## 实验七 Array II

## 学号: 09017423 姓名：杨彬

### 实验目的

Objectives This lab was designed to reinforce programming concepts from Chapter 7 of C++ How To Program: Sixth Edi-tion. In this lab, you will practice:

.Sorting data using the bubble sort algorithm.

The follow-up question and activity also will give you practice:

.Optimizing a program to be more efficient.

(本实验旨在加强规划概念从C++如何计划7章：第六EDI等。在这个实验室里，你要练习：

使用冒泡排序算法对数据进行排序。

后续的问题和活动也会给你练习：

优化程序以提高效率。)

### 实验内容

Exp1

Experiments In the bubble sort algorithm, smaller values gradually ―bubble‖ their way upward to the top of the array like air bubbles rising in water, while the larger values sink to the bottom. The bubble sort makes several passes through the array. On each pass, successive pairs of elements are compared. If a pair is in increasing order (or the values are identical), we leave the values as they are. If a pair is in decreasing order, their values are swapped in the array.

Write a program that sorts an array of 10 integers using bubble sort

(在冒泡排序算法实验，较小的值逐渐―泡沫‖路上行到数组像气泡在水中上升的顶部，而较大的值沉到底部。冒泡排序使数个数组通过。在每个传递过程中，比较连续的元素对。如果一对在递增的顺序（或值是相同的），我们保留值，因为它们是。如果一对是递减的，它们的值将在数组中交换。

编写一个程序，用冒泡排序对10个整数数组进行排序。）

Follow-Up Question and Activity

1. This bubble sort algorithm is inefficient for large arrays. Make the following simple modifications to improve the performance of the bubble sort:

a) After the first pass, the largest number is guaranteed to be in the highest-numbered element of the array; after the second pass, the two highest numbers are “in place,” and so on. Instead of making nine comparisons on every pass, modify the bubble sort to make eight comparisons on the second pass, seven onthe third pass, and so on.

b) The data in the array may already be in the proper order or near-proper order, so why make nine passesif fewer will suffice? Modify the sort to check at the end of each pass if any swaps have been made. If none have been made, then the data must already be in the proper order, so the program should terminate. If swaps have been made, then at least one more pass is needed.

(跟进问题和活动

1。这种冒泡排序算法对大型数组无效。进行以下简单的修改以提高冒泡排序的性能：

a）第一次传递后，最大的数字被保证在数组的最高编号元素中；第二次传递后，两个最高的数字是“就位”，等等。不要在每次传球时进行九次比较，修改泡泡排序，在第二次传球八次，第三次传球七次，等等。

b）数组中的数据可能已经在正确的顺序或接近正确的顺序，所以为什么要让九passesif少就够了吗？如果有任何交换，请在每次传递结束时修改排序。如果没有，那么数据必须是按正确顺序排列的，所以程序应该终止。如果已经进行了掉期交易，那么至少需要再通过一次交易。)

Exp2

Program Challenge

A prime integer is any integer greater than 1 that is evenly divisible only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It operates as follows:

1. Create an array with all elements initialized to 1 (true). Array elements with prime subscripts will remain. All other array elements will eventually be set to zero. You will ignore elements 0 and 1 in this exercise.

2. Starting with array subscript 2, every time an array element is found whose value is 1, iterate through the remainder of the array and set to zero every element whose subscript is a multiple of the subscript for the element with value 1. For array subscript 2, all elements beyond 2 in the array that are multiples of 2 will be set to zero (subscripts 4, 6, 8, 10, etc.); for array subscript 3, all elements beyond 3 in the array that are multiples of 3 will be set to zero (subscripts 6, 9, 12, 15, etc.); and so on.

When this process is complete, the array elements that are still set to one indicate that the subscript is a prime number. These subscripts can then be printed. Write a program that uses an array of1000 elements to determine and print the prime numbers between 2 and 999. Ignore elements 0 and 1 of the array.

