第15章

设计模式与软件测试

鲁迅先生曾经说过：“地上本没有路，走的人多了也就成了路。”设计模式如同此理，它是经验的传承，并非体系。它是被前人发现，经过总结形成的一套某一类问题的一般性解决方案，而不是被设计出来的定性规则。它不像算法那样可以照搬照用。

设计模式关注的重点在于通过经验提取的“准则或指导方案”在设计中的应用，因此在不同层面考虑问题的时候就形成了不同问题领域上的模式。模式的目标是，把共通问题中的不变部分和变化部分分离出来。不变的部分，就构成了模式。因此，模式是一个经验提取的“准则”，并且在一次一次的实践中得到验证。在不同的层次有不同的模式，小到语言实现，大到架构。在不同的层面上，模式提供不同层面的指导。

和建筑结构一样，软件中亦有诸多的“内力”。和建筑设计一样，软件设计也应该努力疏解系统中的内力，使系统趋于稳定、有生气。一切软件设计都应该由此出发。任何系统都需要有变化，任何系统都会走向死亡。作为设计者，应该拥抱变化，利用变化，而不是逃避变化。

经典设计模式一共有 23 种，面试时重要的不是你熟记了多少个模式的名称，关键还在于付诸实践的运用。为了有效地设计而去熟悉某种模式所花费的代价是值得的，因为很快你会在设计中发现这种模式真的很好，很多时候它会令你的设计更加简单。

软件测试是指使用人工或者自动手段来运行或测试某个系统的过程，其目的在于检验软件是否满足规定的需求或弄清预期结果与实际结果之间的差别。测试工程师是目前 IT 行业极端短缺的人才，未来 5 年 IT 行业最炙手可热的职位。中国软件业每年新增约 20 万测试岗位就业机会，而企业、学校培养出的测试人才却不足需求量的 1/10，这种测试人才需求与供给间的差距仍在拉大。

随着软件市场的成熟，软件对社会运转的巨大贡献已经得到了广泛认可，但是，人们对软件作用期望值也越来越高，更多人将关注点转移到软件的质量和功能可靠性上，而软件产业在产

品性能测试领域存在着严重不足，软件测试水平的高低可以说是决定了软件产业的前途命运。

15.1　设计模式

面试例题 1: Which came first, chicken or the egg?(请用设计模式观点描述先有鸡还是先有蛋?)

[德国某著名软件咨询企业 2005 年 10 月面试题]

解析:

请先看下面的程序:

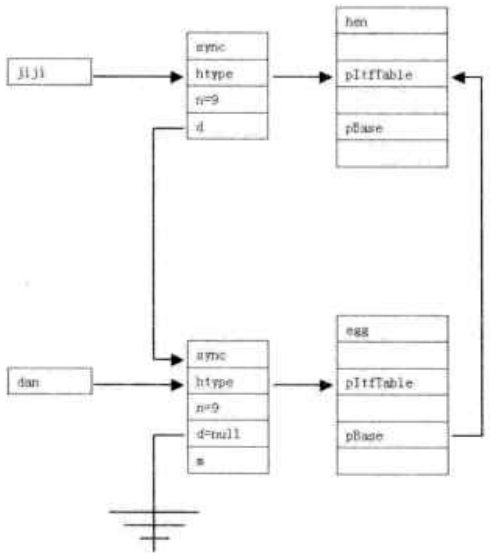
```
using System;
 class Client
{
     public static void Main ()
     {
          hen jiji = new hen();
          egg dan = new egg();
          jiji.d = dan;
          Console.WriteLine(jiji.d.m);
     }
}
```

```
 class hen
{
     public int n = 9;
     public egg d;
}


class egg : hen
{
     public int m = 10;
}
```

egg 继承自 hen，可以说没有 hen 就没有 egg，可 hen 里面有一个成员是 egg 类型。到底是先有鸡还是先有蛋？这个程序可以正常编译执行并打印结果 10。

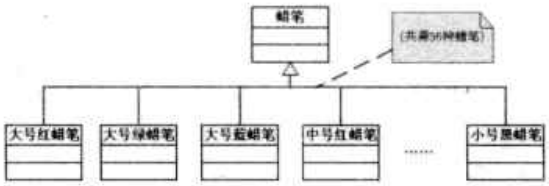
答案:“先有鸡还是先有蛋”问题只是对面向对象本质的一个理解，将人类的自然语言放在此处来理解并不合适。由下图可知，根本不存在鸡与蛋的问题，而是型与值的问题，以及指针引用的问题，因为鸡和蛋两个对象间是“引用”关系而不是“包含”关系。



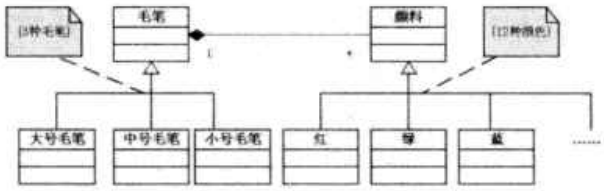
面试例题 2： 请用设计模式的思想描述蜡笔和毛笔有什么不同，并用 C#程序实现。

