第七章作业

7.2

7.3

7.5

7.2

（1）传值调用

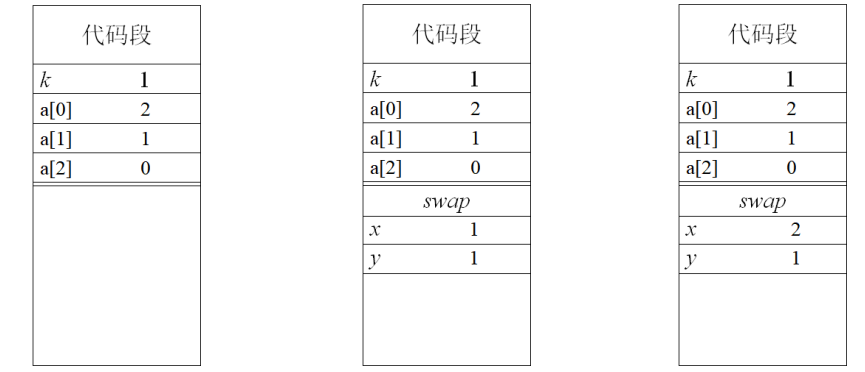
全局变量初始化后的状态如图 1(a)所示；

第一次调用过程 *swap*(*k*, *a*[*k*])时，*swap* 的活动记录入栈，并将变量 *k* 的右值 1 和 *a*[1]的右值 1 传递给形参 *x* 和 *y*，状态如图 1(b)所示；*swap*(*k*, *a*[*k*])的过程体执行期间，执行语句 x=x+y 之后的状态如图 1(c)所示，执行语句 y=x-y 之后的状态同图 1(c)所示，执行语句 x=x-y 之后的状态同图 1(b)所示；从 *swap*(*k*, *a*[*k*])返回，*swap* 的活动记录出栈后，状态同图 1(a)所示。

printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2])的执行结果是 1, 2, 1, 0。

第二次调用过程 *swap*(*a*[*k*], *a*[*k*])时，*k*=1，*swap* 的活动记录入栈，将变量 *a*[1]的右值 1 同时传递给形参 *x* 和 *y*，状态如图 1(b)所示，*swap*(*a*[*k*], *a*[*k*])的过程体执行期间，执行语句 x=x+y 之后的状态如图 1(c)所示，过程体执行完之后从 *swap*(*a*[*k*], *a*[*k*])返回之前的状态同图 1(b)所示，从 *swap*(*a*[*k*], *a*[*k*])返回，*swap* 的活动记录出栈后，状态同图 1(a)所示。

printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2])的执行结果是 1, 2, 1, 0。



(a) 过程调用前　　(b) 参数传递后　　(c) x=x+y 执行后

图 1　传值调用情况下，程序运行空间的状态变化

（2）引用调用

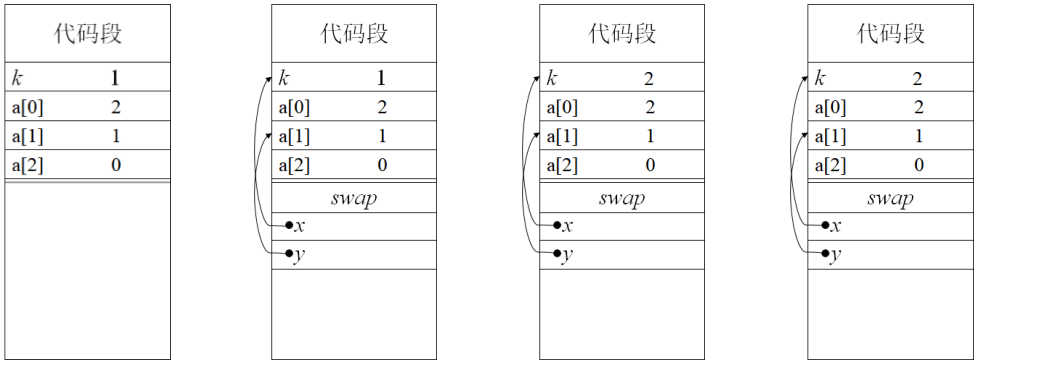
全局变量初始化后的状态如图 2(a)所示；

第一次调用过程 *swap*(*k*, *a*[*k*])时，*swap* 的活动记录入栈，并将变量 *k* 的左值和 *a*[1]的左值传递给形参 *x* 和 *y*，状态如图 2(b)所示；*swap*(*k*, *a*[*k*])的过程体执行期间，执行语句 x=x+y 之后的状态如图 2(c)所示，执行语句 y=x-y 之后的状态如图 2(d)所示，执行语句 x=x-y 之后的状态如图 7-(e)所示；从 *swap*(*k*, *a*[*k*])返回，*swap* 的活动记录出栈后，状态同图 2(a)所示。

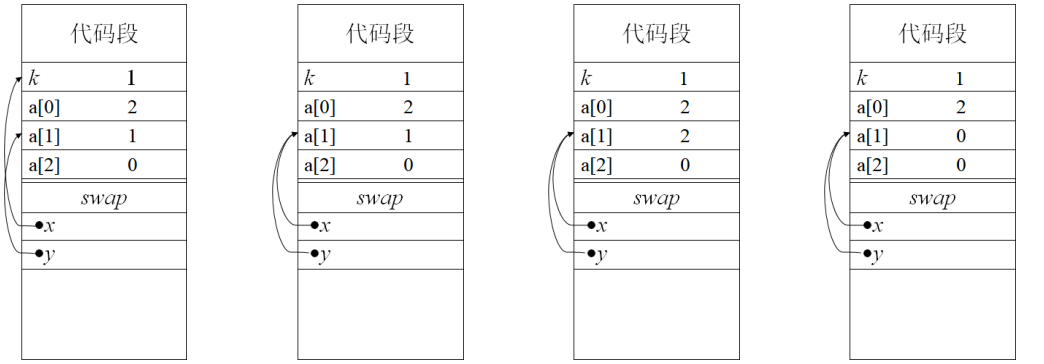
printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2])的执行结果是 1, 2, 1, 0。

第二次调用过程 *swap*(*a*[*k*], *a*[*k*])时，*k*=1，*swap* 的活动记录入栈，将变量 *a*[1]的左值同时传递给形参 *x* 和 *y*，状态如图 2(f)所示，*swap*(*a*[*k*], *a*[*k*])的过程体执行期间，执行语句 x=x+y 之后的状态如图 1(g)所示，过程体执行完之后从 *swap*(*a*[*k*], *a*[*k*])返回之前的状态如图 1(h)所示，从 *swap*(*a*[*k*], *a*[*k*])返回，*swap* 的活动记录出栈。

printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2])的执行结果是 1, 2, 0, 0。



(a) 过程调用前　(b) 参数传递后　(c) x=x+y 执行后　(d) y=x-y 执行后



(e) x=x-y 执行后　(f) 参数传递后　(g) x=x+y 执行后　(h) 过程体执行后

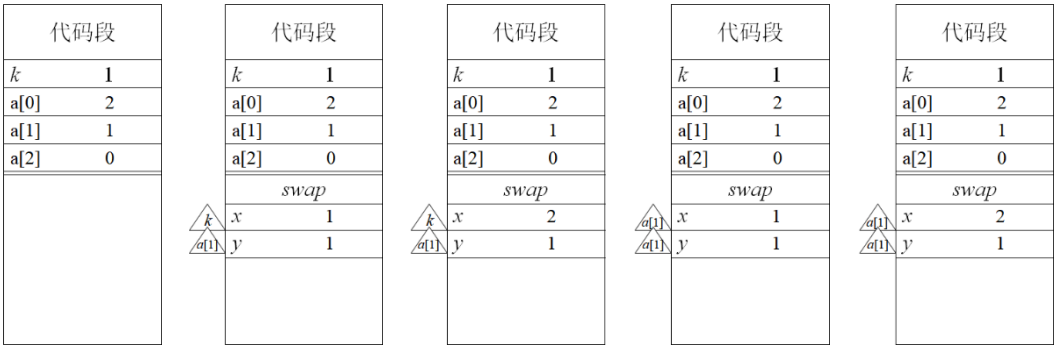
图 2　引用调用情况下，程序运行空间的状态变化

（3）复制恢复

全局变量初始化后的状态如图 3(a)所示；

第一次调用过程 *swap*(*k*, *a*[*k*])时，*swap* 的活动记录入栈，并将变量 *k* 的右值 1 和 *a*[1]的右值 1 传递给形参 *x* 和 *y*，状态如图 3(b)所示，并记录与参数 *x* 相应的实参 *k* 的左值，与参数 *y* 相应的实参 *a*[1]的左值。*swap*(*k*, *a*[*k*])的过程体执行期间，执行语句 x=x+y 之后的状态如图 3(c)所示，执行语句 y=x-y 之后的状态同 7-5(c)所示，执行语句 x=x-y 之后的状态同如图 3(b)所示；从 *swap*(*k*, *a*[*k*])返回时，根据记录的与各参数相应实参的左值，把参数的当前值复制到实参的空间中，*swap* 的活动记录出栈后，状态同图 3(a)所示。

printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2])的执行结果是 1, 2, 1, 0。



(a) 过程调用前　(b) 参数传递后　(c) x=x+y 执行后　(d) 参数传递后　(e) x=x+y 执行后

图 3　复制恢复调用情况下，程序运行空间的状态变化

第二次调用过程 *swap*(*a*[*k*], *a*[*k*])时，*k*=1，*swap* 的活动记录入栈，将变量 *a*[1]的右值 1

同时传递给形参 *x* 和 *y*，状态如图 3(d)所示，并记录与参数 *x* 和 *y* 相应的实参 *a*[1]的左值。*swap*(*a*[*k*], *a*[*k*])的过程体执行期间，执行语句 x=x+y 之后的状态如图 3(e)所示，执行语句 y=x-y 之后的状态同 7-5(e)所示，执行语句 x=x-y 之后的状态同如图 3(d)所示；从 *swap*(*a*[*k*], *a*[*k*])返回时，根据记录的与各参数相应实参的左值，按照从左向右的顺序把参数的当前值复制到实参的空间中，*swap* 的活动记录出栈后，状态同图 3(a)所示。

printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2])的执行结果是 1, 2, 1, 0。

（4）传名调用

在该题目中，过程调用语句 swap(k, a[k])将被展开为以下语句序列：

```
k=k+a[k];
a[k]=k-a[k];
k=k-a[k];
```

过程调用语句 swap(a[k], a[k])将被展开为以下语句序列：

```
a[k]=a[k]+a[k];
a[k]=a[k]-a[k];
a[k]=a[k]-a[k];
```

经过展开之后的程序体代码如下：

```
k=1;
a[0]=2;
a[1]=1;
a[2]=0;
k=k+a[k];
a[k]=k-a[k];
k=k-a[k];
printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2]);
a[k]=a[k]+a[k];
a[k]=a[k]-a[k];
a[k]=a[k]-a[k];
printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2]);
```

