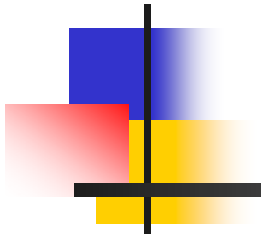
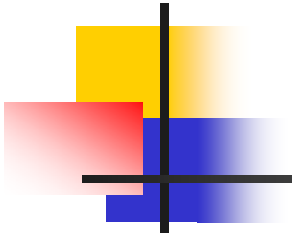


编译原理

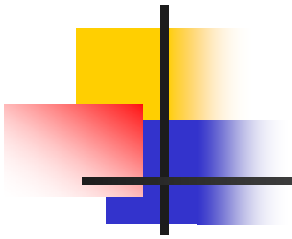
武汉大学计算机学院

编译原理课程组



前述内容回顾

- 属性文法
- 目标代码结构
- 中间代码
- 控制语句的翻译



第9章 运行时的存储组织与分配

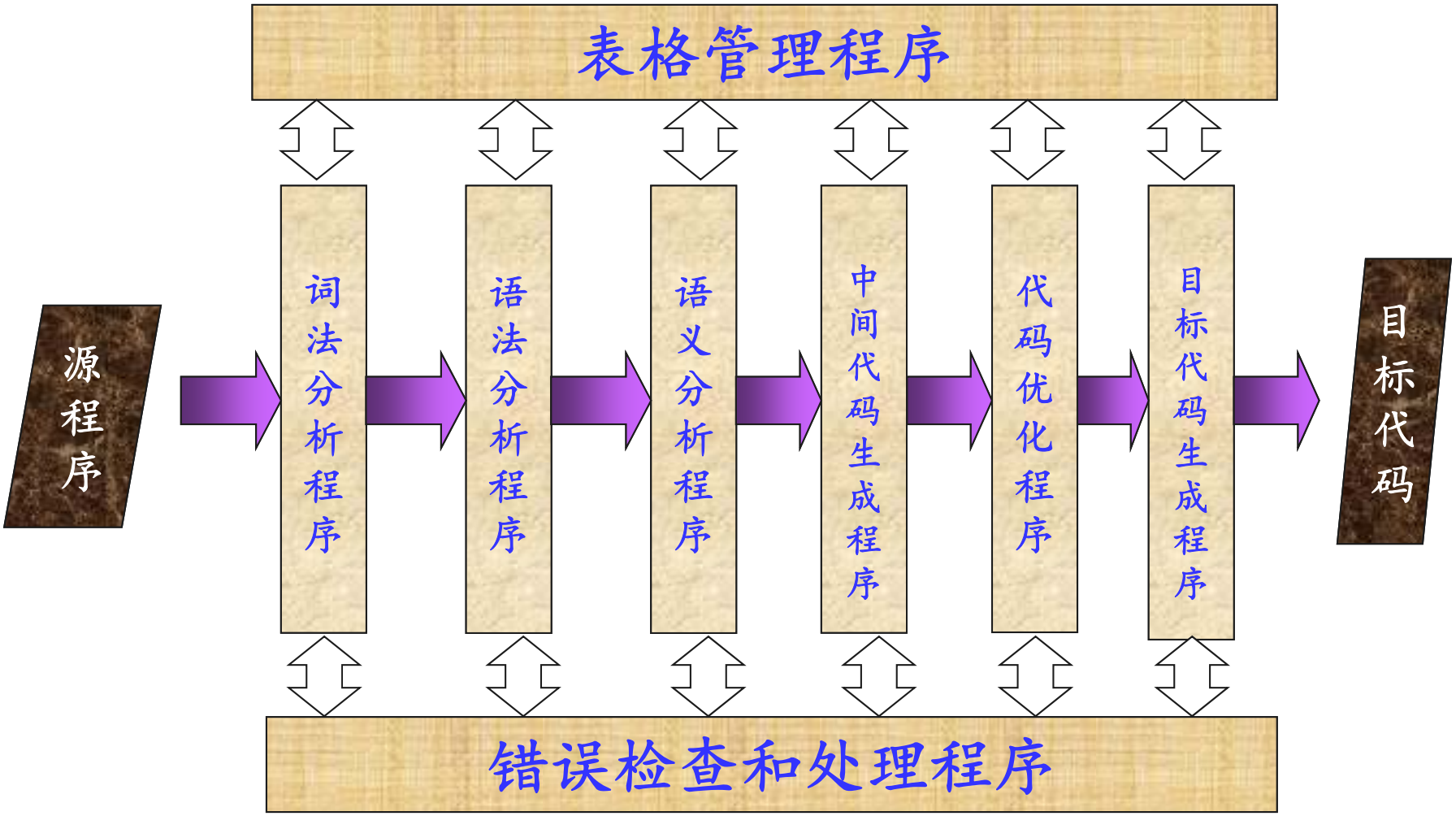
- 基本思想
- 基本知识
- 需解决的问题
- 具体实例

第9章 运行时的存储组织与分配

运行时环境：目标计算机的寄存器和存储器的结构。用以管理和组织存储，维护指导运行过程所需要的信息。

为了让目标程序运行，Compiler要从操作系统中得到一块存储区域，载入目标程序及其运行时需要和产生的数据，也就是需要把静态的目标程序正文和实现这个程序的运行活动联系起来，明确将来代码运行时源程序中的各个变量、常量、类型、类等程序定义的量是如何存放的，到哪里去访问。

