运行时刻环境
Part3：非局部名字的访问

徐 伟

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年4月10日

非局部名字的访问

- 无过程嵌套（C语言）
- 有过程嵌套（Pascal语言）

非局部名字的访问

无过程嵌套时的数据访问

- 过程体中的非局部引用可以直接使用静态确定的地址（非局部数据此时就是全局数据）
- 局部变量在栈顶的活动记录中，可以通过*base_sp*指针来访问
- 无须深入栈中取数据，无须访问链
  - 因此，访问链是optional的

```
int a[11];
void readArray(){};
int partition(int m, int n){};
void quickSort(int m, int n){};
main(){
   readArray();
   a[0] = -9999;
   a[10] = 9999;
   quickSort(1,9);
}
```

非局部名字的访问

有过程嵌套的静态作用域

- PASCAL中，如果过程A的声明中包含了过程B的声明，那么B可以使用在A中声明的变量
- 然而，当B的代码激活执行时，如何找到A的活动记录呢?
  - 这里需要建立访问链

```
void A() {
    int x, y;
    void B() {
        int b;
        x = b + y;
    }
    void C() { B(); }
    C();
    B();
}
```

当A调用C，C又调用B时：

当A直接调用B时：

非局部名字的访问

- 有过程嵌套的静态作用域

sort
  readarray
  exchange
  quicksort
    partition

```
(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}.
```

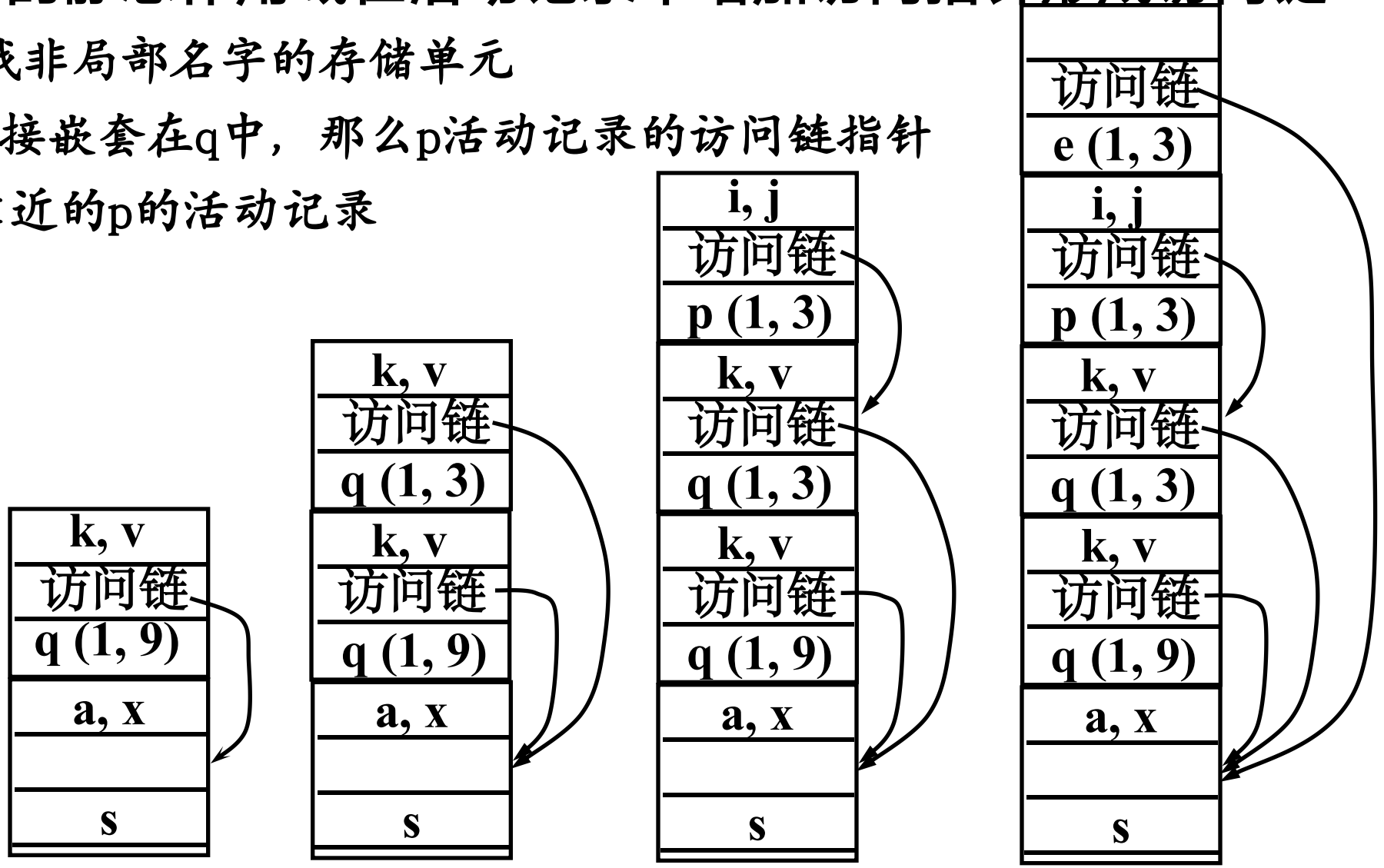
非局部名字的访问

- 有过程嵌套的静态作用域
  - 过程嵌套深度：主程序为1，进入一个被包围的过程时加1
  - 变量的嵌套深度：它的声明所在过程的嵌套深度作为该名字的嵌套深度

```
sort                    1
  readarray             2
  exchange              2
  quicksort       2
       partition        3
```

非局部名字的访问

- 过程嵌套的静态作用域在活动记录中增加访问指针形成访问链
  - 用来寻找非局部名字的存储单元
  - 假设p直接嵌套在q中，那么p活动记录的访问链指针指向最靠近的p的活动记录



非局部名字的访问

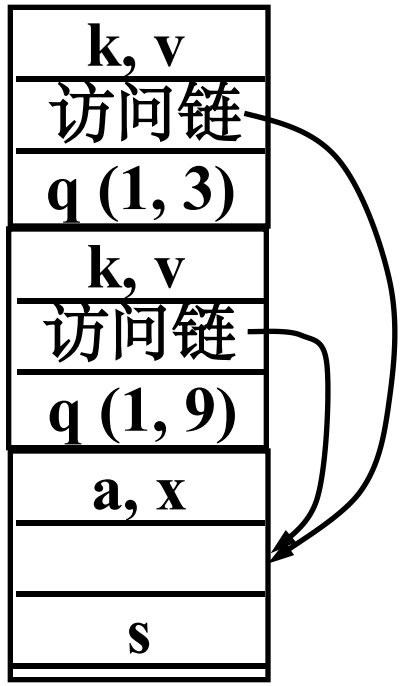
- 两个关键问题需要解决：
  - 通过访问链访问非局部引用
  - 访问链的建立

