## 实验四 Control Statements II

## 学号: 09017423 姓名：杨彬

### 实验目的

**Objective**

**The essentials of counter-controlled repetition.**

**To use the for and do…while repetition statements to execute statements in a program repeatedly.**

**To understand multiple selection using the switch selection statement.**

**To use the break and continue program control statements to alter the flow of control.**

**To use the logical operators to form complex conditional expressions in control statements.**

**To avoid the consequences of confusing the equality and assignment operators.**

（目的

反控制重复的要领。

在重复语句中使用for和do来重复执行程序中的语句。

使用开关选择语句理解多个选择。

使用中断和继续程序控制语句来更改控制流。

使用逻辑运算符在控制语句中形成复杂的条件表达式。

为了避免混淆相等和赋值操作符的后果。）

### 实验内容

**Exp1**

Integer Average (课后习题5.6)

**Lab Objectives**

• Using sentinel-controlled repetition with a for loop. The follow-up question and activity also will give you practice:

• Using counter-controlled repetition with a for loop.

**Description of the Problem**

Write a program that uses a for statement to calculate and print the average of several integers. Assume the last value read is the sentinel 9999. A typical input sequence might be 10 8 11 7 9 9999 indicating that the program should calculate the average of all the values preceding 9999.

(例1：整数的平均值（课后习题5.6）

实验室目标-使用哨兵控制重复与for循环。后续的问题和活动也会给你练习：使用反控制重复循环。对问题的描述编写一个程序，它使用for语句计算和打印多个整数的平均值。假定最后读取的值是哨兵9999。一个典型的输入序列可能是10、8、11、9、9999、7，表示程序应该计算前9999的所有值的平均值。)

**Follow-up Question**

Modify the program to perform counter-controlled repetition with a for loop. Assume that the first integer entered by the user represents the number of subsequent integers that the user will input to be averaged. Sample output:

（接下来的问题

修改程序以循环执行计数器控制重复。假设用户输入的第一个整数表示用户将输入的平均整数。）

Exp2

Lab Objectives

• Using counter-controlled repetition.

• Using ―brute force‖ to solve a problem.

• Nesting for loops. The follow-up questions and activities will also give you practice:

• Using break statements.

• Using continue statements.

• Using long integers.

Description of the Problem

A right triangle can have sides that are all integers. A set of three integer values for the sides of a right triangle is called a Pythagorean triple. These three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Find all Pythagorean triples for side1, side2 and hypotenuse all no larger than 500. Use a triple-nested for loop that tries all possibilities. This is an example of ―brute force computing.‖ You will learn in more advanced computer-science courses that there are many interesting problems for which there is no known algorithmic approach other than using sheer brute force.

(问题的描述：直角三角形的边都是整数。直角三角形的边的一组三个整数值叫做勾股数。这三个方面必须满足的两个边的平方和等于斜边的平方的关系。找到适合于side1所有勾股数，两和斜边都不大于500。使用三个嵌套循环来尝试所有可能性。这是―蛮力计算的一个例子。‖您将学习更高级的计算机科学课程，有很多有趣的问题，有没有已知的算法比用纯粹的蛮力。)

**Follow-up Question**

1. How many times did this program execute the innermost for loop? Add another counter to the program that counts the number of times this loop iterates. Declare a new variable of type long, named loopCounter and initialize it to 0. Then add a statement in the innermost for statement that increments loopCounter by 1. Before exiting the program, print the value of loopCounter. Do the numbers match?

2. Add a break statement to the program inside the innermost for loop. This break statement should be called after the 20th Pythagorean triple is found. Explain what happens to the program after the 20th triple is found. Are all three for loops exited, or just the innermost one? What happens when the break statement is placed inside the middle loop? The outermost loop?

3. Add a continue statement to the program that prevents a Pythagorean triple from being found when side1 is equal to 8. Using your solution to Follow-Up Question 1, calculate how many times this new program executes the innermost for loop. Explain why the continue statement affected the output.

4. Explain why a long variable is used for hypotenuseSquared and sideSquared. Modify the program so that they are both of type short instead of type long. Rerun the program. What happens?

（1。这个程序执行了多少次for循环？添加一个计数器的程序，计数的次数循环。声明式长一个新的变量，命名为loopcounter并初始化为0。然后把最里面的声明，声明loopcounter增量的1。在退出程序，打印loopcounter价值。这些数字匹配吗？

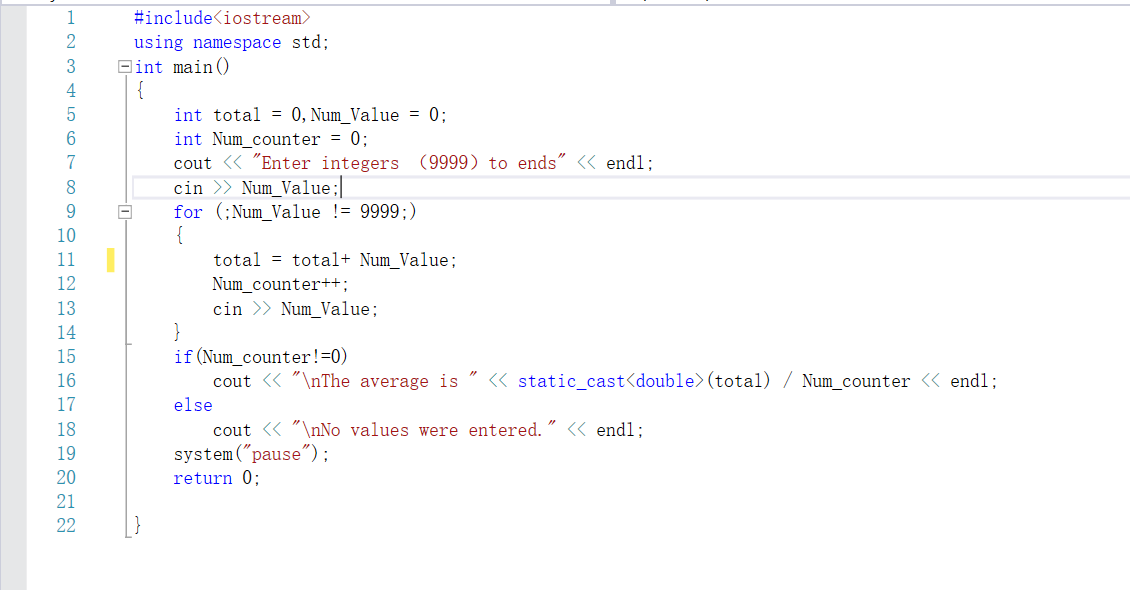
2。在最内层的循环中添加一个中断语句到程序中。这个中断语句应该在第二十勾股三元组被找到之后调用。解释在找到第二十个三元组之后程序会发生什么。循环的三个都是退出的，还是最里面的？当中断语句放置在中间循环时会发生什么情况？最外层的循环？