编译程序的结构


表格管理程序
源程序
词法分析程序
语法分析程序
语义分析程序
中间代码生成程序
代码优化程序
目标代码生成程序
目标代码
错误检查和处理程序

9.1 概述

1. 任务

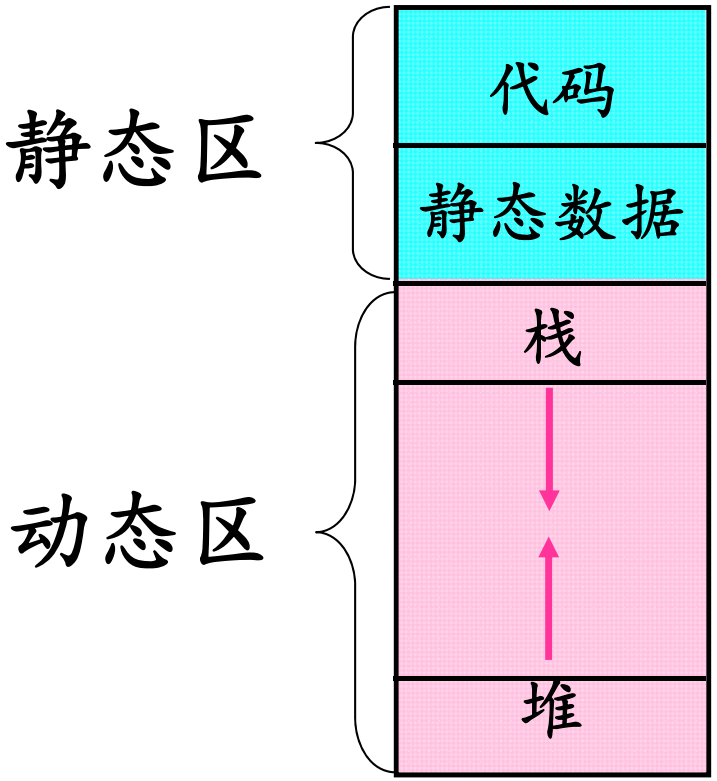
当一个目标程序运行时，应该有相应的存储区来存放目标代码、数据对象。

静态数据：所占存储空间的大小在编译时刻已知。

动态数据：所占存储空间的大小在编译时刻无法确定。

9.1 概述

运行时刻相关的存储区域：



9.1 概述

2. 基本思想

运行阶段的存储组织与分配，是指编译程序在编译阶段负责为源程序中出现的用户定义的变量与常量、临时工作单元、过程或函数调用时需要的连接单元与返回地址，以及输入/输出所需的缓冲区等组织好在运行阶段的存储空间。

9.1 概述

2. 基本思想

存储分配是在运行阶段进行的，但编译程序在编译阶段要为其设计好存储组织形式，并将这种组织形式通过生成的目标代码体现出来，有时还需要辅之以一定形式的服务子程序和信息表，为运行阶段实现存储奠定基础。