非局部名字的访问

访问非局部名字的存储单元

- 假定过程p的嵌套深度为n_p，它引用嵌套深度为n_a的变量a，$n_a \le n_p$，如何访问a的存储单元?

```
sort                1
  readarray         2
  exchange          2
  quicksort 2
      partition     3
```

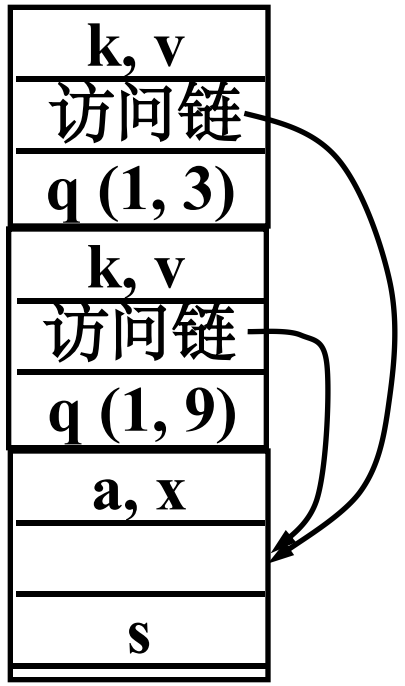


非局部名字的访问

访问非局部名字的存储单元

- 假定过程p的嵌套深度为n_p，它引用嵌套深度为n_a的变量a，$n_a \le n_p$，如何访问a的存储单元?
  - 从栈顶的活动记录开始，追踪访问链$n_p - n_a$次
  - 到达a的声明所在过程的活动记录
  - 访问链的追踪可用间接操作完成

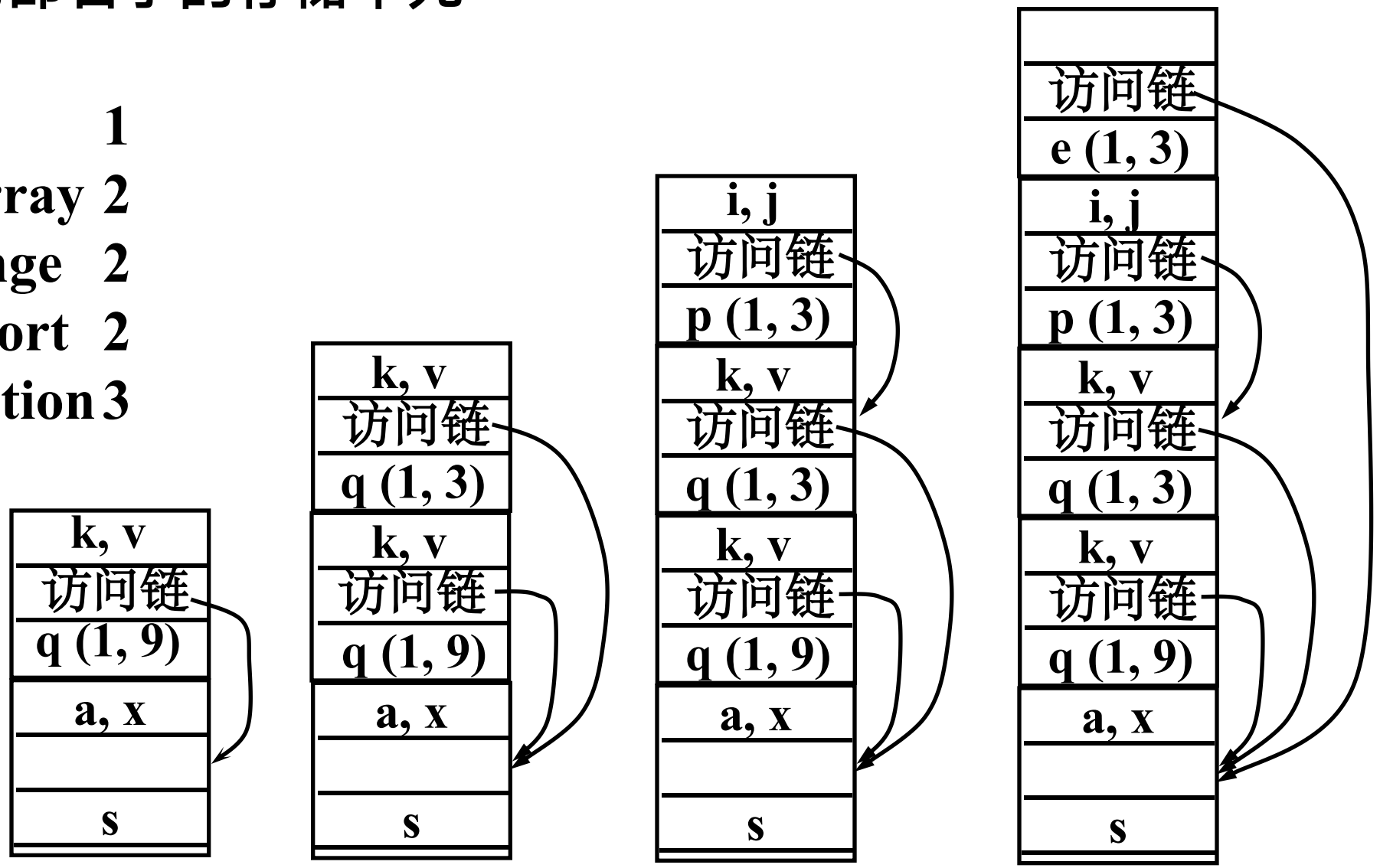
```
sort                1
  readarray         2
  exchange          2
  quicksort 2
      partition     3
```



非局部名字的访问——举例

- 访问非局部名字的存储单元

```
sort          1
  readarray   2
  exchange    2
  quicksort   2
    partition 3
```



非局部名字的访问

- 建立访问链(过程调用序列代码的一部分)
  - 假定嵌套深度为n_p的过程p调用嵌套深度为n_x的过程x

(1) $n_p < n_x$的情况

```
sort                1
    readarray       2
    exchange        2
    quicksort       2
        partition   3
```

这时x肯定就声明在p中

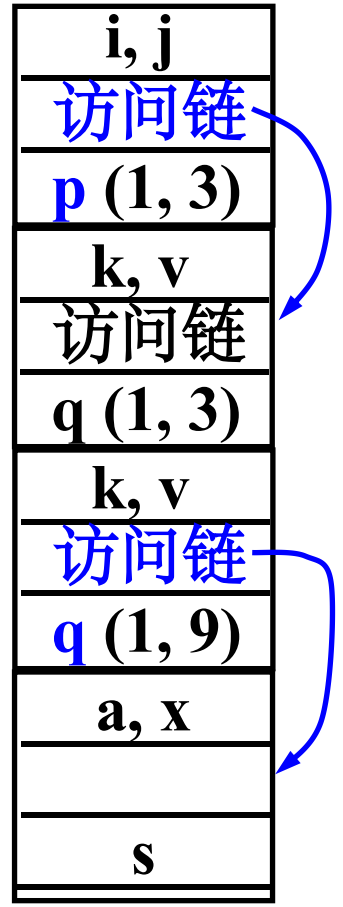
非局部名字的访问

建立访问链

- 假定嵌套深度为n_p的过程p调用嵌套深度为n_x的过程x

(1) $n_p < n_x$的情况

- 这时x肯定就声明在p中(嵌套)
- 被调用过程的访问链必须指向调用过程的活动记录（次栈顶）的访问链
- sort调用quicksort、quicksort 调用 partition



