

如何进行调试

怎么debug?

测试点信息

#1 AC 83ms/202.54MB	#2 AC 81ms/202.46MB	#3 WA 163ms/199.55MB	#4 WA 188ms/203.71MB	#5 RE	#6 TLE 1.20s/207.76MB	#7 TLE 1.20s/217.64MB
#8 RE	#9 RE	#10 TLE 1.20s/212.87MB	#11 TLE 1.20s/213.28MB	#12 TLE 1.20s/213.41MB	#13 TLE 1.20s/211.50MB	#14 TLE 1.20s/210.65MB
#15 TLE 1.20s/210.54MB	#16 TLE 1.20s/209.85MB	#17 TLE 1.20s/218.29MB	#18 WA 732ms/206.05MB	#19 TLE 1.20s/213.29MB	#20 AC 562ms/205.91MB	#21 WA 189ms/199.49MB

- 相信大家在做题的过程中经常碰到各种报错信息，这些信息说明大家的算法出现了某些问题，不能正确的通过评测。
- 那么我们在代码出现问题之后，应该如何去纠正错误呢？
- 我想在座大部分人的选择是

看就完了！

然而直接将代码硬生生的看个几十遍的或者随机瞎改再运行的效率并不高，而且并不能真正地理解自己错在哪里



所以我们要思路有技巧的debug

首先来看一下各类错误提示的含义

- **编译错误 (Compile Error, CE)**

很显然，如果代码没有办法通过编译，那么就会返回Compile Error

这种错误应该在提交到OJ之前在编译器内就解决，因为无法通过编译的代码在编译器上是无法正常运行的。

注意把主类名改成Main，不然会报CE

- **答案错误 (Wrong Answer, WA)**

很多初学算法的童鞋经常会问：“啊我这跟样例答案一模一样
凭什么错啊”

拜托OJ如果只测它的样例的话那跟写题直接告诉你答案有什么区别（你们说对吧）

OJ的题目通常会提供一些较为简单的数据，目的是让我们**初步**验证我们程序的正确性。OJ的后台还有大量复杂的数据和标准答案来判断我们写的程序到底是否正确。因此，可以说，样例没过肯定不能AC，能过样例不一定能AC。

- **运行错误 (Runtime Error, RE)**

导致这一错误的可能性非常多，最常见的有

数组越界

/0或%0错误

目前遇到RE先往以上两个方向思考

- **内存超限 (Memory Limit Exceed, MLE)**

内存超限是因为数组开太大的缘故，一般来说128MB的内存限制，开int型变量的数组大小最多大概是 $3e7$ ，了解一下即可，该课程基本不会出现。

- **运行超时 (Time Limit Exceeded, TLE)**

TLE是一种常见的错误类型，指程序运行的时间超过了题目的限制。一般的题目要求时限为1s，而计算机1s能运行的语句数大约为1亿次。因此只要程序运行次数大于1亿这个量级就很可能出现TLE。

出现TLE的一种情况是写了死循环，这种情况会显示TLE

当然大部分情况是你的算法在数据较小的时候正确，数据大时会超时。想要在大数据下又快又对的得到结果没有那么容易，你的算法往往需要一些优化甚至重新设计，这就体现了我们在编写程序前设计算法分析复杂度的重要性，关于算法的时间以及空间复杂度，我们这门课程几乎不考察，如果有想进计系的同学在后续课程中会学到。

调试步骤

- 1、重审题目

包括但不限于

- (1) 题目已知什么，要求什么
- (2) 参数的范围和类型
- (3) 输入输出的格式

- 2、静态查错

就是想着自己的思路把自己写的代码完整的看一遍，看看自己的代码能不能正确地表达自己的思路

- 3、构造错误数据

我们想要找到哪里出错，得先获得让我们出错的数据。对于构造数据，可以构造随机数据或针对算法可能出现的错误情况构造极端条件。构造的数据要尽量简单，可以进行手动计算正确答案，然后再与程序输出进行对比。若程序输出跟理论正确答案不同，恭喜你已经看到了成功的希望。

