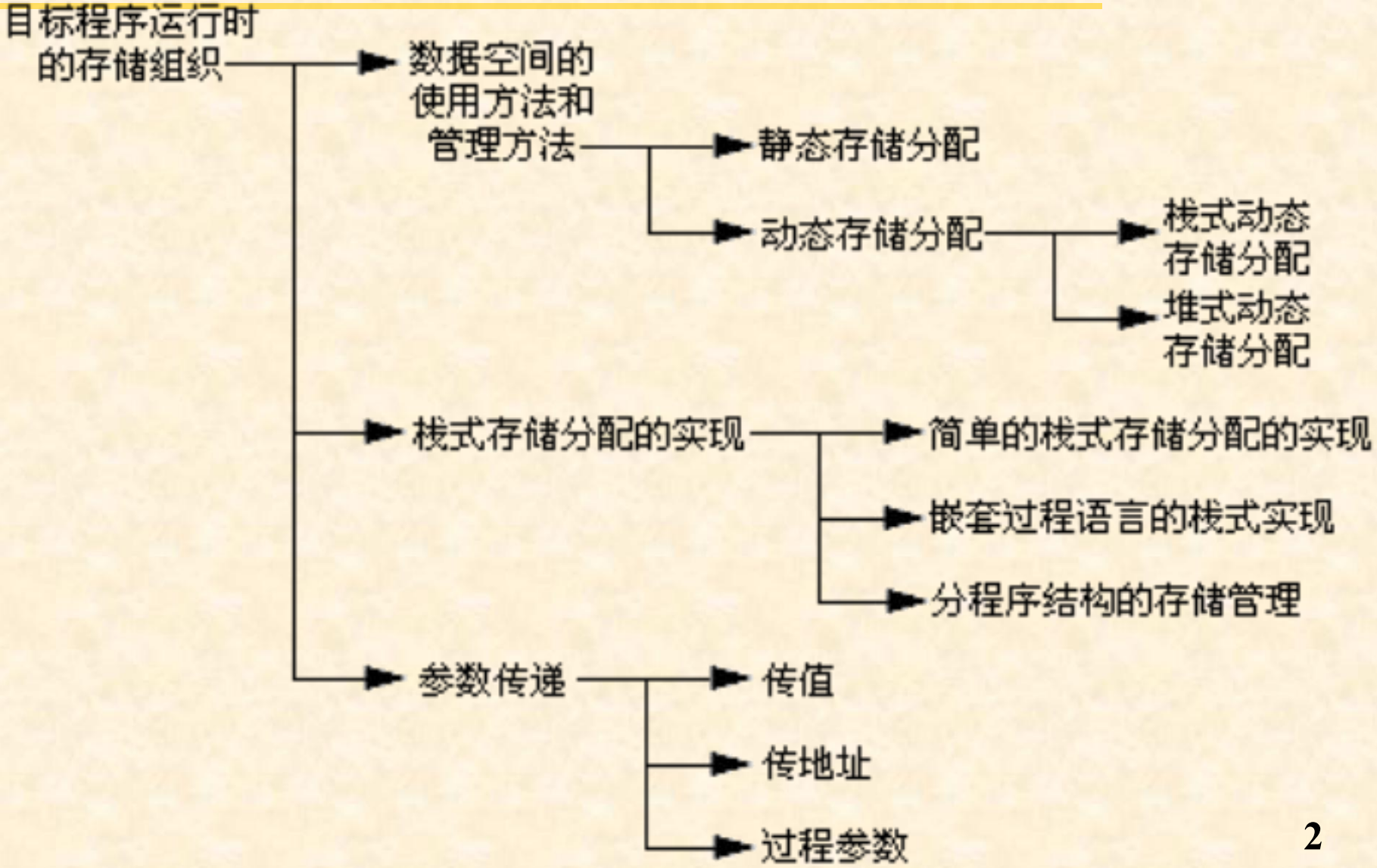
第9章　目标程序运行时的存储组织

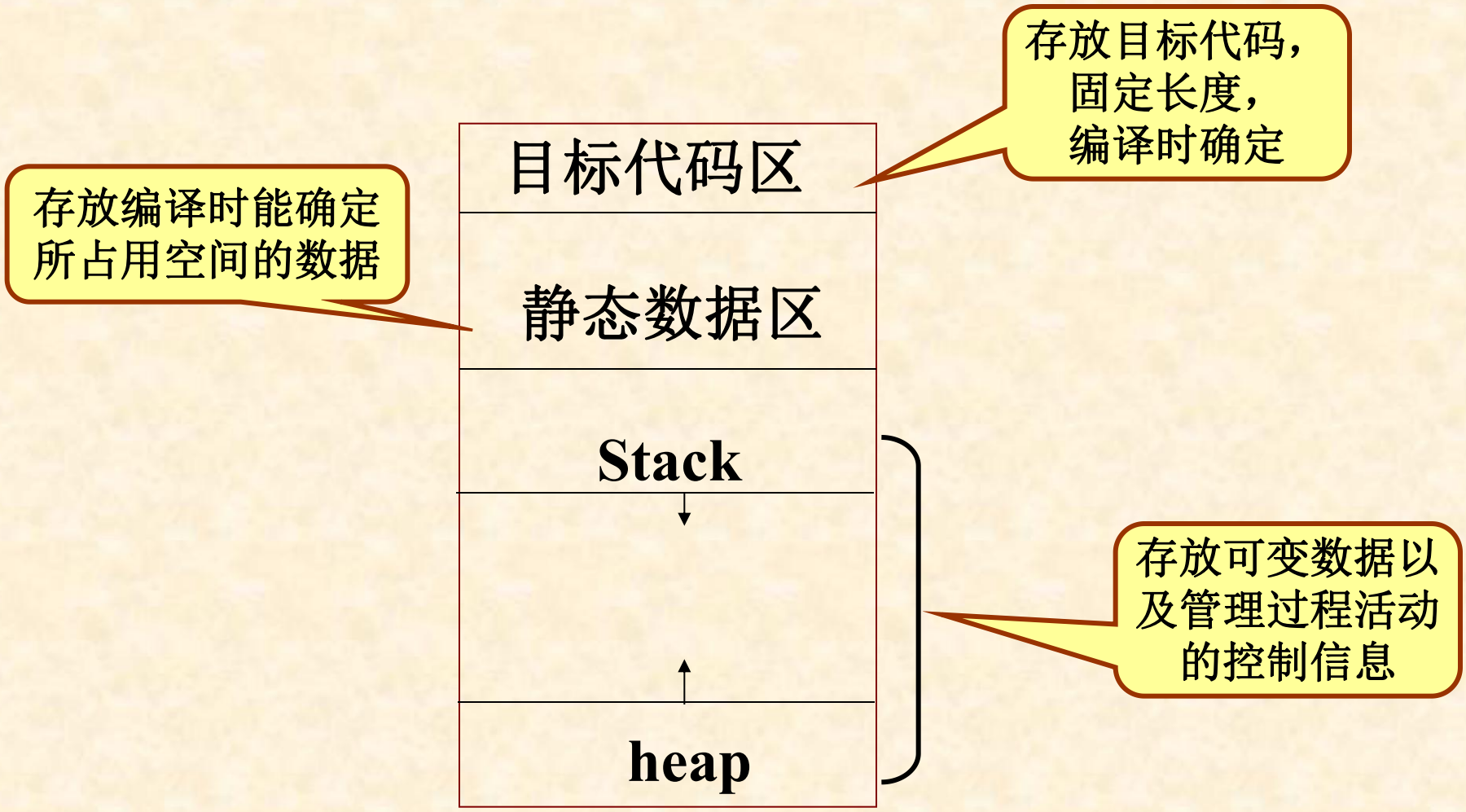
知识结构


目标程序运行时的存储组织
数据空间的使用方法和管理方法
静态存储分配
动态存储分配
栈式动态存储分配
堆式动态存储分配
栈式存储分配的实现
简单的栈式存储分配的实现
嵌套过程语言的栈式实现
分程序结构的存储管理
参数传递
传值
传地址
过程参数

概述

- 从逻辑上看，在代码生成前，编译程序必须进行目标程序运行环境的设计和数据空间的分配。
- 该数据空间需容纳生成的目标代码和目标代码运行时的数据空间。
- 数据空间应包括：
  - 用户定义的各种类型的数据对象(变/常量) 的存储空间
  - 作为保留中间结果和传递参数的临时工作单元
  - 调用过程时所需的连接单元
  - 组织输入输出所需的缓冲区