非局部名字的访问

- 建立访问链
  - 假定嵌套深度为n_p的过程p调用嵌套深度为n_x的过程x

(2) $n_p \geq n_x$的情况

```
sort                1
    readarray       2
    exchange        2
    quicksort       2
        partition   3
```

这时p和x的嵌套深度分别为1，2，…，n_x-1的外围过程肯定相同

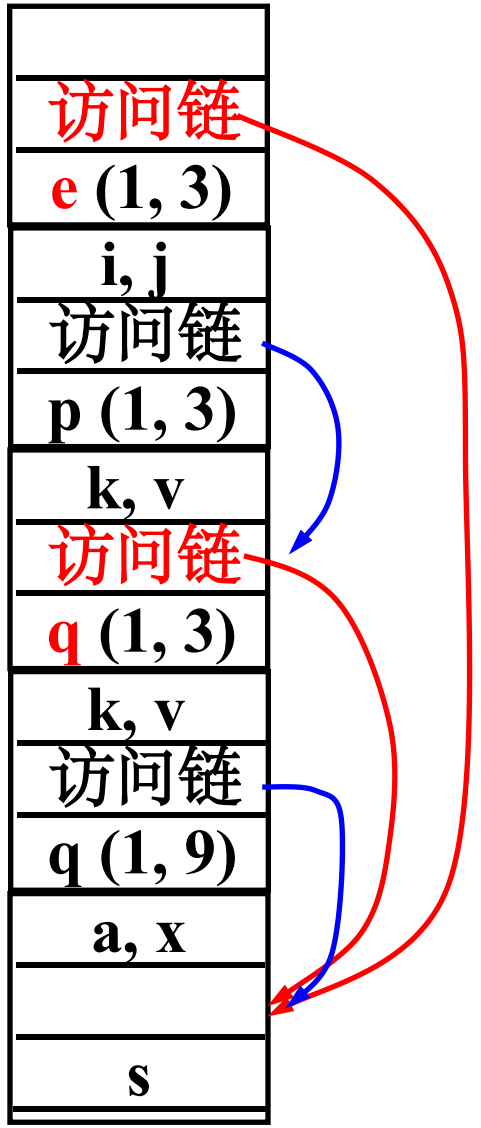
非局部名字的访问

建立访问链

- 假定嵌套深度为n_p的过程p调用嵌套深度为n_x的过程x

(2) $n_p \geq n_x$的情况

- 沿着访问链追踪$n_p - n_x + 1$次，到达了静态包围x和p的且离它们最近的那个过程的最新活动记录
- 所到达的活动记录就是x的活动记录中的访问链应该指向的那个活动记录
- partition调用exchange, quicksort调用自身



参数传递

- 实参与形参
  - 存储单元（左值）
  - 存储内容（右值）

根据所传递的实参的“内容”，参数传递可分为：

- 传值调用：传递实参的右值到形参单元；
- 引用调用：传递实参的左值到形参单元。

参数传递举例

```
procedure  swap( a , b )
  a, b : int;  temp : int;
begin
  temp := a ;
  a := b;
  b := temp;
end.
```

讨论下面程序在不同参数传递方式下输出：

```
x := 10 ; y := 20;
    swap( x,y );
    print ( x, y );
```

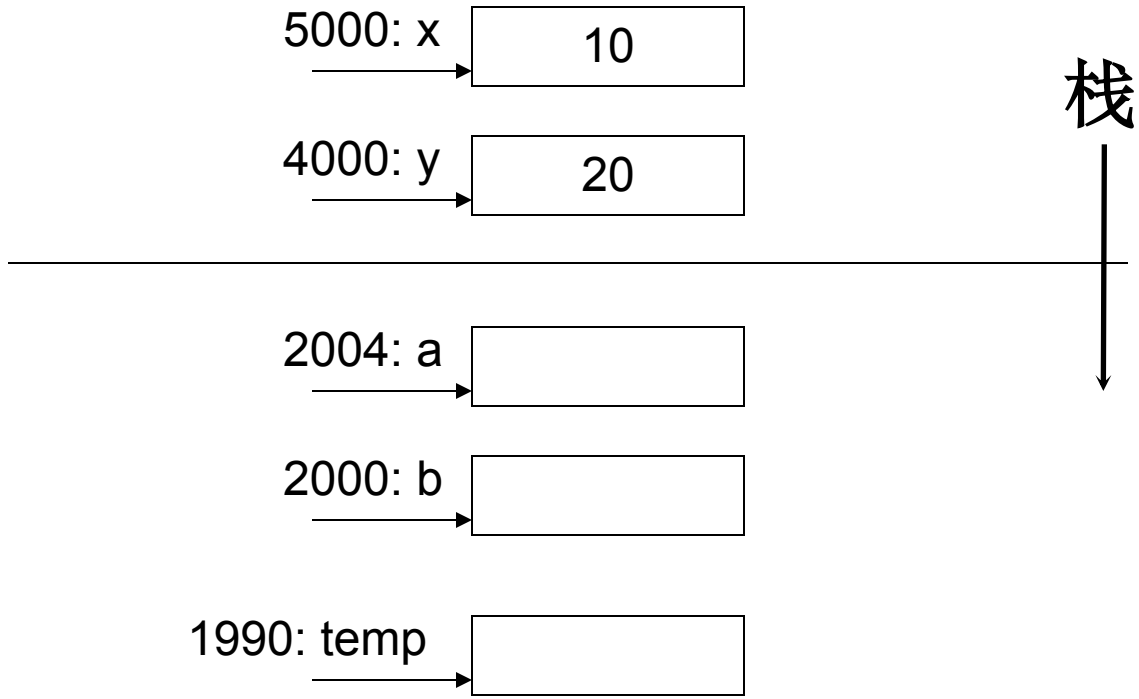
参数传递举例

讨论下面程序在不同参数传递方式下输出：

1) x := 10 ; y := 20;

swap(x,y);

print (x, y);

