山东大学
SHANDONG UNIVERSITY

编译原理

第九章 运行时存储空间组织

授 课 教 师 ：郑艳伟
手 机 ：18614002860（微信同号）
邮 箱 ：zhengyw@sdu.edu.cn

第九章 运行时存储空间组织

第九章 运行时存储空间组织

- 运行时存储空间组织：代码运行时刻，源代码中的各种变量、常量等用户定义的量是如何存放的，如何去访问它们？
  - 在程序语言中，程序中使用的存储单元都由标识符表示，它们对应的内存地址由编译程序在编译时或由其生成的目标程序在运行时分配；
  - 存储组织与管理，就是将标识符和存储单元关联起来，进行存储分配、访问和释放。

第九章 运行时存储空间组织

```
① program sort (input, output)
②     var a: array[0..10] of integer;
③     procedure readarray;                          // 嵌套的过程定义
④         var i: integer;
⑤         begin
⑥             for i := 1 to 9 do read(a[i])
⑦         end;
⑧     function partition(y, z: integer) : integer;  // 嵌套的函数定义，y, z是形参
⑨         var i : integer;  begin ... end;
⑩     procedure quicksort(m, n: integer)            // 嵌套的过程定义，m,n是形参
⑪         var i: integer;
⑫         begin
⑬             if (n > m) then begin
⑭                 i := partition(m, n);             // 表达式中的函数调用，实参为m,n
⑮                 quicksort(m, i - 1);              // 过程调用，实参为m和i-1
⑯                 quicksort(i + 1, n);              // 过程调用，实参为i+1和n
⑰             end;
⑱         end;
⑲     begin
⑳         a[0] := -9999; a[10] := 9999;
21        readarray;                                // 主程序调用过程
22        quicksort(1, 9);                          // 主程序调用过程，实参为1,9
23    end;
```

9.1.1 过程的活动

- 一个过程的活动指该过程的一次执行。
- 关于过程P的一个活动的生存期，指从执行该过程体第一步操作到最后一步操作之间的操作序列。
  - 在像Pascal这样的语言里，每次控制流从过程P进入过程Q后，如果没有错误，最后都返回过程P；
  - 如果a和b都是过程的活动，那么它们的生存期或者是不重叠的，或者是嵌套的
- 一个过程是递归的，指该过程在没有退出当前活动时，又开始其新活动。
  - 一个递归过程P并不一定需要直接调用它本身，它可以通过调用过程Q，而Q经过若干调用又调用P。

9.1.1 过程的活动

- 语言中的说明是规定名称含义的语法结构。
  - 显式说明，如Pascal的变量说明var i: integer；
  - 隐式说明，如Fortran中，在无其它说明的情况下，认为变量i是整型的。
- 一个说明在程序里起作用的范围称为该说明的作用域。
  - 如果一个说明的作用域是在一个过程里，那么在这个过程里出现的该说明中的名称都是局部于本过程的；
  - 除此之外的名称就是非局部的。

第九章 运行时存储空间组织

- 9.1 目标程序运行时的活动
  - 9.1.1 过程的活动
  - 9.1.2 参数传递
- 9.2 运行时存储器的划分
  - 9.2.1 运行时存储器的划分
  - 9.2.2 活动记录
  - 9.2.3 存储分配策略
- 9.3 静态存储分配
- 9.4 简单的栈式存储分配
  - 9.4.1 C的活动记录
  - 9.4.2 C的栈式动态分配
- 9.5 嵌套过程语言的栈式实现
  - 9.5.1 非局部名字的访问的实现
  - 9.5.2 参数传递的实现
- 9.6 堆式动态存储分配
  - 9.6.1 堆式动态分配的实现
  - 9.6.2 隐式存储回收

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

9.1.2 参数传递

- 参数
  - 形式参数，简称形参（Fortran中称为哑元），即函数的自变量，其初值来源于函数的调用，函数被调用之前并不分配内存；
  - 实在参数，简称实参，其本质是一个变量，已经占用内存空间，函数调用时实参赋值给形参。
- 实参如何传递给形参？
  - 传值（call by value）
  - 传地址（call by address ）
  - 引用（call by reference ）
  - 传名字（call by name）
- 函数工作完毕返回时，如何把函数值送回？
  - 编译器可以把函数值保留在某个寄存器中。

9.1.2 参数传递

- 传值
  - 调用段把实参的值计算出来，存放到一个被调用段可以拿的到的地方；
  - 被调用段开始工作时，把这些值抄进自己的形式单元中；
  - 使用形参时，就像使用局部变量一样使用这些形式单元。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

9.1.2 参数传递

- 传地址：把实在参数的地址作为值传递给相应的形式参数
  - 在过程段中每个形参都有一个相应的单元，称为形式单元，用来存放相应的实参地址；
  - 当调用一个过程时，调用段必须把实参地址传递到被调用段可以拿得到的地方
    - 如果实参是一个变量，则直接传递它的地址；
    - 如果实参是常数或表达式，则先计算它的值，并存放到一个临时单元，然后传递这个临时单元的地址。
- 程序控制转入被调用段后
  - 被调用段首先把实参地址抄进自己相应的形式单元；
  - 过程体对形式参数的任何引用或赋值，都被处理成对形式单元的间接访问；
  - 被调用段工作完毕返回时，形式单元所指的实参单元就持有了所期望的值。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

9.1.2 参数传递

- 引用：把实在参数的地址传递给相应的形式参数
  - 以地址的方式传递参数；
  - 传递以后，形参和实参都是同一个对象，只是它们名字不同。

```
1  void Swap(int &x, int &y)
2  {
3    int a = x;
4    x = y;
5    y = a;
6  }
7  int main()
8  {
9    int a = 0, b = 1;
10   Swap(a, b);
11   printf("a␣=␣%d,␣b␣=␣%d\n", a, b);
12   return 0;
13 }
```

```
1  void Swap(int *x, int *y)
2  {
3    int a = *x;
4    *x = *y;
5    *y = a;
6  }
7  int main()
8  {
9    int a = 0, b = 1;
10   Swap(&a, &b);
11   printf("a␣=␣%d,␣b␣=␣%d\n", a, b);
12   return 0;
13 }
```

9.1.2 参数传递

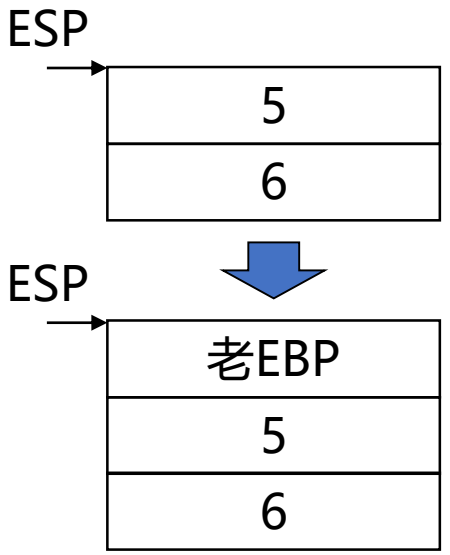
- 传结果（call by result）：与传地址类似但不完全等价
  - 每个形参对应两个单元，第一个单元存放实参地址，第二个单元存放实参值；
  - 过程体中对形式参数的任何引用或赋值，都看成是对第二个单元的直接访问；
  - 过程工作返回前，把第二个单元的内容存放到第一个单元所指的实参单元中。
- 传名字：是Algol60定义的一种特殊的形-实参数结合方式
  - 过程调用段作用相当于把被调用段的过程体抄到调用出现的地方；
  - 把其中任一出现的形参都替换成相应的实参（文字替换）；
  - 如果在替换时发现过程体中的局部名和实参中的名字相同，则必须用不同的标识符来表示这些局部名；
  - 为了表现实在参数的整体性，必要时在替换前先把它用括号括起来。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

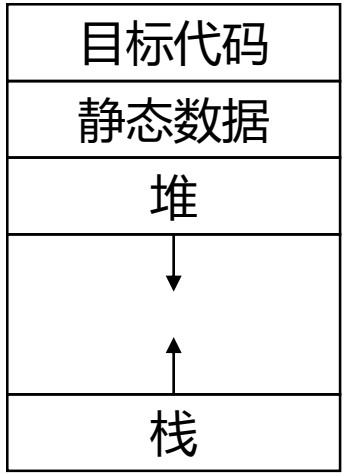
9.1.2 参数传递

- C调用规范(cdecl)

```
Example proc
    push 6
    push 5
    call AddTwo
    add esp, 8    ; 从堆栈移除传递的参数
    ret
Example endp
```




```
AddTwo proc
    push ebp
    mov ebp, esp    ; 堆栈帧的基值
    mov eax, [ebp + 12]    ; 第二个参数
    add eax, [ebp + 8]      ; 第一个参数
    pop ebp
    ret
AddTwo endp
```



