7.1.1

程序设计语言是人和计算机通信的最基本的工具，会影响人的思维和解题方式，影响人和计算机通信的方式和质量，影响其他人阅读和理解程序的难易程度。

那么选择适宜的程序设计语言的原因是什么？

答案：1.根据设计去完成编码时，困难最少；

2.可以减少需要的程序测试量；

3.可以得到更容易阅读和更容易维护的程序

相比于汇编语言，高级语言更加的实用高效，除了在特殊领域如对程序执行时间和使用时间和使用空间都有很严格限制的情况，需要产生任意的甚至非法的指令序列等：

高级语言和汇编语言的区别是：

1.汇编语言编码需要把软件设计翻译成机器操作的序列，既困难又容易出差错；

2.高级语言写程序比用汇编语言写程序生产率可以提高好几倍；

3.用高级语言写的程序容易阅读、容易测试、容易调试、容易维护。

选择程序语言有理想标准和实用标准两种标准

选择程序语言的理想标准有以下几点：

1.应该有理想的模块化机制，以及可读性好的控制结构和数据结构；使程序容易测试和维护2.以减少软件的总成本

3.使编译程序能够尽可能多地发现程序中的错误；便于调试和提高软件的可靠性

4.应该有良好的独立编译机制。降低软件开发和维护的成本

选择程序语言的实用标准有以下几点：

1.系统用户的要求；

2.可以使用的编译程序；

3可以得到的软件工具；

4工程规模；

5程序员的知识；

6软件可移植性要求；

7软件的应用领域。

7.1.2编码风格

源程序代码的逻辑简明清晰、易读易懂是好程序的一个重要标准，为了做到这一点，应该遵循以下的规则。

程序内部的文档的规范，数据说明的次序标准化，语言构造的简单明了，输入输出的规则以及高效率

第一点程序内部的文档

所谓程序内部的文档包括恰当的标识符、适当的注解和程序的视觉组织等

第二点数据说明

数据说明的次序应该标准化；当多个变量名在一个语句中说明时，应该按字母顺序排列这些变量；如果设计时使用了一个复杂的数据结构，则应该用注解说明用程序设计语言实现这个数据结构的方法和特点。

第三点是语句构造

1不要为了节省空间而把多个语句写在同一行；

2尽量避免复杂的条件测试；

3尽量减少对“非”条件的测试；

4避免大量使用循环嵌套和条件嵌套；

5利用括号使逻辑表达式或算术表达式的运算次序清晰直观。

第四点输入输出

1在设计和编写程序时需考虑有关输入输出风格的规则：

2对所有输入数据都进行检验；

3检查输入项重要组合的合法性；

4保持输入格式简单；

5使用数据结束标记，不要要求用户指定数据的数目；

6明确提示交互式输入的请求，详细说明可用的选择或边界数值；

7程序设计语言对格式有严格要求时，应保持输入格式一致；

8设计良好的输出报表；

9给所有输出数据加标志。

第五点是效率

效率主要指处理机时间和存储器容量两个方面。效率是性能要求，因此应该在需求分析阶段确定效率方面的要求；而效率是靠好设计来提高的；同时程序的效率和程序的简单程度是一致的，不要牺牲程序的清晰性和可读性来不必要地提高效率。

效率又从三个方面进行分析，

1）程序运行时间

包括了写程序之前先简化算术的和逻辑的表达式；仔细研究嵌套的循环，以确定是否有语句可以从内层往外移；尽量避免使用多维数组；尽量避免使用指针和复杂的表；使用执行时间短的算术运算；不要混合使用不同的数据类型；尽量使用整数运算和布尔表达式。

1. 存储器效率

包括了在大型计算机中必须考虑操作系统页式调度的特点，一般说来，使用能保持功能域的结构化控制结构，是提高效率的好方法。在微处理机中如果要求使用最少的存储单元，则应选用有紧缩存储器特性的编译程序，在非常必要时可以使用汇编语言