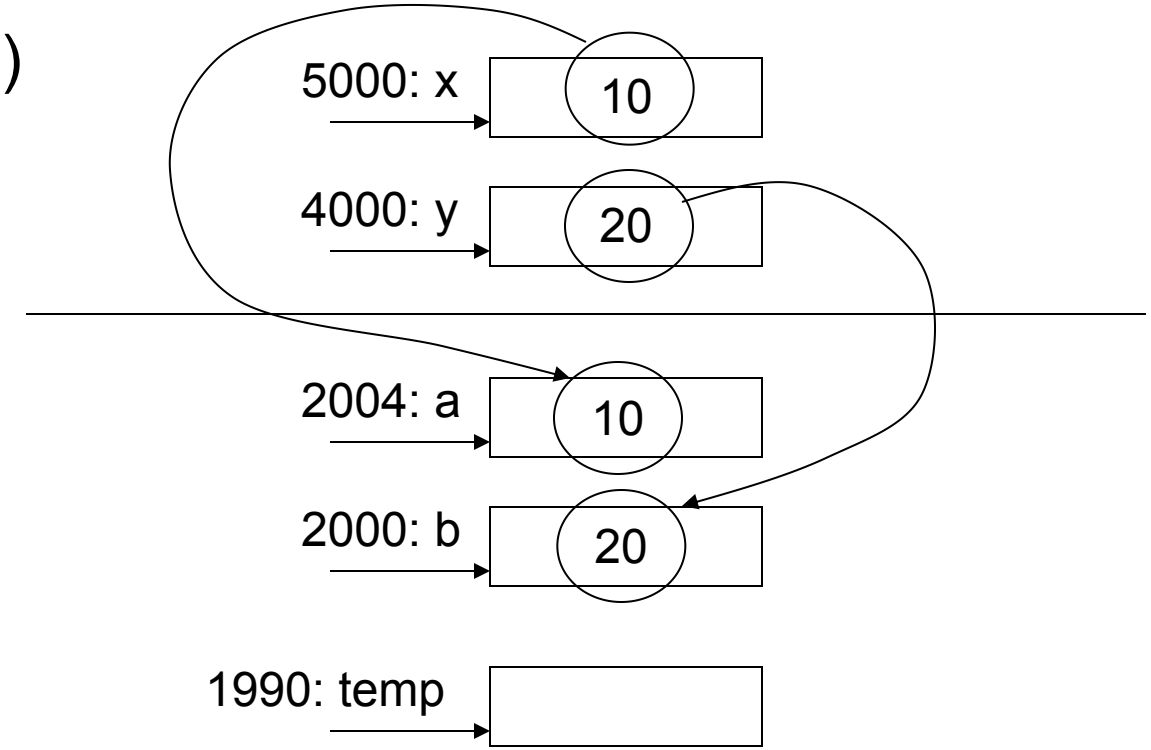


实参x,y和过程swap中形参a,b,和局部数据temp的存储分布示意

参数传递 - 传值调用

过程调用 – swap(x,y)

- 传参 – 形、实结合

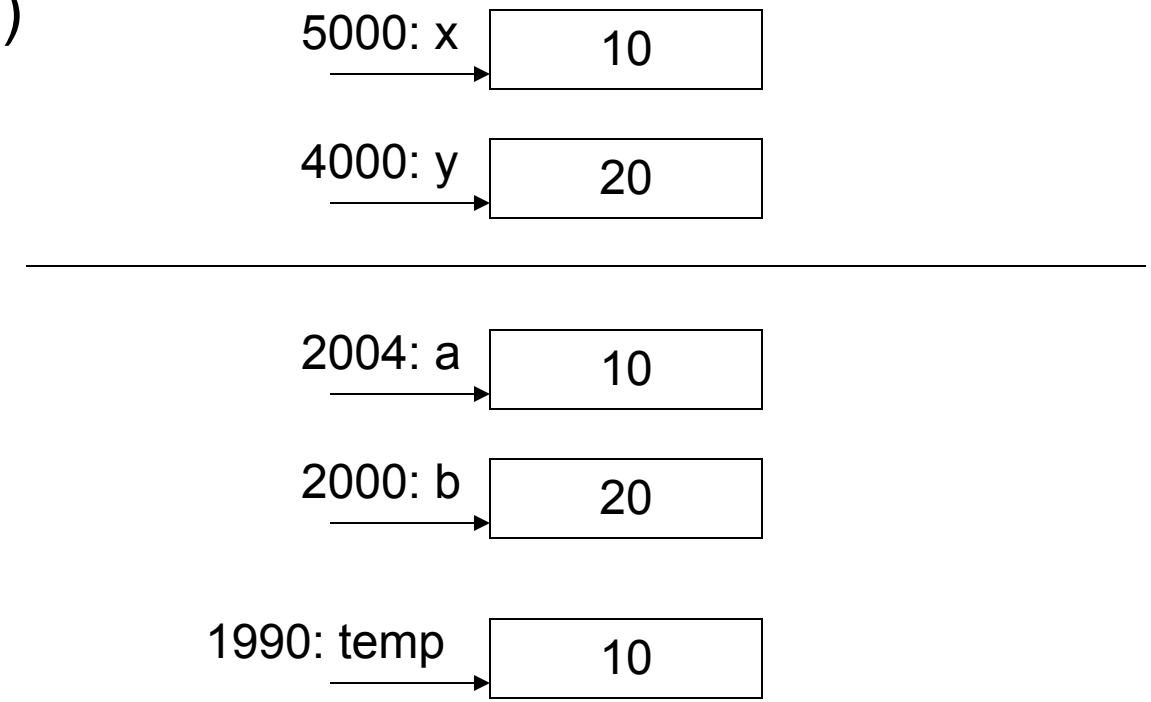


参数传递 - 传值调用

过程调用 – swap(x,y)

- 过程执行

  temp := a

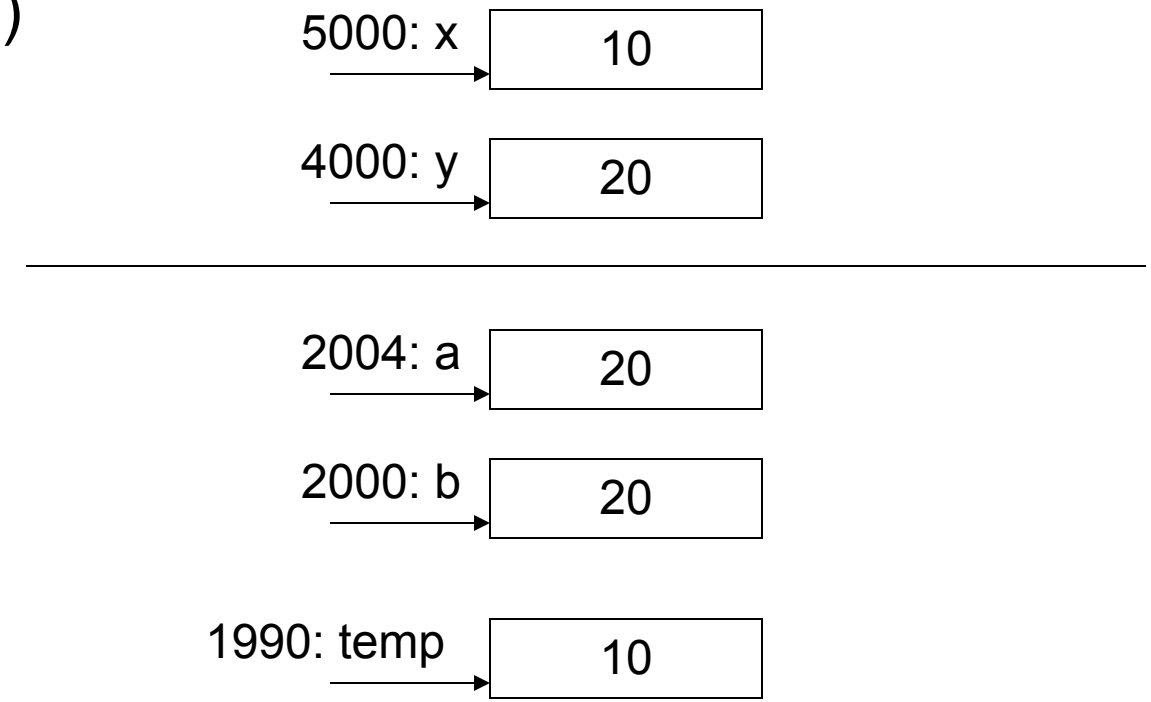


参数传递 - 传值调用

过程调用 – swap(x,y)