目标程序运行时的存储区的典型划分：



Windows32下对数据空间的划分

```
#include <stdio.h>
#include <stdlib.h>
int global1;                                    //未初始化全局变量
int global2=200;                                //初始化全局变量
int main(void)
{
    int a, b, i;                                //a, b, i, p局部变量
    int *p;                                     //指向整型的指针变量
    scanf("%d%d%d", &a, &b, &i);                //读入a, b, i
    scanf("%d%d", &global1, &global2);          //读入global1, global2
    p=(int *)malloc(20*sizeof(int));            //分配80个字节空间给p
    for (i=0; i<20; i++)
        *(p+i)=i;                               //给p数组赋值
    free(p);                                    //释放p
    return 0;
}
```

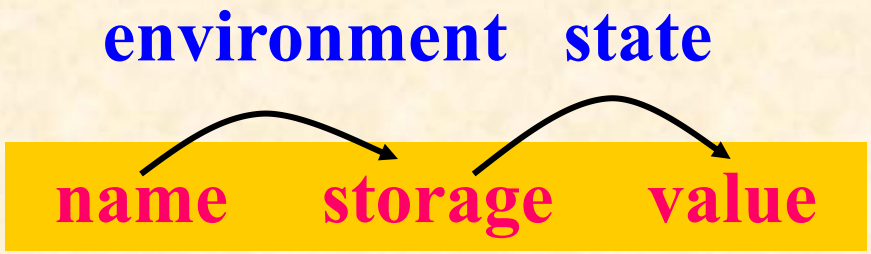
- Windows32下，每个段由多个页组成，每个页的大小为4K。如果某页32位线性地址为：XXXXX000~XXXXXFFF，

  则该页记为第XXXXX页。

- Windows32的段主要有如下几种：
  - 代码段：存放指令代码　(00401页)
  - 只读数据段：只读的常量(字符常量和程序运行中的错误信息 （00422页, 如“%d%d”）
  - 可读写的经初始化的数据段：全局变量和静态变量 (00424页, global1)
  - 可读写的未经初始化的数据段：全局变量/静态变量（00425页global2）
  - 栈段(Stack)：局部变量和返回地址等数据 (0012f页，a, b, i等)
  - 堆段(Heap)：动态申请/释放的存储段 (00431页，p所指向的内存区域)

数据空间的分配的本质

- 将程序中的每个名字与一个存储位置关联起来，该存储位置用以容纳名字的值。
- 用environment表示将一个名字映射到一个存储位置。
- 用state表示存储位置到值的映射。



名字到存储、到值的映射

存储管理复杂度

取决于源语言本身，具体包括：

1. 允许的数据类型的多少
2. 语言中允许的数据项是 { 静态确定 / 动态确定 }
3. 程序决定名字的作用域的规则和结构
   - ① 段结构
   - ② 过程定义不嵌套，只允许过程递归调用
   - ③ 分程序结构 { 分程序嵌套 / 过程定义嵌套 }

9.1 数据空间的三种不同使用方法和管理方法

- **静态存储分配**
- **栈式动态存储分配**
- **堆式动态存储分配**

静态存储分配

- 如果在编译时能确定 目标程序运行中所需的全部数据空间的大小，编译时安排好目标程序运行时的全部数据空间，确定每个数据对象的存储位置，称这种分配策略为静态存储分配。

动态存储分配

- 如果一个程序设计语言允许递归过程、可变数组或允许用户自由申请和释放空间，那么就需要使用动态存储管理技术。
- 对于这种程序，在编译时无法知道它在运行时需要多大的存储空间，它所需要的数据空间的大小需待程序运行时动态地确定。
- 有两种动态存储分配方式：栈式和堆式。

栈式动态存储分配

- 将整个程序的数据空间设计为一个栈
- 适用于Pascal、C和ALGOL之类的语言的实现，每当调用一个过程时，它所需的数据空间就分配在栈顶，每当过程工作结束时就释放这部分空间。
- 过程所需的数据空间包括两部分：
  - 生存期在本过程这次活动中的数据对象
  - 用以管理过程活动的记录信息

堆式动态存储分配

- 如果一个程序语言提供用户自由地申请数据空间和退还数据空间的机制，或者不仅有过程而且有进程的程序结构，即空间的使用未必服从“先申请后释放，后申请先释放”的原则，适用堆式动态存储分配。
- 假设程序运行时有一个大的存储空间，每当需要时就从这片空间中借用一块，不用时再退还。
- 空间分配机制和垃圾回收

9.2 栈式存储分配的实现

- **过程活动记录：AR(Activation Record)**

  一个过程的一次执行所需要的信息使用一个连续的存储区来管理，这个区（块）叫做一个活动记录。

- **一般这个段要记录：**

1)临时值，如计算表达式时的中间工作单元

2)局部变量

3)机器状态信息：保存运行过程前状态（返回地址.寄存器值）

4)存取链（可选）对于非局部量的引用

5)控制链（可选）指向调用者的活动记录

6)实参（形式单元）

7)返回值（对函数返回调用后的地址）。

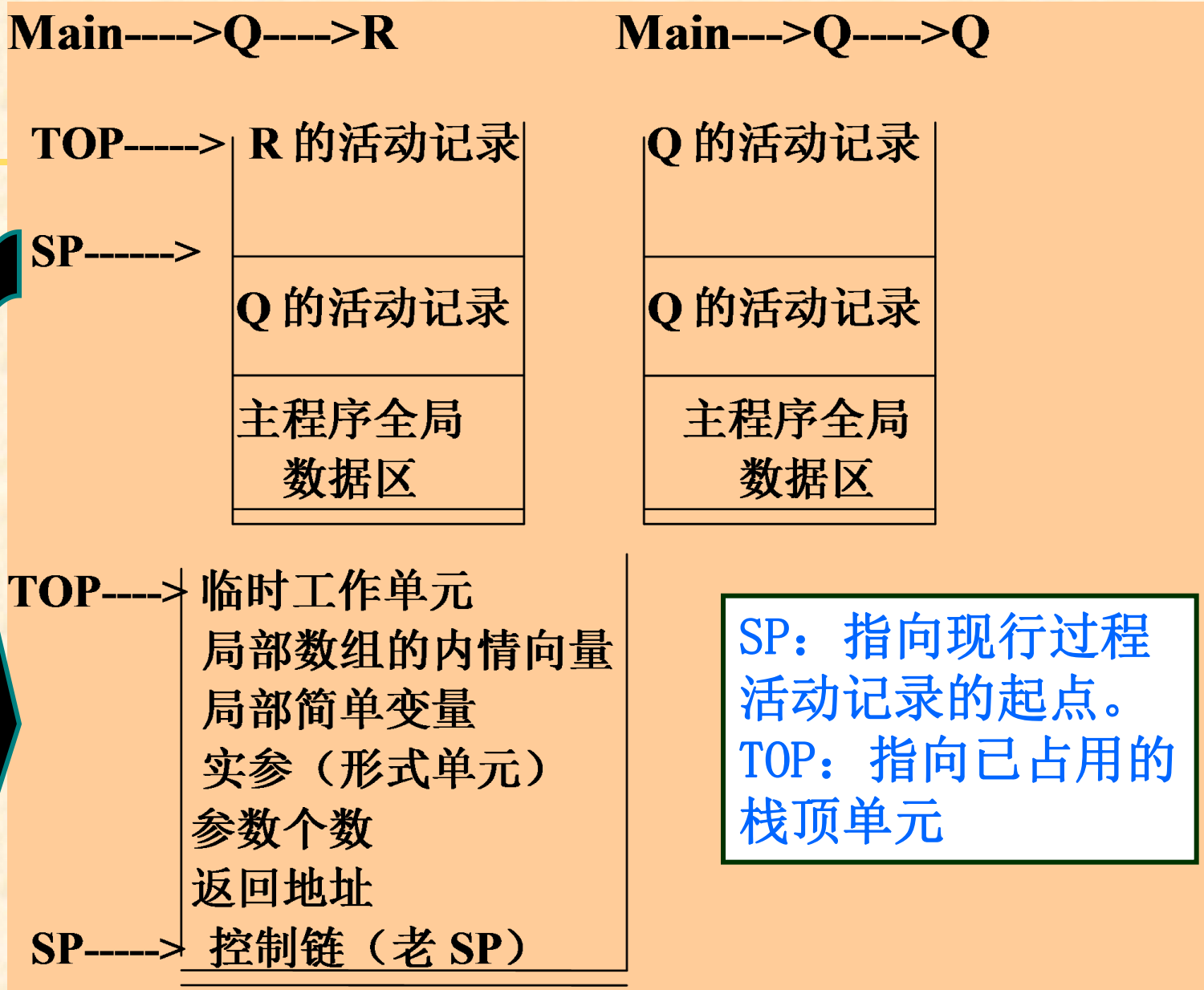
简单的栈式分配方案

- 程序结构特点：没有分程序结构，过程定义不嵌套，过程可递归调用，含可变数组。

例: main
　　全局变量或数组的说明
　　proc　R
　　……
　　end　R;
　　proc　Q
　　……
　　end　Q;
主程序执行语句
end　main