语句 k=k+a[k]执行后，k=2。

语句 a[k]=k-a[k]执行后，a[2]=2。

语句 k=k-a[k]执行后，k=0。

printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2])的执行结果是 0, 2, 1, 2。

语句 a[k]=a[k]+a[k]执行后，a[0]=4。

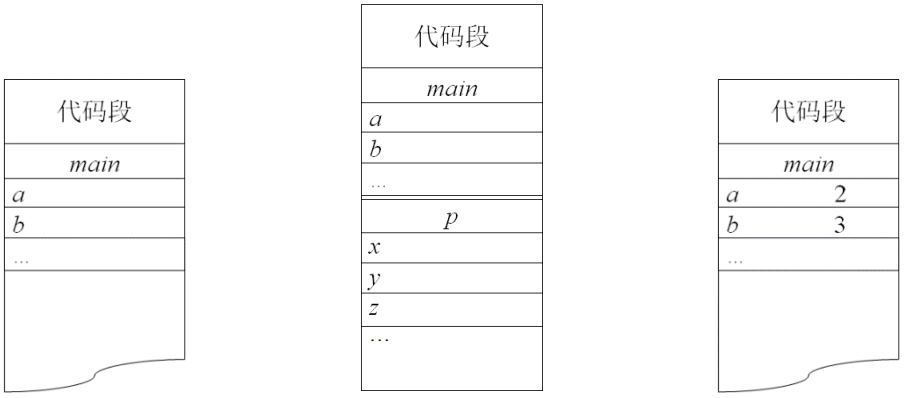
语句 a[k]=a[k]-a[k]执行后，a[0]=0。

语句 a[k]=a[k]-a[k]执行后，a[0]=0。

printf(″k=%d, a[0]=%d, a[1]=%d, a[2]%d\n″, k, a[0], a[1], a[2])的执行结果是 0, 0, 1, 2。

7.3

程序在运行过程中，当控制在主程序 main 中时，控制栈组织如图 4(a)所示，当控制在过程 p 中时，控制栈组织如图 4(b)所示，当控制在 main 中，变量初始化之后，调用过程 p 之前的状态如图 4(c)所示。



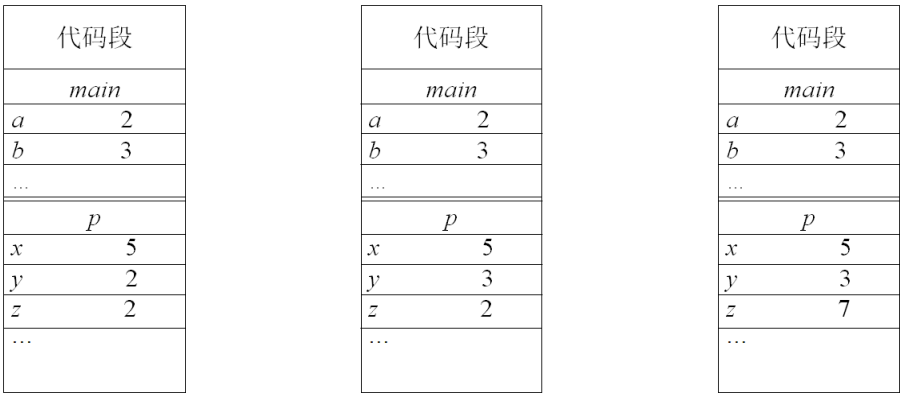
(a) 控制在 main 中时　　(b) 控制在过程 p 中时　　(c) 调用过程 p 之前

图 4　程序运行时控制栈

(1) 传值调用

主程序计算实参 a+b、a 和 a 的右值，并传递给过程 p 的参数空间，控制栈状态如图 5(a)所示。控制进入过程 p 中，对参数的所有访问均在过程 p 的活动记录中参数空间上进行。语句 y:=y+1 的执行结果，如图 5(b)所示。语句 z:=z+x 的执行结果，如图 5(c)所示。控制从过程 p 返回后，P 的活动记录出栈，控制栈状态同 7-6(c)所示。

语句 writeln (′a=′,a)的执行结果是：a=2。



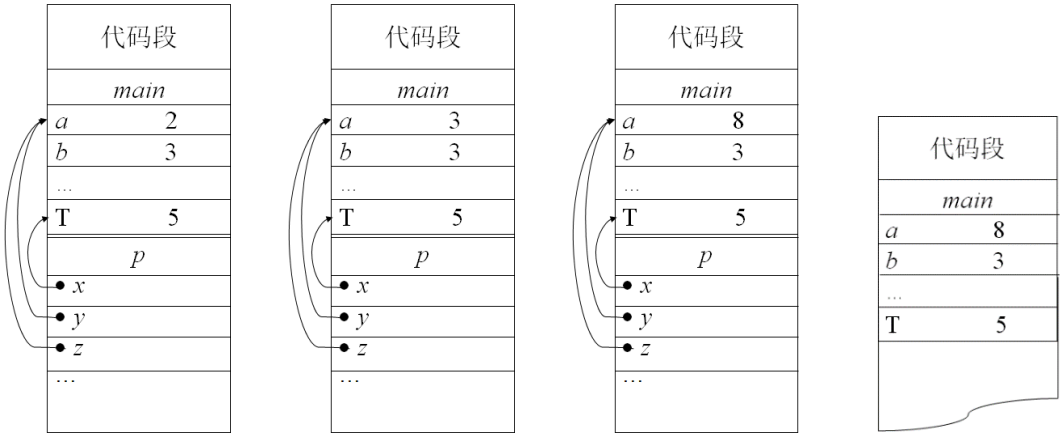
(a) 参数传递后　　(b) y:=y+1 执行后　　(c) z:=z+x 执行后