解析：大家小时候都有用蜡笔画画的经历吧。红红绿绿的蜡笔一大盒，根据想象描绘出各式的图样。而毛笔下的国画更是工笔写意各展风采。而今天我们的故事从蜡笔与毛笔说起。

设想要绘制一幅图画，蓝天、白云、绿树、小鸟。如果画面尺寸很大，那么用蜡笔绘制就会遇到点儿麻烦。毕竟细细的蜡笔要涂出一片蓝天是有些麻烦的。如果有可能，最好有套大号蜡笔，粗粗的蜡笔很快能涂抹完成。至于色彩嘛，最好每种颜色来支粗的，除了蓝天还有绿地呢。这样，如果一套 12 种颜色的蜡笔，我们需要两套 24 支，同种颜色的一粗一细。要是再有一套中号蜡笔就更好了，于是，不多不少总共 36 支蜡笔（见下图）。



再看看毛笔这一边，居然如此简陋：一套水彩 12 色，外加大、中、小 3 支毛笔（见下图）。你可别小瞧这“简陋”的组合，画蓝天用大毛笔，画小鸟用小毛笔，各有专长。



这就是 Bridge 模式。为了一幅画，我们需要准备 36 支型号不同的蜡笔，而改用毛笔的话 3 支就够了，当然还要搭配上 12 种颜料。通过 Bridge 模式，我们把乘法运算 3 × 12＝36 改为了加法运算 3＋12＝15，这一改进可不小。那么这里蜡笔和毛笔到底有什么区别呢？

实际上，蜡笔和毛笔的一个关键区别就在于笔和颜色是否能够分离。桥梁模式的用意是“将抽象化（Abstraction）与实现化（Implementation）脱耦，使得二者可以独立地变化”。关键就在于能否脱耦。蜡笔的颜色和蜡笔本身是分不开的，所以必须使用 36 支色彩、大小各异的蜡笔来绘制图画。而毛笔与颜料能够很好地脱耦，各自独立变化，简化了操作。在这里，抽象层面的概念是“毛笔用颜料作画”，而在实现时，毛笔有大、中、小 3 号，颜料有红、绿、蓝等 12 种，于是便可出现 3 × 12 种组合。每个参与者（毛笔与颜料）都可以在自己的自由度内随意转换。

蜡笔由于无法将笔与颜色分离，造成笔与颜色两个自由度无法单独变化，使得只有创建 36 种对象才能完成任务。Bridge 模式将继承关系转换为组合关系，从而降低了系统间的耦合，

减少了代码编写量。

答案:

代码如下:

```
// 桥接模式类型举例
using System;

// "Abstraction"类
class Abstraction
{
  // 字段
  protected Implementor implementor;

  // 属性
  public Implementor Implementor
  {
    set{ implementor = value; }
  }

  // 函数
  virtual public void Operation()
  {
    implementor.Operation();
  }
}

// "Implementor"类
abstract class Implementor
{
  // 函数
  abstract public void Operation();
}

// "RefinedAbstraction"类
class RefinedAbstraction : Abstraction
{
  // 函数
  override public void Operation()
  {
    implementor.Operation();
  }
}

// "ConcreteImplementorA"类
classConcreteImplementorA:Implementor
{
 // 函数
  override public void Operation()
  {
Console.WriteLine("ConcreteImplementorA
Operation");
  }
}

// "ConcreteImplementorB"
classConcreteImplementorB:Implementor
{
  // 函数
  override public void Operation()
  {
Console.WriteLine("ConcreteImplementorB
Operation");
  }
}

/**//// <summary>
/// Client test
/// </summary>
public class Client
{
  public  static  void  Main(string[]
args )
  {
    Abstraction        abstraction=new
RefinedAbstraction();
    // 设置 implementation 委托并且调用
abstraction.Implementor=newConcreteImplem
entorA();
    abstraction.Operation();
    // 修改 implementation 委托并且调用
abstraction.Implementor=newConcreteImplem
entorB();
    abstraction.Operation();
  }
}
```

面试例题 3: 以下代码实现了设计模式中的哪种模式? [美国某著名搜索引擎公司 2005 年面试题]

```
public sealed class SampleSingleton1
{
private int m Counter = 0;
private SampleSingleton1()
{
Console.WriteLine(""初始化SampleSingleton1。
"");
}
publicstatic readonly SampleSingleton1
Singleton = new
  SampleSingleton1();
public void Counter()
{
m Counter ++;
}
}
```

A．原型　　B．抽象工厂　　C．单键　　D．生成器

解析： 这是一个典型的单键模式。

答案： C

面试例题 4： Consider a context-sensitive help feature for graphical user interface. The user can obtain help information on any part of the interface just by clicking on it. The help that's provided depends on the part of interface that's selected and its context; for example, a button widget in a dialog box might have different help information than a similar button in the main window. If no specific help information exists for that part of the interface, then the help system should display a more general help message about the immediate context——the dialog box as a whole, for example.