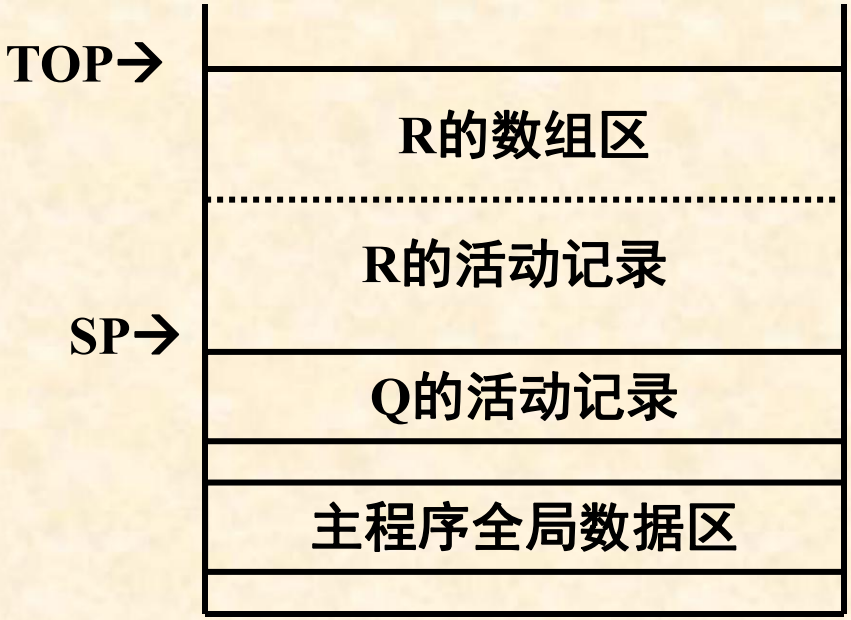
采用栈式存储分配策略，即，运行时，每当进入一个过程，则为该过程分配一段存储区，当一个过程工作完毕返回时，它所占用的存储区则可释放。

程序运行时，栈中在某一时刻可能会包含某个过程的几个活动记录；同样的一个存储位置，可能不同运行时刻分配给不同的数据对象。



无嵌套定义的过程活动记录内容

Main→Q→R：如果R中存在可变数组，则分配了数组区之后的运行栈如下：



- 在此种分配方式下，对任何局部变量x的引用都可表示为变址访问x[SP]，此处x代表变量x的相对数，也就是相对于活动记录起点的地址。

嵌套过程中的栈式分配方案

主要特点：

- （语言）一个过程可以引用包围它的任一外层过程所定义的标识符（如变量，数组或过程等）。
- （实现）一个过程可以引用它的任一外层过程的最新活动记录中的某些数据。

```
((1)) program sort(input, output)
((2))   var a:array [0..10] of integer;
((3))   x: integer;
((4))   procedure readarray;
((5))     var i:integer;
((6))   begin …a… end {readarray};
((7))   procedure exchange(i,j: integer)
((8))   begin
((9))     x:=a[i]; a[i]:=a[j]; a[j]:=x;
((10))  end{exchange};
((11))  procedure quicksort(m,n:integer)
((12))    var k, v: integer;
((13))    function partition(y,z:integer):integer;
((14))      var i,j:integer;
((15))    begin …a…
((16))      …v…
((17))      …exchange(i,j); …
((18))    end{partition};
((19))  begin…end{quicksort};
((20)) begin … end {sort}.
```

过程定义的嵌套情况：

sort

- readarray
- exchange
- quicksort
  - partition

readarray, exchange, quicksort中引用的a都不是局部变量，而是sort过程的局部变量。

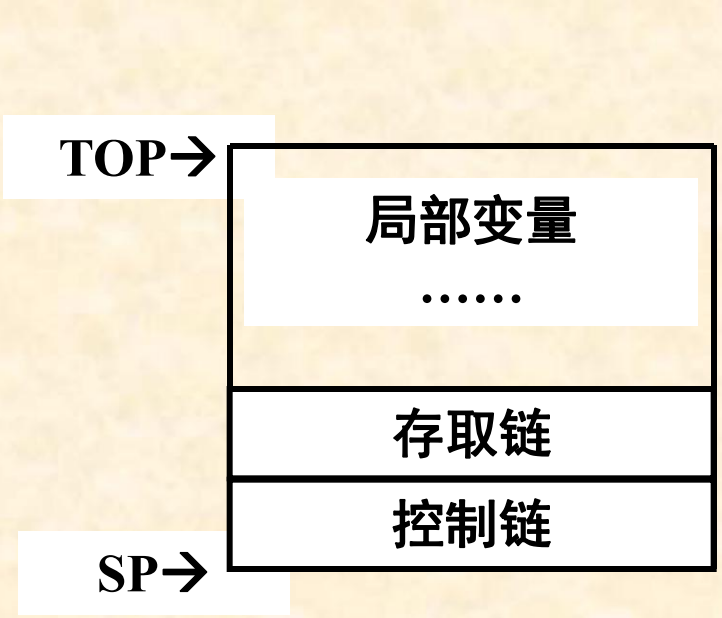
- **关键技术：**解决对非局部量的引用（存取）。
- **方法：**设法跟踪每个外层过程的最新活动记录AR的位置。
- **跟踪办法：**

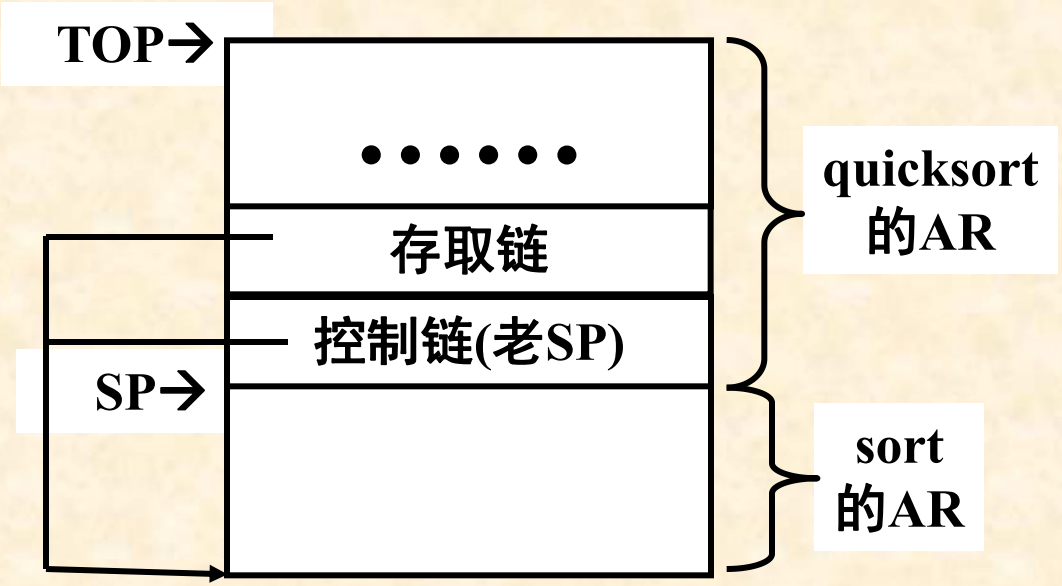
  1. 在活动记录中增设存取链，指向包括该过程的直接外层过程的最新活动记录的地址。

