一、实验 输入三个整数，显示是普通三角，等腰还是等边

测试用例 ：可以组成普通三角 ？ 可以组成等腰 ？ 可以组成等边？ 长度为0 长度为负？ 两边和等于第三边？ 三个边的输入顺序组成的情况（不同输入，相同结果）

两边和小于第三边？ 顺序？ 三边都为0？ 为非整数？ 少输入参数？

预期输出值？ 大整数赋值？长度溢出？

测试代码超大的系统，测试面向对象

一个完美的程序应该多么健壮？

1. 理想是测试程序所有可能执行的情况，覆盖所有的输入和预计的输出 不可能（输入还行，输出不一定会正确） **建表**
2. 测试不是证明软件没错 而是找错 不是证明软件能完成，而是发现不能完成 对软件不应该有信心，要相信有Bug,或者是功能不能完美使用

**需要尽可能地发现最多的问题 是一个破坏的过程**

**设计测试用例（测试数据）**

没发现错误的测试不能说明成功

大部分程序都是有错误的，正常情况运行可能没问题，极端时就有问题了

1. 发现所有的错误是很难的 需要经济地考虑，选择策略

黑盒测试：数据驱动测试/输入输出测试 调用api，穷举所有输入， 把程序输出和应有的输出进行对比

要使用所有可能的输入，包括边界和违法输入 ，**超边界和违法输入的 理论输出应该是报错（异常或者错误代码-1等）**

如果是一个编译器—— 穷举所有语法？验证编译的所有程序正确性？太难了

数据库 查询，和事务处理 。。。要测试**执行顺序**

**并发测试**

完全穷举是不可能的，也没意义，需要的是有限测试。。。发现最多的错误 需要探究软件内部

2、白盒测试 （逻辑驱动的测试） 检查内部 对逻辑结构进行检查

穷举路径测试。。。。所有的控制流swith分支 for循环 都尝试 不实际，每次循环都有不同分支选择

原则：不应测试自己的软件？ 必须定义预期输出结果 每个测试的结果都要验证

输入不仅是应该有的输入，也要包括不应该有的输入（类型错误，长度错误，范围错误）

检查程序有没有**输出不想要的输出 ，任何情况（输入）下都不应该输出不想要的输出**

测试需要保存下来 发现的错误越多，说明可能有更多错误（错误聚集）

调试

3、代码检查 走查和评审

给人阅读的代码 软件规模 复杂度 团队规模 和时限

人工测试（在编码结束后就该开始），最好在编程中，甚至编程前就开始共同设计如何测试

错误发现越早，越好改

头脑风暴，如何找出错误，不考虑改正方法

逻辑设计错误，编码错误 ，可以发现70%左右已知错误 （未知错误永远是未知的）

人工测试和计算机测试是互补的

代码检查：阅读 ，讲解逻辑 回答问题 大声**朗读程序**

获得错误清单

**常见编码错误列表：**

**太讲究编程风格， 太过模糊（需求不明确）的错误列表是没用的**

数据引用错误： 未赋值 未初始化（为需要的值） ，包括变量，数组元素，属性

证明数据是有值的

2、数组访问时 下标过界？ 下标是整数吗？（可能不是整型变量）

3、指针和引用（对象） 指向的地址是正确数据吗？ （销毁的局部，释放的空间）

4、 类型是否正确 注意隐式类型转换

5、字符串 索引 只差一个 6、 继承之后，该重写的是否都重写了？

声明错误 变量使用前是否声明？

变量声明的默认属性是否正确，比如java中属性的默值

初始化是否正确（数组和字符串）

变量长度和类型

运算错误：

类型不同值的运算（包括字符串 ，对象等）

混合运算的转换规则 整数和浮点 误差 位丢失

相同类型不同字长的运算？

赋值右值的数据类型和左边

运算中是否存在溢出？ 中间值有溢出吗？ 上溢下溢

可能存在除0？ （可以除0.0）

浮点的运算精度。。。。

变量值超范围。

**整数除法导致了丢失余数， 5/2==2.5 不正确**

比较错误

不同类型比较 （包括对象，数组等）

混合比较 1 1.5 不同长度比较，转换规则

比较运算符用错了吗？ < <=

布尔表达式是否正确 && || 最后的范围正确吗？

比较运算符两边不应用布尔类型 1<2<3 实际上是true<3

浮点精度丢失

多个与或非放一起，顺序和相互关系是否正确？

控制错误：

循环是否可以终止？ 终止时返回什么？

函数最终终止了吗？ 错误 ，返回

循环有执行吗？

循环如果没有终止，会有什么输出？

仅差一个错误——循环多一次或者少一次 循环起点是0还是1 结束是<还是<= 当前值 最终值s

If判断 是否覆盖了所有的情况？ Else包含多少情况？