- 过程执行

  a := b

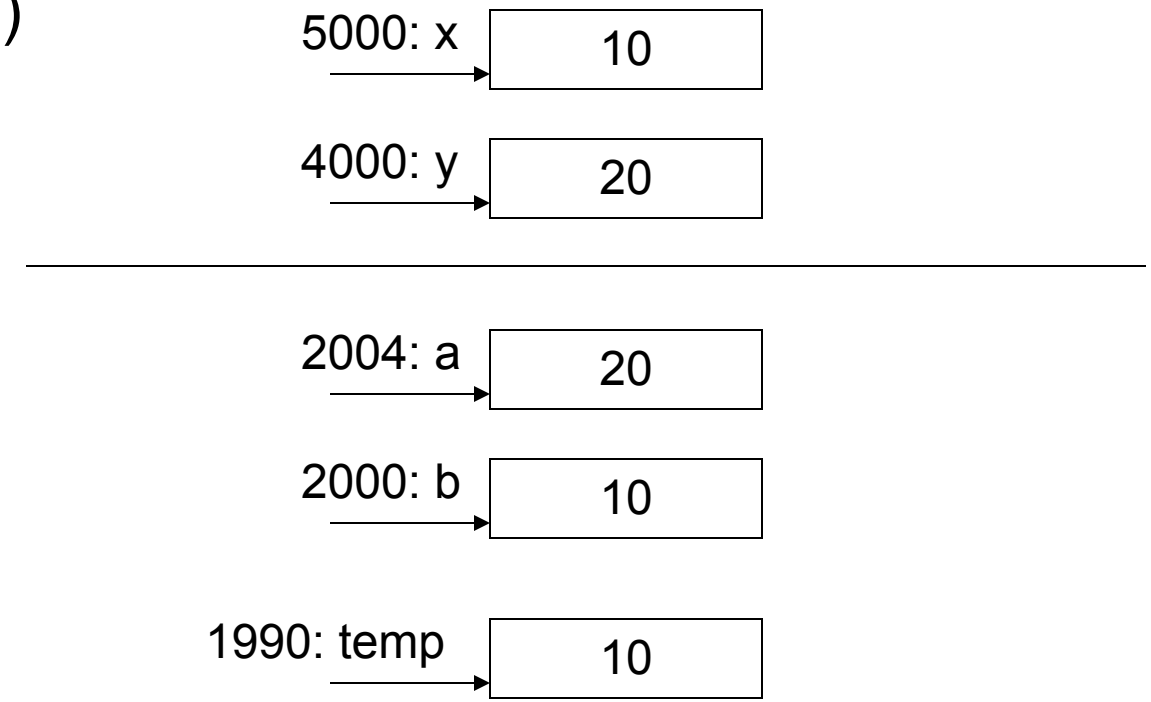


参数传递 - 传值调用

过程调用 – swap(x,y)

- 过程执行

  b := temp

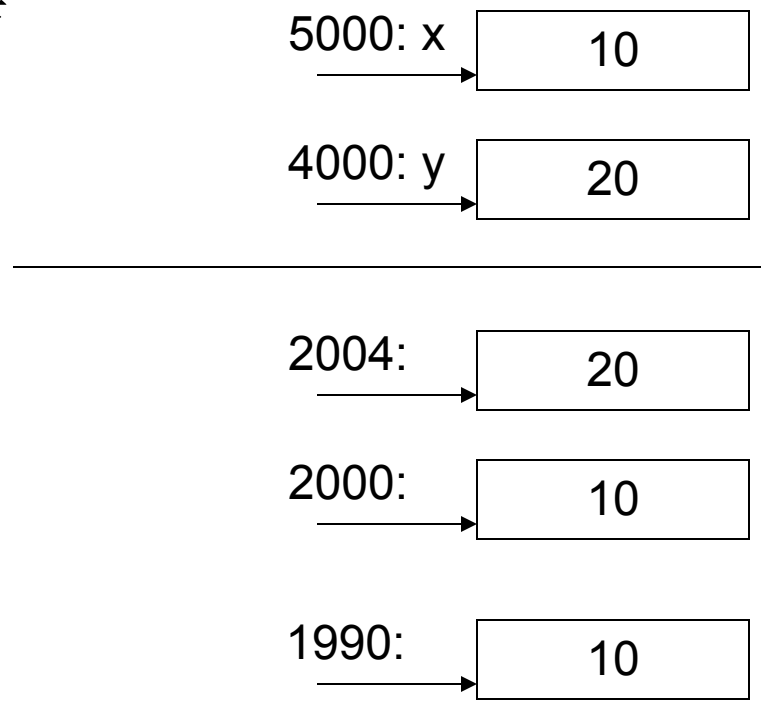


参数传递 - 传值调用

过程swap(x,y)执行后

- print(x, y)