三.添加一个continue语句的程序，防止三从被发现适合于side1等于8个毕达哥拉斯。使用您的解决方案跟踪问题1，计算这个新程序执行最内层循环的次数。解释为什么连续语句影响输出。

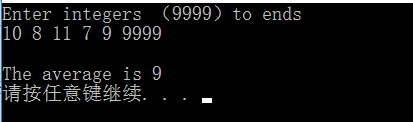
4。解释为什么长变量用于hypotenusesquared和sidesquared。修改程序，使它们都是类型短的，而不是类型long的。重新运行程序。发生了什么事？）

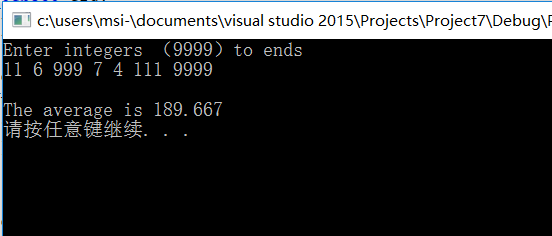
### 实验代码及结果：

Exp1 源码



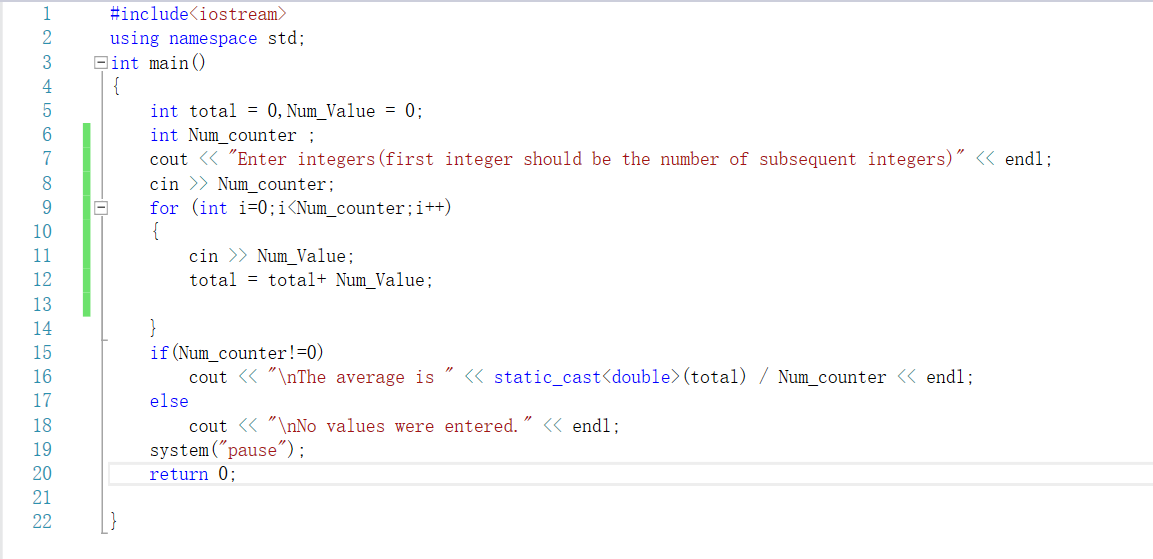
结果

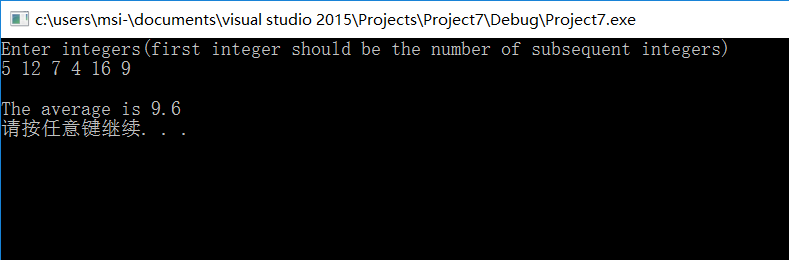


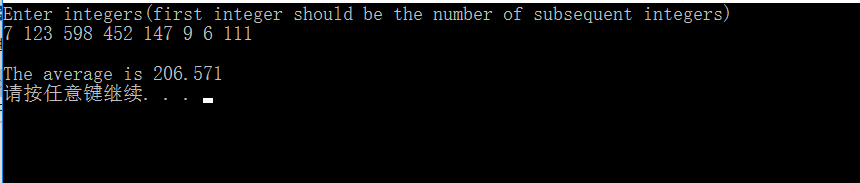


Exp1 follow-up

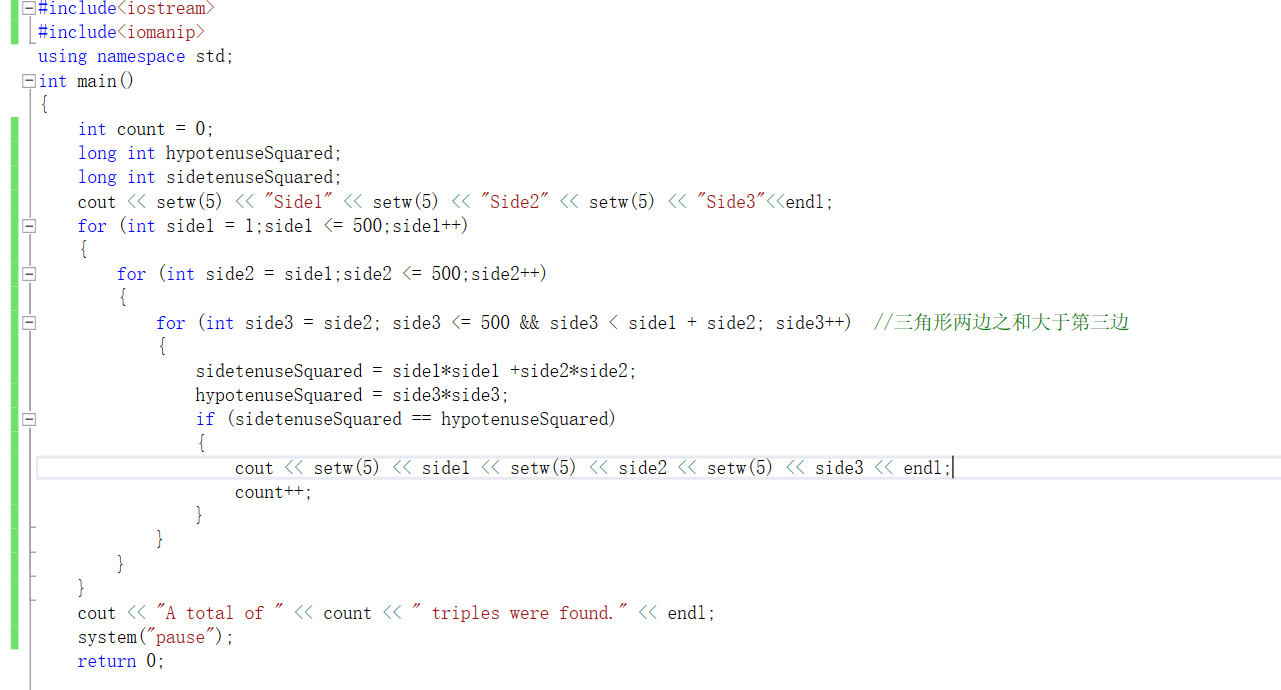
源代码



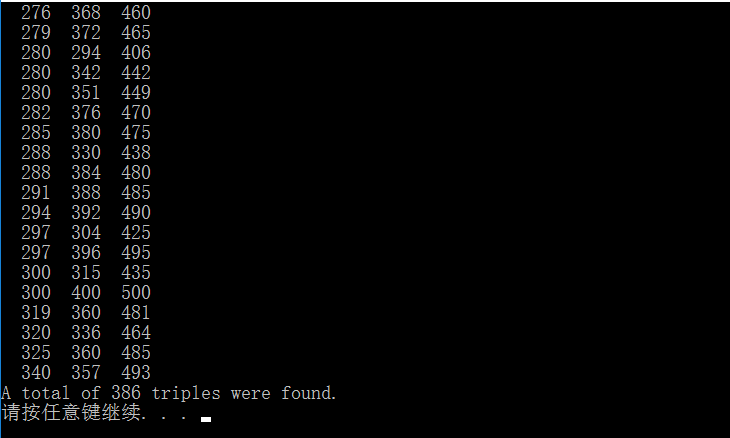
结果



Exp2 源码



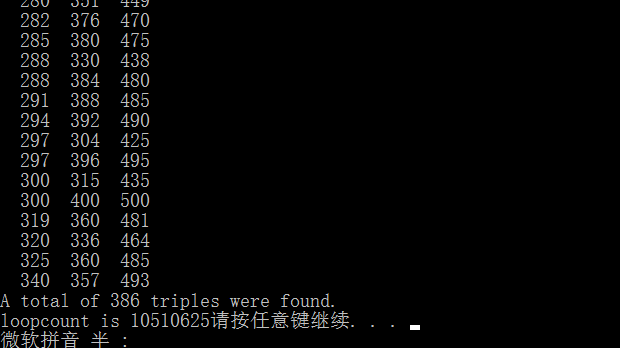
结果