运行阶段，随着目标代码的运行，数据的存储组织形式便得以实现。

9.1 概述

3．典型的存储组织与分配方案

◆ 静态存储分配

在编译阶段对源程序中的量分配以固定单元，运行时始终不变。

◆ 动态存储分配

在运行阶段动态地为源程序中的量进行存储分配。

栈式存储分配　　堆式存储分配

9.1 概述

3. 典型的存储组织与分配方案—— 静态存储组织与分配

对语言的要求：

◆程序中的每一个数据对象的大小在编译阶段能够确定（即不允许有可变体积的数据如可变数组等）；

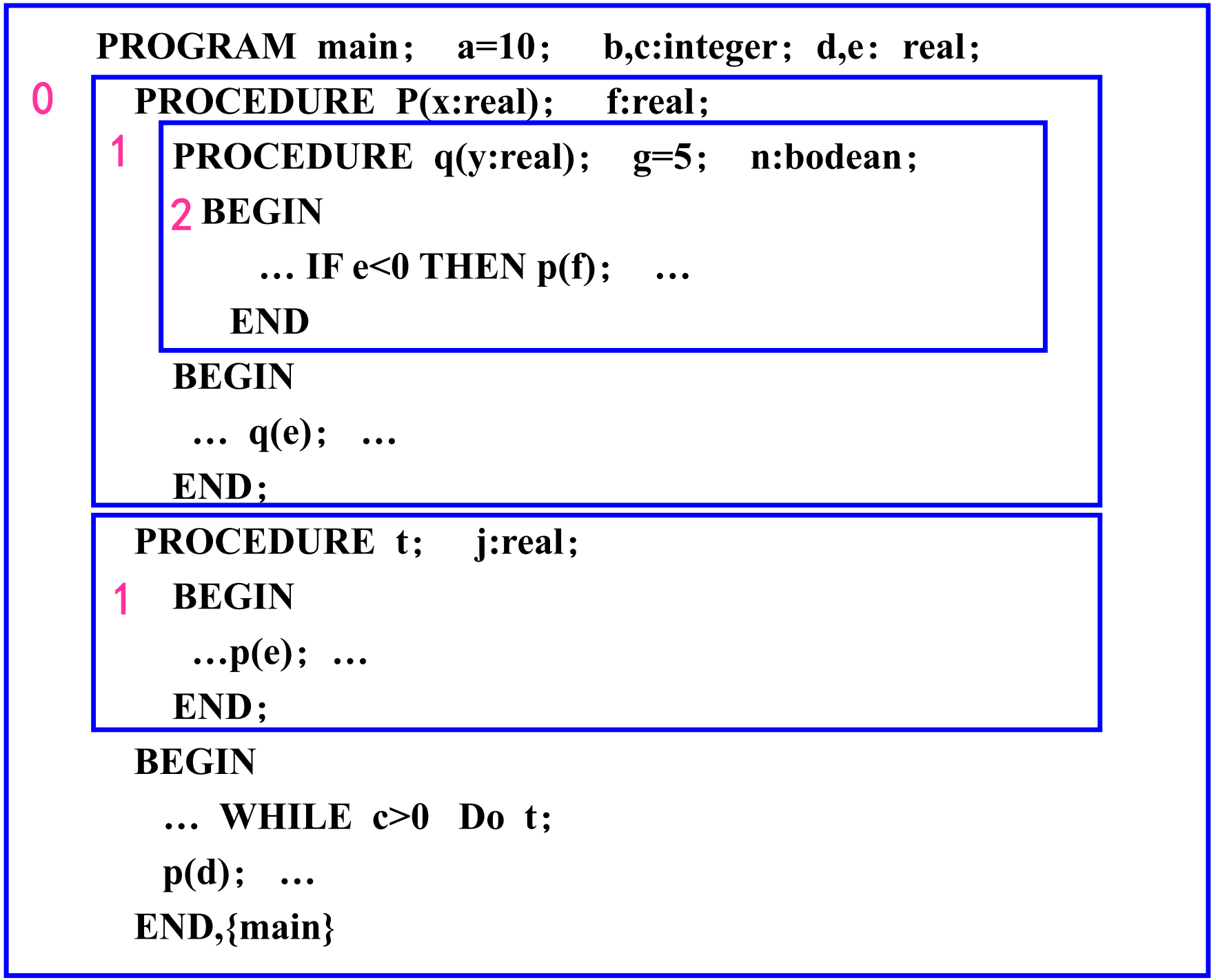
◆程序运行过程中的给定时刻，每一个数据对象只允许存在一个实例（即不允许有递归等）；

◆所有数据的性质是完全确定的（即不允许有需在运行时动态确定性质的名字）。

9.1 概述

3．典型的存储组织与分配方案—— 动态存储组织与分配

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。


PROGRAM main; a=10; b,c:integer; d,e: real;
0
PROCEDURE P(x:real); f:real;
1
PROCEDURE q(y:real); g=5; n:bodean;
2 BEGIN
… IF e<0 THEN p(f); …
END
BEGIN
… q(e); …
END;
PROCEDURE t; j:real;
1 BEGIN
…p(e); …
END;
BEGIN
… WHILE c>0 Do t;
p(d); …
END,{main}

9.1 概述

3．典型的存储组织与分配方案——动态存储组织与分配

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：在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，凡申请则从堆中分给一块，凡释放则退回给堆。

9.2 基本知识

1. 数据区

数据区是指一片连续的存储单元。在编译时，任何变量运行时的存储单元都可由一个对偶（数据区编号，位移）表示，数据区编号是分配给数据区的唯一编号，位移是指该存储单元相对于该数据区首地址的距离（或单元数）。

9.2 基本知识

2. 属性字

编译阶段在给变量分配存储地址的同时，还为它分配一个属性字，用以记录该变量在运行时所确定的属性。

如对动态数组，一旦运行时知道了存储空间的属性，就分配存储空间并将所分配存储空间的首地址存放在相应的属性字中，以后总是通过这个属性字去访问相应数组元素的地址。

9.2 基本知识

$address(A[i_1,i_2,\ldots,i_n])$

$= address(A[l_1,l_2,\ldots,l_n]) + (i_1-l_1)*d_2*d_3*\ldots*d_n$

$+ (i_2-l_2)*d_3*d_4*\ldots*d_n$

$+ \ldots$

$+ (i_{n-1}-l_{n-1})*d_n$

$+ (i_n-l_n)$

$= Conspart + Varpart$

$= address(A[l_1,l_2,\ldots,l_n]) -((\ldots((l_1*d_2+l_2)*d_3+l_3)d_4\ldots+l_{n-1})*d_n+l_n)$