接口错误： 形参和实参 数量 顺序 类型，长度（long） 是否相等

**传入的参数是否符合要求？**

**不要使用不希望的参数类型转换**

默认参数是否正确？

仅用于get的形参是否在程序中被修改？

全局变量

Io错误 文件的属性是否正确？ 有无分配正确的空间来保存读出的内容

文件是否打开？ 用完是否关闭？ Io错误处理？

修改文件时指针位置是否正确？ 文件内容是否正确？

标识符交叉引用列表？从未使用的变量 每个变量的属性列表

警告信息

**程序的健壮性，需要输入合法检查**

**程序是否遗漏功能？**

走查： 测试带上用例，大家一起推演……过一遍流程，写程序数据变化图 提问解答

桌面检查：效率不高

同行评分： 代码是否易理解 层次设计是否清晰（从高到低） 是否好修改？ 好扩展？

四、设计测试用例

所有用例中 采取哪些才能发现最多的错误，而且消耗最少？

随机测试效率最低

黑盒白盒结合

白（知道内部语句是怎么样的，合理设计用例）：

逻辑覆盖测试 最好是每条语句路线都能运行。。。存在循环，不可能无限覆盖

判定覆盖 每条分支都一次 If –else do-while switch

条件覆盖：每个条件分支的所有可能输入都执行一次 用例要能覆盖所有条件

2、等价划分 黑

减少无用用例数，增加用例类型

划分用例的范围，每个区间选代表性用例，一个区间内的用例都是等价的（发现同样的错误，或者同样能通过）

参数数量 参数类型 参数范围（过大，过小，边界0，负）

有效等价类：有效输入

无效等价类:非法或者不希望的输入

取值范围：范围内 大于范围 小于范围

个数：0个 或小于个数 等于个数 大于个数

输入类型： 正确类型 错误类型

输入的限制要求 ： 满足要求，不满足要求

覆盖有效等价类

无效等价类

1、确认输入 条件 划分有效和无效（非法）输入 范围

边界值分析 对有效和无效等价类 边界分析

需要思考输出

输入范围的边界 正好边界 边界超一点

参数数量，最少参数，最多的参数，少一个 多一个

输出的边界：输出有什么限制？ 结果空间边界 返回值 超过规定的情况 如何获得错误输出 输出范围，输出数量，**输出类型** (比如类型转换丢失数据)

设计用例 使输出超出需求。。。 恰好满足需求 满足最小需求 满足最大需求

输入一连串数据（串，数组，对象，注意第一个和最后的值）

**使用边界条件进行判断，可以让我们写最少的用例满足有效和无效**

**进行输入报告测试和输出测试…… 合法输入和非法输入，合法输出和非法输出全部覆盖**

1. 因果图

输入条件存在组合 即……又 不组合的话覆盖还是不足

**参数多了而且参数范围错误 （覆盖很麻烦）**

测试单元应该是可测的

因果 输入条件导致的输出 或者是系统转换（修改外部数据）

**输入会导致什么结果 然后把条件组合一下，分析会有什么结果**

**不能同时满足的原因（输入） 输出之间的关联影响**

5、错误猜测 靠经验和直觉

输入了0 输入仅一个值 输入为空 输入顺序错误 重复输入（两次输入相同，误输入）

类型错误

奇数偶数

五、单元测试（模块测试） 单个程序或者类测试 每个类的api都应该被测试，每个类有个main()或者test()

找出和设计上的差距

1、用例设计 单元是比较偏白盒的 白盒分析每条逻辑路线是否正确

应该有的返回值

逻辑覆盖

足够多的用例 让所有条件判断都触发一次 ，注意短路和中断的判断

2、增量测试

自顶而上 自底而下

先独立测试再组装起来测试？ 非增量

还是测试完了就组装，然后下一个测试？

测试需要驱动模块（test或者main函数），将测试用例（数据）传递给被测模块，驱动模块显示结果

桩模块： 当被测试的模块需要调用其他**api模块 或者数据的时候，**而另一个模块没准备好，就用桩提供给他 （）

增量测试：测完一个就组装一个。不需要太多的桩;可以更快检查出接口错误（调用不对的api）;可以提前发现组装的错误 不方便并行（分布测试），

自顶向下的测试 开发 设计

先测调用模块，然后测一个个需要被调用的模块（开始可能需要很多桩模块）

先测被调模块，再测调用模块

桩模块本身也可能是输入（测试用例/**如果被调模块不是某种固定的输出或者副作用的话**），下层本身就是上层的输入。。。所以桩也需要覆盖。。。**但是桩只需要覆盖被调模块可能的输出就行了**