- 4、利用调试方法进行Debug

获取错误数据后我们就可以采用调试方法来对我们程序的运行过程进行观察，根据错误的情况进行猜测和推理，从而找到出错的位置和问题所在。一般分为两种调试方法

- 在程序中添加调试输出语句
- 利用IDE的调试功能

添加调试输出语句的Debug方式

如果我们程序出现了错误，肯定是从某个地方开始出问题。我们可以在做某些操作时输出过程中一些变量的值，并增加一些内容去标识这些输出是作为调试使用的，在程序运行结束后的输出中我们即可观察到每一个步骤中你想要看到的变量的值。一般调试结束后用单行注释掉即可，也方便下次再次调试。

```

public static void main(String[] args)
{
    Scanner in=new Scanner(System.in);
    int L=in.nextInt(),R=in.nextInt();
    int sum=0;
    for(int i=L;i<=R;i++)
    {
        sum+=i;
        System.out.printf("loop i=%d:sum=%d\n",i,sum);
    }
    System.out.println(sum);
}

```

```

2 5
loop i=2:sum=2
loop i=3:sum=5
loop i=4:sum=9
loop i=5:sum=14
14

```

如何利用IDE的调试功能Debug

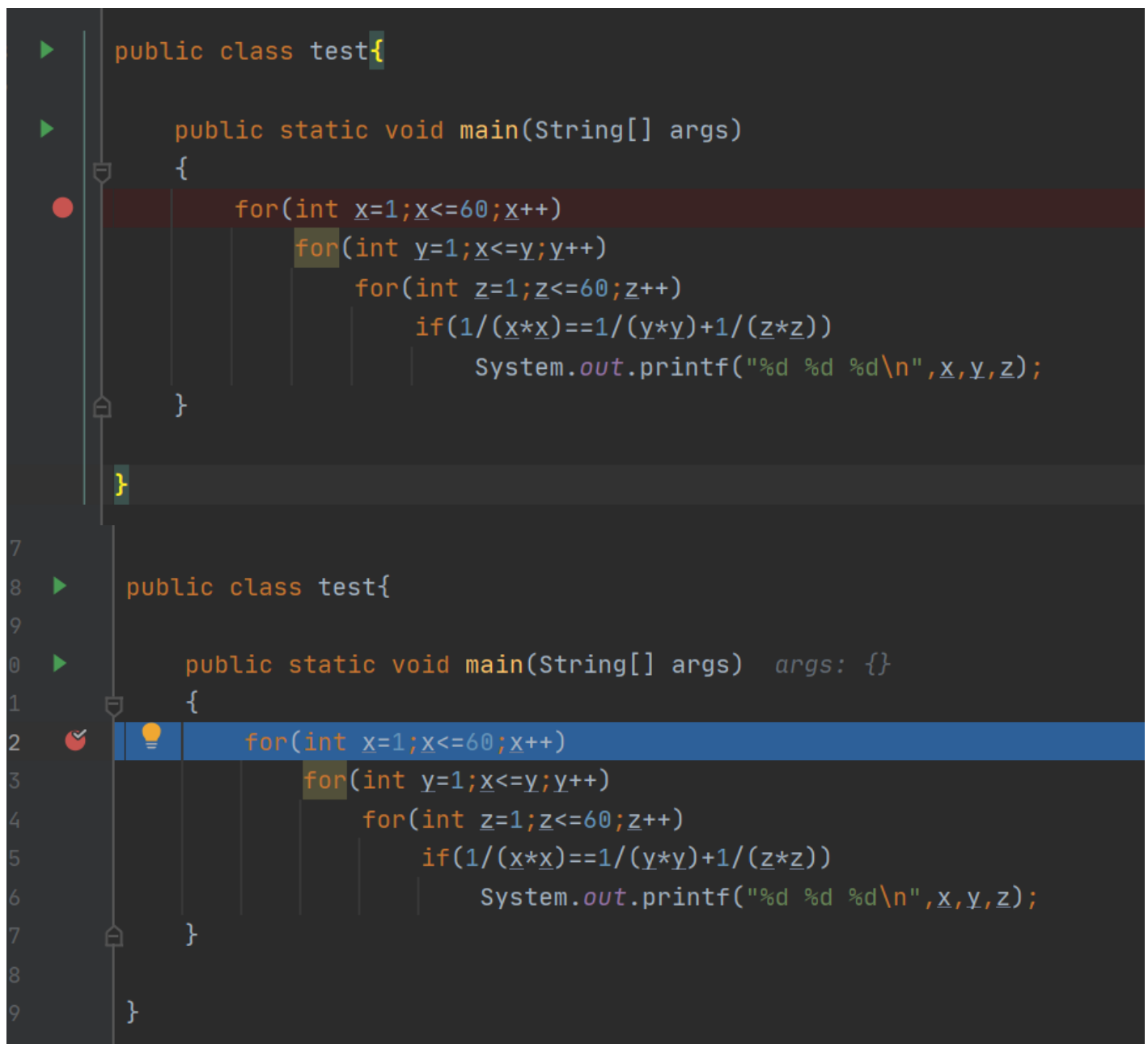
大部分IDE都有自带的调试功能，IntelliJ的调试按钮即为右上角运行箭头旁边的昆虫符号。