$+ (\ldots(i_1*d_2+l_2)*d_3+\ldots+i_n-1)*d_n+i_n$

l_1	u_1	d_1
l_2	u_2	d_2
…	…	…
l_n	u_n	d_n
n	Conspart	
type	Baseloc	

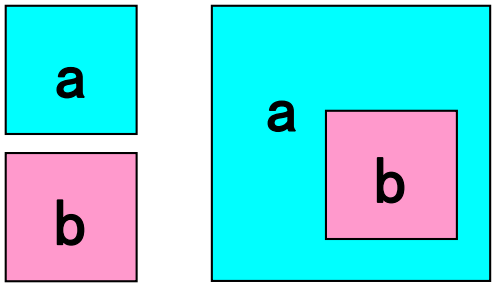
9.2 基本知识

3. 过程活动与过程活动记录

一个过程的活动指的是该过程的一次执行。即，每次执行一个过程体，就产生该过程的一个活动。

过程P的一个活动的生存期指从执行P的过程体第一步操作开始到最后一步操作之间的操作序列，包括执行P时调用其它过程花费的时间。

如果a和b都是过程的活动，则它们的生存期或者是不重叠的，或者是嵌套的。



9.2 基本知识

3. 过程活动与过程活动记录

过程是递归的：如果该过程在退出当前活动时，又开始其新的活动。

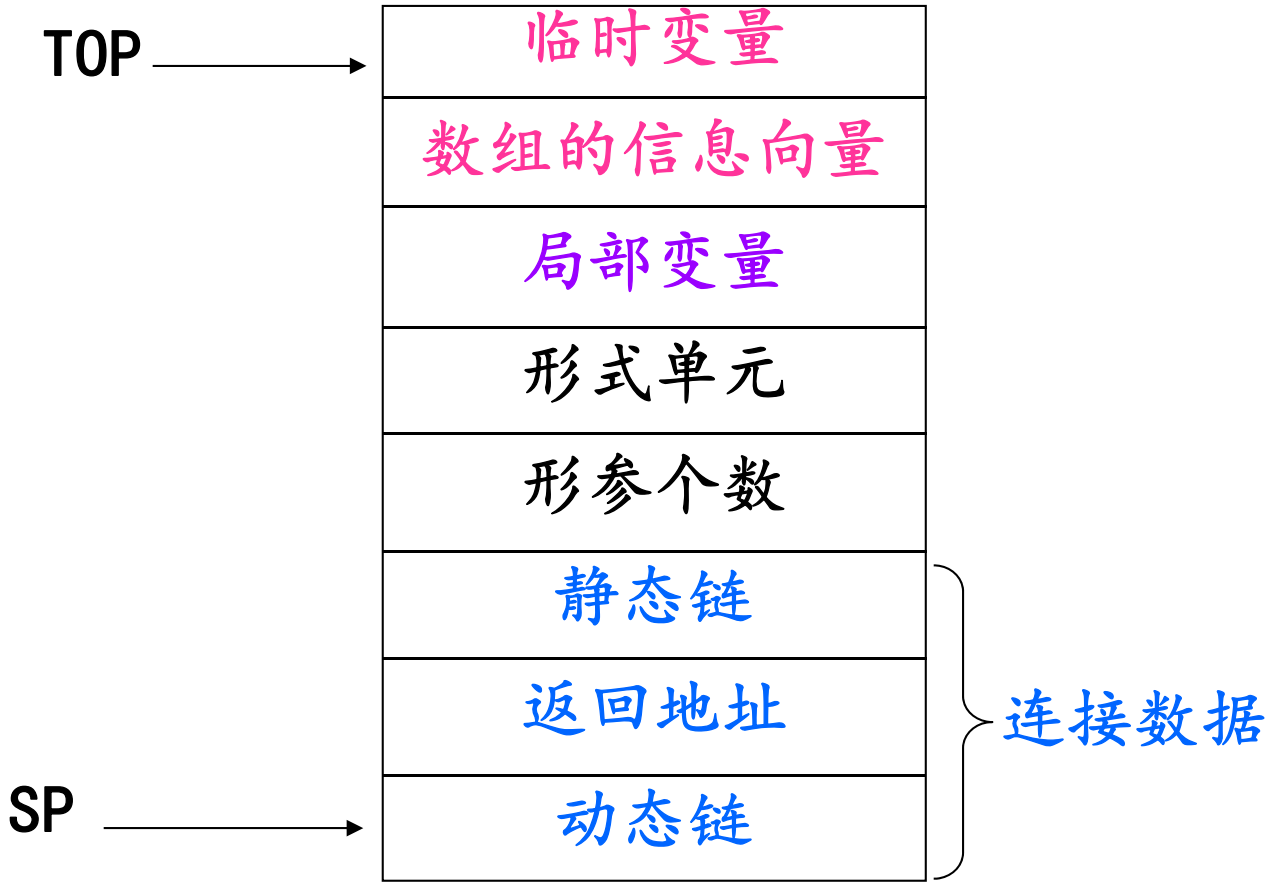
并不一定要直接调用它本身。P可调用Q，Q又经过若干调用又调用P。若P递归，在某一时刻可能有它的几个活动活跃着。

为了管理过程在一次执行中所需要的信息，使用一个连续的存储块，称其为活动记录，或该过程的数据区。

9.2 基本知识

3. 过程活动与过程活动记录

当过程返回（活动结束）时，当前活动记录一般包含如下内容：



动态链：
调用者活动记录的首地址(老SP)。

静态链：
解决非局部量的访问。

9.3 需解决的问题

1. 标识符的作用域问题　　P158

全局变量：编译时刻就可知道其大小。在编译时刻可以映射到存储单元。

局部量：仅在过程/函数被调用时才存在。

程序具有嵌套结构：调用时如何访问非局部量？

```
PROGRAM main;   a=10;    b,c:integer;  d,e:  real;
0   PROCEDURE  p(x:real);    f:real;
1     PROCEDURE  q(y:real);   g=5;    n:bodean;
        BEGIN
2       … IF e<0 THEN p(f);    …
        END
      BEGIN
        …  q(e);   …
      END;
    PROCEDURE  t;    j:real;
1     BEGIN
        …p(e);  …
      END;
    BEGIN
      …  WHILE  c>0   Do  t;
      p(d);   …
    END,{main}
```

