Practice Problem 2.11 (solution page 182)

Armed with the function **inplace_swap** from Problem 2.10, you decide to write code that will reverse the elements of an array by swapping elements from opposite ends of the array, working toward the middle.

You arrive at the following function:

```
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  }
```

When you apply your function to an array containing elements 1, 2, 3, and 4, you find the array now has, as expected, elements 4, 3, 2, and 1. When you try it on an array with elements 1, 2, 3, 4, and 5, however, you are surprised to see that the array now has elements 5, 4, 0, 2, and 1. In fact, you discover that the code always works correctly on arrays of even length, but it sets the middle element to 0 whenever the array has odd length.

A. For an array of odd length **cnt** $= 2k + 1$, what are the values of variables **first** and **last** in the final iteration of function **reverse_array**?

B. Why does this call to function **inplace_swap** set the array element to 0?

C. What simple modification to the code for **reverse_array** would eliminate this problem?

Practice Problem 2.10 (solution page 182)

As an application of the property that $a \hat{} a = 0$ for any bit vector a, consider the following program:

```
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  }
```

As the name implies, we claim that the effect of this procedure is to swap the values stored at the locations denoted by pointer variables **x** and **y**. Note that unlike the usual technique for swapping two values, we do not need a third location to temporarily store one value while we are moving the other. There is no performance advantage to this way of swapping; it is merely an intellectual amusement.

Starting with values a and b in the locations pointed to by **x** and **y**, respectively, fill in the table that follows, giving the values stored at the two locations after each step of the procedure. Use the properties of **^** to show that the desired effect is achieved. Recall that every element is its own additive inverse (that is, $a \hat{} a = 0$).

Step	*x	*y
Initially	a	b
Step 1	a	a^b
Step 2	b	a^b
Step 3	b	a

A. first = last = k (for 5,4,0,2,1, the middle element has index 2).

B. inplace_swap(x,y) will set y to 0 if x=y.

Suppose *x = *y = a. Then Step 1: *y = *x ^ *y = a ^ a = 0 = *x

Step 2: *x = *x ^ *y = 0 ^ 0 = 0

Step 3: *y = *x ^ *y = 0 ^ 0 = 0

C. Change first <= last to first < last in the for loop condition.