(程序的挑战

一个素数是大于1的整数，它只能被它自己和1整除。筛法是一种寻找素数的方法。它的操作如下：

1。创建一个数组，所有元素初始化为1（true）。素数下标数组元素将保留。所有其他数组元素最终将被设置为零。您将忽略此练习中的元素0和1。

2。从数组下标2开始，每次发现一个值为1的数组元素时，遍历数组的其余部分，并将每个元素设置为零，每个元素的下标是值为1的元素的下标的倍数。数组下标2，所有的元素都超过2，是2的倍数，将设置为零的数组（下标4, 6, 8，10，等）；数组下标3，所有的元素都超过3，是3的倍数，将设置为零的数组（下标6, 9, 12，15，等）；等在。

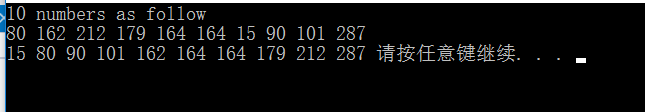
当这个过程完成时，仍然设置为一个数组元素表示下标是一个素数。这些下标就可以印刷。写一个程序，使用一个数组元素确定1000打印质数2和999之间。忽略数组的元素0和1。)

### 实验代码及结果：

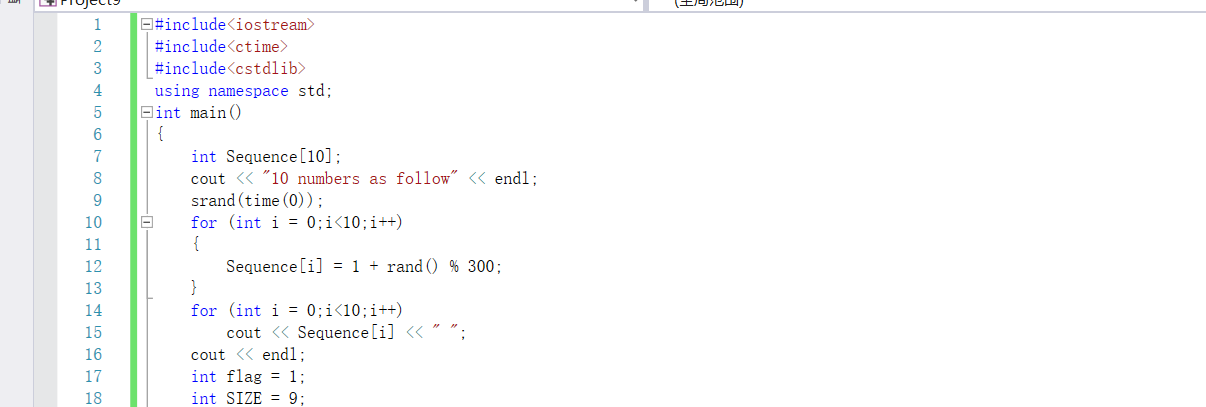