（为图形用户界面设计一种特性——能领会语境并提供帮助，使得用户仅仅通过在界面的任何部分上单击鼠标就可以获得相应的帮助信息。它所提供的帮助主要依赖于所选择的界面及相应的语境。

举个例子，在具备这样特性的一个对话框中的一个按钮部件上单击鼠标可能就会有一个与主窗口中类似的按钮但有不同的帮助信息。如果没有该界面的这部分内容的具体帮助信息，那么帮助系统就会根据即时的具体语境提供较为通用的帮助信息，这就是对于上述问题的一个具体的例子。）

为了设计上面的东西该选用（　　）。[中国台湾某著名杀毒软件公司 2005 年 10 月面试题]

A．观察者模式　　B．桥接模式

C．供应链模式　　D．原型模式

解析： 本题应该选择观察者模式。为了形象描述观察者模式，举一个给教工发大米的例子。

每年到年底的时候，工会都要给每位教工发大米、油什么的。正好我家的米快吃完了，所以就盼着发大米。可是等了好多天也没见什么动静，扳着手指头数数离春节还有多半个月呢，看我心急的。不过，盼大米的老师恐怕不只是我一个，都等着工会有什么动静就去领大米呢。

教工等着发大米是一个典型的观察者模式。当工会大米来了，教工就会做出响应。不过这观察也有两种说头法：一种是拉（Pull）模式，要求教工时不时到工会绕一圈，看看大米来了没有，恐怕没有人认为这是一种好办法。当然，还有另外一种模式，就是推（Push）模式，大米到了工会，工会会给每家每户把大米送过去。第二种方法好不好呢？好？呵呵，谁说好了，谁说好我让谁到工会上班去。

其实，我们还有一种方法，就是大米到了工会后，工会不把大米给每人送去，而是给每人发个轻量级的“消息”，教工得到消息后，再把大米拉回各家。这要求每位教工有一个工会的引用，在得到消息后到指定的地点领取大米。这样工会不用给每家教工送大米，而教工也不用每天到工会门口巴望着等大米了。

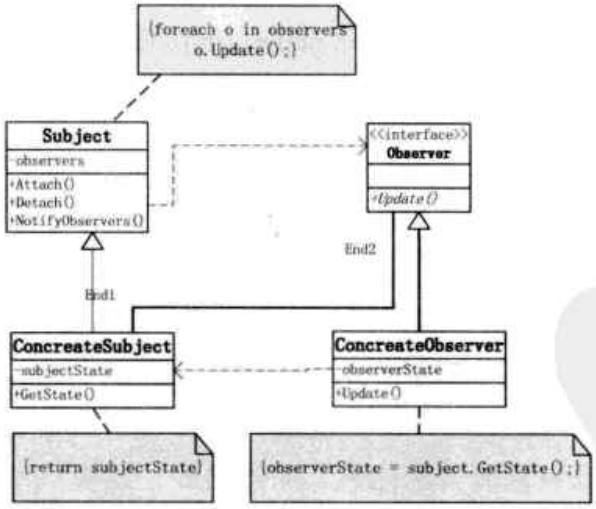
而本题鼠标帮助的例子也与教工拉大米相似。观察者模式定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题帮助。这个帮助对象在状态上发生变化时，会通知所有观察者对象，使它们能够自动更新自己。

一个软件系统常常要求在某一个对象的状态发生变化的时候，某些其他的对象做出相应的改变。做到这一点的设计方案有很多，但是为了使系统能够易于复用，应该选择低耦合度的设计方案。减少对象之间的耦合有利于系统的复用，但是同时设计师需要使这些低耦合度的对象能够维持行动的协调一致，保证高度的协作（collaboration）。观察者模式是满足这一要求的各种设计方案中最重要的一种。

答案：A

扩展知识（观察者模式的结构）

观察者模式的类如下图所示。



可以看出，在这个观察者模式的实现里有下面这些角色。

- 抽象主题（Subject）角色：主题角色把所有对观察者对象的引用保存在一个聚集里，每个主题都可以有任何数量的观察者。抽象主题提供一个接口，可以增加和删除观察者对象。抽象主题角色又叫做抽象被观察者（Observable）角色，一般用一个抽象类或者一个接口实现。