在进行调试之前，我们首先得设置断点。

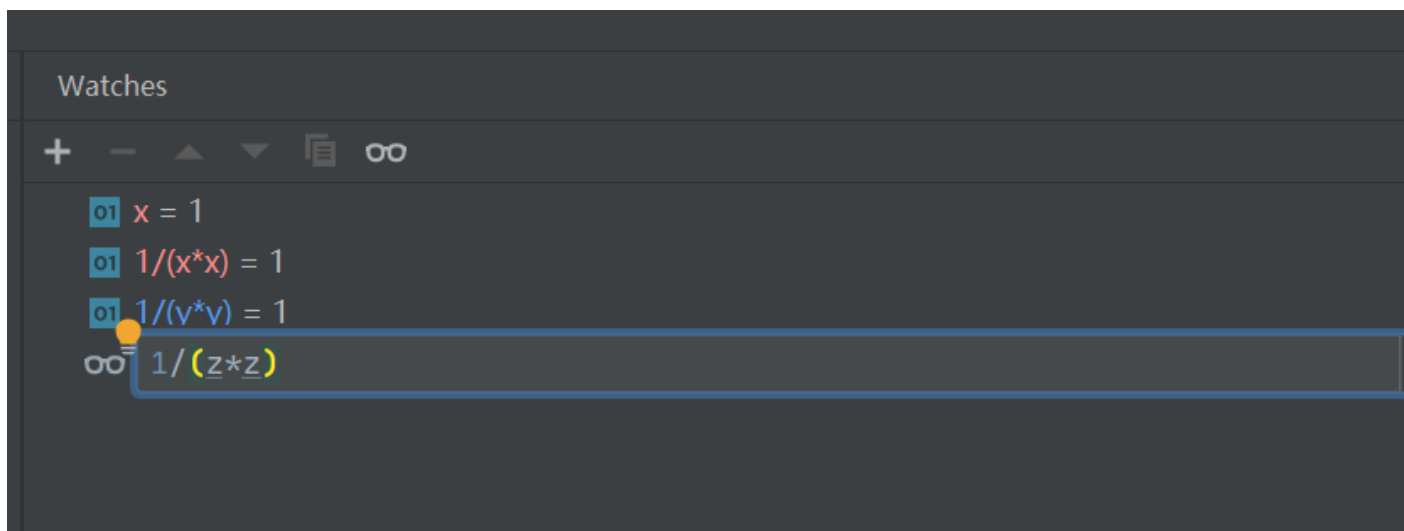
在调试模式下，程序按照其运行顺序正常运行，遇到断点时便会停下，便于我们查看程序运行到这个地方时候的一些情况是否符合我

们的预期，从而检查出到底是哪个地方出了问题。

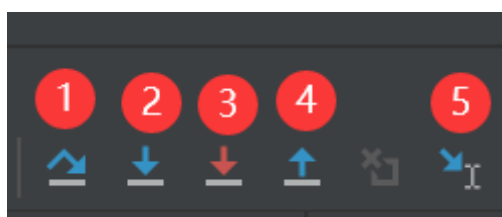


```
public class test{  
    public static void main(String[] args)  
    {  
        for(int x=1;x<=60;x++)  
            for(int y=1;x<=y;y++)  
                for(int z=1;z<=60;z++)  
                    if(1/(x*x)==1/(y*y)+1/(z*z))  
                        System.out.printf("%d %d %d\n",x,y,z);  
    }  
}  
  
7  
8 public class test{  
9  
0     public static void main(String[] args)  args: {}  
1     {  
2         for(int x=1;x<=60;x++)  
3             for(int y=1;x<=y;y++)  
4                 for(int z=1;z<=60;z++)  
5                     if(1/(x*x)==1/(y*y)+1/(z*z))  
6                         System.out.printf("%d %d %d\n",x,y,z);  
7                 }  
8  
9     }  
0
```

在代码左侧处任意行点击即可设置断点，再按下调试按钮即可进行调试。它会在其运行顺序遇到第一个断点的位置停下。高亮蓝色的位置即为程序当前将要执行的下一条语句。调试过程中可以任意增加或减少断点。