Exp 1

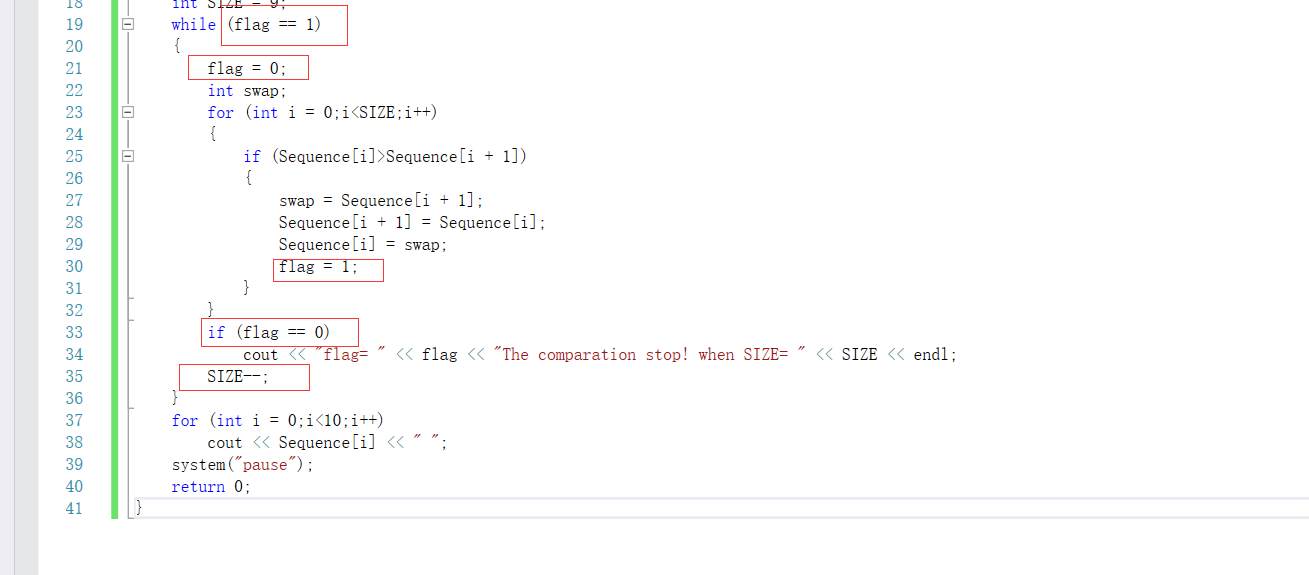
源代码

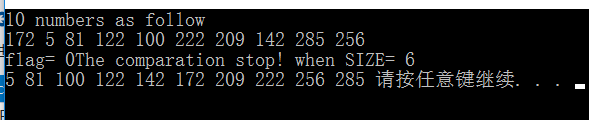




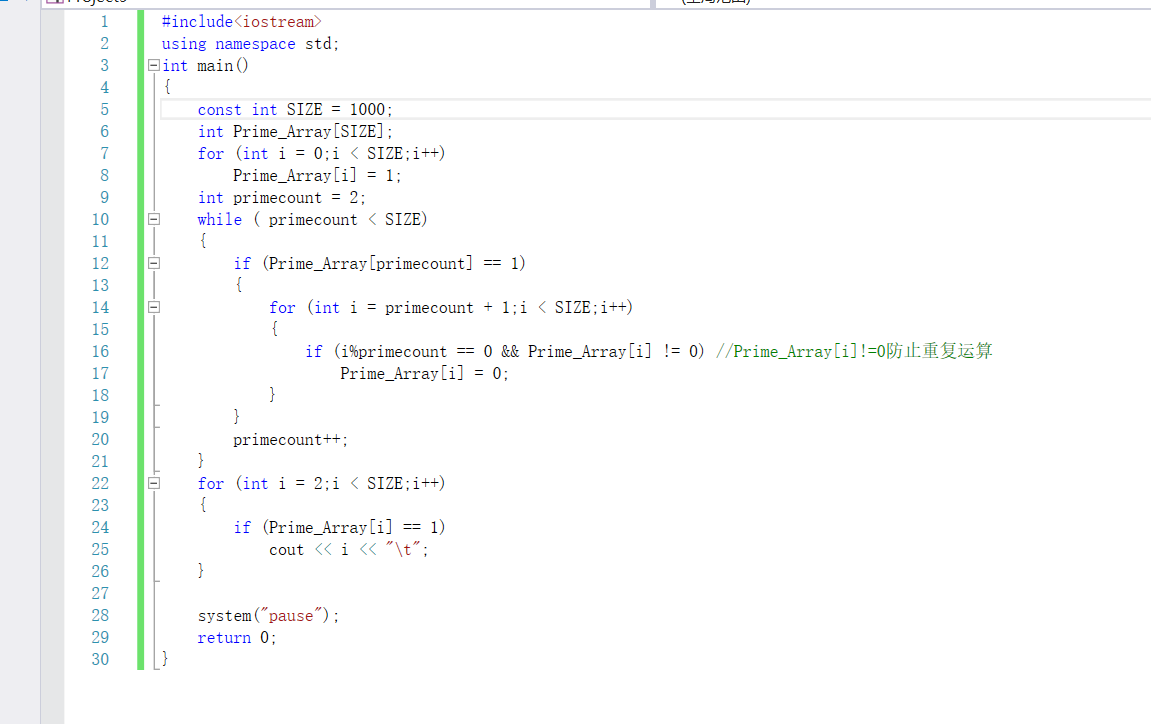
Exp1 Follow-up

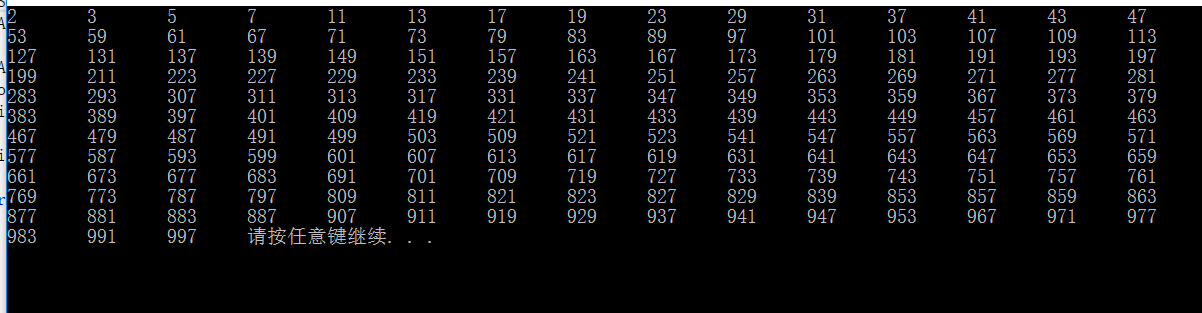


(红色标记的是对程序的改进)