- 抽象观察者（Observer）角色：为所有的具体观察者定义一个接口，在得到主题的通知时更新自己。这个接口叫做更新接口。抽象观察者角色一般用一个抽象类或者一个接口实现。在这个示意性的实现中，更新接口只包含一个方法（即 Update()方法），这个方法叫做更新方法。
- 具体主题（ConcreteSubject）角色：将有关状态存入具体现察者对象；在具体主题的内部状态改变时，给所有登记过的观察者发出通知。具体主题角色又叫做具体被观察者角色（ConcreteObservable）。具体主题角色通常用一个具体子类实现。
- 具体观察者（ConcreteObserver）角色：存储与主题的状态自恰的状态。具体观察者角色实现抽象观察者角色所要求的更新接口，以便使本身的状态与主题的状态相协调。如果需要，具体观察者角色可以保存一个指向具体主题对象的引用。具体观察者角色通常用一个具体子类实现。

从具体主题角色指向抽象观察者角色的合成关系，代表具体主题对象可以有任意多个对抽象观察者对象的引用。之所以使用抽象观察者而不使用具体观察者，意味着主题对象不需要知道引用了哪些 ConcreteObserver 类型，而只需知道抽象 Observer 类型。这就使得具体主题对象可以动态地维护一系列的对观察者对象的引用，并在需要的时候调用每一个观察者共有的 Update()方法。这种做法叫做针对抽象编程。

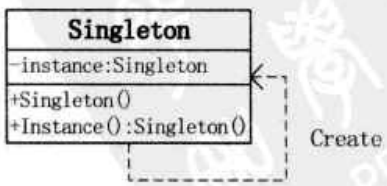
面试例题 5: Implement the simplest singleton pattern(initialize if necessary).（写一个单例模式，如果有必要的话就初始化。）[德国某著名软件咨询企业 2005 年 10 月面试题]

解析:

单例模式的特点有 3 个:

- 单例类只能有一个实例。
- 单例类必须自己创建自己的唯一实例。
- 单例类必须给所有其他对象提供这一实例。

Singleton 模式包含的角色只有一个，就是 Singleton。Singleton 拥有一个私有构造函数，确保用户无法通过 new 直接实例化它。除此之外，该模式中包含一个静态私有成员变量 instance 与静态公有方法 Instance()。Instance 方法负责检验并实例化自己，然后存储在静态成员变量中，以确保只有一个实例被创建。单例模式的结构如右图所示。



答案:

完整代码如下：

```
// 单件模式类型举例
using System;

// "Singleton"
class Singleton
{
    // 字段
    private static Singleton instance;

    // 结构
    protected Singleton() {}

    // 函数
    public static Singleton Instance()
    {
        // 设定初值进行初始化
        if( instance == null )

instance=newSingleton();

        return instance;
    }
}

/**//// <summary>
/// Client test
/// </summary>
public class Client
{
    public static void Main()
    {
    // 构造函数被保护——不能使用 new 字符

Singleton s1=Singleton.Instance();

Singleton s2=Singleton.Instance();

        if( s1 == s2 )
            Console.WriteLine(  "The
same instance" );
    }
}
```

15.2　软件测试基础

面试例题 1：Why is test automation important?（自动化测试为何重要？）[美国数据库公司 S2009 年 11 月面试题]

解析：自动化测试从根本上提高 QA 们的职业素质，让 QA 们从彻底上摆脱重复繁重的测试工作，而更着重在 QA 的流程上已经完成项目质量的重复保证上。此外某些人对 QA 有些偏见：对 QA 的普遍认识就是只是测试而已。因为他们能看到 QA 最直接的劳动就是在反反复复勤勤恳恳的测试。

答案：自动化测试可以让测试人员从枯燥无味的手工重复性测试中解放出来，并且提高工作效率，通过自动化测试结果来分析功能和性能上的缺陷。

面试例题 2：Describe criterions that testing is completed。（描述一个测试结束的准则。）[美国数据库公司 S2009 年 11 月面试题]

答案：一个测试结束的标准可以查看已提交的 bug 是否已经全部解决并已验证关闭，一般来说，bug 验证率在 95%以上，并且没有大的影响功能的 bug 处于未解决状态，就可以测试通过。

面试例题 3：What kinds of content should be included in a Test Plan?(For example, available

human resource)（在一个测试计划中能包含哪些内容，如可用的人力资源？）[美国数据库公司 S2009 年 11 月面试题]

答案：在一个测试计划中可以包含需要测试的产品的特点和主要功能模块，列出需要测试的功能点，并标明侧重点；测试的策略和记录（测试工具的确认，测试用例等文档模板，测试方法的确定）；测试资源配置（确定测试每一阶段的任务和所需资源）。

面试例题 4：Please describe differences between functional testing and usability testing.(请描述功能测试和可用性测试之间的区别。) [美国数据库公司 S2009 年 11 月面试题]

解析：本题涉及几个测试的重要概念：

Functional testing（功能测试），也称为 behavioral testing（行为测试），根据产品特征、操作描述和用户方案，测试一个产品的特性和可操作行为以确定它们满足设计需求。本地化软件的功能测试，用于验证应用程序或网站对目标用户能正确工作。使用适当的平台、浏览器和测试脚本，以保证目标用户的体验将足够好，就像应用程序是专门为该市场开发的一样。