1. 设计不同的桩，多次执行调用测试 2、测试数据从文件读取 生成不同的桩来测试
2. 测完用实际模块代替桩
3. 应该早点测核心模块 早点把io加入进来 （文件，网络，数据库，减少创建桩的压力）

自顶而上的好处——只要做好桩就能运行最上层模块，方便客户演示，方便整体功能是否合理

不利于程序设计迭代……上层改进 上层有时候给下层提供资源，这在桩中是无法验证正确的（不应该）

2、桩设计并不简单， 可能导致不能穷举完全

自底而上：先测不调用其他模块的被调模块，然后测调用模块，要求所有被调模块测过了，全部组装上去再测。。。。

不需要桩

缺点：直到最后一个模块组装好，才是完整的程序

1. 执行测试

输出和预期不对：模块错误 或者预期是错的（用例数据错误）

一开始就要检查测试用例（数据）

自动化测试工具：流程分析工具

程序的副作用，不应该执行的操作， 是否修改了不改该的输入

六、更高级别的测试

无法实现最终用户要求的合理功能 ——软件错误

信息沟通和转换时的故障和差错——需求（要实现） 目标（具体） 规格（黑盒，外部和用户的交互） 设计系统（分割模块） 设计程序（模块功能层次，间的接口） 说明接口

编码

阶段测试：在开发完成后，对各层次上都进行测试

模块测试（外部接口可用性） 功能测试

功能测试

系统测试：作为用户，对整个系统进行测试 。。。理论上找用户测试是最好的

参考文档进行测试

容量测试 大量处理

压力测试（强度测试） 短时间高速输入（飞快点击），超级并发，同时使用N个功能

易用性测试： 用户界面是否好用？ 错误信息用户是否能够理解？

即时交互反馈

准确性和安全性

性能测试：

存储测试：

兼容性测试

安装测试（是否好安装）

可靠性测试 （对安全测试的强度评估），要求一个很高的指标

可恢复性测试（从崩溃，数据丢失中恢复） 看恢复方法是否达到要求，数据是否正确还原

文档测试：文档正确性

测试的总错误数 编码和逻辑错误——设计错误

七、调试 成功测试之后（即发现错误之后，非法或合法输入导致非法输出，实现了不需要的功能）

对错误调试，找出原因并修改处理

调试很麻烦 可能是漏了条件 也可能是语句错误 可能是没处理非法输入和边界

1、暴力调试 内存信息 （静态或者动态）

插入打印语句（print或者日志打印变量等）

自动化工具 断点 call

打印堆栈轨迹

2、思考 使用归纳法 从细节线索分析原因

需要一定的知识量（可以查询）

相关数据 错误输出是什么导致的—— 把所有相关的数据之间的关系组织清楚——做出假设——验证

需要考虑所有可用数据 列举正确的输出 和错误的输出 影响输出的数据，为什么正确，为什么错误

归纳是从特殊到一般- 不正确的输出是什么？输入的是什么？ 相关的值是什么？ 相关值正确吗？ 正确的输出是什么？输入是？

3、演绎法

所有可能的原因——排除法——现场情况进行排除——剩下的可能——验证

1. 回溯法 顺着输出往上看程序结构，一步步反推 （小程序）

5、测试法 使用用例 进行筛选 需要的用例少

6、定位错误 动脑 ，收集信息

想不出来就放放，做别的事

和别人交流

少用修改语句的方式试错

先思考，再用调试工具

1. 修改错误

有错的地方可能会有其他的错误

要修改错误本身，不要只是一部分

修改错误之后要进行更严格的测试（输入枚举）

改正错误可能引入新错误

错误修改也是一种设计

7、错误分析

错误在哪里（编码还是设计或者需求，要追到源头） 错误是谁的？ 错误的原因？（人员方面） 如何避免下次 为什么没有更早发现， 如何更早发现（测试用例）

八、极限测试 XP极限编程 设计 反馈