  2. 每进入一个过程后，在建立它的活动记录的同时建立一张嵌套层次显示表DISPLAY。

1、增设存取链

存取链指向包含该过程的直接外层过程的最新活动记录的起始位置。





嵌套定义的活动记录

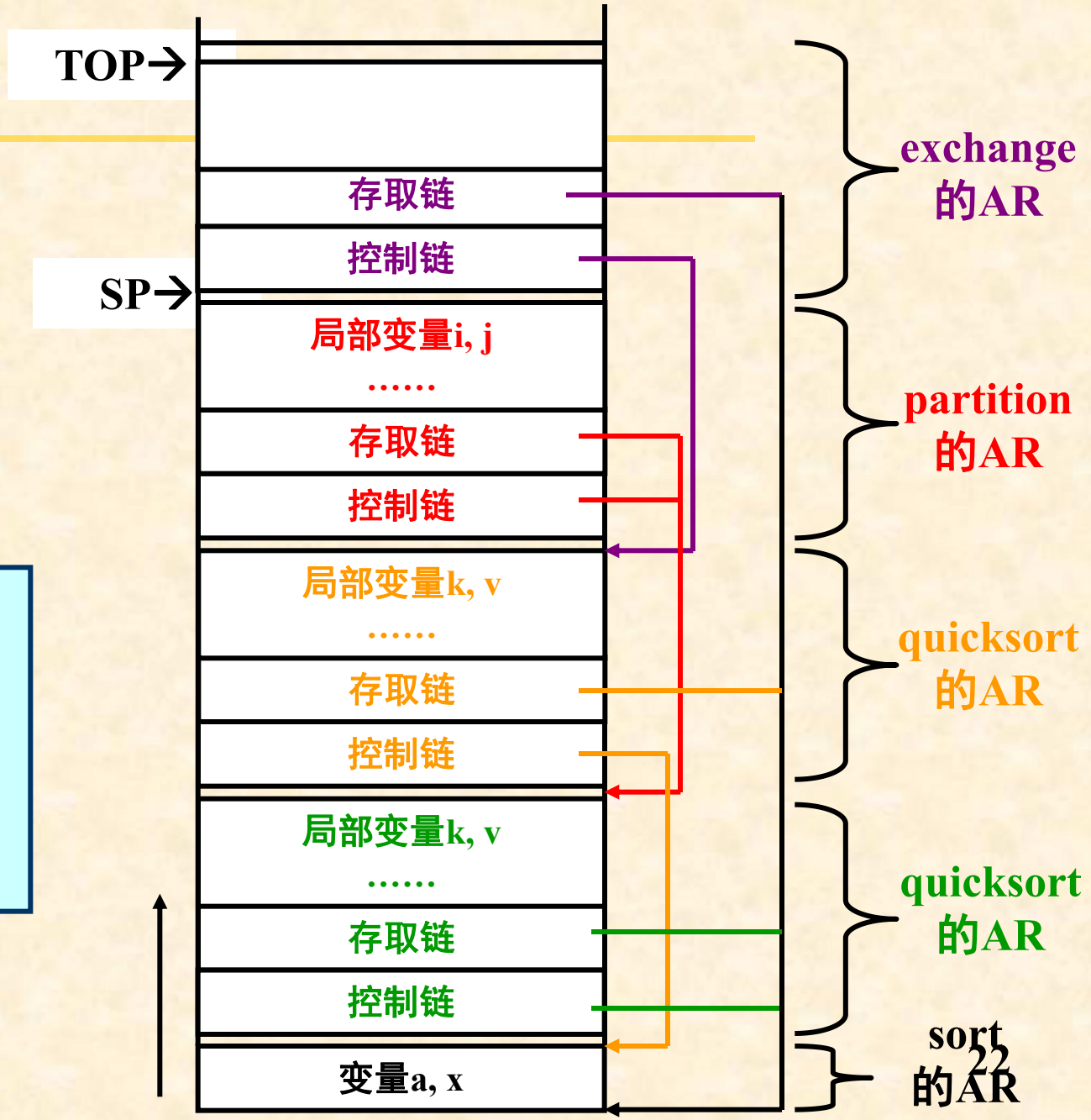
sort →quicksort的存储栈

存取链指向包含该过程的直接外层过程的最新活动记录的起始位置。

如果该程序的某执行顺序为：

sort→quicksort
→quicksort
→partition
→exchange

过程定义的嵌套情况：
sort
readarray
exchange
quicksort
partition



2、建立嵌套层次显示表

- **实现方法：**每进入一个过程后，在建立它的活动记录的同时建立**一张嵌套层次显示表display**，以此来解决非局部量的引用。
- **嵌套层次：**指过程定义的层数，可以用一个计数器来实现。
- **display表：**一个指针数组。

  设当前激活过程的层次为K，它的Display表含有K+1个单元，依次存放着现行层，直接外层…直至最外层的每一过程的最新活动记录的基地址。

```
((1)) program sort(input, output)
((2))     var a:array [0..10] of integer;
((3))     x: integer;
((4))     procedure readarray;
((5))         var i:integer;
((6))     begin …a… end {readarray};
((7))     procedure exchange(i,j: integer)
((8))     begin
((9))         x:=a[i]; a[i]:=a[j]; a[j]:=x;
((10))    end{exchange};
((11))    procedure quicksort(m,n:integer)
((12))        var k, v: integer;
((13))        function partition(y,z:integer):integer;
((14))            var i,j:integer;
((15))        begin …a…
((16))            …v…
((17))            …exchange(i,j); …
((18))        end{partition};
((19))    begin…end{quicksort};
((20)) begin … end {sort}.
```

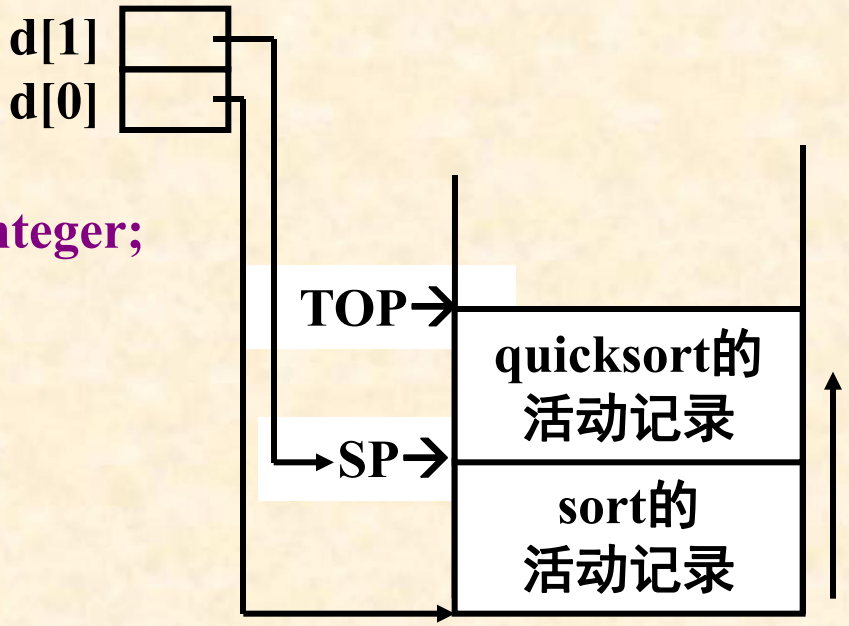
调用情况：

sort→quicksort

…

quicksort的

display表



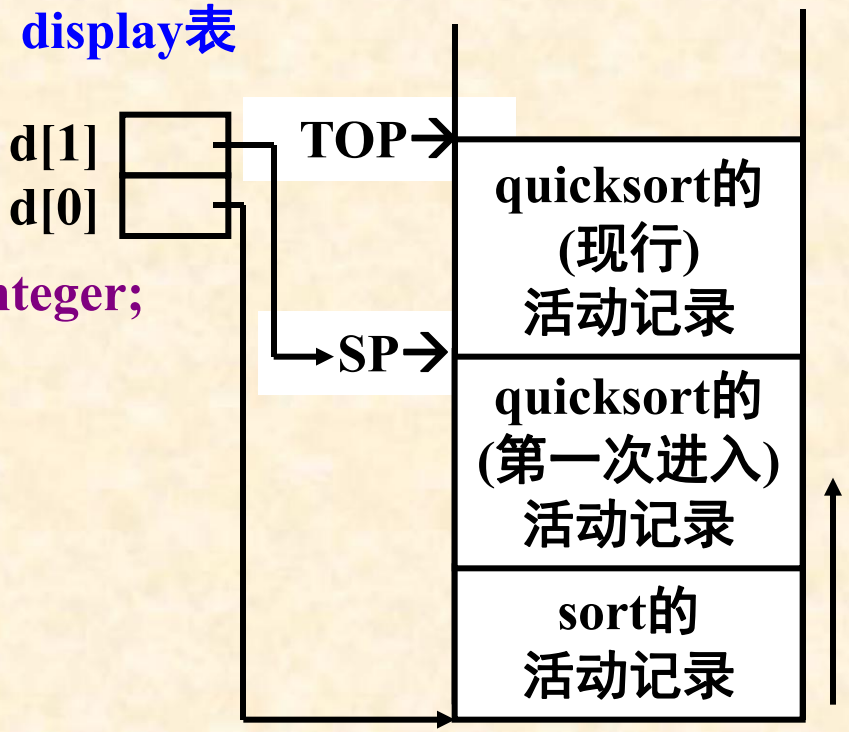
```
((1)) program sort(input, output)
((2))     var a:array [0..10] of integer;
((3))     x: integer;
((4))     procedure readarray;
((5))         var i:integer;
((6))     begin …a… end {readarray};
((7))     procedure exchange(i,j: integer)
((8))     begin
((9))         x:=a[i]; a[i]:=a[j]; a[j]:=x;
((10))    end{exchange};
((11))    procedure quicksort(m,n:integer)
((12))        var k, v: integer;
((13))        function partition(y,z:integer):integer;
((14))            var i,j:integer;
((15))        begin …a…
((16))            …v…
((17))            …exchange(i,j); …
((18))        end{partition};
((19))    begin…end{quicksort};
((20)) begin … end {sort}.
```