功能测试也叫黑盒子测试或数据驱动测试，只需考虑各个功能，不需要考虑整个软件的内部结构及代码。一般从软件产品的界面、架构出发，按照需求编写出来的测试用例，输入数据在预期结果和实际结果之间进行评测，进而提出使产品更加符合用户使用的要求。

可用性测试是用户在和系统（网站、软件应用程序、移动技术或任何用户操作的设备）交互时对用户体验质量的度量。可用性（Usability）是交互式 IT 产品/系统的重要质量指标，指的是产品对用户来说有效、易学、高效、好记、少错和令人满意的程度，即用户能否用产品完成他的任务，效率如何，主观感受怎样？实际上是从用户角度所看到的产品质量，是产品竞争力的核心。

答案：功能测试主要是黑盒测试，由测试人员进行，主要验证产品是否符合需求设计的要求；可用性测试主要是由用户（或者测试人员模拟用户行为）来进行的测试，主要是对产品的易用性进行测试，包括有效性（effectiveness）、效率（efficiency）和用户主观满意度（satisfaction）。其中有效性指用户完成特定任务和达到特定目标时所具有的正确和完整程度；效率指用户完成任务的正确和完整程度与所使用资源（如时间）之间的比率；满意度指用户在使用产品过程中所感受到的主观满意和接受程度。

15.3　黑盒测试

面试例题 1: A student needs to score at least 60 points to pass. If they score at least 80 points they will achieve a Merit and if they score 100 points they will achieve the Maximum. Which of the following would be the most likely set of values identified by the boundary Value Testing?[美国著名软件公司 M2009 年 12 月笔试题]

（一个学生需要 60 分才能及格；80 分就可以得优；100 分就是满分了。下面的 4 个选项中哪一个边界测试是最好的，为什么？）

A. -1,0,59,60,79,80,99,100

B. 0,59,79,100

C. 0,1,59,69,70,80,100

D. 60,80,100

解析: 边界值测试，就是找到边界，然后在边界及其边界附近（这里应该包括边界两侧）选点。因此边界 0（隐含需求边界），60，80，100 要测试，边界另一侧的-1，59，79，99 也要测试。对于选项 B、D，只覆盖了边界的一侧，而选项 C 中的 69 和 70 跟边界无关，所以 A 相对最好。

答案: A

扩展知识:

除边界值测试外，面试前最好了解这几个相关概念。

- 健壮性测试：健壮性测试是边界值分析的一种简单扩展。除了变量的 5 个边界值分析之外，还要分析变量值比最高值高出一点和比最低值低一点的情况下会出现什么反应。
- 最坏情况测试：边界值分析时是在单缺陷的假设下进行的。如果不做此假设，那么就会出现同时有多个变量取边界值的情况。最坏情况测试的测试用例的获取，是对每个变量，先进性包含 5 个边界值元素集合的测试，然后对这些集合进行笛卡儿积计算，以生成测试用例。
- 特殊值测试：这种测试不需要使用任何测试方针，只使用最佳工程判断。因此，该方法与测试人员的能力密切相关。
- 随机测试：这种方法不是永远选取有界变量的最小值、略高于最小值、正常值、略

低于最大值、最大值，而是使用随机数生成器生成测试用例值。这种测试用例的获取需要用程序来得出，而且还涉及测试覆盖率的问题。

面试例题 2: Function club is used to simulate guest in a club. With 0 guests initially and 50 as max occupancy, when guests beyond limitation, they need to wait outside;when some guests leave the waiting list will decrease. The function will print out number of guests in the club and waiting outside. The function declaration as follows:

```
void club(int x);
```

positive x stands for guests arrived, nagative x stands for guests left from within the club.（club 函数用来模拟一个俱乐部的顾客。初始化情况下是 0 个顾客，俱乐部最大规模只能有 50 个顾客，当用户超过了最大规模，他们必须等在外面。当一些顾客离开了等待队列将减少。这个 club 函数将打印在俱乐部里面的顾客人数，和外面的等待人数。函数声明如下：

```
void club(int x);
```

正数 x 代表客人来了，负数 x 代表客人离开了俱乐部。）

For example, club (40) prints 40,0; and then club (20) prints 50,10; and then club (-5) prints 50,5; and then club (-30) prints 25,0; and then club (-30) prints N/A; since it is impossible input.（举例而言，club (40)打印 40，0；接着 club (20)打印 50，10；接着 club (-5)打印 50，5；接着 club (-30)打印 25，0；接着 club (-30)打印 N/A；因为这是不可能实现的。）

To make sure this function works as defined, we have following set of data to pass into the function and check the result are correct.（为了确保函数工作正常，我们使用下列数据来测试函数是否正常，你认为该选哪个选项？）[美国著名软件公司 M2009 年 12 月笔试题]

```
a 60
b 20 50 -10
c 40 -30
d 60 -5 -10 -10 10
e 10 -20
f 30 10 10 10 -60
g 10 10 10
```

```
h 10 -10 10

A a d e g
B c d f g
C a c d h
D b d g h
E c d e f
```