**Follow-up**

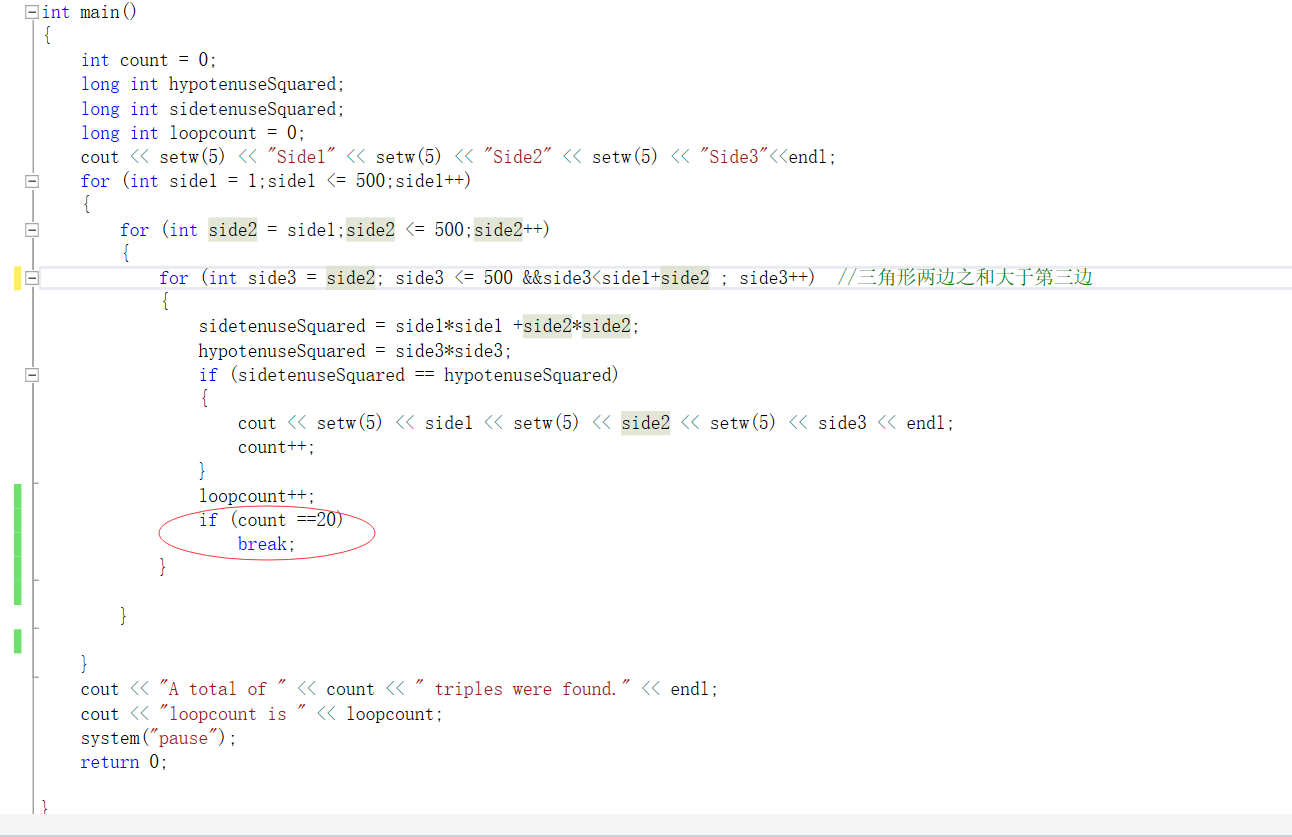
**1**

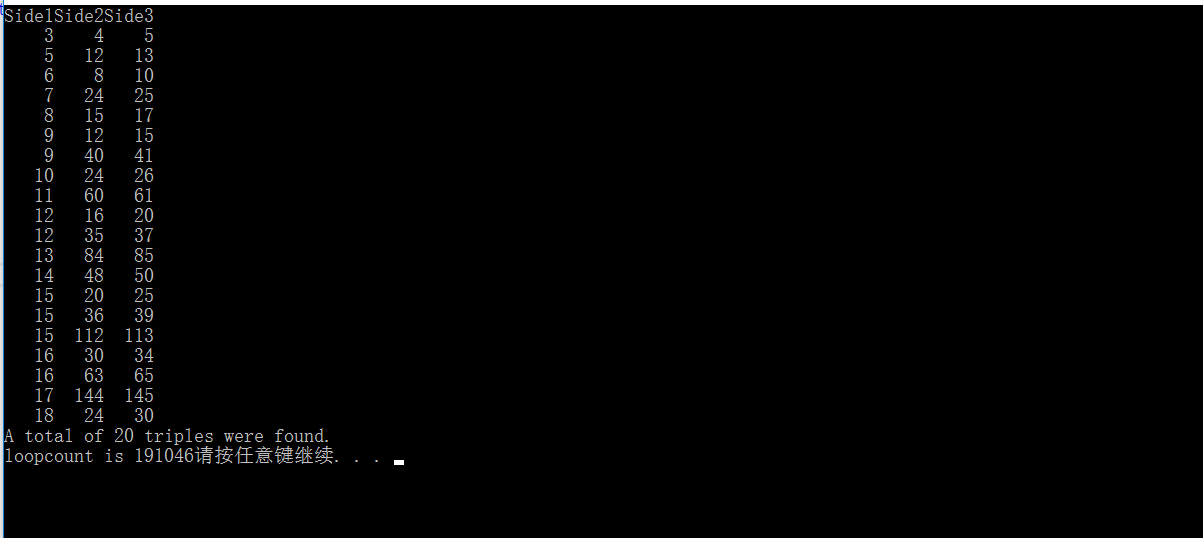
源码

结果

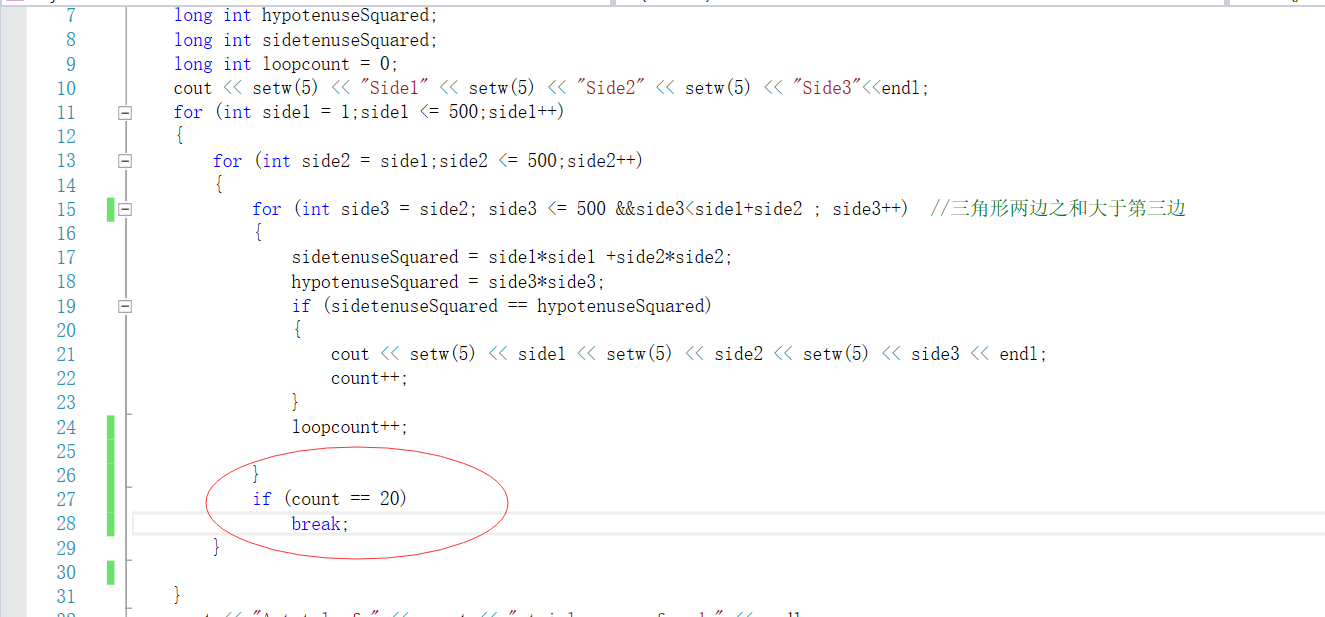
2

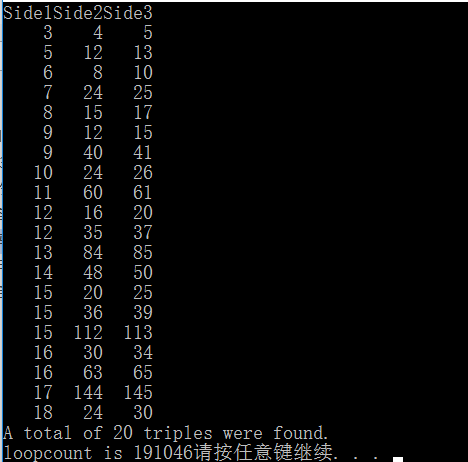
代码最内层 break



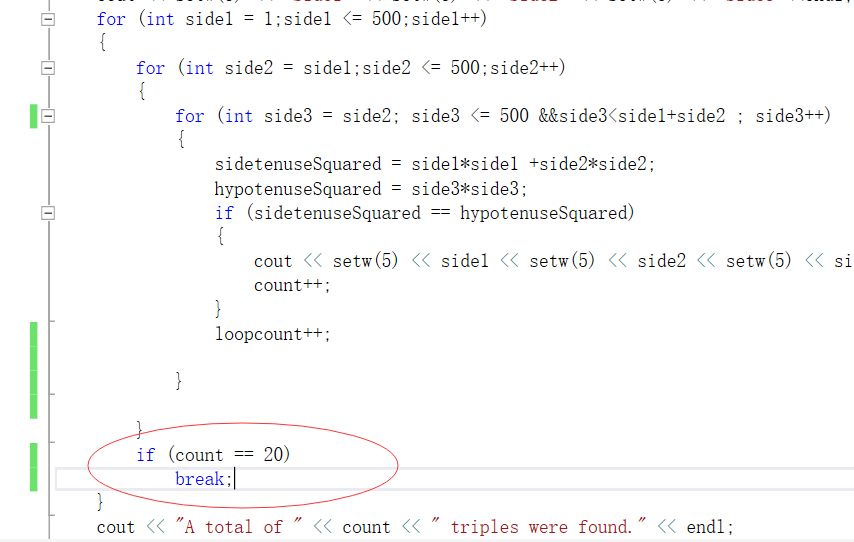
结果

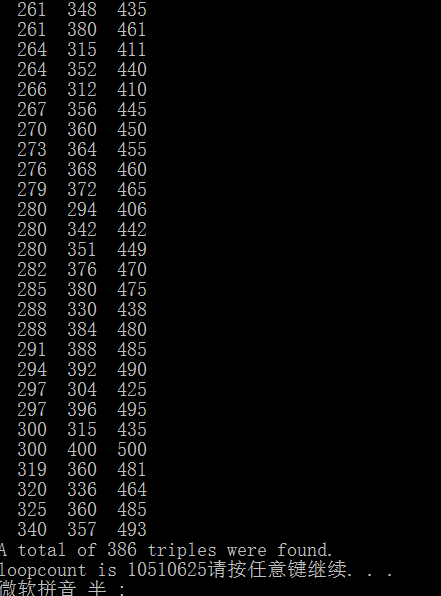
代码 第二层break



