
  - 辅助易于正确性检查
    - 编译程序扫描说明部分，将收集到的有关名字的属性填写到符号表，这些属性信息可以为语义的合法性检查提供依据。
  - 辅助代码生成
    - 编译程序扫描到说明语句时，将对变量存储单元的分配情况记录在符号表中。当代码生成时，由语义分析程序或代码生成程序读取并安排在生成的代码中。
- 符号表的生存期
  - 编译开始时，符号表中或者为空，或者预先存放一些保留字。随着编译过程的进行，符号表的信息将在各阶段陆续填入。各阶段需要从符号表中获取不同的属性信息，符号表中的信息使用有时会延续到目标代码的运行阶段

- 符号表的内容
  - 名字信息
  - 名字的其他信息
    - 类型信息
    - 地址码
    - 层次信息
    - 行号信息
    - 存储类别等

- **符号表的组织**
  - 符号表的数据结构
    - 线性符号表
    - 有序表
    - 散列表
  - 符号表的内容组织
    - 名字域的组织：设立一个存放标识符的字符数组，在名字域仅给出标识符在该字符数组中的序号
    - 属性域的组织：采用信息向量表的方法。即把名字相关联的不等长属性域信息保存在符号表的某处，而把相应的指针或序号放在名字的属性域内

- 栈式符号表
  - 对于允许分程序嵌套结构的语言，不同嵌套层次的分程序中定义的标识符具有不同的作用域。
  - 为解决不同层次中同名标识符的作用域问题，通常为每个分程序建立一个子符号表，分程序是按层次嵌套的，所以，子符号表也是多层嵌套，并按序生成。这样，不同层次的同名标识符不会导致混乱。