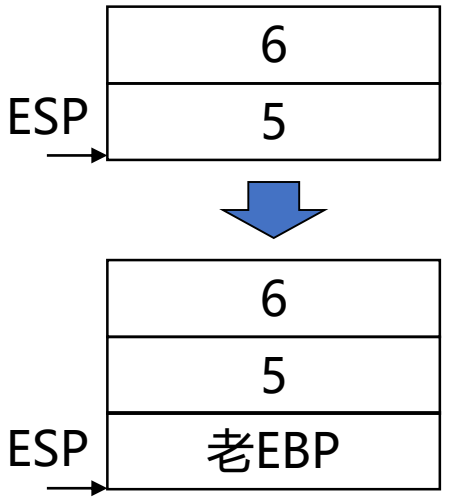
山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

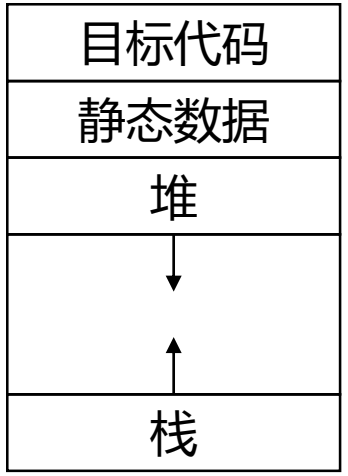
9.1.2 参数传递

- StdCall

```
Example proc
    push 6
    push 5
    call AddTwo
    ; add esp, 8 这个没有了
    ret
Example endp

AddTwo proc
    push ebp
    mov ebp, esp    ; 堆栈帧的基址
    mov eax, [ebp + 12]    ; 第二个参数
    add eax, [ebp + 8]     ; 第一个参数
    pop ebp
    ret 8                  ; 清除堆栈
AddTwo endp
```





山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

9.1.2 参数传递

```
1  void Fun(uint x, uint y)
2  {
3    uint a, b, var;
4    a = x + y;
5    b = x + y;
6    var = a * b;
7  }
```

(100) [+, x, y, $1]
(101) [=, $1, -, a]
(102) [+, x, y, $2]
(103) [=, $2, -, b]
(104) [*, a, b, $3]
(105) [=, $3, -, var]

(100) [+, x, y, $1]
(101) [=, $1, -, a]
(102) [=, $1, -, $2]
(103) [=, $1, -, b]
(104) [*, $1, $1, $3]
(105) [=, $3, -, var]

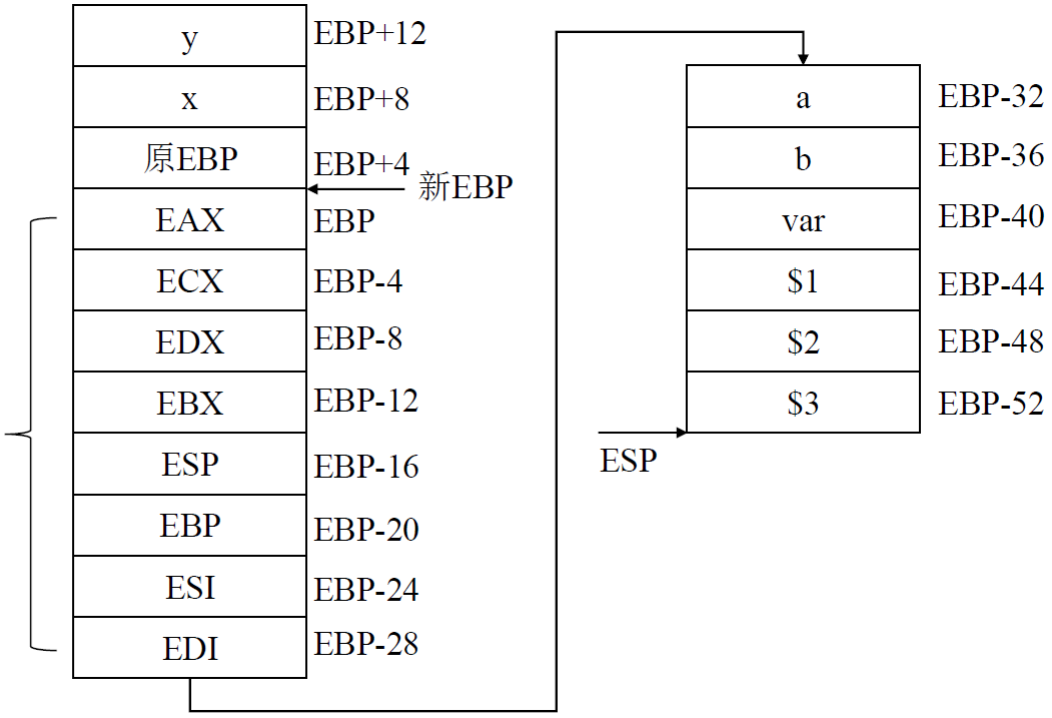
名字	类别	类型	字宽	偏移量
a	局部变量	dword	4	0
b	局部变量	dword	4	4
var	局部变量	dword	4	8
$1	临时变量	dword	4	12
$2	临时变量	dword	4	16
$3	临时变量	dword	4	20

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

(100) [+, x, y, $1]
(101) [=, $1, -, a]
(102) [=, $1, -, $2]
(103) [=, $1, -, b]
(104) [*, $1, $1, $3]
(105) [=, $3, -, var]

```
1   Fun proc
2     push ebp
3     mov ebp, esp
4     pushad
5     sub esp, 24   ; 局部变量
6     mov eax, [ebp + 12]
7     add eax, [ebp + 8]
8     mov [ebp - 32], eax
9     mov [ebp - 48], eax
10    mov [ebp - 36], eax
11    mov ebx, eax
12    mul ebx
13    mov [ebp - 52], eax
14    mov [ebp - 40], eax
15    add esp, 24
16    popad
17    pop ebp
18    ret 8
19  Fun endp
```

名字	类别	类型	字宽	偏移量
a	局部变量	dword	4	0
b	局部变量	dword	4	4
var	局部变量	dword	4	8
$1	临时变量	dword	4	12
$2	临时变量	dword	4	16
$3	临时变量	dword	4	20



山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

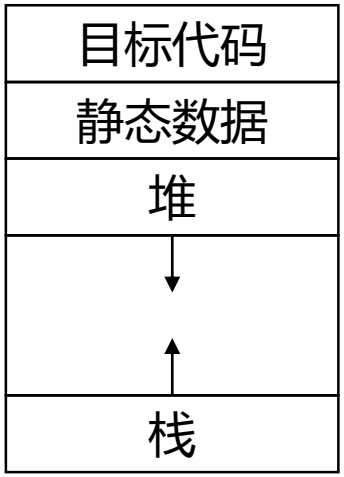
9.1.2 参数传递

- 编译程序为了组织存储空间，需要考虑到几个问题
  - 过程是否允许递归?
  - 当控制从一个过程的活动返回时，对局部名称的值如何处理?
  - 过程是否允许引用非局部名称?
  - 过程调用时如何传递参数，过程是否可以作为参数被传递，以及作为结果被返回?
  - 存储空间可否在程序控制下进行动态分配?
  - 存储空间是否必须显式的释放?

第九章 运行时存储空间组织

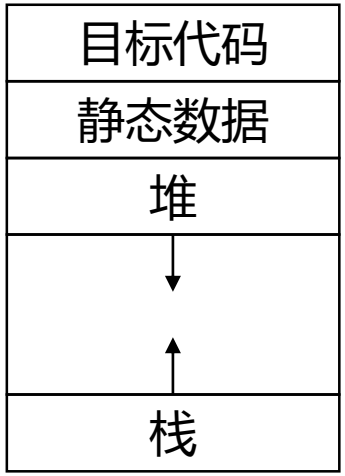
9.2.1 运行时存储器的划分

- 目标代码区：目标代码的大小在编译时可以确定，所以编译程序可以把它放在一个静态确定的区域。
- 静态数据区：编译时完全确定的数据可以放在静态数据区，应尽可能多的静态分配数据对象。
  - 如Fortran语言，所有数据对象都可静态地进行存储分配。