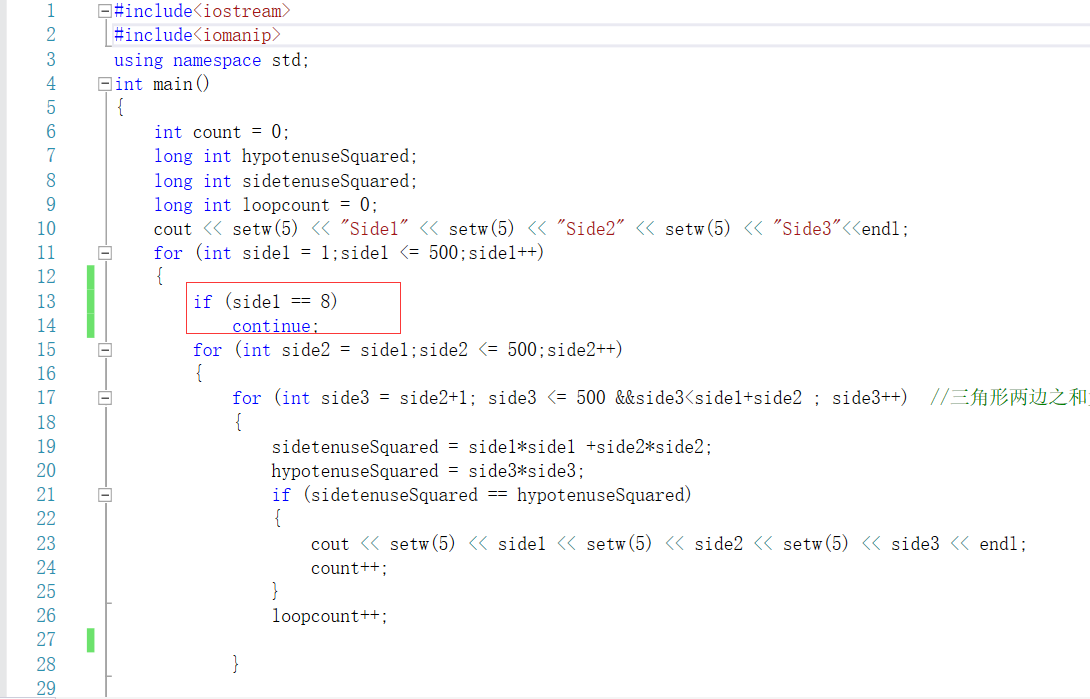
结果

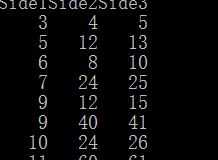
3最外层break

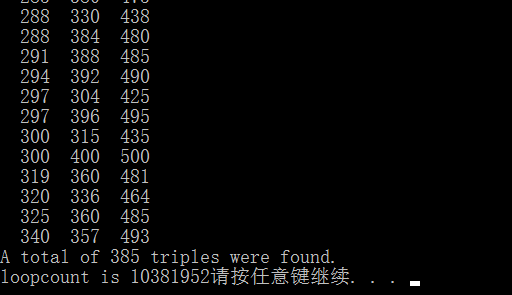


、

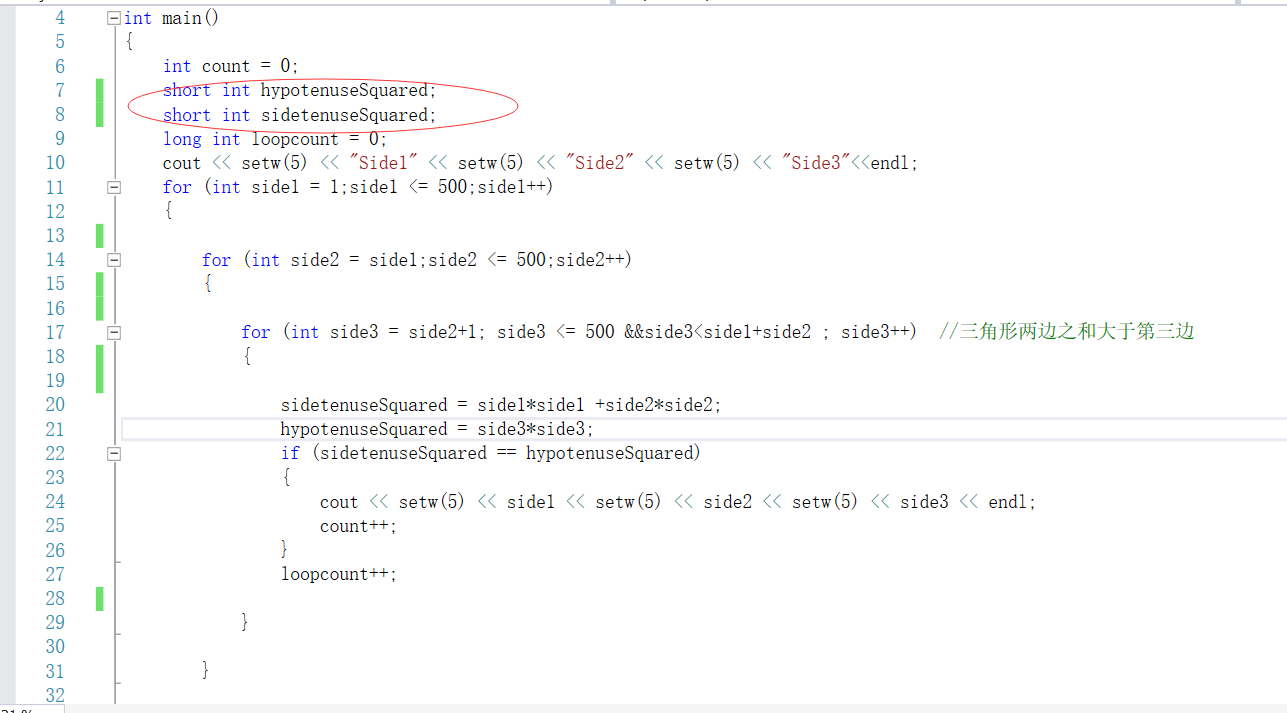
Follow-up3

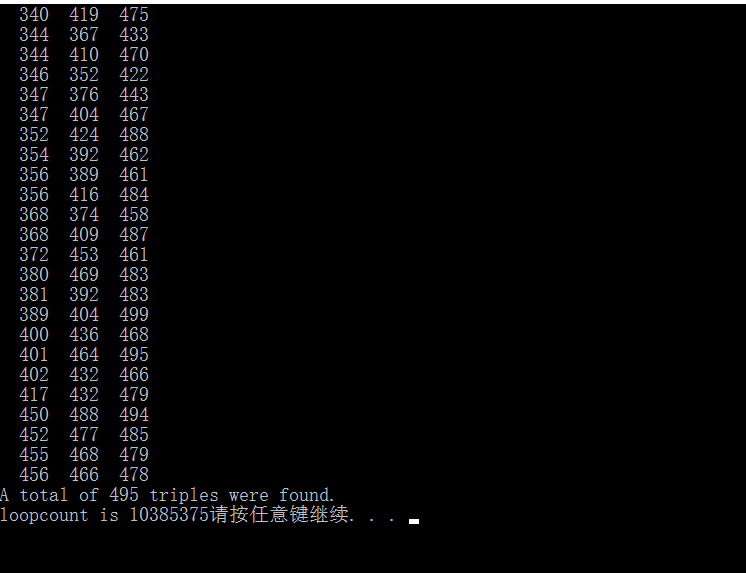






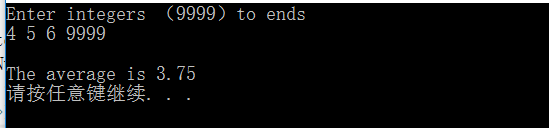
Follow-up4