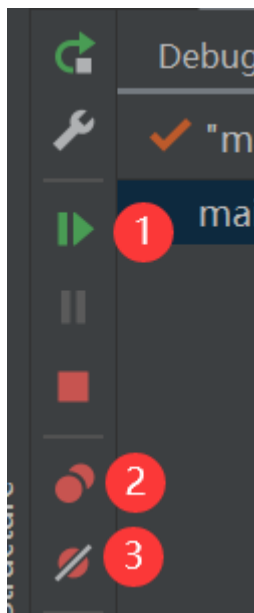


在调试过程中会自动显示循环变量的值，非常便于观察。当然也可以自行在下面的watches框内添加自己想查看的变量的值



- ① Step over(F8)，步过该语句，即直接执行完该语句到下一个语句。若该语句调用了其他的方法等也直接执行完。只会在当前作用域中移动
- ② Step into(F7)，步入该语句，若为最基本的语句则直接到下一个语句。若该语句调用了自己写的方法，则进入到编写该方法的程序中执行。可能会在各个作用域中反复横跳，但一定是符合程序运行顺序。
- ③ Force step into(Alt+Shift+F7)，与step into类似，但能进入不是自己写的方法，用处不大。
- ④ Step Out(Shift+F8)跳出子方法，将子方法执行完毕，跳到调用该方法后的下一个语句。

- ⑤Run to Cursor(Alt+F9)运行到光标所在处。



- 在调试界面左侧也有一些控制按钮
- ①Resume Program(F9),让程序继续运行，到下一个断点停止
- ②View breakpoints,查看在所有project中添加的所有断点位置
- ③Mute breakpoints,屏蔽所有断点，使断点失效。
- 断点还有一个非常好用的功能，可以右击断点给断点添加悬停条件，也就是在满足某些条件的时候再停下程序。合理地设置悬停条件对我们的Debug会带来极大的便利，例如直接以出错的条件作为悬停条件可以快速找到错误位置。但条件断点在找悬停位置的速度比正常跑程序会慢很多很多，所以如果跑的过慢还是采用第一种方式定位错误位置更合适。

两种方式的优缺点分析

- 在程序中添加调试输出语句

- 优点

- 操作相对简单
- 可以通过观察输出快速找到出错的位置

- 缺点

- 较难深入观察出错的原因，需要另加输出语句
- 对于运行错误的情况可能无法观察

- 利用IDE的调试功能

- 优点

- 可以动态查看代码运行过程，既可以查找错误，也可以帮助理解代码的运行逻辑和顺序
 - 可以进行步入、步出、修改等便于查找错误的操作

- 缺点

- 定位错误位置需要一定时间
 - 观察运行过程时需要注意力集中
 - 需要一定的经验和熟练度

一般来说通常利用前者快速定位错误位置，再利用后者深入观察错误原因。前者使用的不多，因此大家需要多练习IDE的调试功能，这会为大家查找错误带来很大帮助。

下面我们用一个例子来示范一下怎么利用调试找出bug

题面：满足 $\frac{1}{x^2} + \frac{1}{y^2} = \frac{1}{z^2}$ 的一组 $x, y, z (x \leq y)$ 称为倒立勾股数，请设计

程序打印出60以内的倒立勾股数

```
public static void main(String[] args)
{
    for(int x=1;x<=60;x++)
        for(int y=1;y<=x;y++)
            for(int z=1;z<=60;z++)
                if(1/(x*x)==1/(y*y)+1/(z*z))
                    System.out.print("%d %d %d\n",x,y,z);
}
```

请复制以上代码并利用调试功能修改正确。

正确答案为

```
15 20 12
30 40 24
45 60 36
```

练习题

题面：求出一个数组中最长连续严格单调递增区间的长度

($1 \leq n \leq 100$)

错误代码

```
public static void main(String[] args)
{
    Scanner in=new Scanner(System.in);
    int n=in.nextInt();
    int[] a=new int[n+10];
    int len=0,maxLen=n;
```

```
for(int i=1;i<=n;i++) a[i]=in.nextInt();
for(int i=2;i<=n;i++)
{
    if(a[i]>=a[i-1]) len++;
    else
    {
        maxLen=Math.max(maxLen,len);
        len=0;
    }
}
System.out.println(maxLen);
}
```

测试样例

```
//input
1
3
//output
1
```

```
//input
5
1 2 2 3 4
//output
3
```

```
//input
10
1 2 7 2 3 1 5 9 10 11
//output
5
```