提高执行效率的技术通常也能提高存储器效率。

1. 输入输出的效率

包括了在写程序的角度有以下原则所有输入输出都应该有缓冲，以减少用于通信的额外开销；

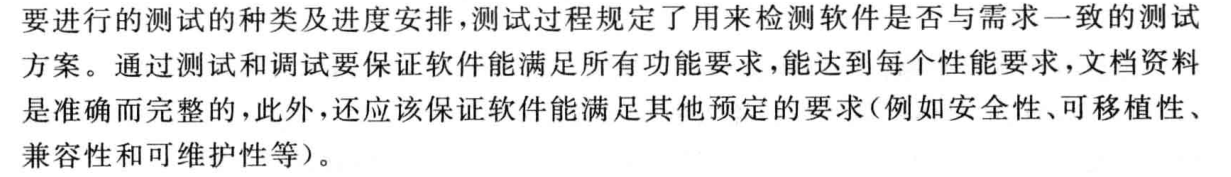
对二级存储器(如磁盘)应选用最简单的访问方法；二级存储器的输入输出应该以信息组为单位进行；如果“超高效的”输入输出很难被人理解，则不应采用这种方法。

7.5确认测试

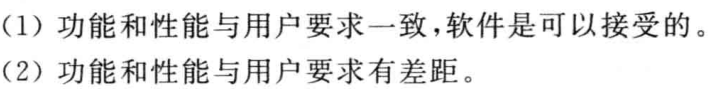
确认测试也叫验收测试，需求分析阶段产生的软件需求规格说明书，准确地描述了用户对软件的合理期望，是进行确认测试的基础。

7.5.1确认测试的范围

首先在确认测试的时候必须有用户的参与，根据用户的需求对各方面需求进行完善。确认测试使用黑盒测试法。分为测试计划和测试过程，测试计划包括



确认测试后会有两种可能



7.5.2软件的配置复查

复查的目的是保证软件配置的所有成分都齐全,质量符合要求,文档与程序完全一致,具有完成软件维护所必须的细节，而且已经编好目录。

7.5.3Alpha和 Beta 测试

当软件是为了许多用户开发的那么就需要Alpha和 Beta 测试

Alpha 测试由用户在开发者的场所进行,并且在开发者对用户的“指导”下进行测试开发者负责记录发现的错误和使用中遇到的问题。总之,Alpha 测试是在受控的环境中进行的。

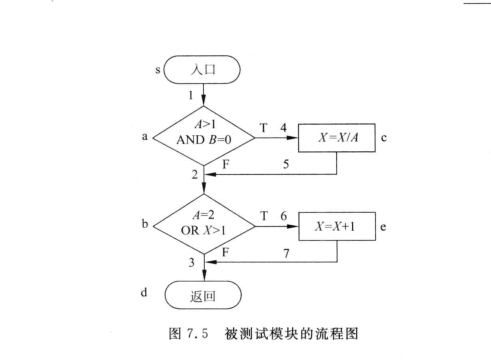
Beta 测试由软件的最终用户们在一个或多个客户场所进行。与 Alpha 测试不同开发者通常不在 Beta 测试的现场,因此,Beta 测试是软件在开发者不能控制的环境中的“真实”应用。用户记录在 Beta 测试过程中遇到的一切问题(真实的或想象的),并且定期把这些问题报告给开发者。接收到在 Beta 测试期间报告的问题之后,开发者对软件产品进行必要的修改,并准备向全体客户发布最终的软件产品。

7.6白盒测试技术

设计测试方案是测试阶段的关键技术问题。所谓测试方案包括具体的测试目的(例如,预定要测试的具体功能),应该输入的测试数据和预期的结果。通常又把测试数据和预期的输出结果称为测试用例。接下来我们讲一下白盒测试技术。

7.6.1逻辑覆盖

有选择地执行程序中某些最有代表性的通路是对穷尽测试的唯一可行的替代办法所谓逻辑覆盖是对一系列测试过程的总称，这组测试过程逐渐进行越来越完整的通路测试。



1. 语句覆盖

语句覆盖的含义是,选择足够多的测试数据,使被测程序中每个语句至少执行一次。语句覆盖对程序逻辑覆盖很少，此外语句覆盖只关心判定表达式的值,而没有分别测试判定表达式中每个条件取不同值时的情况。

语句覆盖只关心判定表达式的值，而没有分别测试判定表达式中每个条件取不同值时的情况。为了执行sacbed路径，以测试每个语句，只需两个判定表达式(A>1)AND(B=0)和(A=2)OR(X>1)都取真值，因此使用上述一组测试数据就够了。但是，如果程序中把第一个判定表达式中的逻辑运算符AND错写成OR，或把第二个判定表达式中的条件X>1误写成X<1，使用上面的测试数据并不能查出这些错误。