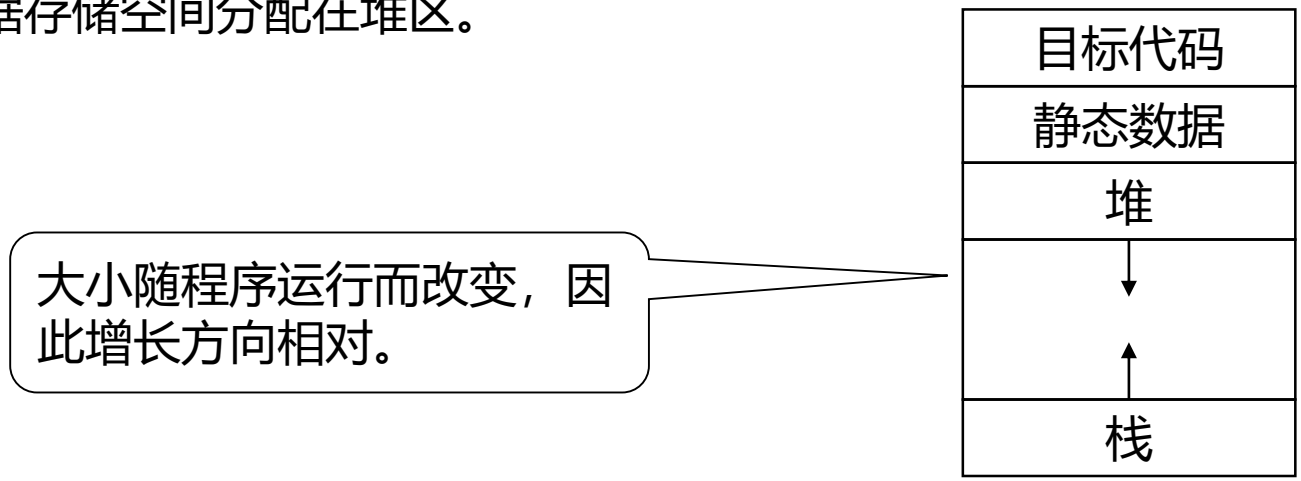
山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

```
1   .386            ; 伪指令，表示是32位程序
2   .model flat, stdcall  ; 内存模式flat，子程序调用规范stdcall
3   .stack 4096           ; 堆栈大小4096
4
5   ; 标准Windows服务ExitProcess，需要一个退出码做参数
6   ExitProcess PROTO, dwExitCode: DWORD
7
8   .data       ; 数据区
9   sum DWORD 0
10
11  .code       ; 代码区
12  main PROC   ; 过程名（函数名）
13    mov eax, 7
14    add eax, 70
15    mov sum, eax
16
17    invoke ExitProcess, 0  ; 调用操作系统的标准退出服务
18  main ENDP         ; 函数结束
19
20  END main          ; 程序结束，end后面跟入口地址，再往后的内容会忽略
```



9.2.1 运行时存储器的划分

- 栈区：如Pascal和C，使用扩充的栈来管理过程的活动
  - 当发生过程调用时，中断当前活动的执行，激活被调用过程的活动，并把包含在这个活动生存期中的数据对象以及和该活动有关的信息存入栈中；
  - 当控制从调用返回时，将所占存储空间弹出栈顶，同时，被中断的活动恢复运行。
- 堆区：Pascal和C都允许数据对象在程序运行时分配空间，以便建立动态数据结构，这样的数据存储空间分配在堆区。



第九章 运行时存储空间组织

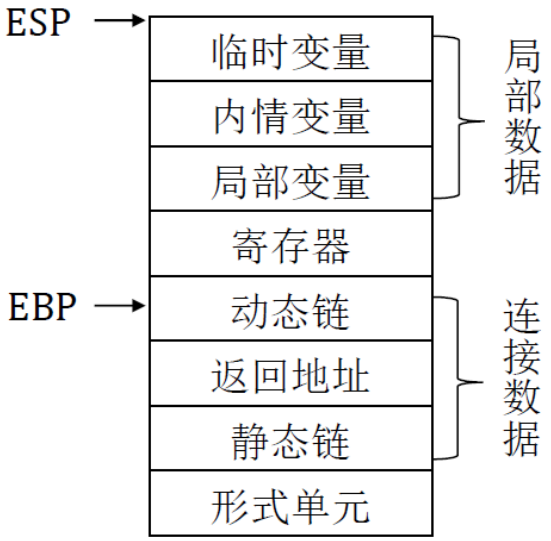
9.2.2 活动记录

- 活动记录：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，这样的一个连续存储块称为活动记录（Activation Record）。
  - 如Pascal和C语言，当过程调用时，产生活动记录，并压入栈；
  - 过程返回时，弹出栈。

9.2.2 活动记录

活动记录的内容

- 连接数据
  - 返回地址
  - 动态链：指向该过程前的最新活动记录的指针，运行时，使运行栈上各数据区按动态建立的次序结成链，链头是栈顶起始位置；
  - 静态链：指向静态直接外层最新活动记录的指针，用来访问非局部数据。
- 形式单元：存放相应的实在参数的地址或值。
- 局部数据区：局部变量、内情向量（数组的有关信息）、临时工作单元（如存放表达式求值结果）。



第九章 运行时存储空间组织

- 9.1 目标程序运行时的活动
  - 9.1.1 过程的活动
  - 9.1.2 参数传递
- 9.2 运行时存储器的划分
  - 9.2.1 运行时存储器的划分
  - 9.2.2 活动记录
  - 9.2.3 存储分配策略
- 9.3 静态存储分配
- 9.4 简单的栈式存储分配
  - 9.4.1 C的活动记录
  - 9.4.2 C的栈式动态分配
- 9.5 嵌套过程语言的栈式实现
  - 9.5.1 非局部名字的访问的实现
  - 9.5.2 参数传递的实现
- 9.6 堆式动态存储分配
  - 9.6.1 堆式动态分配的实现
  - 9.6.2 隐式存储回收

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

9.2.3 存储分配策略

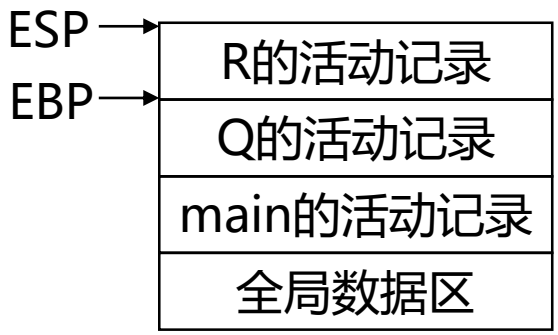
- 静态分配策略：在编译时对所有数据对象分配固定的存储单元，且在运行时始终保持不变。
- 栈式动态分配策略：在运行时把存储器作为一个栈进行管理，每当调用一个过程时，它所需要的存储空间就动态地分配于栈顶；一旦退出，它所占用的空间就予以释放。
- 堆式动态分配策略：在运行时把存储器组织成堆结构，以方便用户关于存储空间的申请与回收；用户申请时从堆中分配一块空间，释放时退回给堆。

第九章 运行时存储空间组织

- 9.1 目标程序运行时的活动
  - 9.1.1 过程的活动
  - 9.1.2 参数传递
- 9.2 运行时存储器的划分
  - 9.2.1 运行时存储器的划分
  - 9.2.2 活动记录
  - 9.2.3 存储分配策略
- 9.3 静态存储分配
- 9.4 简单的栈式存储分配
  - 9.4.1 C的活动记录
  - 9.4.2 C的栈式动态分配
- 9.5 嵌套过程语言的栈式实现
  - 9.5.1 非局部名字的访问的实现
  - 9.5.2 参数传递的实现
- 9.6 堆式动态存储分配
  - 9.6.1 堆式动态分配的实现
  - 9.6.2 隐式存储回收

9.4 简单点栈式存储分配

```
全局数据说明;
void main()
{
    main中的数据说明;
    Q();
}
void R()
{
    R中的数据说明;
}
.....
void Q()
{
    Q中的数据说明;
    R();
}
```



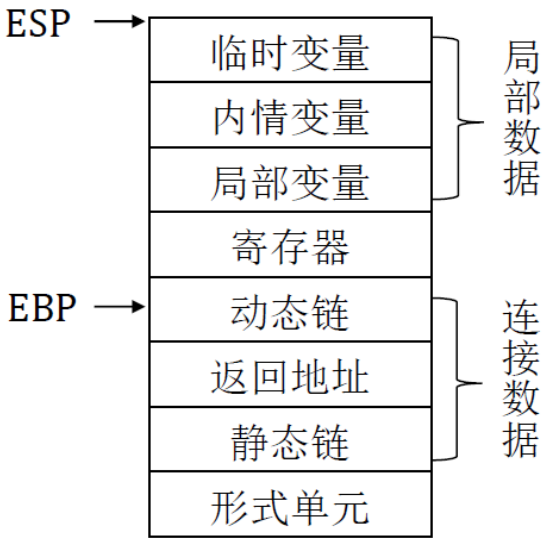
- 两个指针
  - EBP总是指向现行过程活动记录的固定点，用于访问局部数据；
  - ESP始终指向（已占用）栈顶单元。

第九章 运行时存储空间组织

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

9.4.1 C的活动记录

- C的活动记录包括4个项目
  - 连接数据
    - 老SP值，即前一活动记录的地址；
    - 返回地址
  - 参数个数
  - 形式单元：存放相应的实在参数的地址或值。
  - 局部数据区：过程的局部变量、数组内情向量、临时工作单元。