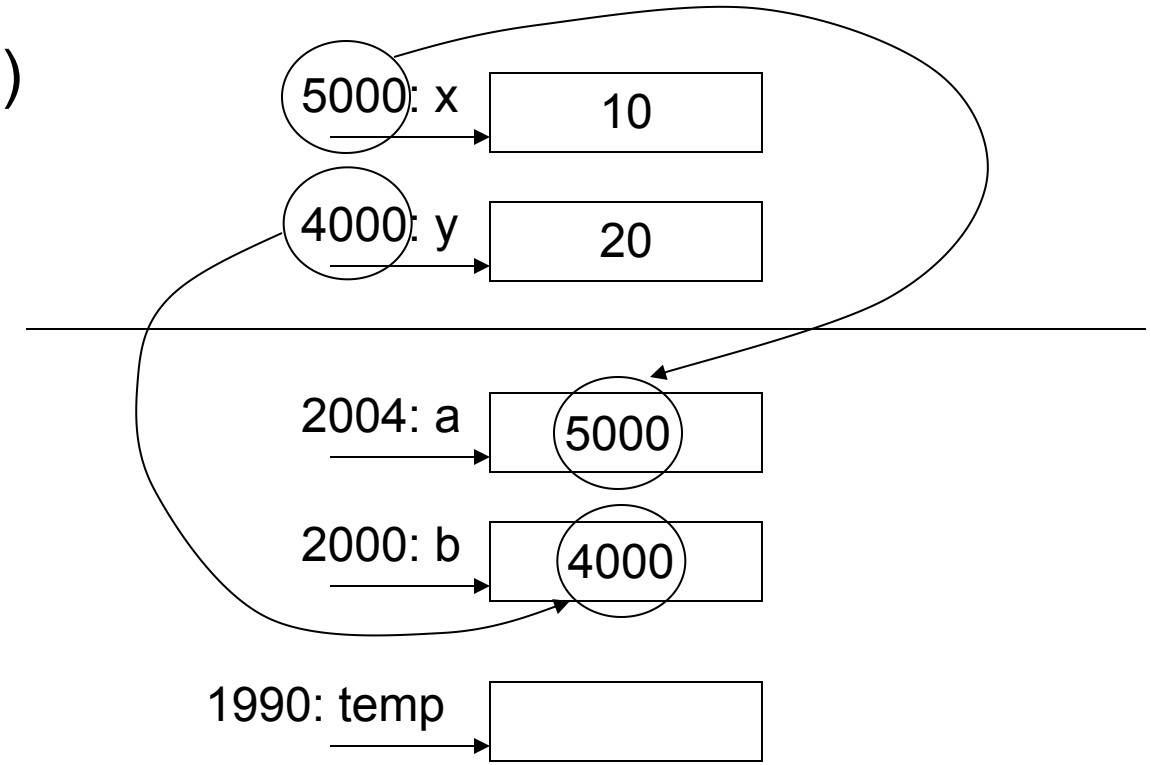
10, 20



参数传递 - 引用调用

过程调用 – swap(x,y)

- 传参 – 形、实结合

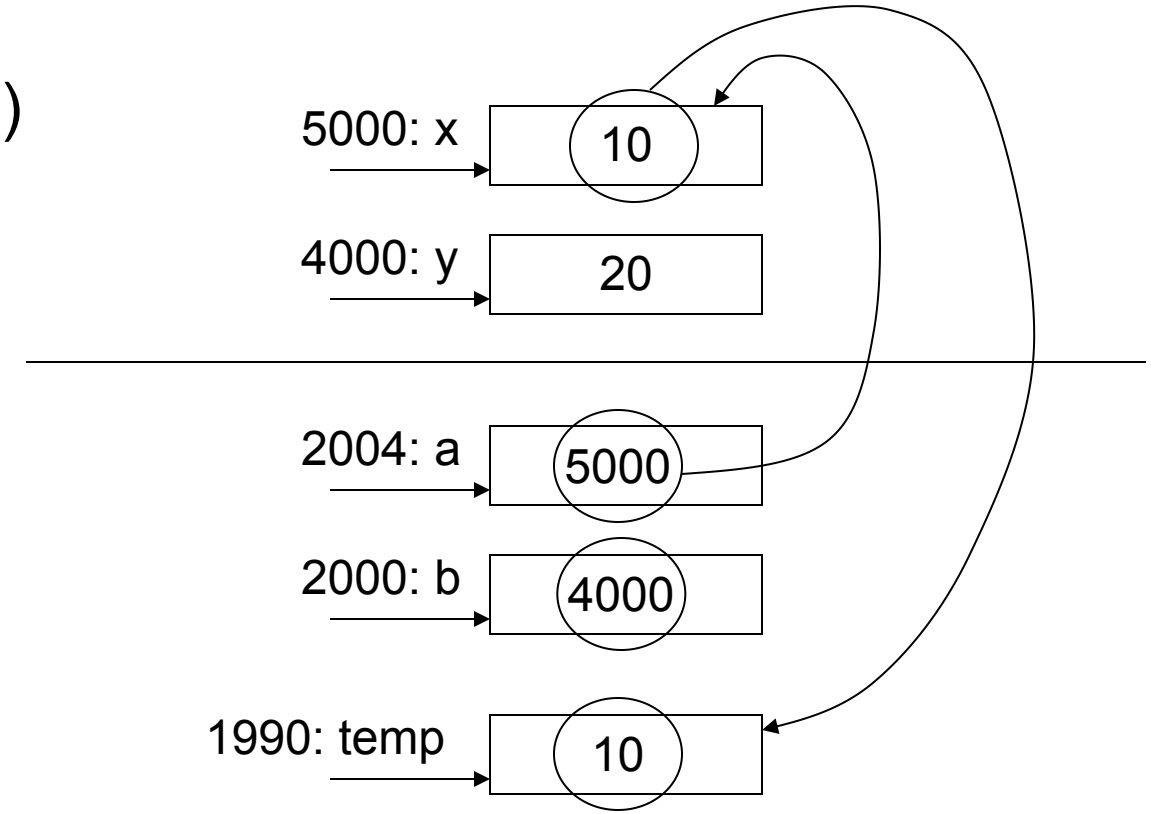


参数传递 - 引用调用

过程调用 – swap(x,y)

- 过程执行

  temp := a

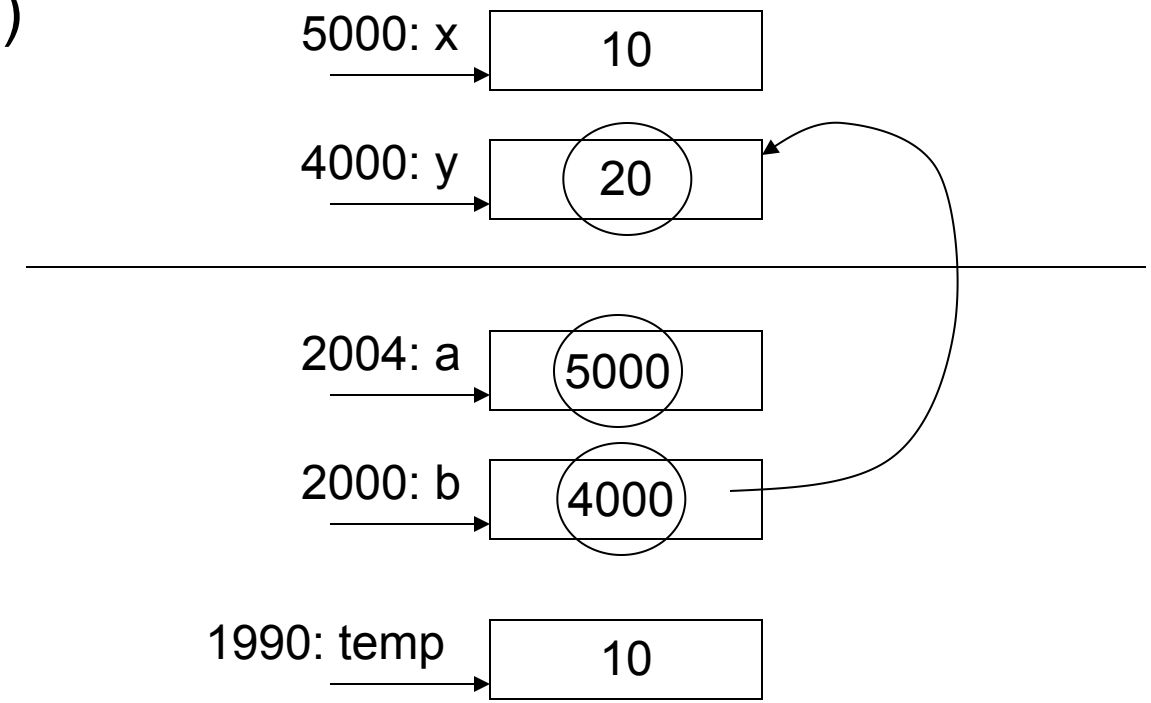


参数传递 - 引用调用

过程调用 – swap(x,y)