9.3 需解决的问题

2. 字边界对齐问题

对不同类型的量所分配的存储单元的起始编址都必须符合边界要求。

基本数据类型——如字符、整型、实型等存放在连续几个字节中。

整型变量——通常占用数据区中的一个单元。

实型变量——通常占用一个字或采用双倍字长的浮点形式。

布尔变量——占用一个字或用一个二进位表示一个布尔变量，并尽可能把许多布尔变量合并在一个字中。

指示器型数据——通常在数据区中占用一个单元。

9.3 需解决的问题

2. 字边界对齐问题

结构类型：所分配的区域足以存放所有分量，可连续分配以便存取。

(1)对数组的存储分配有两种：　　9.3节

◆单块存储方式

把数据区中的一片内相连的单元分配给数组的元素，数组的所有元素按次序连续地存储在这片数据区中。

◆多块存储方式

每行分配一单块数据区，每行元素按递增次序放在该数据区中。另设一指示器表，指示这些单块数据区的开始位置。

(2)记录结构的存储分配

最简单的方法——将分量依次连续存储在一个数据区中。

9.3 需解决的问题

3. 参数的传递方式及其实现

不同语言的形实参数结合方式不同，解决的方法也各不相同。常见的形实参数结合方式有：传地址、传值、传结果、传名（换名）。

9.3 需解决的问题

◆ 传地址(Call by Reference)：被调用者数据区中需要一个存放实参地址的单元，对实参间接引用。

◆ 传值(Call by Value)：被调用者数据区中需要一个存放实参值的单元，对存放实参值的形参单元的引用。

◆ 传结果(Call by Result)：被调用者数据区中需要一个存放实参值的单元X和一个存放实参地址的单元Y。通过X取值，调用结束后，在返回时，按Y中地址将结果写入实参对应的单元。

◆ 传名/换名（Call by Name）：被调用者数据区中需要一个存放实参地址计算程序入口地址的单元Y。

9.3 需解决的问题

◆ 传地址(Call by Reference)：被调用者数据区中需要一个存放实参地址的单元，对实参间接引用。

```
procedure    p(x);                    ……
  begin                               a:=3;
    …                                 ……
    x:=x+5;                           p(a);
    writeln(x, a)…                    write(a);
    end;                              ……
```

9.3 需解决的问题

◆ 传值(Call by Value)：被调用者数据区中需要一个存放实参值的单元，对存放实参值的形参单元的引用。

```
procedure    p(x);                    ……
  begin                               a:=3;
    …                                 ……
    x:=x+5;                           p(a);
    writeln(x, a)…                    write(a);
    end;                              ……
```

9.3 需解决的问题

◆ 传结果(Call by Result)：被调用者数据区中需要一个存放实参值的单元X和一个存放实参地址的单元Y。通过X取值，调用结束后，在返回时，按Y中地址将结果写入实参对应的单元。

```
procedure    p(x);                    ……
  begin                               a:=3;
    …                                 ……
    x:=x+5;                           p(a);
    writeln(x, a)…                    write(a);
    end;                              ……
```

9.3 需解决的问题

◆ 传名/换名（Call by Name）：被调用者数据区中需要一个存放实参地址计算程序(形实替换程序thunk)入口地址的单元Y。

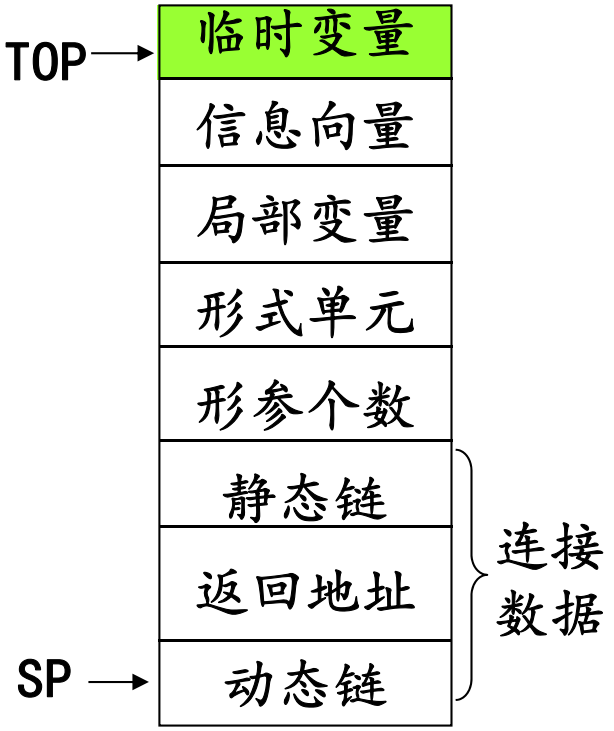
```
procedure  R(X, I);
  begin
    I:=2;
    X:=5;
    I:=3;
    X:=I-1
  end;
```

```
.....J:=2;
 R（B[J*2], J）;
.....
```