### 遇到的问题及解决过程

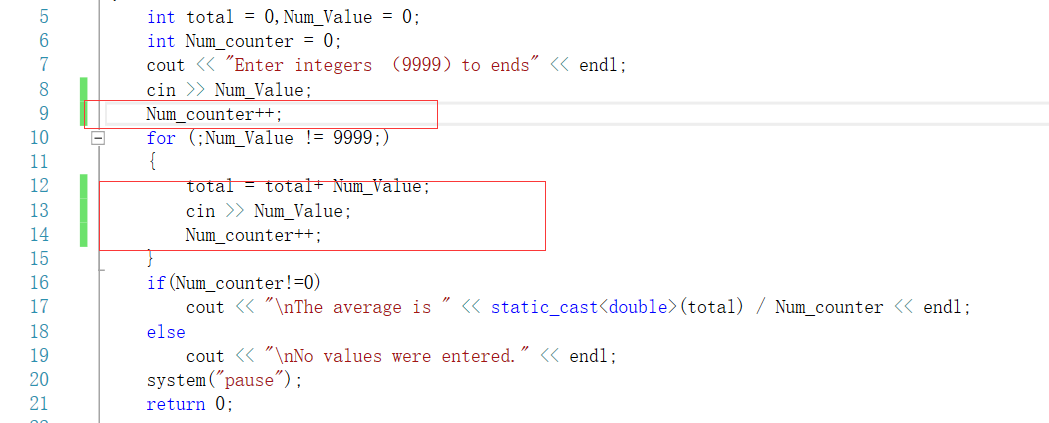
Exp1 1一开始的时候得到的平均值都比预期值要小。



2于是我在total 和 Num\_counter 这两个变量处设置断点

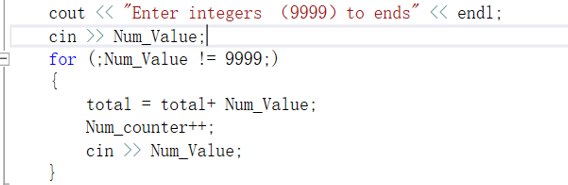
3发现counter 多了1

4 检查代码 发现了问题



问题就在于，我在循环外第一次输入数之后 ，将Num\_counter加了1，我认为这是正常的想法，但是在for循环语句中，我这种写法，意味着我所输入每一个值之后counter都会加一 包括不被计入总数的9999

