**练习题 2.10**　对于任一位向量 $a$，有 $a \hat{} a = 0$。应用这一属性，考虑下面的程序：

```
1  void inplace_swap(int *x, int *y) {
2      *y = *x ^ *y;  /* Step 1 */
3      *x = *x ^ *y;  /* Step 2 */
4      *y = *x ^ *y;  /* Step 3 */
5  }
```

正如程序名字所暗示的那样，我们认为这个过程的效果是交换指针变量 x 和 y 所指向的存储位置处存放的值。注意，与通常的交换两个数值的技术不一样，当移动一个值时，我们不需要第三个位置来临时存储另一个值。这种交换方式并没有性能上的优势，它仅仅是一个智力游戏。

以指针 x 和 y 指向的位置存储的值分别是 $a$ 和 $b$ 作为开始，填写下表，给出在程序的每一步之后，存储在这两个位置中的值。利用^的属性证明达到了所希望的效果。回想一下，每个元素就是它自身的加法逆元($a \hat{} a = 0$)。

| 步骤 | *x | *y |
|---|---|---|
| 初始 | $a$ | $b$ |
| 第1步 | | |
| 第2步 | | |
| 第3步 | | |

**练习题 2.11**　在练习题 2.10 中的 inplace_swap 函数的基础上，你决定写一段代码，实现将一个数组中的元素头尾两端依次对调。你写出下面这个函数：

```
1  void reverse_array(int a[], int cnt) {
2      int first, last;
3      for (first = 0, last = cnt-1;
4           first <= last;
5           first++,last--)
6           inplace_swap(&a[first], &a[last]);
7  }
```

当你对一个包含元素 1、2、3 和 4 的数组使用这个函数时，正如预期的那样，现在数组的元素变成了 4、3、2 和 1。不过，当你对一个包含元素 1、2、3、4 和 5 的数组使用这个函数时，你会很惊奇地看到得到数字的元素为 5、4、0、2 和 1。实际上，你会发现这段代码对所有偶数长度的数组都能正确地工作，但是当数组的长度为奇数时，它就会把中间的元素设置成 0。

A. 对于一个长度为奇数的数组，长度 cnt = $2k+1$，函数 reverse_array 最后一次循环中，变量 first 和 last 的值分别是什么？

B. 为什么这时调用函数 inplace_swap 会将数组元素设置为 0？

C. 对 reverse_array 的代码做哪些简单改动就能消除这个问题？

位级运算的一个常见用法就是实现**掩码**运算，这里掩码是一个位模式，表示从一个字中选出的位的集合。让我们来看一个例子，掩码 0xFF(最低的 8 位为 1)表示一个字的低位字节。位级运算 x&0xFF 生成一个由 x 的最低有效字节组成的值，而其他的字节就被置为 0。比如，对于 x= 0x89ABCDEF，其表达式将得到 0x000000EF。表达式~0 将生成一个全 1 的掩码，不管机器的字大小是多少。尽管对于一个 32 位机器来说，同样的掩码可以写成 0xFFFFFFFF，但是这样的代码不是可移植的。