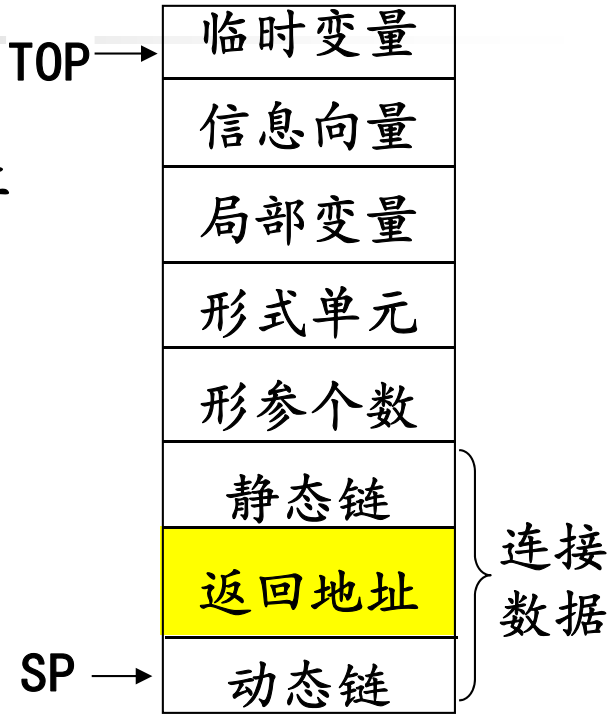
9.3 需解决的问题

4. 临时单元的分配

临时工作单元(变量)的分配涉及其作用域(——从该变量开始确定其值到最后一次使用它之间的时间间隔)。



9.3 需解决的问题



5. 过程或函数调用时需要的连接单元与返回地址

◆ 过程调用开始时：

① 进行形、实参数替换

② 建立参数表——连接数据等

③ 转到被调用者过程(体)起址

◆ 过程调用结束时——恢复现场，从断点处继续执行

① 参数返回

② 恢复调用者的数据区——隐参数

③ 恢复调用者的执行——从断点处继续执行（将返回地址送PSW）

9.4 栈式存储分配

1. 基本思想

以过程调用为单位来设置数据区。

设置一个数据空间栈，当程序运行时，每调用某过程一次，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量的存贮单元，称为该过程的数据区；当调用完毕，则释放该过程的数据区。

如果是递归调用，则根据调用深度分别为各次调用按先后顺序设置不同的数据区。

将这种数据空间栈称为运行栈S。

9.4 栈式存储分配

2. 存储分配实现时需要解决的问题

◆ 临时变量、数组等特殊量的存储分配

临时变量可直接在数据区中分配。对于数组，则只需在数据区中安排数组的信息向量所需单元即可，实际分量则安排在栈顶。

◆ 实在参数与形式参数结合的问题

不同的语言的形实参数结合方式不同，解决的方法也各不相同。

◆ 调用结束的返回地址

每个被调用者相应的数据区内增加一项：返回地址。

◆ 嵌套过程之间对非局部量的访问

在被调用者数据区内开辟一些单元，存放嵌套过程间的联系信息。

9.4 栈式存储分配

3．嵌套过程之间对非局部量的访问

嵌套结构中，一个过程可以引用包围它的任一外层过程所定义的名字。为非局部名字，被调用者必须知道它的所有外层过程的最新活动记录的地址——必须设法跟踪每个外层过程的最新活动记录的位置。