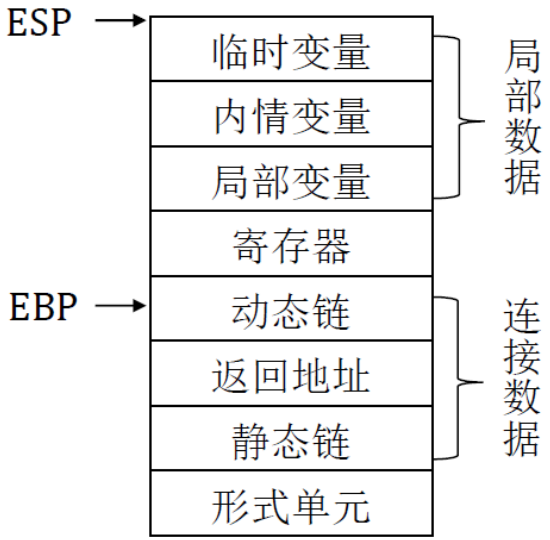


- C语言不允许过程嵌套定义
  - 所以，C语言的非局部量仅能出现在源程序头，非局部变量可以作为最顶层活动记录。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

9.4.1 C的活动记录

- 过程的每一个局部变量或形参在活动记录中的位置是确定的，即它们都分配了存储单元，其地址是相对于活动记录的基地址SP的。
  - 变量和形参地址：绝对地址 = 活动记录基地址 + 相对地址；
  - 即变量或形参X的引用可以表示为：X[SP]；
  - 相对地址X在编译时也可以完全确定下来。



第九章 运行时存储空间组织

9.4.2 C的栈式动态分配

- 过程调用的四元式：

  $param\ T_1$

  ……

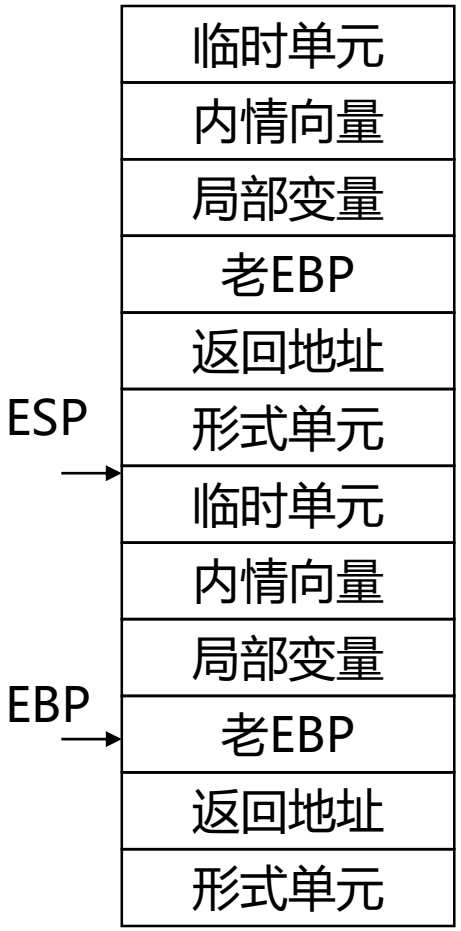
  $param\ T_n$

  $call\ P, n$

- 形式单元与活动记录EBP之间的距离总是2（元素数不是字节数）：

  $(i+2)[EBP] = T_i$ // 传值，i从1开始

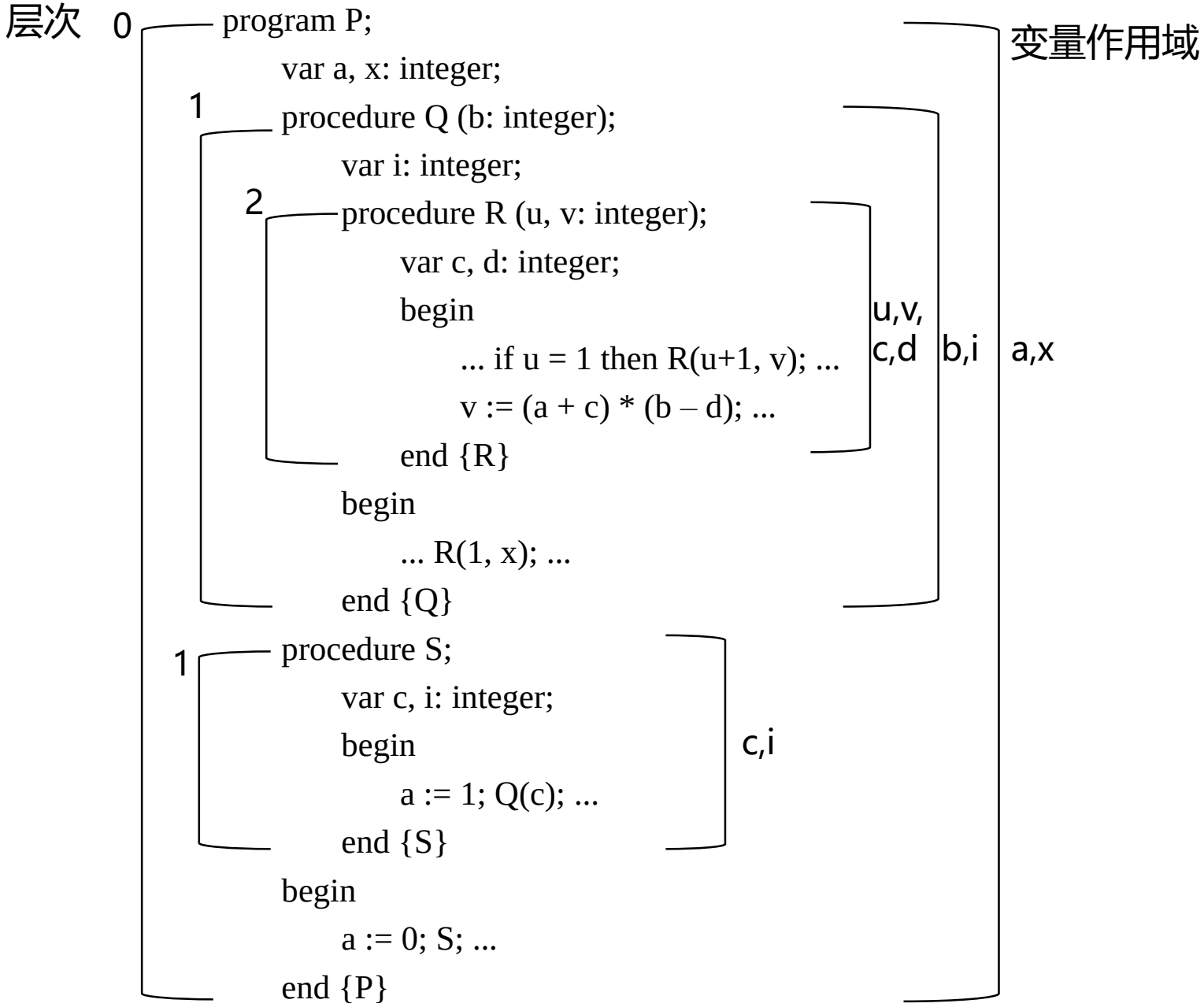
  $(i+2)[EBP] = addr(T_i)$ // 传地址



第九章 运行时存储空间组织

- 9.1 目标程序运行时的活动
  - 9.1.1 过程的活动
  - 9.1.2 参数传递
- 9.2 运行时存储器的划分
  - 9.2.1 运行时存储器的划分
  - 9.2.2 活动记录
  - 9.2.3 存储分配策略
- 9.3 静态存储分配
- 9.4 简单的栈式存储分配
  - 9.4.1 C的活动记录
  - 9.4.2 C的栈式动态分配
- 9.5 嵌套过程语言的栈式实现
  - 9.5.1 非局部名字的访问的实现
  - 9.5.2 参数传递的实现
- 9.6 堆式动态存储分配
  - 9.6.1 堆式动态分配的实现
  - 9.6.2 隐式存储回收