- 过程执行

  a := b

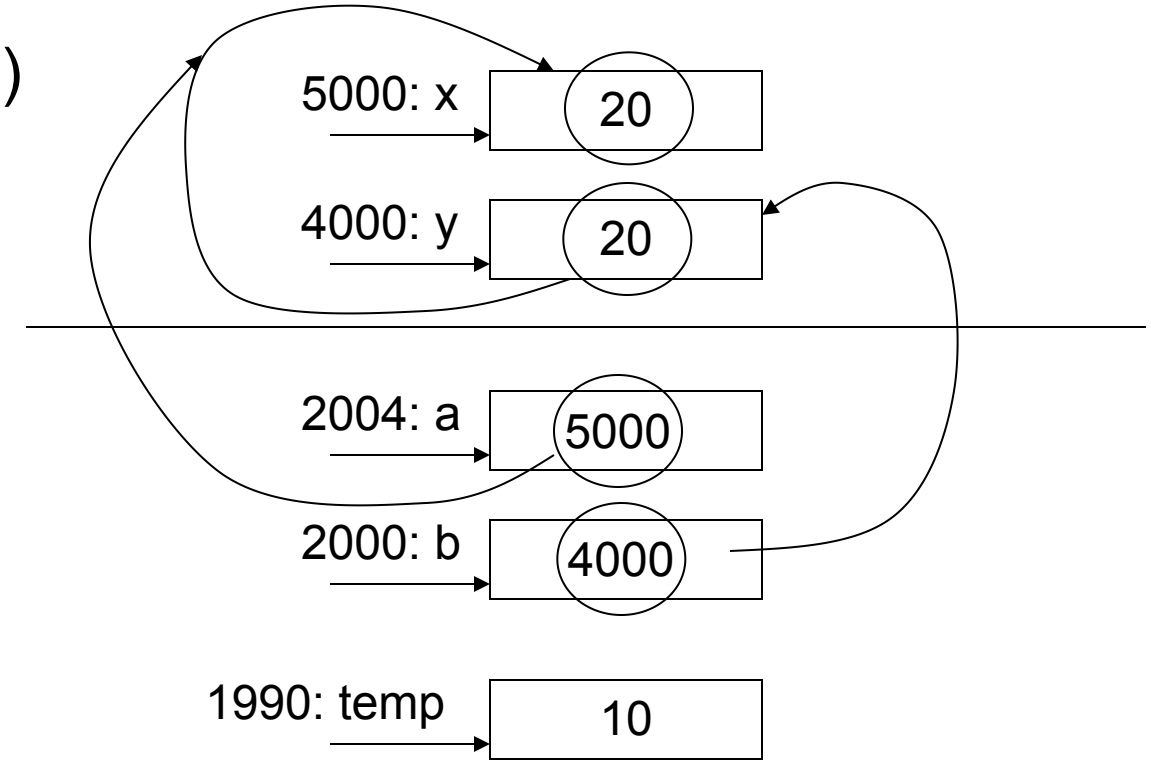


参数传递 - 引用调用

过程调用 – swap(x,y)

- 过程执行

  a := b

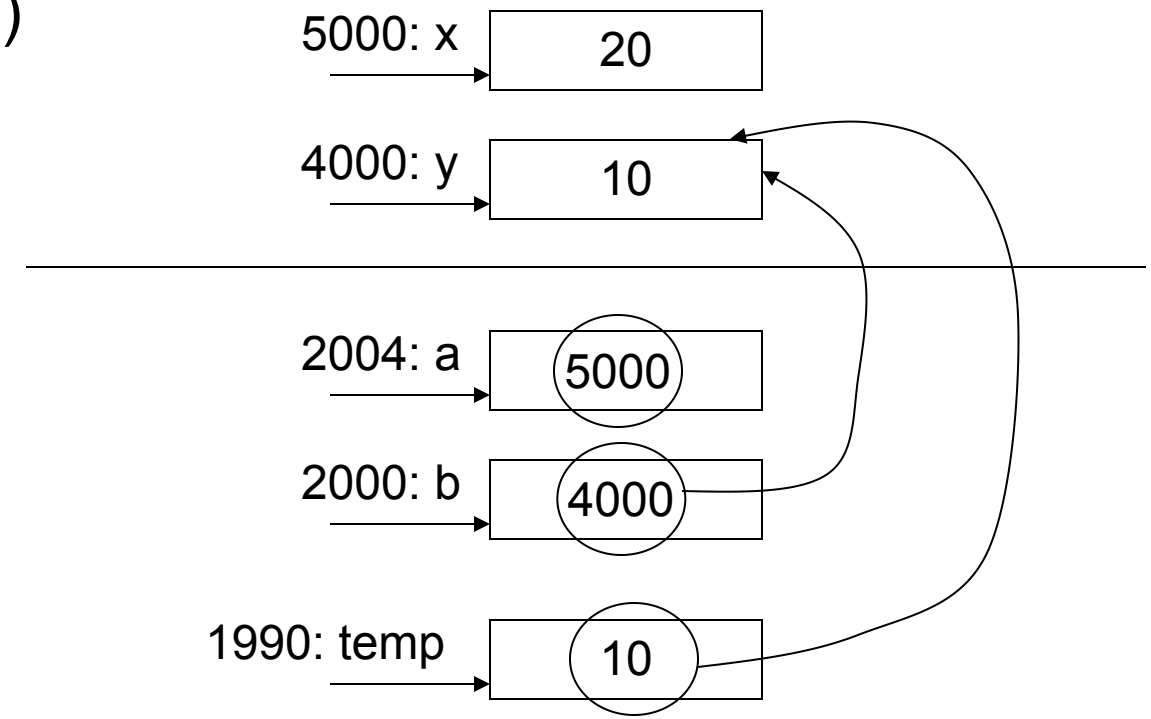


参数传递 - 引用调用

过程调用 – swap(x,y)