图 5　程序运行时控制栈

(2) 引用调用

主程序计算实参 a+b 的右值，在栈顶为表达式 a+b 分配空间 T，并将 T 和变量 a 的空间地址传递给过程 p 的参数空间，控制栈状态如图 6(a)所示。控制进入过程 p 中，对参数的所有访问均通过参数空间中保存的指针找到实参的空间，在实参的空间上进行。语句 y:=y+1 的执行结果，如图 6(b)所示。语句 z:=z+x 的执行结果，如图 6(c)所示。控制从过程 p 返回后，P 的活动记录出栈，控制栈状态如图 6(d)所示。

语句 writeln (′a=′,a)的执行结果是：a=8。



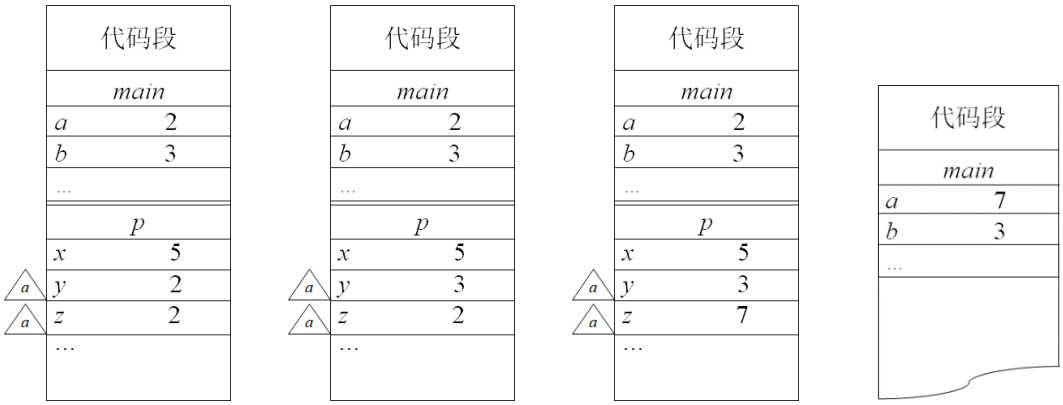
(a) 参数传递后　(b) y:=y+1 执行后　(c) z:=z+x 执行后　(d) 从 p 中返回后

图 6　程序运行时控制栈

(3) 复制恢复（假定按从左到右的顺序把结果复制回实参）

主程序计算实参 a+b、a 和 a 的右值，并传递给过程 p 的参数空间，记录与参数 y 和 z 相应的实参 a 的地址，控制栈状态如图 7(a)所示。控制进入过程 p 中，对参数的所有访问均在参数空间中进行。语句 y:=y+1 的执行结果，如图 7(b)所示。语句 z:=z+x 的执行结果，如图 7(c)所示。控制从过程 p 返回时，根据记录的参数相应的实参的地址，按照从左向右的顺序把参数空间的内容复制到实参的空间，P 的活动记录出栈，控制栈状态如图 7(d)所示。

语句 writeln (′a=′,a)的执行结果是：a=7。



(a) 参数传递后　(b) y:=y+1 执行后　(c) z:=z+x 执行后　(d) 从 p 中返回后

图 7　程序运行时控制栈

(4) 传名调用

将过程调用语句 p(a+b, a, a)被展开为：

a:=a+1;

a:=a+(a+b);

经过展开之后的程序体代码如下：

a:=2;

b:=3;

a:=a+1;

a:=a+(a+b);

writeln (′a=′,a)