调用情况：

sort→quicksort

→quicksort …

quicksort的（现行）display表



```
((1)) program sort(input, output)
((2))     var a:array [0..10] of integer;
((3))     x: integer;
((4))     procedure readarray;
((5))         var i:integer;
((6))     begin …a… end {readarray};
((7))     procedure exchange(i,j: integer)
((8))     begin
((9))         x:=a[i]; a[i]:=a[j]; a[j]:=x;
((10))    end{exchange};
((11))    procedure quicksort(m,n:integer)
((12))        var k, v: integer;
((13))        function partition(y,z:integer):integer;
((14))            var i,j:integer;
((15))        begin …a…
((16))            …v…
((17))            …exchange(i,j); …
((18))        end{partition};
((19))    begin…end{quicksort};
((20)) begin … end {sort}.
```

调用情况：

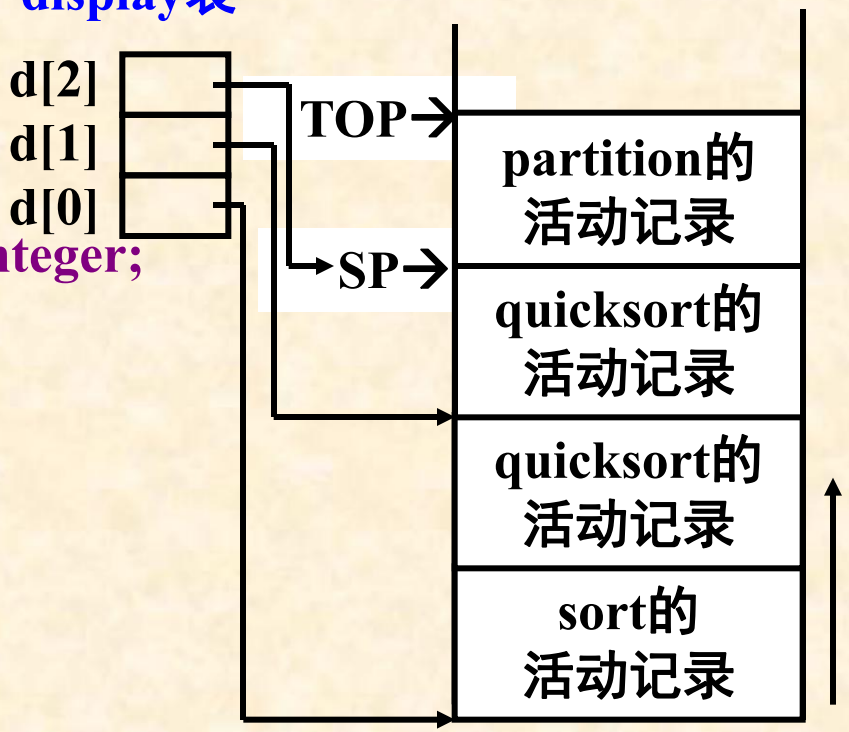
sort→quicksort

→quicksort

→partition …

partition的

display表



```
((1)) program sort(input, output)
((2))     var a:array [0..10] of integer;
((3))     x: integer;
((4))     procedure readarray;
((5))         var i:integer;
((6))     begin …a… end {readarray};
((7))     procedure exchange(i,j: integer)
((8))     begin
((9))         x:=a[i]; a[i]:=a[j]; a[j]:=x;
((10))    end{exchange};
((11))    procedure quicksort(m,n:integer)
((12))        var k, v: integer;
((13))        function partition(y,z:integer):integer;
((14))            var i,j:integer;
((15))        begin …a…
((16))            …v…
((17))            …exchange(i,j); …
((18))        end{partition};
((19))    begin…end{quicksort};
((20)) begin … end {sort}.
```

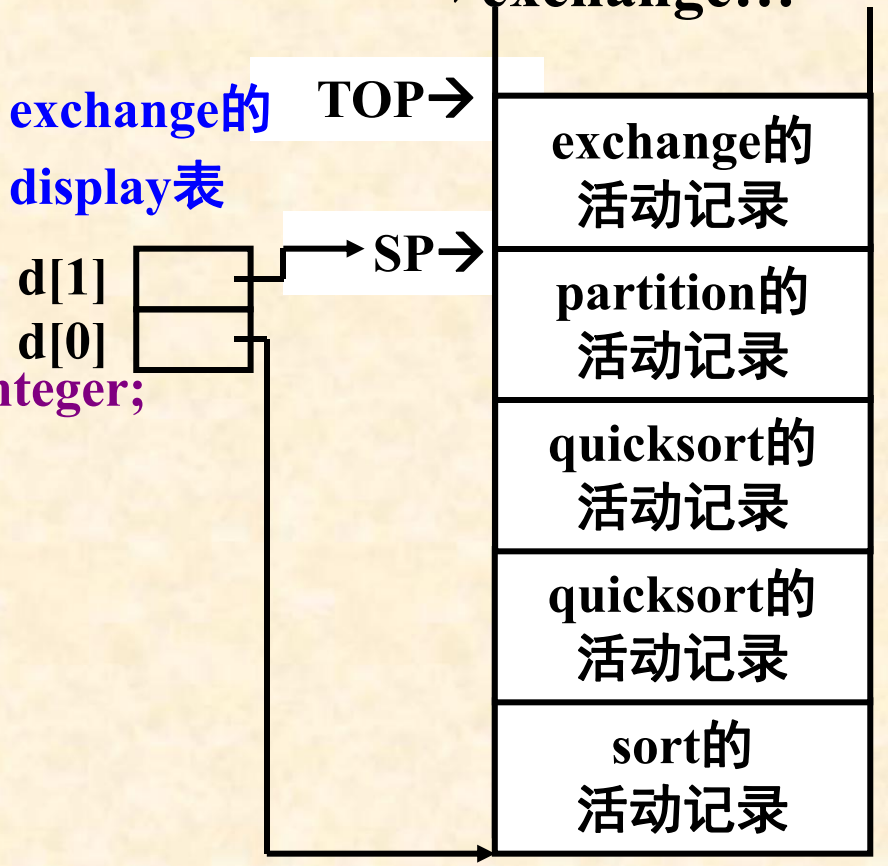
调用情况：

sort→quicksort

→quicksort

→partition

→exchange…



display表的存储结构

- display本身可以作为单独的表分配存储，也可以作为活动记录的一部分(如下图所示)。

分程序结构的存储管理

- 一个分程序是一个含有它自己的局部数据(变量)声明的语句。
- 分程序结构的语言中一个声明的作用域是遵循**最近嵌套原则**的，即
  - 一个分程序A中的一个声明的作用域包含在A中。
  - 如果某个名字x未在分程序A中声明，则必须在它的最近包围外层中被声明。

分程序结构举例

ALGOL语言：

```
Procedure  A(m,n);  integer  m,n;
 B1:begin   real   z; array B[m:n];
          B2:begin    real   d,  e;
               L3:
               end;
          B4:begin    array C[1:m];
               B5:begin    real    e;
                    L6:
                    end;
               end;
   L8:end;
```

分程序结构的存储分配方案

- 简单办法:把分程序看成“无名无参过 程”，在哪里定义就在哪里被调用。但效率很低、原因：

  一：每进入 一个分程序，就照样建立连接数据和DISPLAY表,这是不必要的

  二：当从内层分程序向外层转移时，同时要结束多个分程序。

- 按照过程处理办法，意味着必须一层一层地通过“返回” 来恢复所要到达的那个分程序的数据区，但不能直接到达。
- 例如：如果有一个从第5层分程序转出到达第1层分程序的标号L，虽然在第5层分程序工作时知道L所属的层数，我们极易从DISPLAY中获得第1层分程序的活动记录基址（SP），但是怎么知道第1层分程序进入时的TOP呢？唯一的办法是从 5,4,3和2各层顺序退出。但这种办法是很浪费时间的。

为了解决上述问题，可采取两种措施。

- 第一，对每个过程或分程序都建立有自己的栈顶指示器TOP，代替原来仅有过程的栈顶指示器，每个TOP的值保存在各自活动记录中。
- 第二，不把分程序看作“无参过程”，每个分程序享用包围它的那个最近过程的DISPLAY。每个分程序都隶属于某个确定的过程，分程序的层次是相对于它所属的那个过程进行编号的。