1. 判定覆盖

判定覆盖又叫分支覆盖,它的含义是,不仅每个语句必须至少执行一次,而且每个判定的每种可能的结果都应该至少执行一次,也就是每个判定的每个分支都至少执行一次。

虽然判定覆盖比语句覆盖强,但是对程序逻辑的覆盖程度仍然不高。

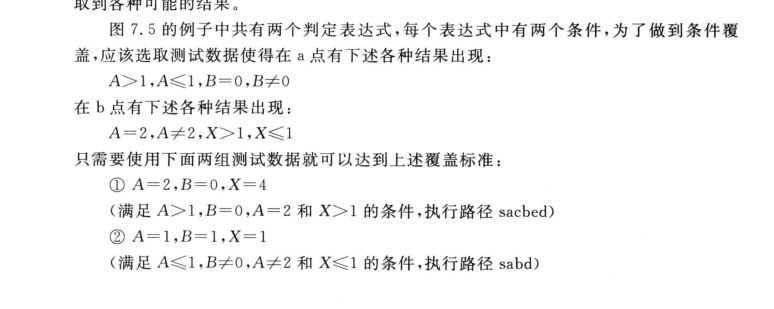
对于上述例子来说，能够分别覆盖路径sacbed和sabd的两组测试数据，或者可以分别覆盖路径sacbd和sabed的两组测试数据，都满足判定覆盖标准。例如，以下两组测试数据就可做到判定覆盖：

① A=3，B=0，X=3 (覆盖sacbd)

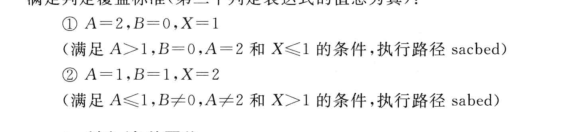
② A=2，B=1，X=1 (覆盖sabed)

1. 条件覆盖

条件覆盖的含义是,不仅每个语句至少执行一次,而且使判定表达式中的每个条件都取到各种可能的结果。

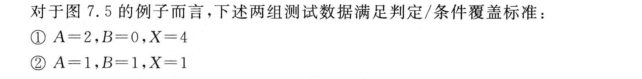


条件覆盖通常比判定覆盖强,因为它使判定表达式中每个条件都取到了两个不同的结果,判定覆盖却只关心整个判定表达式的值。。但是,也可能有相反的情况:虽然每个条件都取到了两个不同的结果,判定表达式却始终只取一个值。。例如,如果使用下面两组测试数据，则只满足条件覆盖标准并不满足判定覆盖标准(第二个判定表达式的值总为真):



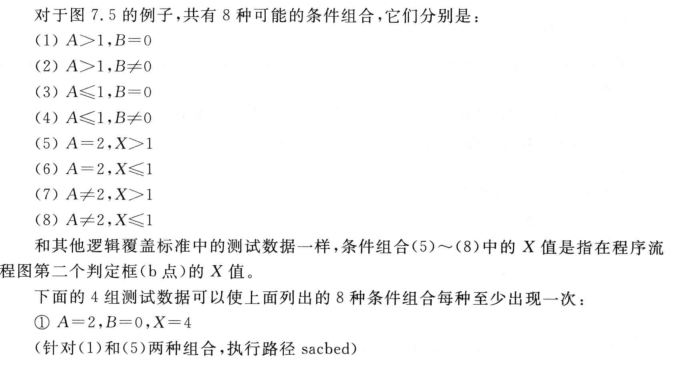
4.判定/条件覆盖

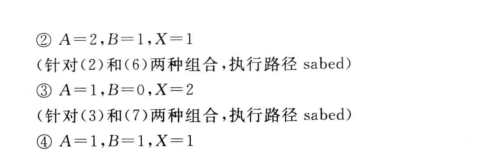
既然判定覆盖不一定包含条件覆盖,条件覆盖也不一定包含判定覆盖,自然会提出一种能同时满足这两种覆盖标准的逻辑覆盖,这就是判定/条件覆盖。它的含义是,选取足够多的测试数据,使得判定表达式中的每个条件都取到各种可能的值，而且每个判定表达式也都取到各种可能的结果。