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

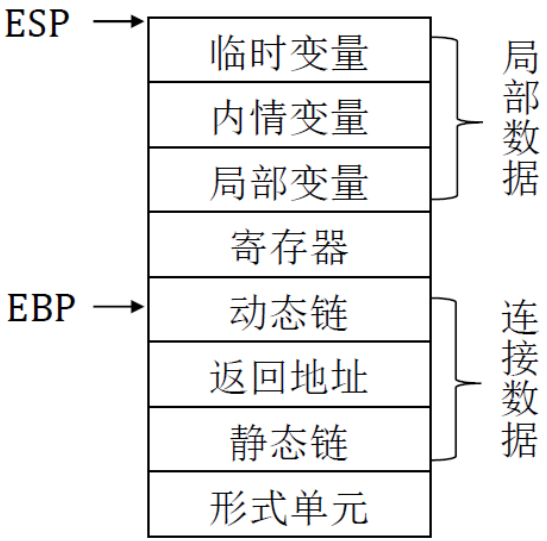


第九章 运行时存储空间组织

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

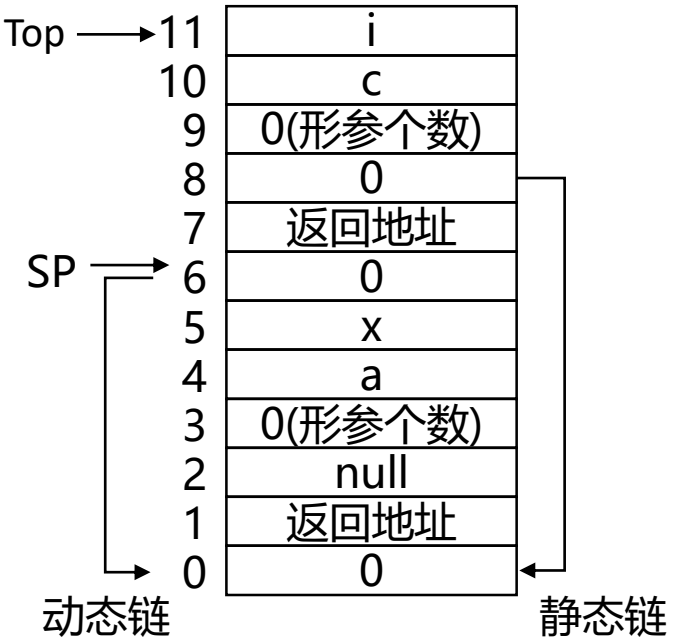
静态链和活动记录

- 运行时，一个过程Q可能引用它的任一外层过程P的最新活动记录中的某些数据，这些数据是Q的非局部量。
  - 解决办法一是引入一个称为静态链的指针，指向直接外层的最新活动记录的地址。



山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

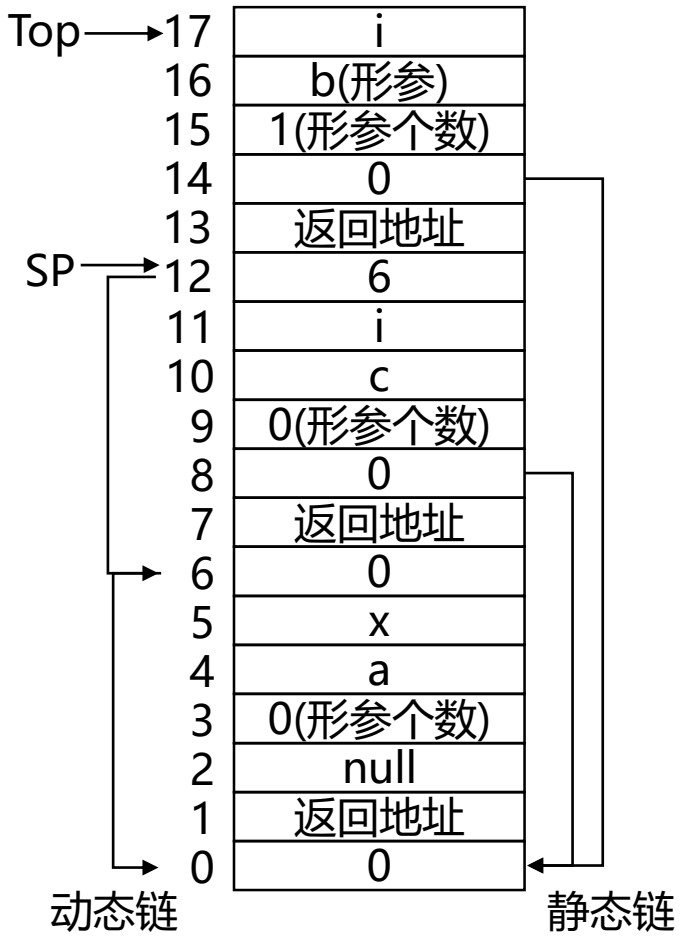
```
program P;
    var a, x: integer;
    procedure Q (b: integer);
        var i: integer;
        procedure R (u, v: integer);
            var c, d: integer;
            begin
                ... if u = 1 then R(u+1, v); ...
                v := (a + c) * (b – d); ...
            end {R}
        begin
            ... R(1, x); ...
        end {Q}
    procedure S;
        var c, i: integer;
        begin
            a := 1; Q(c); ...
        end {S}
    begin
        a := 0; S; ...
    end {P}
```



过程P调用S

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

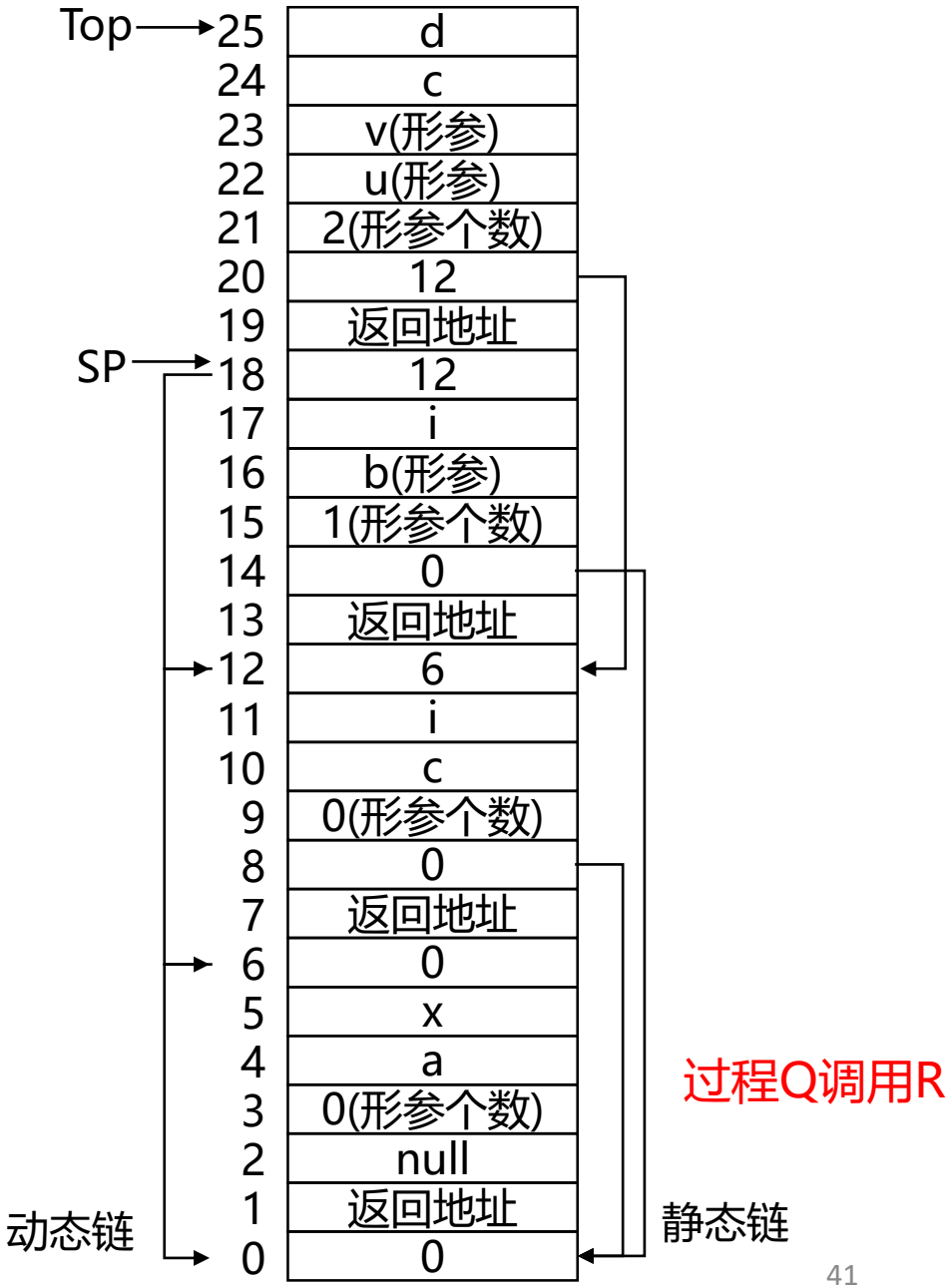
```
program P;
    var a, x: integer;
    procedure Q (b: integer);
        var i: integer;
        procedure R (u, v: integer);
            var c, d: integer;
            begin
                ... if u = 1 then R(u+1, v); ...
                v := (a + c) * (b – d); ...
            end {R}
        begin
            ... R(1, x); ...
        end {Q}
    procedure S;
        var c, i: integer;
        begin
            a := 1; Q(c); ...
        end {S}
begin
    a := 0; S; ...
end {P}
```