◆ 静态链

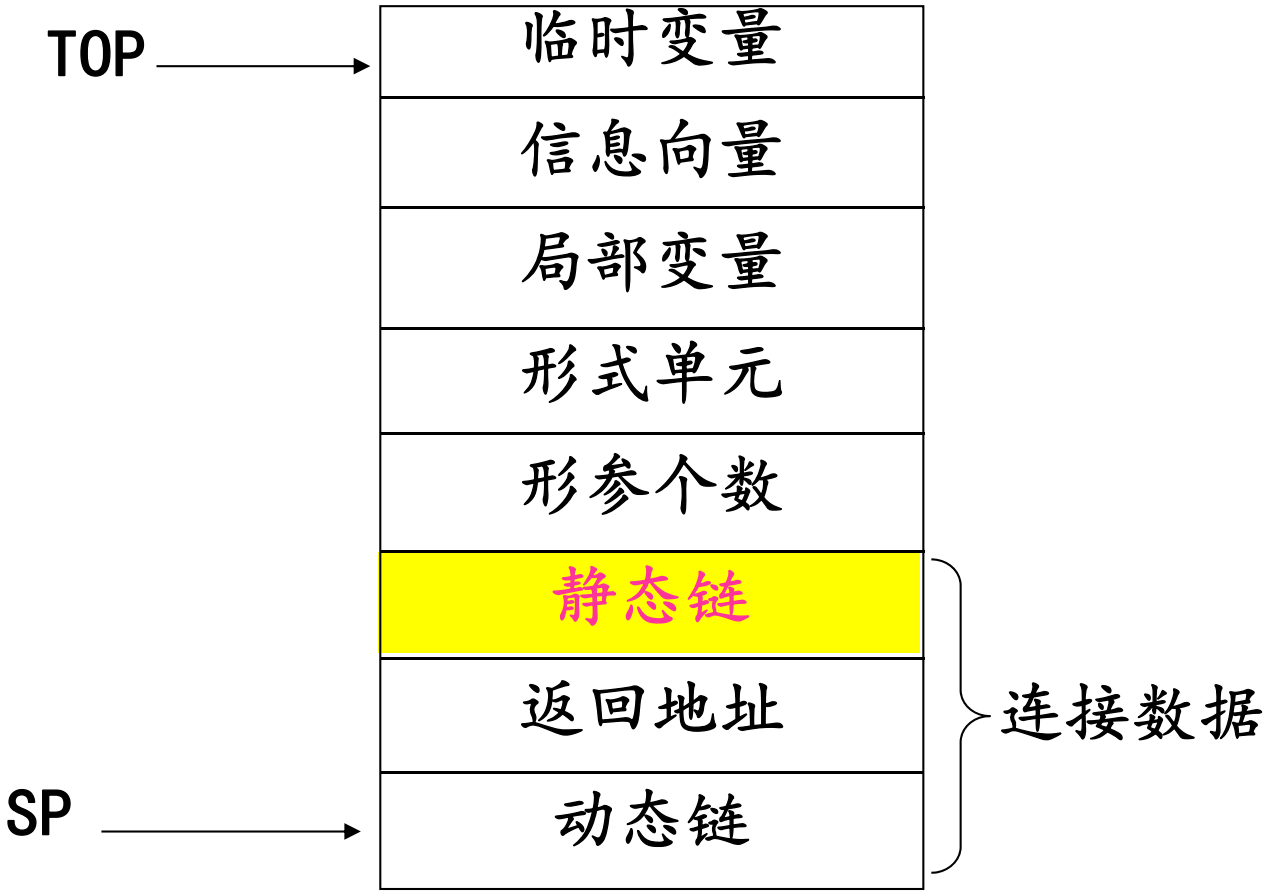
从过程当前活动记录指向其直接外层的最新活动记录的指针。

◆ 嵌套层次显示表(display) ——指针数组

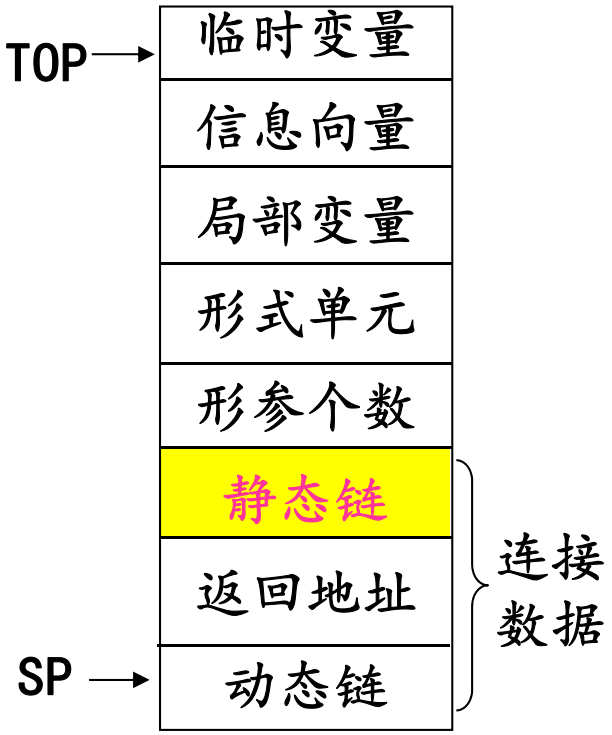
依次指向当前层、直接外层、……，直至最外层(0层，主程序层)等每一层过程的最新活动记录的首地址 。