5.条件组合覆盖

条件组合覆盖是更强的逻辑覆盖标准,它要求选取足够多的测试数据,使得每个判定表达式中条件的各种可能组合都至少出现一次。





6.点覆盖

图论中点覆盖的定义如下：如果连通图G的子图G′是连通的，而且包含G的所有结点，则称G′是G的点覆盖。

满足点覆盖标准要求选取足够多的测试数据，使得程序执行路径至少经过流图的每个结点一次，由于流图的每个结点与一条或多条语句相对应，显然，点覆盖标准和语句覆盖标准是相同的。

7.边覆盖 图论中边覆盖的定义是：如果连通图G的子图G″是连通的，而且包含G的所有边，则称G″是G的边覆盖。为了满足边覆盖的测试标准，要求选取足够多测试数据，使得程序执行路径至少经过流图中每条边一次。通常边覆盖和判定覆盖是一致的。

8.路径覆盖

路径覆盖的含义是，选取足够多测试数据，使程序的每条可能路径都至少执行一次(如果程序图中有环，则要求每个环至少经过一次)。

7.6.2.控制结构测试

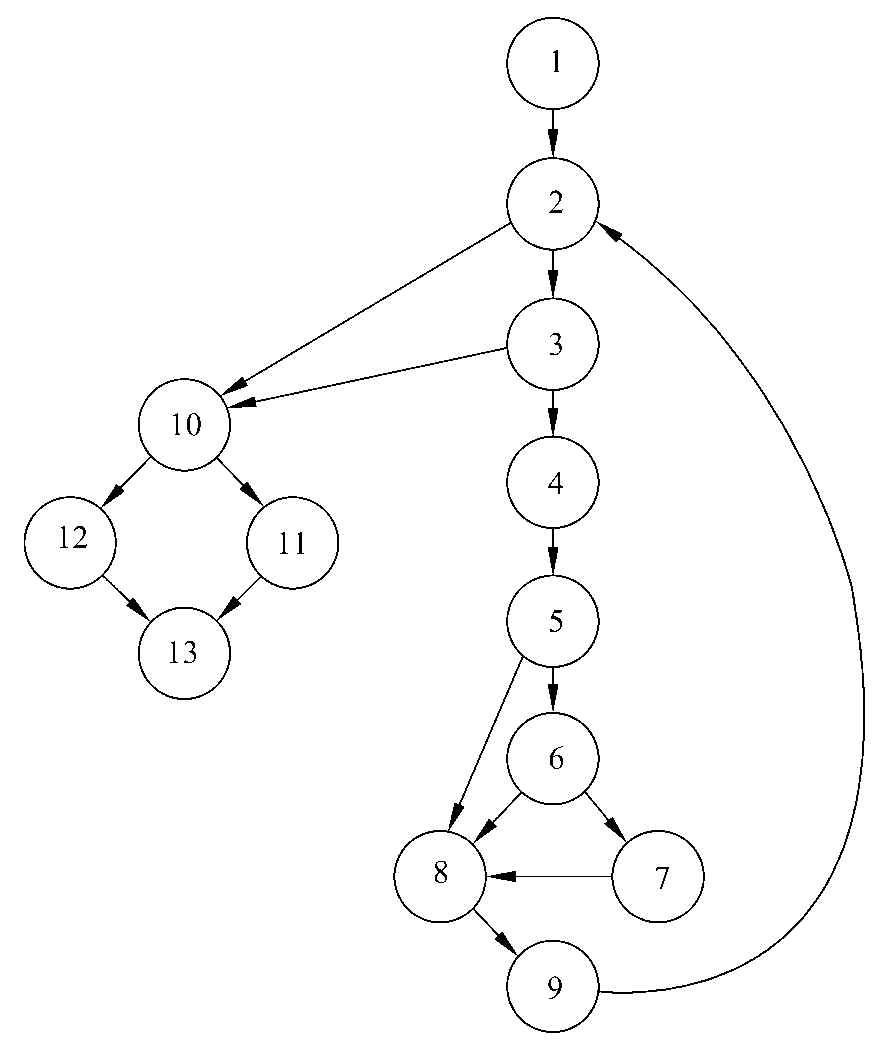
1.基本路径测试

基本路径测试是Tom McCabe提出的一种白盒测试技术。使用基本路径测试设计测试用例时，首先计算程序的环形复杂度，并用该复杂度为指南定义执行路径的基本集合，从该基本集合导出的测试用例可以保证程序中的每条语句至少执行一次，而且每个条件在执行时都将分别取真、假两种值。