过程S调用Q

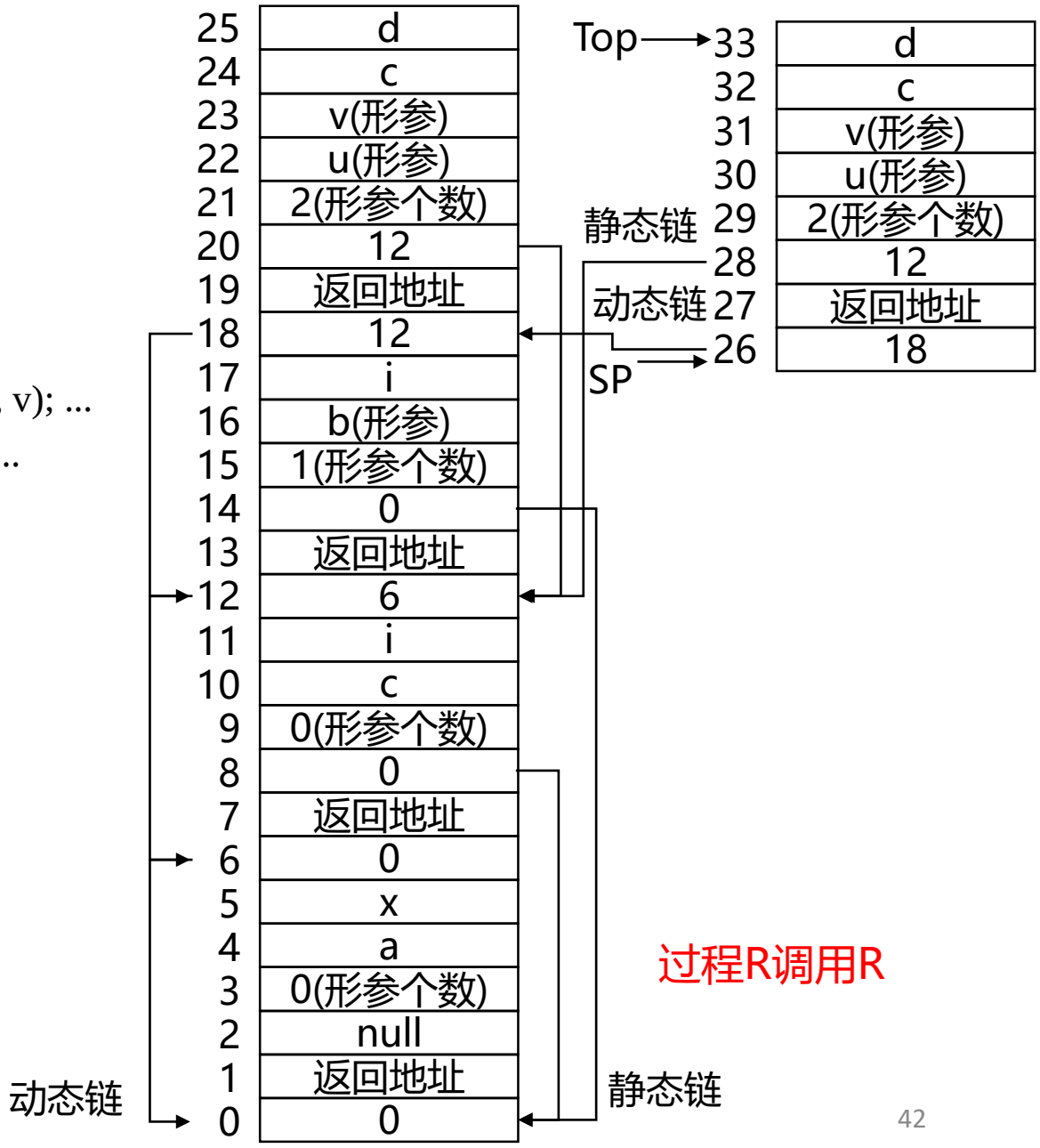
山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

```
program P;
    var a, x: integer;
    procedure Q (b: integer);
        var i: integer;
        procedure R (u, v: integer);
            var c, d: integer;
            begin
                ... if u = 1 then R(u+1, v); ...
                v := (a + c) * (b – d); ...
            end {R}
        begin
            ... R(1, x); ...
        end {Q}
    procedure S;
        var c, i: integer;
        begin
            a := 1; Q(c); ...
        end {S}
begin
    a := 0; S; ...
end {P}
```



山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

```
program P;
    var a, x: integer;
    procedure Q (b: integer);
        var i: integer;
        procedure R (u, v: integer);
            var c, d: integer;
            begin
                ... if u = 1 then R(u+1, v); ...
                v := (a + c) * (b – d); ...
            end {R}
        begin
            ... R(1, x); ...
        end {Q}
    procedure S;
        var c, i: integer;
        begin
            a := 1; Q(c); ...
        end {S}
begin
    a := 0; S; ...
end {P}
```



山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

嵌套层次显示表和活动记录

嵌套层次显示表

- 为提高访问非局部变量的速度，可以引入指针数组指向本过程的所有外层，称为嵌套层次显示表（Display）；
- 嵌套层次显示表是一个小栈，自顶向下依次指向当前层、直接外层、直接外层的直接外层、…、直至最外层（0层）。
- 静态链不再需要，但当P_1调用P_2时，P_2需要知道P_1的直接外层（记为P_0）的display表，因此需要传递调用方的display表，称为全局display。

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

嵌套层次显示表和活动记录

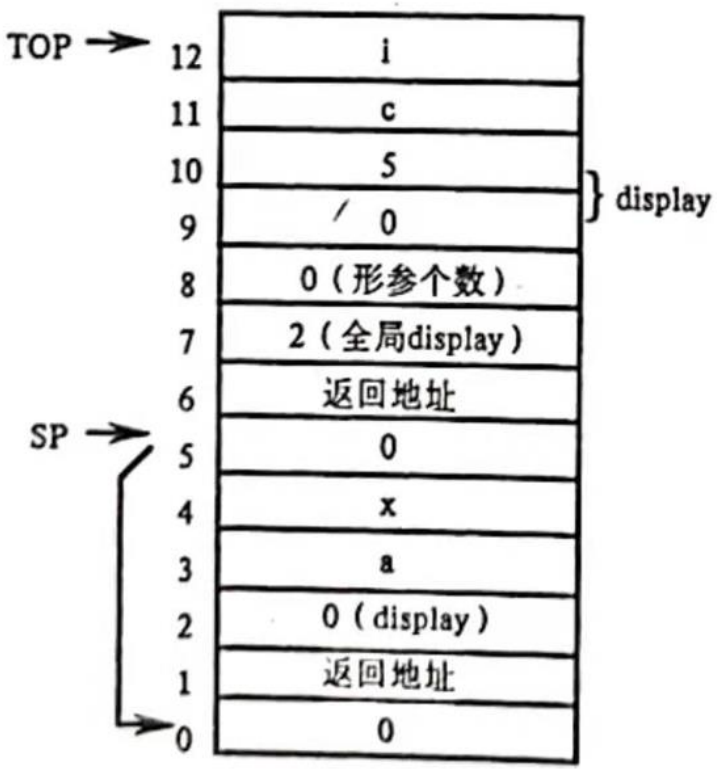
- 访问某外层变量X
  - 由于每个过程的形式单元数目在编译时已知，因此display表的相对地址d编译时可确定；
  - 当前过程引用第k层的变量X，那么可以如下访问：