因此我将代码该成了



这样写相当于 对每一个输入的数要经过判断时候满足条件 之后再加1 这样就解决了问题。

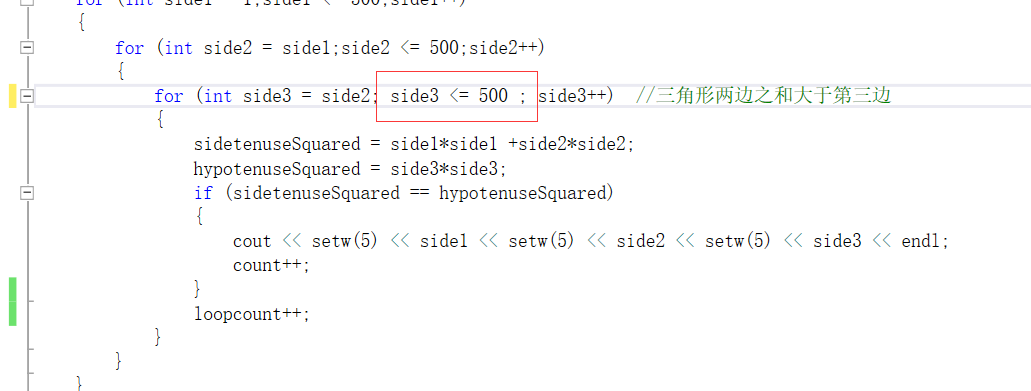
Follow-up :nothing

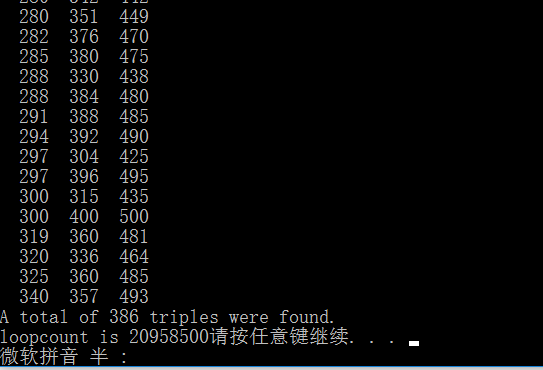
Exp2：nothing

Follow-up

1 这里我发现运算的次数和实际的三角形个数有着极大的反差。

2于是我试着改变算法，去掉我之前加的三角形两边之和大于第三边的这个条件



3运算

可以看到运算次数多了一倍。可见减少无用的计算是多么重要。

Follow up2

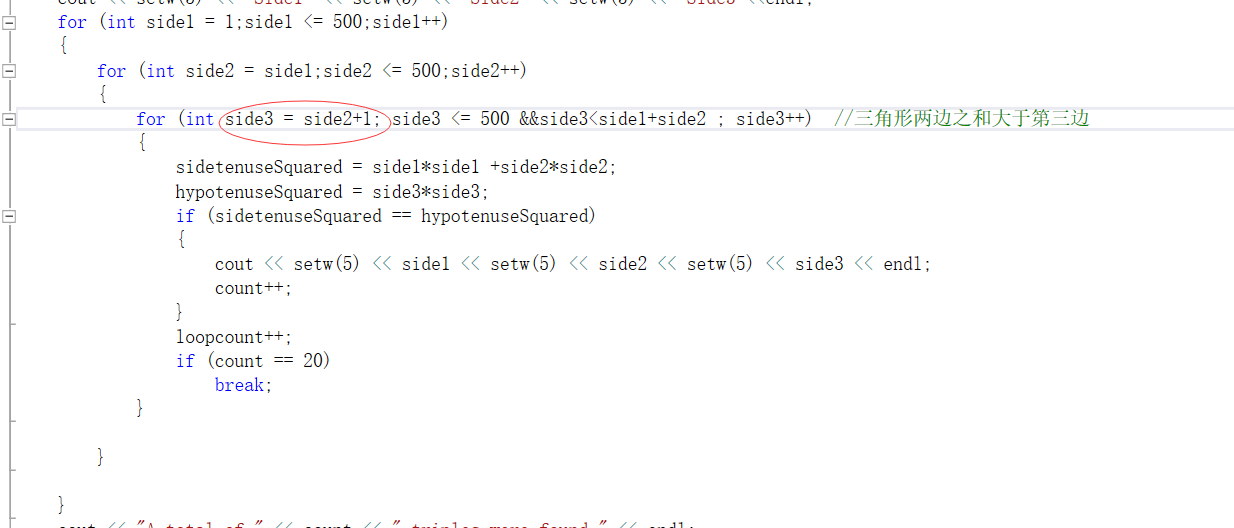
1观察结果有一个疑惑，为什么当break语句放在最内层，和第二层的时候得到的三角形数是一样的，而放在最外层的时候结果又和没有加break语句的结果是相同的？

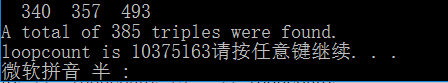
2但是仔细分析发现，第1第2中情况的时候循环的总数是不同的，也就是说两种情况下的break语句工作是不同的。

3因此我使用了调试器，发现break放在那一层，当条件达到的时候，就只是跳出那一层的循环。

4经过思考，得出这样结果的原因和算法本身有关。在本程序的循环语句控制条件上，一但count=20 之后每次第二层循环下的第一次第三层循环，conut都不会被+1，因为三角形的斜边不可能等于直角边。因此之后的第三层循环被跳出了也就得到了这样的结果。Break语句放在第二层的时候，原因是相似的。而放在第一层的时候，因为 在第一次count达到20第一层循环结束后counter就已经大于20，也就是说第一层循环中的break的条件永远不会被满足。（这两个，分别是第20个和第21个三角形）

5为了证明我的想法，我修饰了程序





我们可以看到只有第21个没有被打印出来。

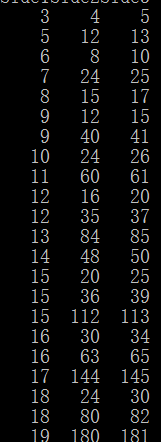
Follow-up 3

对比和follow-up 1的结果，我发现 side=8的这个三角形被忽略了。并且总的循环次数少了很多。

Follow-up4

得出了错误的结论

分析之后的出原因，因为short 类型储存的整数范围较小，所以当数据超出范围时就会的到一个不确定的结果，从而结论是错误的。而当数据没有超出范围的时候结论是正确的