每个过程被当作是0层分程序。而过程体分程序当作是它所管辖的第1层分程序。这样，每个过程的活动记录所含的内容有：

1. 过程的TOP值，它指向过程活动记录的栈顶位置。

2. 连接数据，共四项：

(1)老SP值； (2)返回地址； (3)全局DISPAY地址；

(4)调用时的栈顶单元地址，老TOP。

3. 参数个数和形式单元；

4. DISPAY表；

5. 过程所辖的各分程序的局部数据单元。对于每个分程序来说，它们包括：

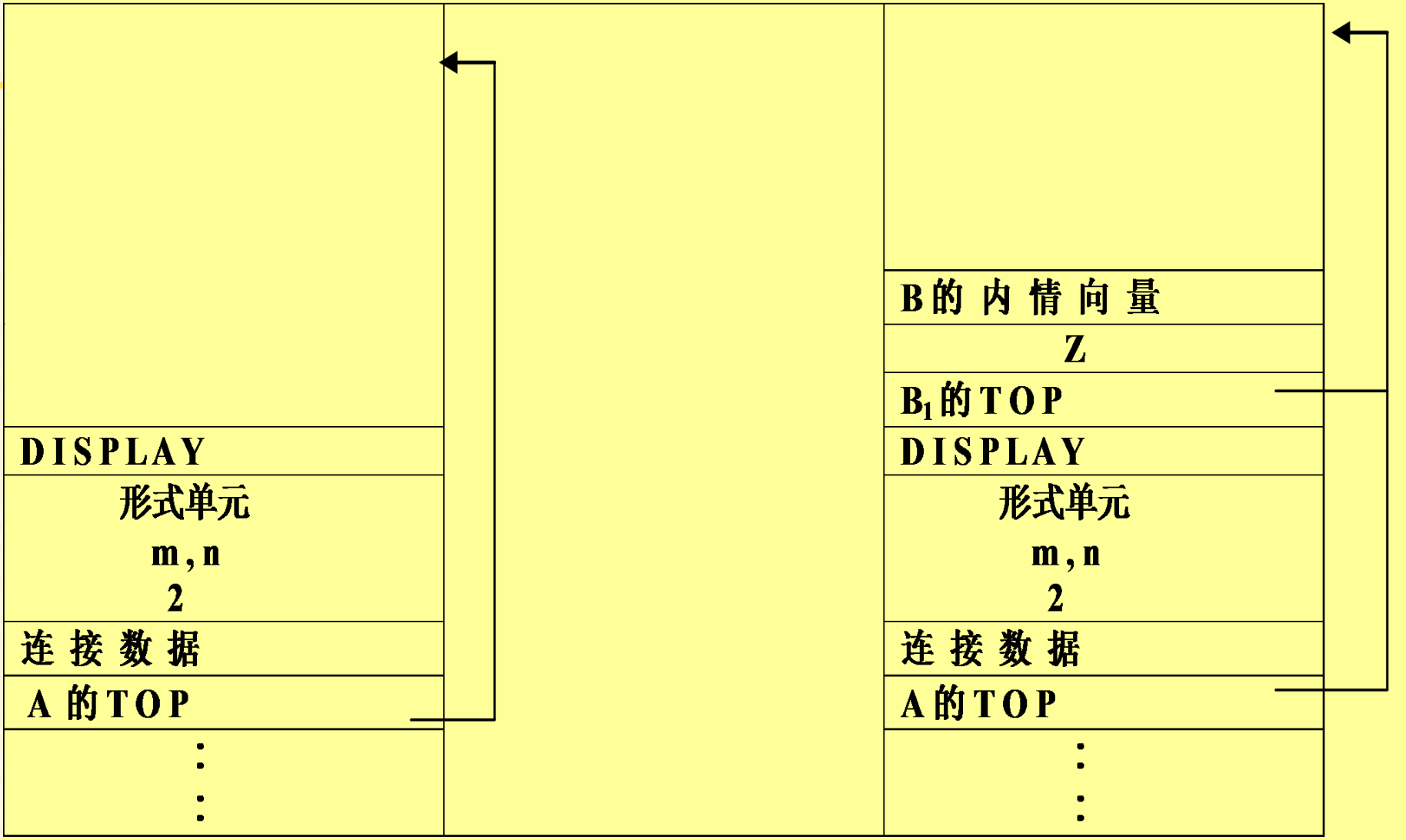
(1)分程序的TOP值。当进入分程序时它含现行栈顶地址，以后，用来定义栈的新高度（分程序的TOP值）；

(2)分程序的局部变量， 数组内情向量和临时工作单元。

	变量e	变量e和d
	B_5 的TOP	
	数组C的内情向量	
	B_4 的TOP	B_2的TOP
	数组B的内情向量	
	变量z	
K	B_1 的TOP	
D	DISPLAY	
6	形式单元m,n	
5	参数个数: 2	
4	调用时的栈顶地址(老TOP)	
3	全局DISPLAY地址	
2	返回地址	
1	老SP	
0	过程的TOP, 指向活动记录之顶	

ALGOL语言:

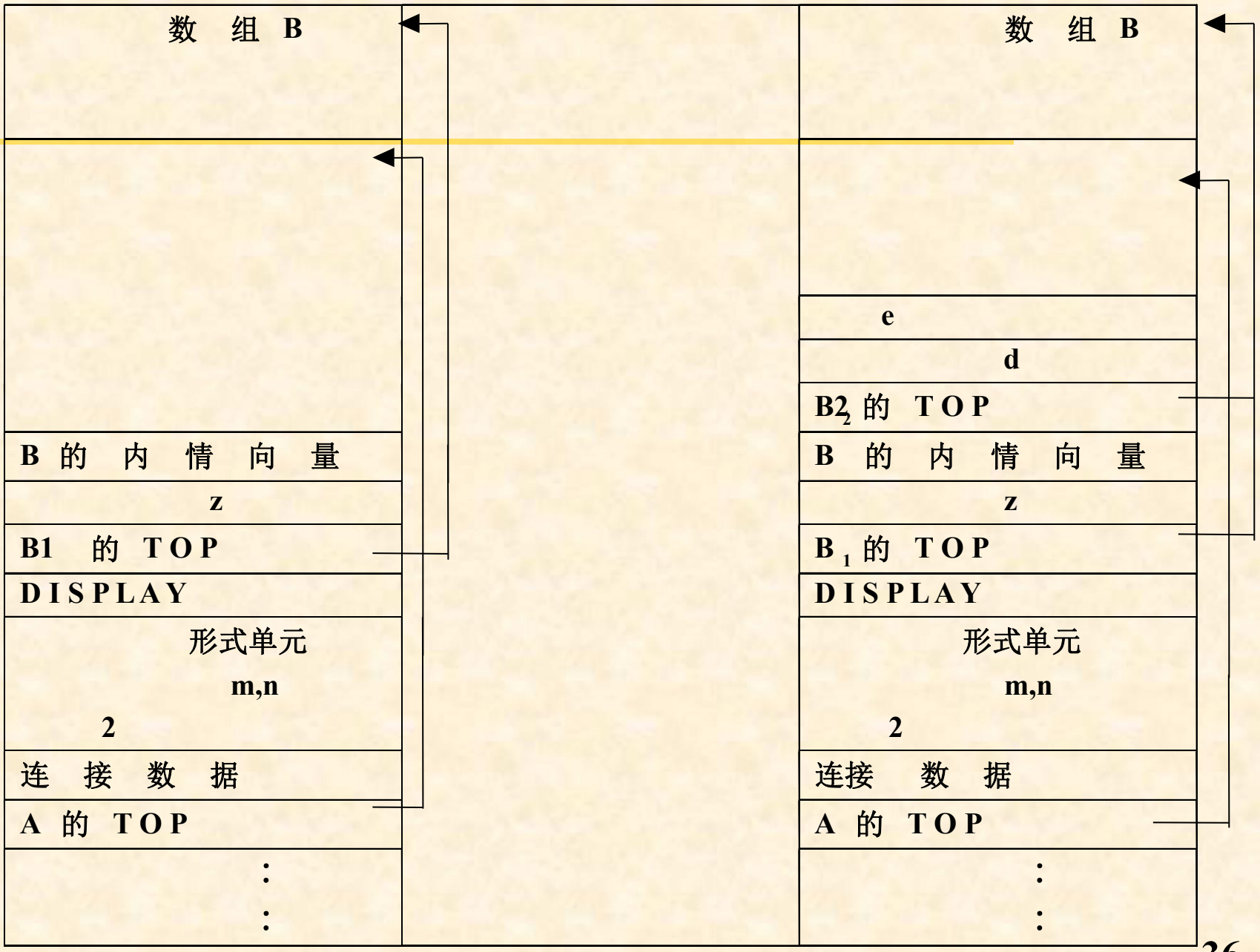
```
Procedure A(m,n); integer m,n;
B1:begin real z; array B[m:n];
        B2:begin real d, e;
              L3:
              end;
        B4:begin array C[1:m];
              B5:begin real e;
                    L6:
                    end;
              end;
   L8:end;
```