$LD\ R_1,(d+k)[SP]$ // 第k层活动记录

$LD\ R_2,X[R_1]$

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

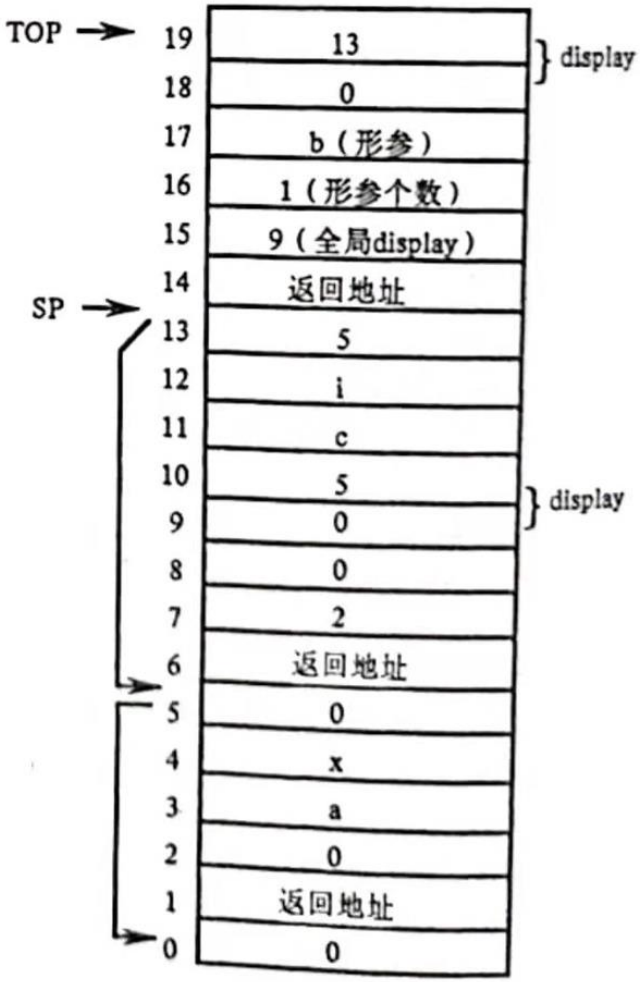
```
program P;
    var a, x: integer;
    procedure Q (b: integer);
        var i: integer;
        procedure R (u, v: integer);
            var c, d: integer;
            begin
                ... if u = 1 then R(u+1, v); ...
                v := (a + c) * (b – d); ...
            end {R}
        begin
            ... R(1, x); ...
        end {Q}
    procedure S;
        var c, i: integer;
        begin
            a := 1; Q(c); ...
        end {S}
begin
    a := 0; S; ...
end {P}
```



过程P调用S

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

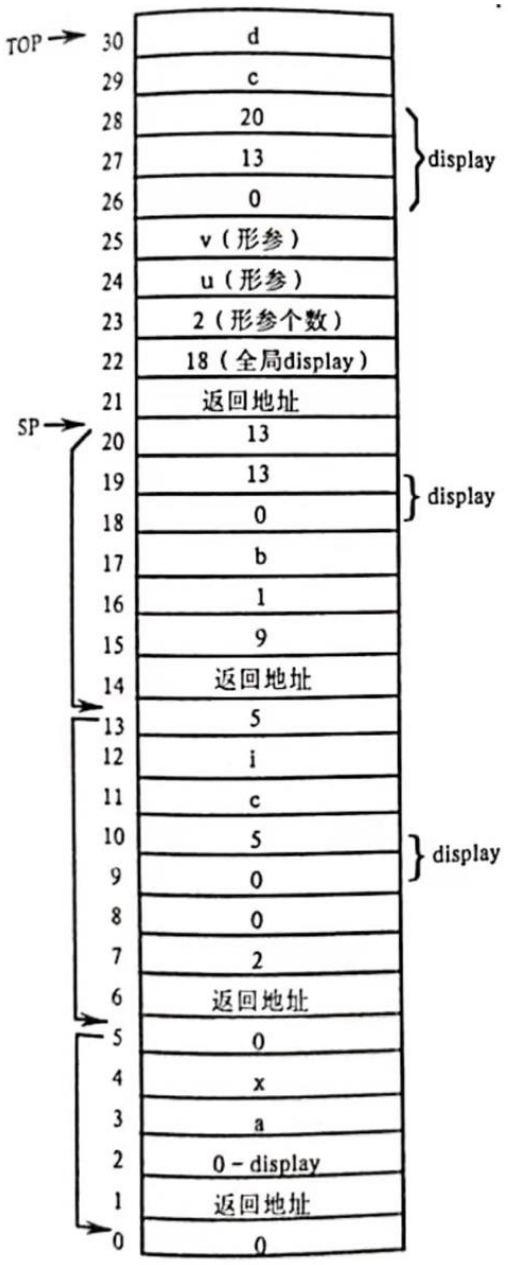
```
program P;
    var a, x: integer;
    procedure Q (b: integer);
        var i: integer;
        procedure R (u, v: integer);
            var c, d: integer;
            begin
                ... if u = 1 then R(u+1, v); ...
                v := (a + c) * (b – d); ...
            end {R}
        begin
            ... R(1, x); ...
        end {Q}
    procedure S;
        var c, i: integer;
        begin
            a := 1; Q(c); ...
        end {S}
begin
    a := 0; S; ...
end {P}
```



过程S调用Q

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

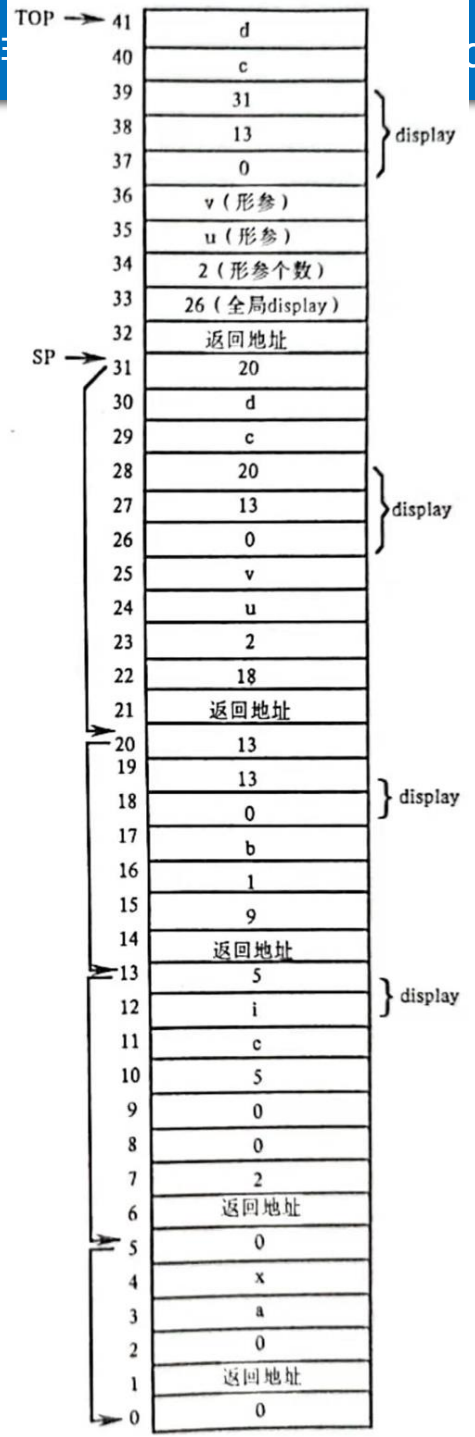
```
program P;
    var a, x: integer;
    procedure Q (b: integer);
        var i: integer;
        procedure R (u, v: integer);
            var c, d: integer;
            begin
                ... if u = 1 then R(u+1, v); ...
                v := (a + c) * (b – d); ...
            end {R}
        begin
            ... R(1, x); ...
        end {Q}
    procedure S;
        var c, i: integer;
        begin
            a := 1; Q(c); ...
        end {S}
begin
    a := 0; S; ...
end {P}
```



过程Q调用R

```
program P;
    var a, x: integer;
    procedure Q (b: integer);
        var i: integer;
        procedure R (u, v: integer);
            var c, d: integer;
            begin
                ... if u = 1 then R(u+1, v); ...
                v := (a + c) * (b – d); ...
            end {R}
        begin
            ... R(1, x); ...
        end {Q}
    procedure S;
        var c, i: integer;
        begin
            a := 1; Q(c); ...
        end {S}
begin
    a := 0; S; ...
end {P}
```

过程R调用R



第九章 运行时存储空间组织

山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

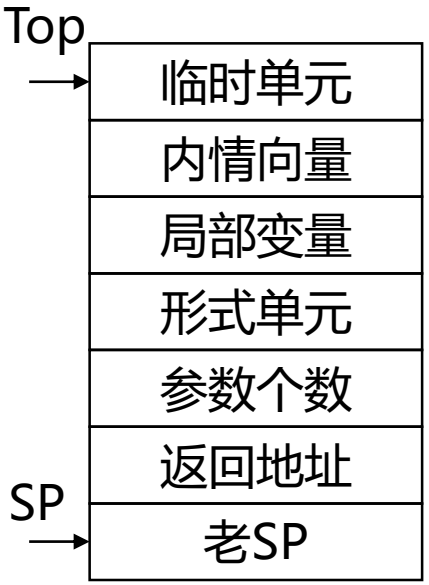
9.5.2 参数传递的实现

- 前述四元式$param\ T$的翻译：

  $(i+3)[Top] = T_i$

  或：$(i+3)[Top] = addr(T_i)$

- 以上做法把问题极大简化了：
  - 如果实参是一个简单变量、数组元素或临时变量，则根据传值或传地址对$param\ T$给予上述解释是正确的；
  - 如果实参是数组、过程或标号，则$param\ T$都应是传地址，需要在传址前做一些额外工作。