使用基本路径测试技术设计测试用例的步骤如下。

① 根据过程设计结果画出相应的流图。

例如，为了用基本路径测试技术测试下列的用PDL描述的求平均值过程，首先画出下图所示的流图。注意，为了正确地画出流图，这里把被映射为流图结点的PDL语句编了序号。



② 计算流图的环形复杂度。

环形复杂度定量度量程序的逻辑复杂性。使用第6.5.1小节讲述的3种方法之一计算环形复杂度。经计算，流图的环形复杂度为6。

③ 确定线性独立路径的基本集合。

独立路径是指至少引入程序的一个新处理语句集合或一个新条件的路径，即独立路径至少包含一条在定义该路径之前不曾用过的边。

程序的环形复杂度决定了程序中独立路径的数量，而且这个数是确保程序中所有语句至少被执行一次所需的测试数量的上界。

④ 设计可强制执行基本集合中每条路径的测试用例。

应该选取测试数据使得在测试每条路径时都适当地设置好各个判定结点的条件。测试第③步得出的基本集合的测试用例如下。

看ppt讲例子。（太多例子，不放了）

2.条件测试基本路径测试技术简单而高效但还要其他的技术来提高白盒测试的质量。

用条件测试技术设计出的测试用例，能够检查程序模块中包含的逻辑条件。一个简单条件是一个布尔变量或一个关系表达式，在布尔变量或关系表达式之前还可能有一个NOT（┐)算符。关系表达式的形式如下：

E1<关系算符>E2

其中，E1和E2是算术表达式，而<关系算符>是下列算符之一：<，≤，=，≠，>或≥。布尔算符有OR(|)，AND(&)和NOT( ┐)。不包含关系表达式的条件称为布尔表达式。

因此，条件成分的类型包括布尔算符、布尔变量、布尔括弧（括住简单条件或复合条件）、关系算符及算术表达式。

如果条件不正确，则至少条件的一个成分不正确。因此，条件错误的类型有：布尔算符错、布尔变量错、布尔括弧错、关系算符错、算术表达式错。

条件测试方法着重测试程序中的每个条件。条件测试策略有两个优点： ①容易度量条件的测试覆盖率； ②程序内条件的测试覆盖率可指导附加测试的设计

条件测试的目的不仅是检测程序条件中的错误，而且是检测程序中的其他错误。如果程序P的测试集能有效地检测P中条件的错误，则它很可能也可以有效地检测P中的其他错误。

BRO测试利用条件C的条件约束来设计测试用例。包含n个简单条件的条件C的条件约束定义为（D1，D2，…，Dn），其中Di(0<i≤n)表示条件C中第i个简单条件的输出约束。如果在条件C的一次执行过程中，C中每个简单条件的输出都满足D中对应的约束，则称C的这次执行覆盖了C的条件约束D。

讲ppt的例子

1. 循环测试

循环测试是一种白盒测试技术，它专注于测试循环结构的有效性。在结构化的程序中通常只有3种循环，即简单循环、串接循环和嵌套循环。

1简单循环，应该使用下列测试集来测试简单循环，其中n是允许通过循环的最大次数。

跳过循环。

只通过循环一次。

通过循环两次。

通过循环m次，其中m<n-1。

通过循环n-1,n,n+1次。

2嵌套循环，如果把简单循环的测试方法直接应用到嵌套循环，测试数就会随嵌套层数的增加按几何级数增长，B.Beizer提出了一种能减少测试数的方法。跳过循环。

从最内层循环开始测试，把所有其他循环都设置为最小值。

对最内层循环使用简单循环测试方法，而使外层循环的迭代参数（例如，循环计数器）取最小值，并为越界值或非法值增加一些额外的测试。

由内向外，对下一个循环进行测试，但保持所有其他外层循环为最小值，其他嵌套循环为“典型”值。

继续进行下去，直到测试完所有循环。

3串接循环，如果串接循环的各个循环都彼此独立，则可以使用前述的测试简单循环的方法来测试串接循环。但是，如果两个循环串接，而且第一个循环的循环计数器值是第二个循环的初始值，则这两个循环并不是独立的。当循环不独立时，建议使用测试嵌套循环的方法来测试串接循环。