比如

前几个三角形仍然是正确的因为他们并没有超出范围，错误是当三角形的三边够大的时候才发生的。

### 实验体会

Exp 1

在循环语句中 counter这个变量往往是十分重要的，也是十分容易出错的，特别是在循环边界的地方，应该十分小心。所以将含counter 的语句放在什么样的位置，取决于具体问题。或者是在循环结束之后，根据具体的问题对counter 进行修饰，已达到目的。

Follow-up：这里通过事先输入一个数，然后用它来控制次数，这种方法又很广泛的应用。不仅可以用在循环语句中——特别是for循环，还可以用在比如控制链表的链结。

Exp2

1这里使用了三个内循环，这种循环的方式，内部的循环都与它最小的外部循环有关，即第一个循环决定第二个循环，第二个循环决定第三个循环，以此类推。循环嵌套，类似于树状图，可以用树状图的方法来理解。

2在使用穷举法的时候，可能会导致巨大的运算量。但是根据已知条件，或者数学上的条件，用其来控制循环语句，就可以达到极大地减少运算量。比如在上述循环中，我加了三角形两边之和大于第三边地条件（注释部分），以及实验模板中的 第二层循环从side1开始 而第三层循环从side2开始。这样就可以减少很多无用的运算。可以优化算法。

Follow-up

1在穷举法当中，总计算次数和实际的计算次数相差非常大，这更加说明了通过增加条件，优化算法，来减少无用计算的重要性。对于小型的数据来说，或许改变不了什么，但是一旦是庞大的数据，将会是天差地别。

2我们通过这个可以明白，当程序运行break的时候，程序跳出break语句所在的循环，比如第二层的break跳出第二层的循环。而不会跳出所有的循环（也就是说不会跳出改循环的外部循环）其次在应用break语句的时候，尤其是多层循环中运用break语句的时候，一定要考虑break语句的条件能不能被满足。就如本实验，在内层循环的影响之下，外层循环的break可能永远也不会被执行。

3.通过实验我明白了continue语句的作用，当程序遇到continue语句的时候程序直接跳出该次循环（不是跳出循环语句）然后执行下一次循环。在本程序中就是，程序跳出side1=8的循环但是side1>=9之后的循环仍然是正常的。

4.我们在声明变量的时候一定要为变量选择何时的类型。小的变量选择小的类型，大的变量选择大的类型。这样才能最大程度地优化程序。因为小的变量选择大的类型会导致空间的浪费，而大的变量选择小的类型会得到错误的结果。