这段代码执行的结果为：a=9。

7.5

Pascal 语言采用静态作用域规则。分析该程序可知，各过程之间的嵌套关系如图 8 所示。

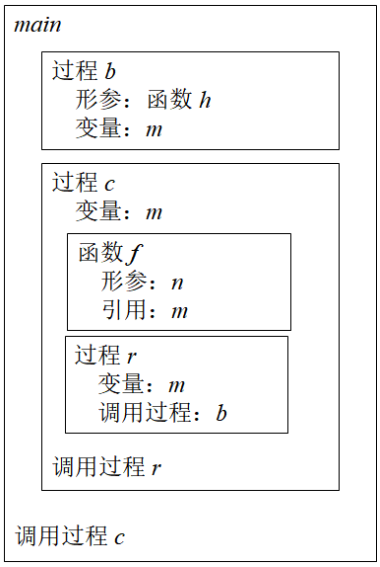


图 8　过程之间的静态嵌套及引用关系

(1)　程序运行过程中，主程序 *main* 调用过程 *c*，*c* 执行第(12)行的代码，为变量 *m* 赋值为 0，然后调用过程 *r*，过程 *r* 执行第(11)行的代码，为局部变量 *m* 赋值为 7，然后，以函数 f 为实参调用过程 b，语句 b(f)执行时，形参过程 h 被实参过程 f 所替代，执行第(4)行的语句，首先为局部变量 *m* 赋值为 3，语句 writeln(h(2))将激活实参过程 f，传递给其形参 n 的值是 2，执行第(8)行的语句，得到函数 f 的返回值 2，通过 writeln 语句打印输出。所以，该程序运行的结果是输出 2。

(2) 当控制在函数 f 中时，其活动树如图 9 所示。

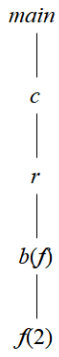


图 9　控制在函数 f 中时的活动树

(3) 在静态作用域规则下，当一个函数作为参数被传递时，它必须携带自己的访问链。

程序运行时，首先主程序 *main* 的活动记录入栈，执行第(13)行的代码，即调用过程 *c*。*c* 的活动记录入栈，其控制链和访问链都指向 *main* 的活动记录。过程 *c* 执行第(12)行的代码，为局部变量 *m* 赋值 0，调用过程 *r*。*r* 的活动记录入栈，其控制链和访问链都指向 *c* 的

活动记录。过程 *r* 执行第(11)行的代码，首先为其局部变量 *m* 赋值为 7，然后以函数 *f* 为实参调用过程 *b*，*r* 计算函数 *f* 的访问链（指向 *c* 的活动记录），并把函数 *f* 及其访问链传递给过程 *b*。过程 *b* 的活动记录入栈，其控制链指向 *r* 的活动记录，访问链指向 *main* 的活动记录。过程 *b* 执行第(4)行的代码，首先为其局部变量 *m* 赋值 3，然后在 writeln 语句中激活函数 *f*。*f* 的活动记录入栈，将 2 传递给它的参数，其控制链指向 *b* 的活动记录，访问链指向 *c* 的活动记录。函数 *f* 执行第(8)行的代码，计算 m+n=2，即函数 f 的返回值是 2，由语句 writeln 打印输出。

程序运行时，控制进入 *f* 后，控制栈的状态如图 10 所示，这里主要给出活动记录中的参数、控制链和访问链的状态。

控制栈状态如下：

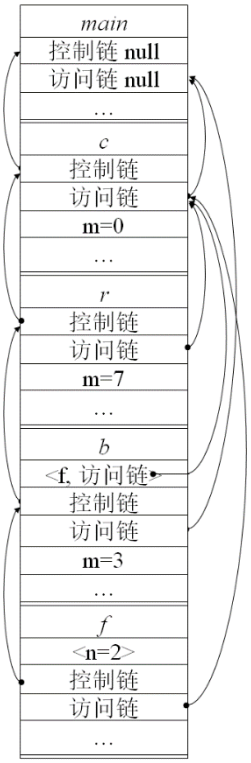


图 10　控制进入 f 后，控制栈的状态