(a) (b)

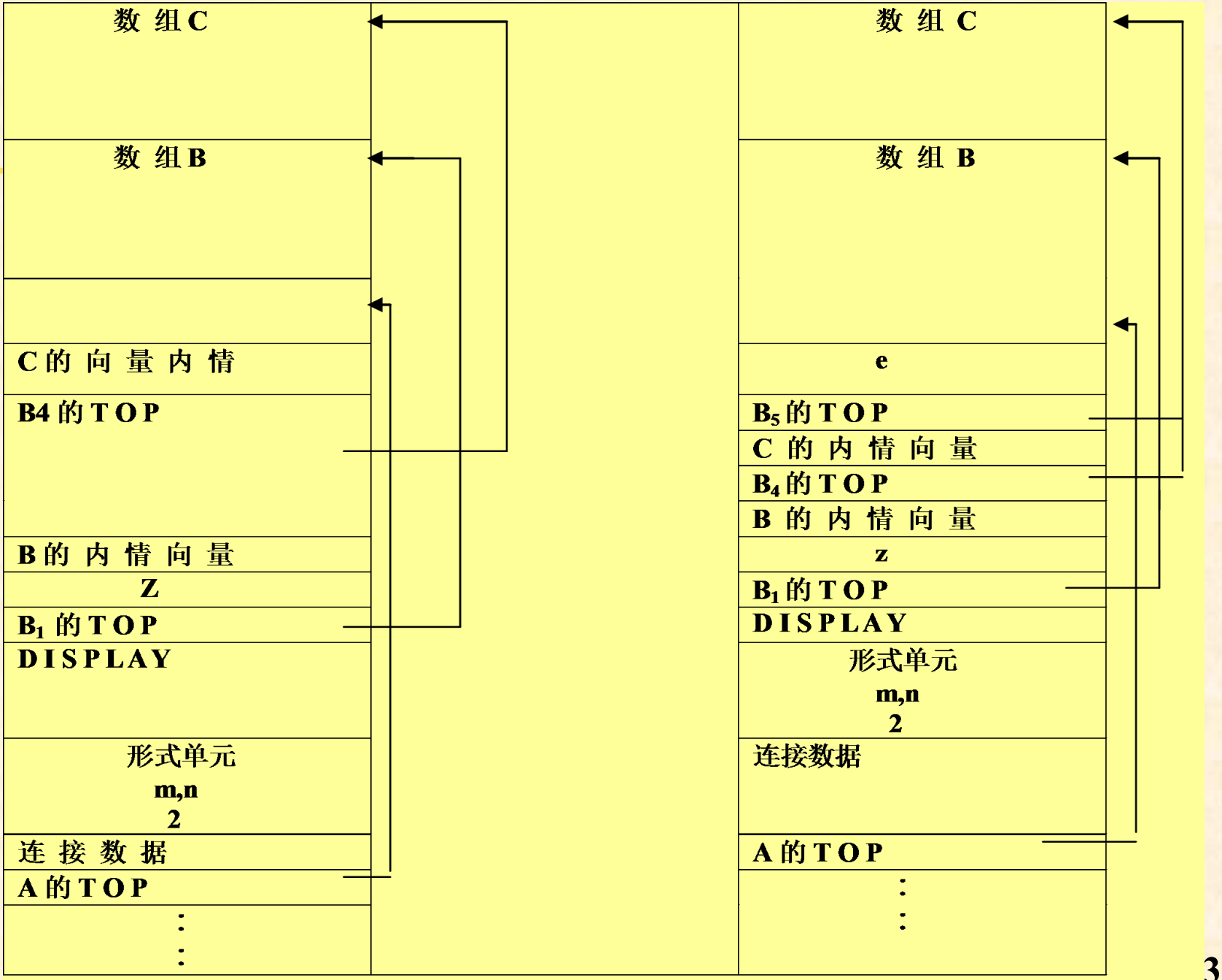
(a) 到达标号B_1处；

(b)进入分程序B_1;



(c)数组 B 分配之后；

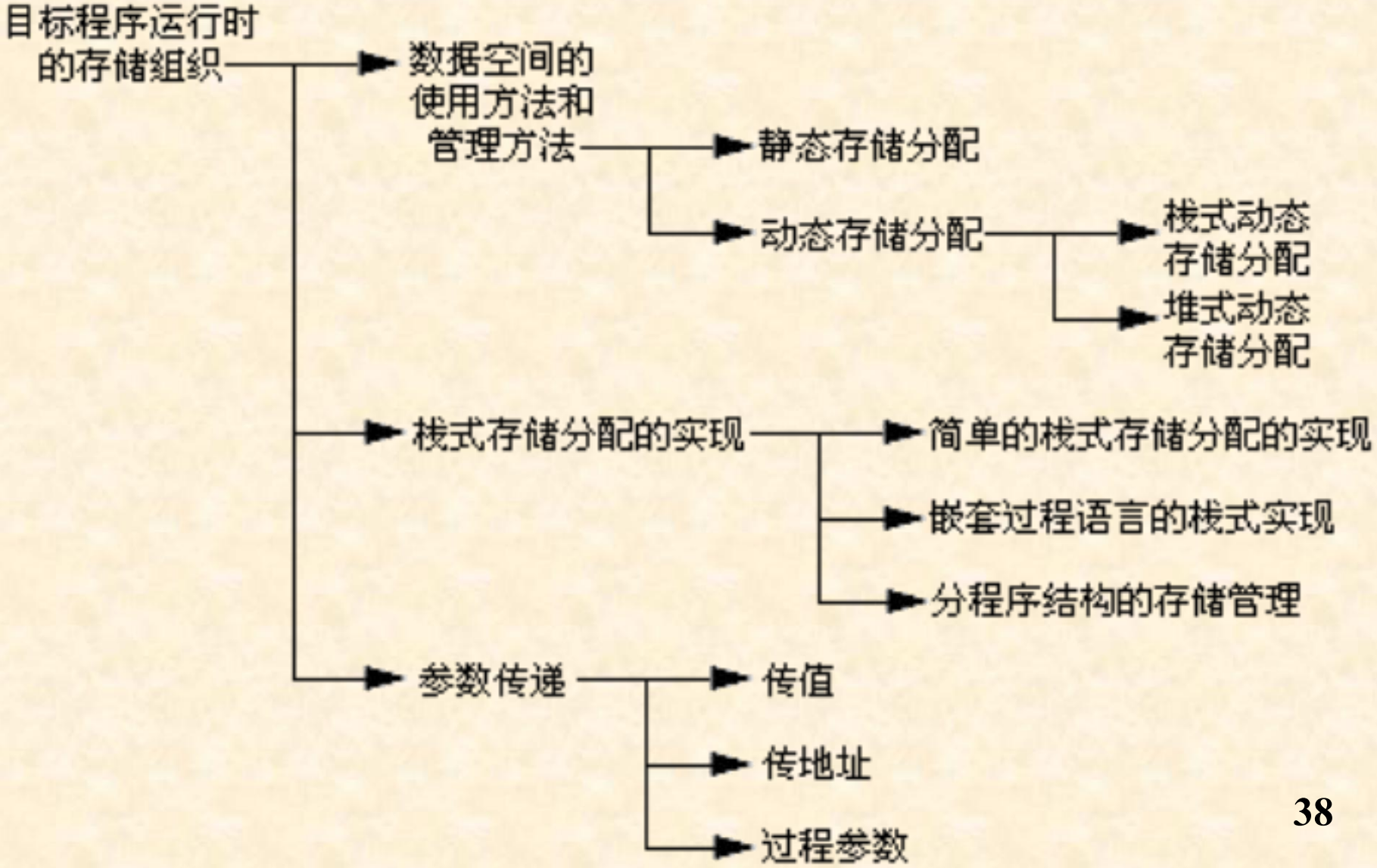
（d）进入分程序 B2



(e)进入分程序 B_4 分配数组 C 之后；

(f)进入分程序 B_5 .