解析: 本题实际上是考边界条件的测试情况。看有没有覆盖所有的边界条件。设 A 为已在俱乐部的成员，B 为排队的人。

```
        A                B
Case1   <=0               0
Case2   <50(Up/Down)       0
Case3   50                >0(Up/Down)
```

对于 a-h 各种情况：

```
a 60                    适用 C3
b 20 50 -10             适用 C2\C3
c 40 -30                适用 C2
d 60 -5 -10 -10 10      适用 C3\C2
```

```
e 10 -20                适用 C1
f 30 10 10 10 -60       适用 C1\C2\C3
g 10 10 10              适用 C2
h 10 -10 10             适用 C2
```

看看条件，肯定要包含 e，因为只有这个 case 能测 N/A 的情况，排除了 B C D 三项；再看 A 和 E，差别在 a 和 c、g 和 f 的选取上，很显然，d 包含 a，f 包含 g，所以排除 A，最终确定 E。

答案：E

扩展知识：本题的测试代码（C++）如下：

```
#include <iostream>
using namespace std;
#define MAX IN CUSTOM (50)
void club(int x)
{
    //这里必须使用静态变量
// 初始情况下，内部客人为 0 个
    static int in custom = 0;
// 初始情况下，外部客人为 0 个
    static int out custom = 0;
    if ( x > 0 ) // 来人的情况
    {
// 需要将人留在外面
        if     (     x+in custom     >
MAX IN CUSTOM )
        {
// 多于的人留在外面
            out custom += x+in custom -
MAX IN CUSTOM;
            in custom = MAX IN CUSTOM;
        }
        else
        {
          out custom = 0; in custom += x;
        }
    }
    else if(x < 0) // 走人
    {
        x = -x; // 转正
        // 如果走的人比内部与外部之和的人还
//要多，那么就出现错误了！
        // 在下面就表示为 in_custom < 0
// 外面人数更多（超过要走掉的人数）
        if ( out custom > x )
        {
            out custom -= x;
        }
        else
        {
            in custom-=x-out custom;
out custom = 0;
        }
    } // 打印输出结果
    if ( in custom < 0 )
    {
        std::cout << "N/A" << std::endl;
    }
    else
    {
        std::cout << in custom << "," <<
out custom << endl;
    }
}

int main()
{
    club(40);
    club(20);
    club(-5);
    club(-30);
    club(-30);
    return 0;
}
```

面试例题 3：int FindMiddle(int a ,int b,int　c)和 int CalMiddle(int a ,int b,int　c)分别为两个 C 函数，他们都号称能返回三个输入 int 中中间大小的那个 int.。你无法看到他们的源代码，那么该如何判断哪个函数的质量好？[中国台湾著名杀毒软件公司 Q2009 年 5 月笔试题]

答案：从编程习惯上看，笔者认为 int FindMiddle(int a ,int b,int　c)比较好，因为名字比较明确，就是找中间那一个数，让人一看就明白。

这道题是考软件测试的分析能力，比如一些特殊情况要特殊处理，例如：先测试 0 0 0 看

他们的测试结果，再测 0 0 1，再随便输入一些不是数字的数，测一下他们的排错功能，如果他们的结果一样，那就该测他们的算法效率。比如可以计算 10000 个数测试用时:

```
    System.out.println(new
Date().toString());
    for(int i=0; i < 10000; i++){
      for(int j=0; j < 10000; j++){
        for(int k=0; k < 10000; k++){
          FindMiddle(i, j, k);
        }
      }
    }
    System.out.println(newDate().toString
())
```

```
    System.out.println(newDate().toString
());
    for(int i=0; i < 10000; i++){
      for(int j=0; j < 10000; j++){
        for(int k=0; k < 10000; k++){
          CalMiddle(i, j, k);
        }
      }
    }
    System.out.println(newDate().toString
())
```

面试例题 4: Write a function bool symmetry - judge(int m) to judge whether the input parameter m is a symmetry number (eg 3, 66, 121, 3883, 45254)Please describe your algorithm and write code.（写一个函数测试输入数是否为回文数，给出算法和代码）[中国台湾著名杀毒软件公司 Q2009 年 5 月笔试题]

解析: 回文数问题 2006 年，2009 年都曾经考过。本题算法是建立数组，按位存储。比较首位和末位是否相同。如不同，则不是回文数。相同则继续比较，首位递增，末位递减直到首位不再小于末位。