山东大学 SHANDONG UNIVERSITY 郑艳伟，zhengyw@sdu.edu.cn

9.5.2 参数传递的实现

- $param\ T$，其中T为数组
  - 在这种情况下，根据不同语言的要求，传输数组T的首地址或内情向量地址；
  - 假定要求在运行时对形-实数组的维度一致性和体积相容性进行动态检查，则应传递内情向量地址，否则传递首地址就够了；
  - 这种一致性和相容性检查意味着所有数组的内情向量都必须保留到运行阶段。

9.5.2 参数传递的实现

- $param\ T$，其中T为过程：过程P把过程T作为实参传给Q，Q通过形参Z调用T
  - 进入T后，为了建立T自己的display，T必须知道它直接外层的display；
  - P的display或者正好是这个外层的display，或者包含了这个外层display；
  - 如果T的层数是l，则T的display由P的display前l个单元，加上SP的当前值组成
  - 为使T工作时知道过程P的display，必须在P把T作为实参传给Q时，把P自身的display地址也传过去。
- $param\ T$建立两个相继的临时单元
  - ① 临时单元B_1：过程T的入口地址；
  - ② 临时单元B_2：当前display地址；
  - ③ 把B_1的地址传给Q：$(i+1)[Top] = addr(B_1)$;
  - ④ 假定Q现在执行到$call\ Z, m$，形参Z中已有B_1的地址，那么B_1的内容用来作为转移指令的目的地址（即转进T），B_2的内容作为全局display地址。

9.5.2 参数传递的实现

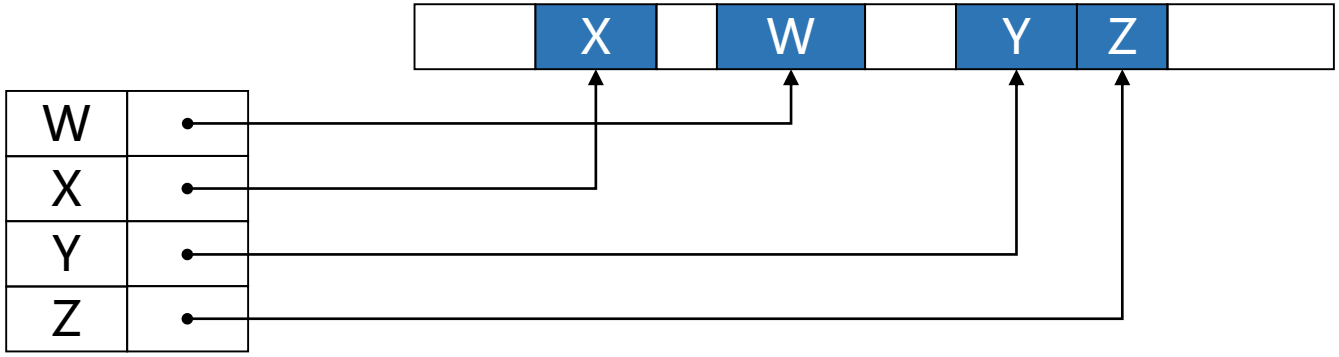
- $param\ T$，其中T为标号：过程P把标号T作为实参传给Q，Q通过形参Z转移到T指向的地方
  - 如果标号T是在过程P_0中定义的，那么P_0或者是P自身，或者是P的某一外层；
  - 当Q要转向T时，必须先把P_0的活动记录变成当前活动记录；
  - 即不仅要把标号T的地址传给Q，还要把P_0的活动记录地址也传过去。
- $param\ T$建立两个相继的临时单元
  - ① 临时单元B_1：标号T的地址；
  - ② 临时单元B_2： P_0的活动记录地址；
  - ③ 把B_1的地址传给Q：$(i+1)[Top] = addr(B_1)$；
  - ④ 假定Q现在执行到$goto\ Z$，形参Z中已有B_1的地址，那么按B_1的内容前，应逐级恢复SP和Top，直至SP指向P_0 的活动记录。

第九章 运行时存储空间组织

- 9.1 目标程序运行时的活动
  - 9.1.1 过程的活动
  - 9.1.2 参数传递
- 9.2 运行时存储器的划分
  - 9.2.1 运行时存储器的划分
  - 9.2.2 活动记录
  - 9.2.3 存储分配策略
- 9.3 静态存储分配
- 9.4 简单的栈式存储分配
  - 9.4.1 C的活动记录
  - 9.4.2 C的栈式动态分配
- 9.5 嵌套过程语言的栈式实现
  - 9.5.1 非局部名字的访问的实现
  - 9.5.2 参数传递的实现
- 9.6 堆式动态存储分配
  - 9.6.1 堆式动态分配的实现
  - 9.6.2 隐式存储回收

9.6 堆式动态存储分配

- 当运行程序请求一块体积为N的空间时，应该分配哪一块？
  - 从比N稍大的一个空闲块中取出N个单元，以便使大的空闲块派更大的用场；
  - 先碰上哪块比N大，就从其中分出N个单元。

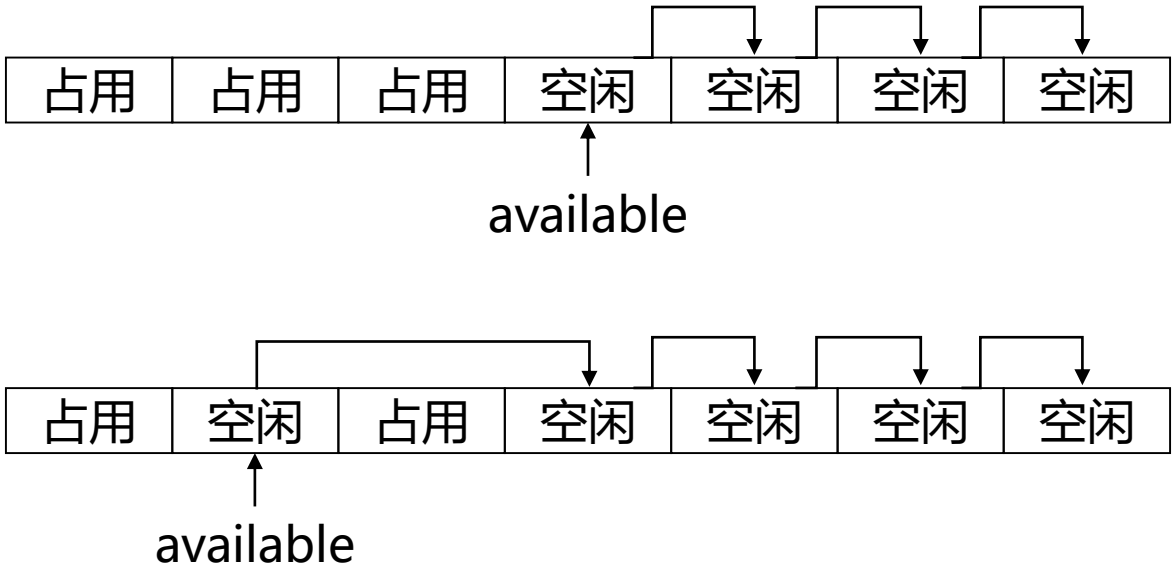


- 运行一段时间后，空间会零碎不堪
  - 没有任何一块比N大，但总和比N大的多。
- 总和也比N小，如何收回空间？
  - 垃圾回收的机制如何设计。

第九章 运行时存储空间组织

9.6.1 堆式动态分配的实现

- 定长块管理



9.6.1 堆式动态分配的实现

- 变长块管理
  - 初始化时，堆区为一整块；
  - 用户申请时，从一个整块里分割出满足需要的一小块；
  - 用户释放时，如果释放块能和现有空闲块合并，则合并，不能则成链。
- 变长块分配策略
  - 首次满足法：只要在空闲块链表中找到满足需要的一块，就分配；如果该块比申请的块大不了多少，就整块分配出去。
  - 最优满足法：将空闲链中不小于申请块，且最接近申请块的空闲块分配给用户；需要将链表块从小到大排序，可能会产生很多小碎片块。
  - 最差满足法：将空闲块中不小于申请块，且最大的空闲的一部分分配给用户；此时链表块从大到小排序，分配时不需要查找，最终结点趋于均匀。

第九章 运行时存储空间组织

9.6.2 隐式存储回收

- 隐式存储回收：垃圾回收子程序与用户程序并行工作，需要知道分配给用户程序的存储块何时不再使用。
  - 第一个阶段为标记阶段，对已分配的块跟踪程序中各指针的访问路径，如果某个块被访问过，就给这个块加个标记。
  - 第二个阶段为回收阶段，所有未加标记的存储块回收到一起，并插入空闲块链表中，然后消除在存储块中所加的全部标记。

块长度
访问计数标记
指针
用户使用空间

存储块格式

山东大学
SHANDONG UNIVERSITY

第九章 运行时存储空间组织

The End

谢谢

授 课 教 师 ：郑艳伟
手 机 ：18614002860（微信同号）
邮 箱 ：zhengyw@sdu.edu.cn