9.4 栈式存储分配

3. 嵌套过程之间对非局部量的访问—— 静态链



静态链：
直接外层的活动记录首地址。



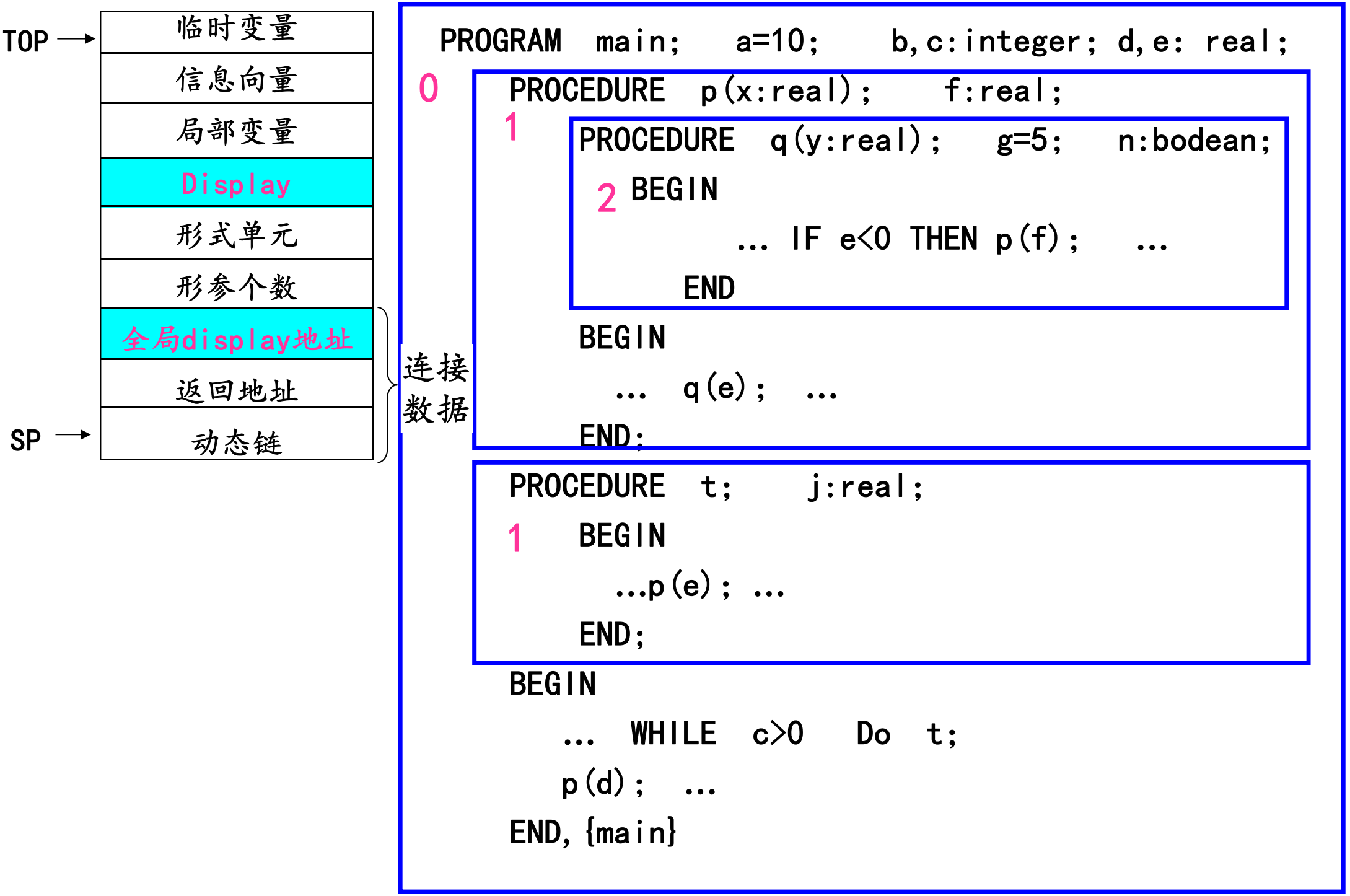

```
PROGRAM  main;   a=10;    b,c:integer; d,e: real;
0  PROCEDURE  p(x:real);    f:real;
1    PROCEDURE  q(y:real);   g=5;   n:bodean;
       2 BEGIN
                ... IF e<0 THEN p(f);   ...
           END
       BEGIN
         ...  q(e);  ...
       END;
     PROCEDURE  t;    j:real;
1    BEGIN
         ...p(e); ...
     END;
     BEGIN
        ...  WHILE  c>0   Do  t;
        p(d);  ...
     END,{main}
```

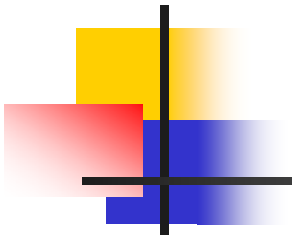
9.4 栈式存储分配

3. 嵌套过程之间对非局部量的访问—— 嵌套层次显示表(display)

display —— 指针数组

依次指向当前层、直接外层、……，直至最外层(0层，主程序层)等每一层过程的最新活动记录的首地址 。


TOP
临时变量
信息向量
局部变量
Display
形式单元
形参个数
全局display地址
返回地址
SP
动态链
连接数据
PROGRAM main; a=10; b,c:integer; d,e: real;
0
PROCEDURE p(x:real); f:real;
1
PROCEDURE q(y:real); g=5; n:bodean;
2 BEGIN
... IF e<0 THEN p(f); ...
END
BEGIN
... q(e); ...
END;
PROCEDURE t; j:real;
1 BEGIN
...p(e); ...
END;
BEGIN
... WHILE c>0 Do t;
p(d); ...
END,{main}



9.5 堆式存储分配

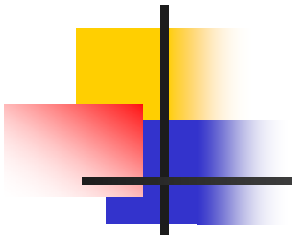
有些数据对象，它们对存贮空间的需求量，既不能在编译阶段，也不能在进入过程语句的时刻知道，而只能在创建它们或给它们赋新值的时候才知道。如PASCAL语言中的new等。

对于这类语法成分，可以采用堆式存贮分配方案：

在存贮空间里专门保留一片连续的存贮块（通常叫做堆），在运行程序的过程中，每遇到这种语法成份，就由一个运行时刻的存贮管理程序从堆中分配一块区域给它，不再需要时，又可由此堆管理程序释放该区域，供以后重新分配使用。

本章内容回顾

- 基本思想
- 基本知识
- 需解决的问题
- 具体实例



下章内容简介 —— 第10章

符号表