答案: 完整代码如下:

```
    #include <iostream>
    using namespace std;

    int main()
    {


intj=10,k=12321,p,a[10],ss,i=0,begin,end;
       cout << "please input" << endl;
       cin >> k;
       p=k;
       while(p)
       {
          ss=p%10;
          a[i]=ss;
          p=p/10;
          i++;

       }
       begin = 0;
```

```
       end = i-1;

       while(begin < end)
       {
          if(a[begin]!=a[end])
            break;
          else
          { begin++; end--;}
       }
       if(begin < end)
       {cout << "jiade" << endl; }
       else
       {cout << "zhende " << endl;}

       cout << "i "  << i  << endl;
       cout << k;

       return 0;
    }
```

面试例题 5: Please write a test plan and test case the elevator functionalities（请写一个电梯功能的测试用例和测试方案。）[中国台湾著名杀毒软件公司 Q2009 年 5 月笔试题]

解析：电梯调度算法的基本原则就是如果在电梯运行方向上有人要使用电梯则继续往那个方向运动，如果电梯中的人还没有到达目的地则继续向原方向运动。此处将电梯一次从下到上视为一次运行（注意不一定是从底层到顶层），同理，电梯一次从上到下也视为一次运行（注意不一定是从顶层到底层）。

当电梯向上运行时：

```
1 位于当前层以下的向上请求都被忽略留到下次向上运行时处理
2 位于当前层以上的向上请求都被记录留到此次运行处理
3 无论何层的向下请求都被忽略留到下次向下运行时处理
```

当电梯向下运行时：

```
1 位于当前层以上的向下请求都被忽略留到下次向下运行时处理
2 位于当前层以下的向下请求都被记录留到此次运行处理
3 无论何层的向上请求都被忽略留到下次向上运行时处理
```

答案：如果只考虑单部电梯的实现情况是很简单的。电梯移动引发状态转换，电梯的主要状态有：

```
1 UpRun，电梯上行中
2 UpStopClose，电梯上行楼层暂停未开门
3 UpStopOpen，电梯上行楼层暂停已开门
4 UpRun,电梯下行中
5 UpStopClose，电梯下行楼层暂停未开门
6 UpStopOpen，电梯下行楼层暂停已开门
7 FullStopClose,电梯楼层停止未开门
8 FullStopOpen,电梯楼层停止未开门
```

电梯类，根据请求转换自身状态，请求分为：

```
1 梯内同向请求
2 前方楼层同向请求
3 前方楼层逆向请求
4 梯内逆向请求
5 后方楼层逆向请求
6 后方楼层同向请求
7 梯内关门请求
8 梯内开门请求
9 停止时同层同向请求
10 停止时同层逆向请求
```

多部电梯情况相对复杂，每部电梯是一个电梯类的实例，在电梯运行到楼层处时检查请求并进行状态转换，但这样的效果：

```
1 电梯只考虑局部状态没有全局观（只看一步），像苍蝇
2 时间一长电梯扎堆（抢任务），像蜜蜂
3 电梯经常空跑（任务被其他电梯完成），像蚂蚁
```

对多部电梯情况进行适当优化：

```
1 增加时间变量
2 增加相应配合（只派一部电梯）
3 电梯空闲时到重要地点（底层或顶层）
```

效果：

1 电梯行为过分依赖于梯内请求
2 电梯不适应突发性集中请求（层层停，没效果）
3 电梯上行的行为与下行不同（底层到各层，各层到底层，停层不是聚集点）
4 公平服务（先来先向服务）
5 反向快速响应（提前转向）
6 最长等待者的最小等待（后来者的跳跃型响应）
7 最少移动（资源最小调度）

更高级的算法是：

1 梯内人优先（判断梯内直达请求）
2 梯内人优先（减少无效开门，考虑超载直达）
3 不响应下行聚集型请求（减少无效开门，考虑超载直达）

甚至测试可以考虑电梯环境：

1 公寓型：只有底层（停车场）到各层或各层到底层，爆发型请求不多
2 写字楼型：有若干次爆发性请求（上下班，午休情况）需要测试，有邻层转移要求（同一公司租用）
3 酒店型：某些层（大堂，餐厅）有阶段持续请求
4 商场型：一般来说均匀调度

面试例题 6：Please design a test plan and test case to test a simplied mobile phone:（写一个测试用例和测试方案来测试手机）[中国台湾著名杀毒软件公司 Q2009 年 5 月笔试题]

```
including following function（包括如下功能）
Calling
SMS
Address book
```

答案：

对于 Calling：

1 是否有拨打电话这个功能，能否接通电话
2 拨打正常号码
3 拨打不正常号码，是否有相应提示

对于 SMS：

1 是否有发送、接收短信功能
2 输入正常号码发送短信
3 输入位数不对等不正常号码是否有提示
4 发送短信能否保存在发件箱，能否保存草稿，短信能否删除等

对于 Address book：

1 是否有电话本这个功能
2 新建一个联系人，联系人信息为空是否有提示
3 新建联系人与已有联系人姓名或电话号码等信息重复是否有提示
4 删除联系人是否成功，删除时是否有提示信息