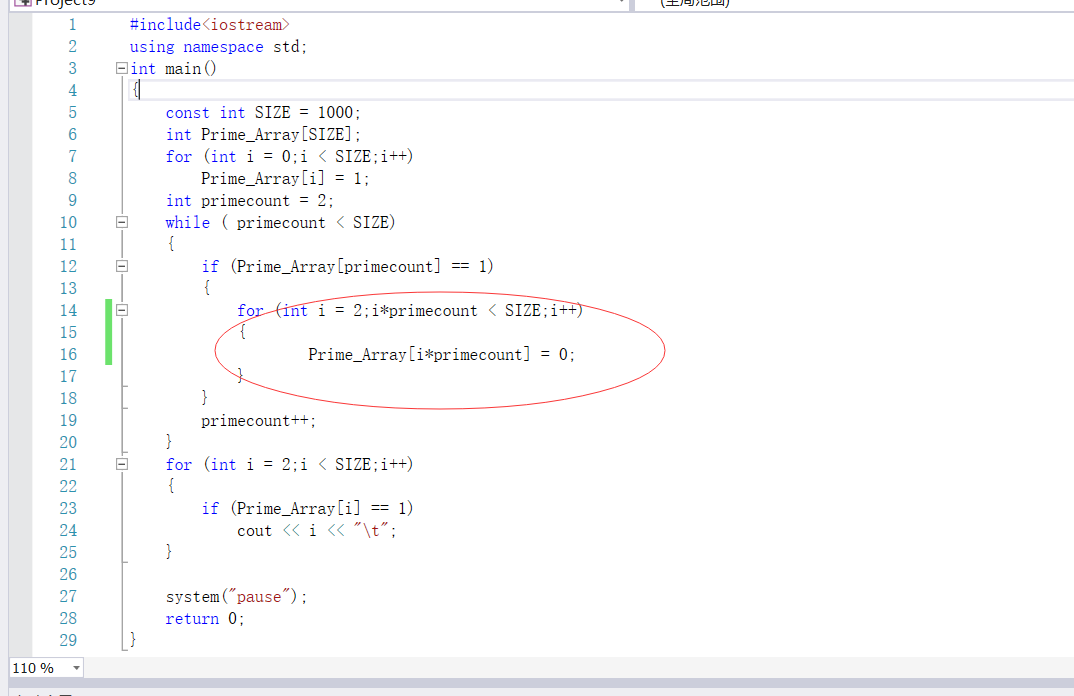


Exp2



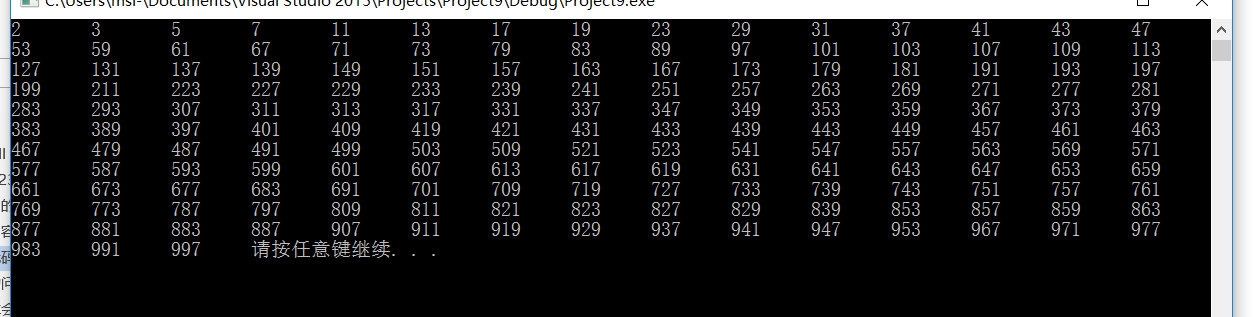


基于程序优化的思想，我又改变了我的程序为



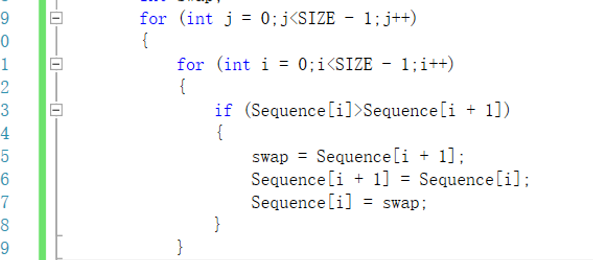
（不用遍历和判断）

得到了相同的结果



### 遇到的问题及解决过程

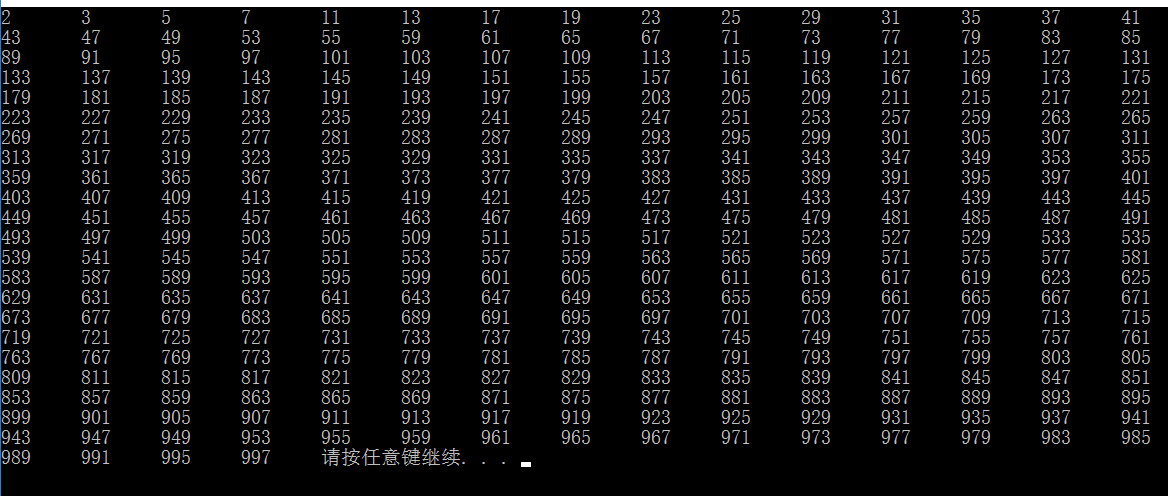
Exp:一开始得到的结果是错误的因为

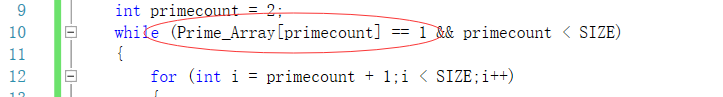


应该使用两层循环，但我只用了一层循环所以得到的结果只是冒泡一次的结果

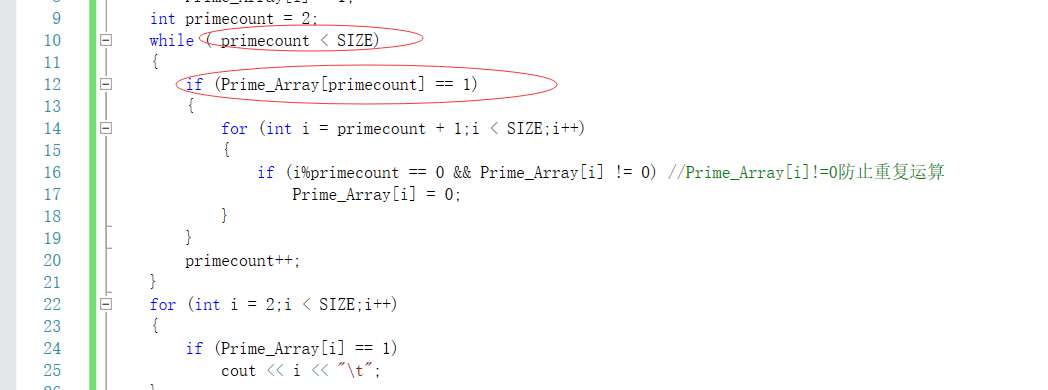
Follow-up nothing

Exp2

一开始运行的时候，结果是错误的，如下

经过调试，我发现了问题所在

就是这个语句造成的，前面的几个数没有问题，但是一旦Prime\_Array[]=0循环就跳出了，也就是说后面的质数就没有用了。我把循环和判断搞混了！！

因此我改变程序为使得问题解决了。

### 实验体会

Exp1 这里我使用了随机数来为我生成测试数据。而且掌握了冒泡排序的方法

Exp1 follow-up这里我使用了flag(小旗子&开关)来控制第一层循环，当没有发生交换的时候

小旗子就倒下，循环就结束了。同时每一次冒泡都将SIZE-1（因为已经冒完泡的不需要在比较了）这里我知道了，在程序中，我们往往可以设立一个flag来作为一个开关，对程序进行控制，同时我们在设计算法的时候应该尽可能地减少无用地运算，来优化程序，最大化地利用计算机性能。

Exp2 这个实验中掌握了一种新的求解质数的方法概括起来就是，从最小的质因子出发，把质因子的所有的倍数排除掉，然后剩下的就是质数了。

其次我明白一定要弄清楚判断和while()循环里的判断，不要弄混了，注意：一旦while()循环语句的条件没有被满足，循环就会跳出了。不要把它当成是continue语句的效果。同时也积累了调试的经验，1缩小数据规模进行调试。2在编写程序的过程中在关键的地方设置断点，来跟踪关键的值。