写到超出 A 数组结尾的地方。因为这些字中的一个很可能是已分配块的边界标记脚部，所以我们可能不会发现这个错误，直到在这个程序的后面很久释放这个块时，此时，分配器中的合并代码会戏剧性地失败，而没有任何明显的原因。这是“在远处起作用(action at distance)”的一个阴险的示例，这类“在远处起作用”是与内存有关的编程错误的典型情况。

9.11.5　造成错位错误

错位(off-by-one)错误是另一种很常见的造成覆盖错误的来源：

```
1   /* Create an nxm array */
2   int **makeArray2(int n, int m)
3   {
4       int i;
5       int **A = (int **)Malloc(n * sizeof(int *));
6
7       for (i = 0; i <= n; i++)
8           A[i] = (int *)Malloc(m * sizeof(int));
9       return A;
10  }
```

这是前面一节中程序的另一个版本。这里我们在第 5 行创建了一个 n 个元素的指针数组，但是随后在第 7 行和第 8 行试图初始化这个数组的 $n+1$ 个元素，在这个过程中覆盖了 A 数组后面的某个内存位置。

9.11.6　引用指针，而不是它所指向的对象

如果不太注意 C 操作符的优先级和结合性，我们就会错误地操作指针，而不是指针所指向的对象。比如，考虑下面的函数，其目的是删除一个有 *size 项的二叉堆里的第一项，然后对剩下的*size-1 项重新建堆：

```
1   int *binheapDelete(int **binheap, int *size)
2   {
3       int *packet = binheap[0];
4
5       binheap[0] = binheap[*size - 1];
6       *size--; /* This should be (*size)-- */
7       heapify(binheap, *size, 0);
8       return(packet);
9   }
```

在第 6 行，目的是减少 size 指针指向的整数的值。然而，因为一元运算符 -- 和 * 的优先级相同，从右向左结合，所以第 6 行中的代码实际减少的是指针自己的值，而不是它所指向的整数的值。如果幸运地话，程序会立即失败；但是更有可能发生的是，当程序在执行过程后很久才产生出一个不正确的结果时，我们只有一头的雾水。这里的原则是当你对优先级和结合性有疑问的时候，就使用括号。比如，在第 6 行，我们可以使用表达式 (*size)--，清晰地表明我们的意图。

9.11.7　误解指针运算

另一种常见的错误是忘记了指针的算术操作是以它们指向的对象的大小为单位来进行的，而这种大小单位并不一定是字节。例如，下面函数的目的是扫描一个 int 的数组，并