- 过程执行

  b := temp

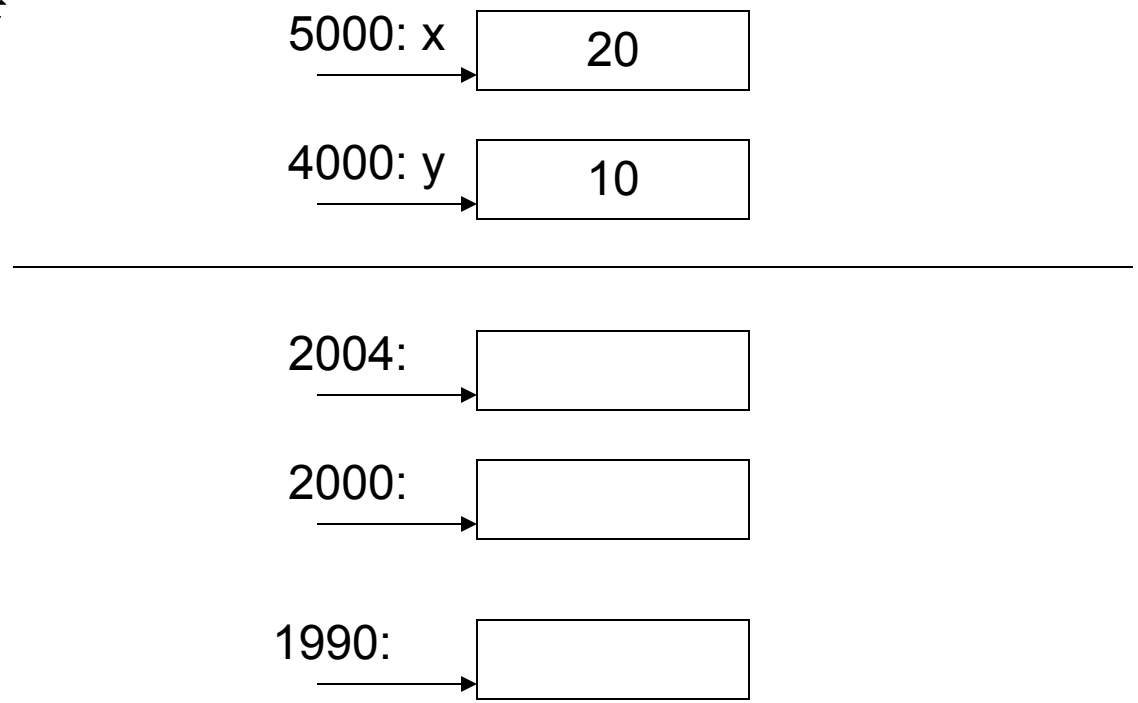


参数传递 - 引用调用

过程swap(x,y)执行后

- print(x, y)

20, 10



参数传递举例

以下C程序的输出是什么？

```
void func(char *s){s = (char*)malloc(10);}
int main()
{
 char *p = NULL;
 func(p);
 if(!p)printf("error\n");else printf("ok\n");
 return 0;
}
```

参数传递 swap

```
void swap1(int p,int q)
{
    int temp;
    temp = p;
    p = q;
    q = temp;
}
```

```
pushl %ebp
movl  %esp, %ebp
subl  $4, %esp//allocation
movl  8(%ebp), %eax //get p
movl  %eax, -4(%ebp)//temp = p
movl  12(%ebp), %eax //get q
movl  %eax, 8(%ebp) //p = q
movl  -4(%ebp), %eax // get temp
movl  %eax, 12(%ebp) // q=temp
leave
ret
```

参数传递 swap

```
void swap2(int *p,int *q)
{
    int  temp;
    temp = *p;
    *p   = *q;
    *q   = temp;
}
```

```
pushl %ebp
movl  %esp, ebp
subl  $4, %esp
movl  8(%ebp), %eax//get p's addr
movl  (%eax), %eax// get p's value
movl  %eax, -4(%ebp)//temp=*p
movl  8(%ebp), %edx//get p's addr
movl  12(%ebp), %eax//get q's addr
movl  (%eax), %eax//get q's value
movl  %eax, (%edx)
movl  12(%ebp), %edx
movl  -4(%ebp), %eax
movl  %eax, (%edx)
leave
ret
```

参数传递 swap

```
void swap3(int *p, int *q)
{
    int *temp   ;
    temp = p    ;
    p    = q    ;
    q    = temp;
}
```

```
pushl %ebp
movl  %esp, ebp
subl  $4, %esp
movl  8(%ebp), %eax
movl  %eax, -4(%ebp)
movl  12(%ebp), %eax
movl  %eax, 8(%ebp)
movl  -4(%ebp), %eax
movl  %eax, 12(%ebp)
leave
ret
```

参数传递 swap

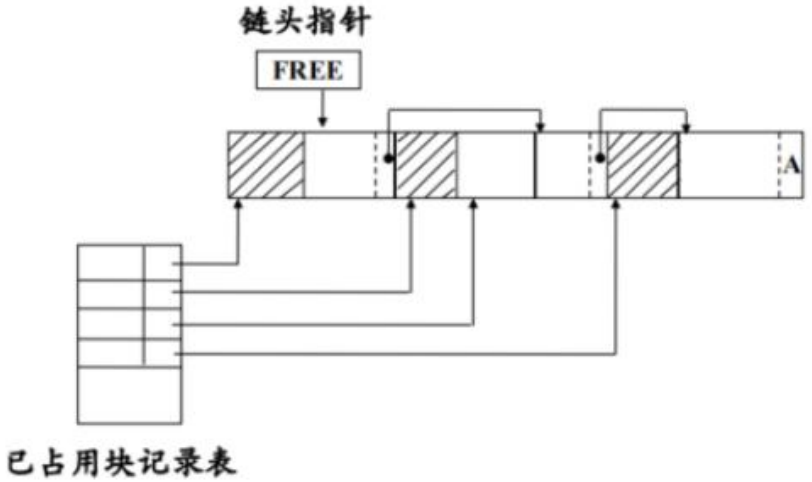
```
void swap4(int &p, int &q)
{
    int temp;
    temp = p;
    p    = q;
    q    = temp;
}
```

```
pushl %ebp
movl  %esp,%ebp
subl  $4, %esp
movl  8(%ebp), %eax
movl  (%eax), %eax
movl  %eax, -4(%ebp)
movl  8(%ebp), %edx
movl  12(%ebp), %eax
movl  (%eax), %eax
movl  %eax, (%edx)
movl  12(%ebp), %edx
movl  -4(%ebp), %eax
movl  %eax, (%edx)
leave
ret
```

堆管理

堆空间

- 用于存放生命周期不确定、或生存到被明确删除为止的数据对象
- e.g., new生成的对象可以生存到被delete为止
  malloc申请的空间生存到被free为止



存储管理器

分配/回收堆空间的子系统

- **分配：为内存请求分配一段连续、适当大小的堆空间**
  - 首先从空闲的堆空间选择
  - 如内存紧张，可以回收一部分内存
    - C、C++需要手动回收空间
    - Java可以自动回收空间
- **评价指标**
  - 空间效率
  - 程序效率
  - 管理效率

一起努力
打造国产基础软硬件体系！

徐 伟

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2025年4月10日