复 习


目标程序运行时的存储组织
数据空间的使用方法和管理方法
静态存储分配
动态存储分配
栈式动态存储分配
堆式动态存储分配
栈式存储分配的实现
简单的栈式存储分配的实现
嵌套过程语言的栈式实现
分程序结构的存储管理
参数传递
传值
传地址
过程参数

9.3 参数传递

```
(1)procedure exchangel(i,j:integer);
(2)     var x:integer;
(3)     begin;
(4)         x:=a[i];  a[i]:=a[j];  a[j]:=x
(5)     end;
```

带有非局部变量和形参的PASCAL过程。
非局变量a[i]和a[j]的值进行交换，i,j为形参。

```
(1)program  reference(input,output);
(2)var  a,b:integer;
(3)procedure  swap({var}  x,y:integer);
(4)        var  temp:integer;
(5)        begin
(6)              temp:=x;
(7)                   x:=y;
(8)                  y:=temp
(9)        end;
(10)begin
(11)       a:=1;  b:=2;
(12)       swap(a,b);
(13)       writeln('a=',a);writeln('b=',b)
(14)end.
```

带有过程swap的PASCAL程序

- 传地址（变量参数）

  例如：过程 swap(var x,y:integer);

  swap(&a,&b）；（ a,b为调用时的实参 ）

  调用结果a,b的值被改变。

- 传值（值调用）

特点是对形式参数的任何运算不影响实参的值。

  例如：过程 swap(x,y:integer);

  swap(a,b）；其结果： a,b调用前的值不改变。

传值的实现

- 1.形式参数当作过程的局部变量处理，即在被调过程的活动记录中开辟了形参的存储空间，这些存储位置即是我们所说的形式单元（用以存放实参）。
- 2.调用过程计算实参的值，并将其放在对应形式单元开辟的空间中。
- 3.被调用过程执行时，就像使用局部变量一样使用这些形式单元。

```
procedure  swap(  x,y:integer);
      var  temp:integer;
      begin   temp:=x;   x:=y;      y:=temp
      end;
```

调用swap(a,b) 过程将不会影响a和b的值。

其结果等价于执行下列运算：

```
x :=a;
y :=b;
temp :=x;
x :=y;
y :=temp
```

传地址的实现

（call- by- reference)(call-by-address)(call-by-location)

- 把实在参数的地址传递给相应的形参，即调用过程把一个指向实参的存储地址的指针传递给被调用过程相应的形参：

1. 实在参数是一个名字，或具有左值的表达式----传递左值
2. 实在参数是无左值的表达式----计算值，放入一存储单元，传此存储单元地址

✓ 目标代码中，被调用过程对形参的引用变成对传递给被调用过程的指针的间接引用

```
(1)swap(x,y)
(2)int   *x,*y;
(3){        int   temp;
(4)             temp=*x;  *x=*y;  *y=temp;
(5)}
(6)main(       )
(7){   int   a=1,b=2;
(8)       swap(&a,&b);
(9)       printf("a is now %d,b is now %d\n",a,b);
(10)}
```

在一个值调用过程中使用指针的C程序在C程序中无传地址所以用指针实现。

过程作为参数传递

一个嵌套过程(函数)可以作为参数传递

```
(1)program param(input,output);
(2)procedure b(function h(n:integer):integer);
(3)       begin  writeln(h(2))  end{b};
(4)procedure  c;
(5)       var m:integer;
(6)       function  f(n:integer):integegr;
(7)            begin  f:=m+n    end{f};
(8)begin m := 0; b(f)  end {c};
(9)begin
(10) c
(11)end
```

当一个嵌套过程作为参数传递时，必须连同它的的存取链（display）一同传递过去。

嵌套过程作为参数传递

9.4 过程调用、过程进入和过程返回

- **过程调用的四元式序列：**

  par T_1

  par T_2

  …

  par T_n

  call P, n

对于par和call产生的目标代码

1) 每个param x_i(i=1,2,…n)可直接翻译成如下指令:

(i+3)[top]:= x_i (传值)

(i+3)[top]:=addr(x_i) (传地址)

2) call p，n 被翻译成如下指令:

1[top]:=sp (保护现行sp)

3[top]:=n (传递参数个数)

JSR p (转子指令)

3) 进入过程p后，首先应执行下述指令:

sp:=top+1　　(定义新的sp)

1[sp]:=返回地址　(保护返回地址)

top:=top+L　　(新top)

L：过程P的活动记录所需单元数，

在编译时可确定。

4) 过程返回时，应执行下列指令:

top:=sp-1　　(恢复调用前top)

sp:=0[sp]　　(恢复调用前SP)

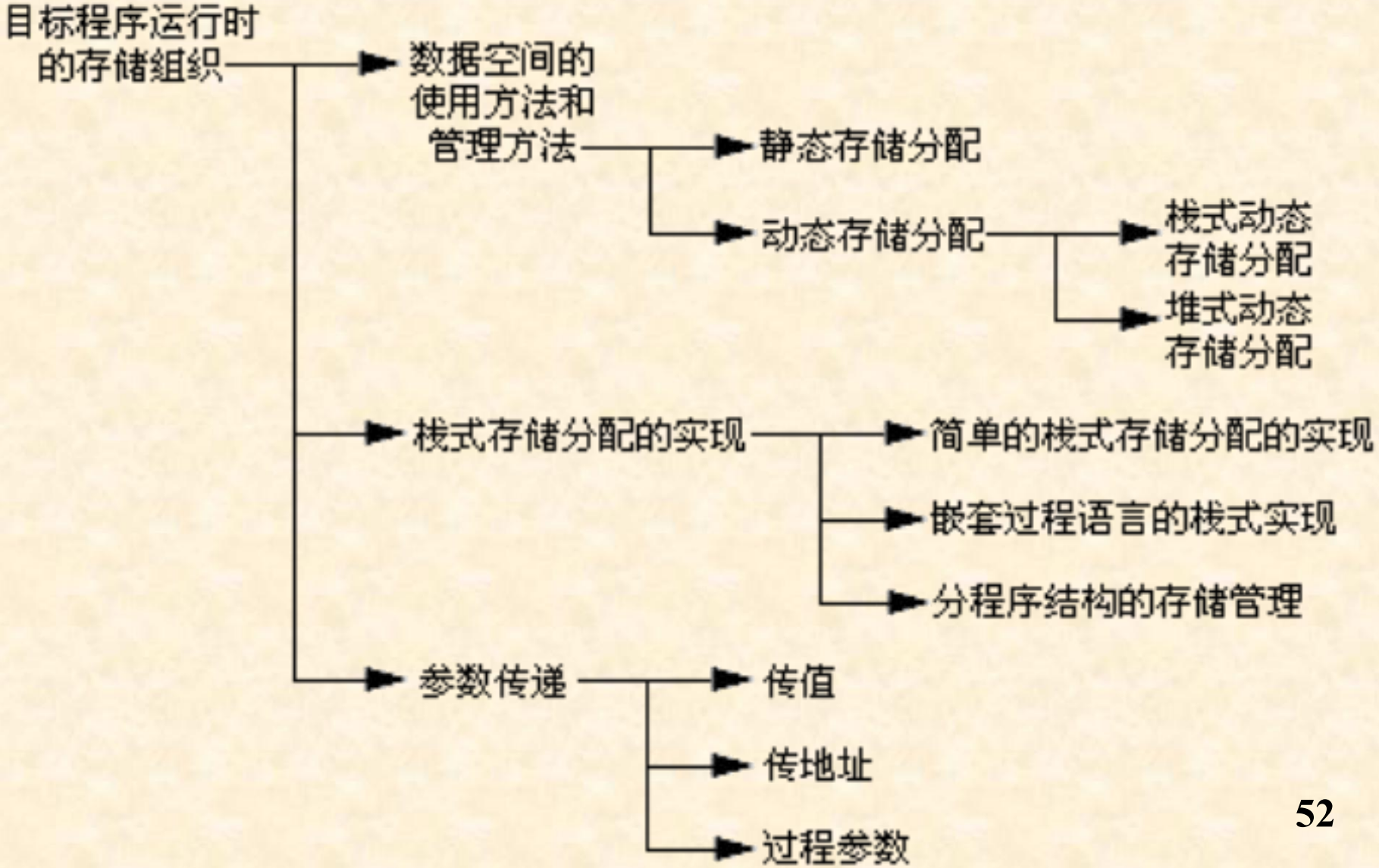
X:=2[top]　　(把返回地址取到X中)

UJ 0[X]　(按X返回)

UJ为无条件转移指令，

即按X中的返回地址实行变址转移

总结


目标程序运行时的存储组织
数据空间的使用方法和管理方法
静态存储分配
动态存储分配
栈式动态存储分配
堆式动态存储分配
栈式存储分配的实现
简单的栈式存储分配的实现
嵌套过程语言的栈式实现
分程序结构的存储管理
参数传递
传值
传地址
过程参数