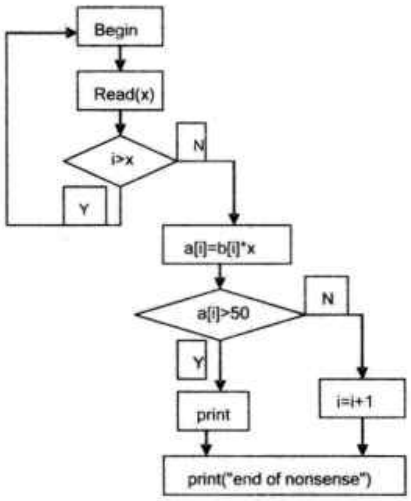
15.4　白盒测试

面试例题 1: The following is a pseudocode fragment that has no redeeming feature except for the purpose of this question.（下面语句是一段伪代码。）[美国某软件公司 2010 年 7 月笔试题]

```
module nonsense[]
/*a[] and b[] are global variables */
begin
int i,x
i=1
read(x)
while(i <x)do begin
a[i]=b[i]*x
if a[i]>50 then
print("array a is over the limit")
else
print("ok")
i=i+1
end
print("end of nonsense")
end
```

Please list test cases to cover all branches in this fragment.(请设计一个测试用例涵盖该段代码各个分支。)

答案: 白盒测试有几种测试方法: 条件覆盖、路径覆盖、语句覆盖、分支覆盖。其中分支覆盖又称判定覆盖，使得程序中每个判断的取真分支和取假分支至少经历一次，即判断的真假均曾被满足。本题目的逻辑分支如下图所示。



根据判定覆盖准则，测试用例涵盖该段代码各个分支如下:

```
1 i>x
2 i<x && a[i]>50
3 i<x && a[i]<50
4 i=x && a[i]>50
5 i=x && a[i]<50
```

面试例题 2: 下面是一道求闰年的 C 程序，这里要求你用基本路径法设计出测试用例。证明程序每一条可执行语句在测试过程中至少执行一次。[中国某互联网公司 2010 年 6 月笔试题]

```
int IsLeap(int year)
{
    if(year%4==0)                    //编号1
    {                                //编号2
        if(year%100==0)              //编号3
        {                            //编号4
          if(year%400==0)            //编号5
              leap=1;                //编号6
          else                       //编号7
              leap=0;                //编号8
        }                            //编号9
        Else                         //编号10
            leap=1;                  //编号11
    }else                            //编号12
        leap=0;                      //编号13
    return leap;                     //编号14
}
```

问题 1：该程序的控制流圆圈复杂度 V(G)是多少，独立线性路径数是多少？

问题 2：假设输入取值范围 1000<year<2001，请使用基本路径测试法为 year 设计测试用例，满足基本路径覆盖要求。（画设计流图及设计路径时可以利用上面的编号）

解析：白盒测试的测试方法有代码检查法、静态结构分析法、静态质量度量法、逻辑覆盖法、基本路径测试法、域测试、符号测试、Z 路径覆盖等方法，其中运用最为广泛的是基本路径测试法。

基本路径测试法是在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例的方法。设计出的测试用例要保证在测试中程序的每个可执行语句至少执行一次。

在程序控制流图的基础上，通过分析控制构造的环路复杂性，导出基本可执行路径集合，从而设计测试用例。包括以下 4 个步骤和一个工具方法。

（1）程序的控制流图：描述程序控制流的一种图示方法。

（2）程序圈复杂度：McCabe 复杂性度量。从程序的环路复杂性可导出程序基本路径集合中的独立路径条数，这是确定程序中每个可执行语句至少执行一次所必须的测试用例数目的上界。

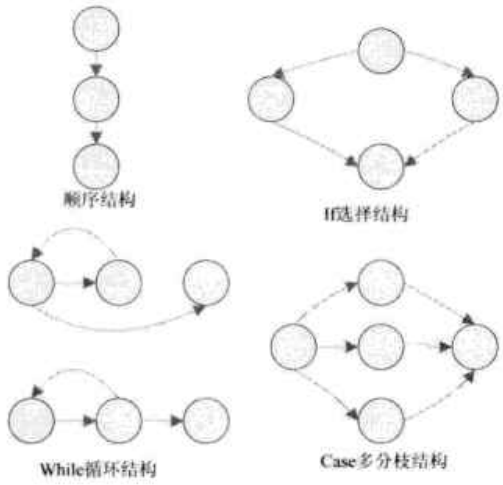
（3）导出测试用例：根据圈复杂度和程序结构设计用例数据输入和预期结果。

（4）准备测试用例：确保基本路径集中的每一条路径的执行。

工具方法：

- 图形矩阵：是在基本路径测试中起辅助作用的软件工具，利用它可以实现自动地确定一个基本路径集。
- 程序的控制流图：描述程序控制流的一种图示方法。

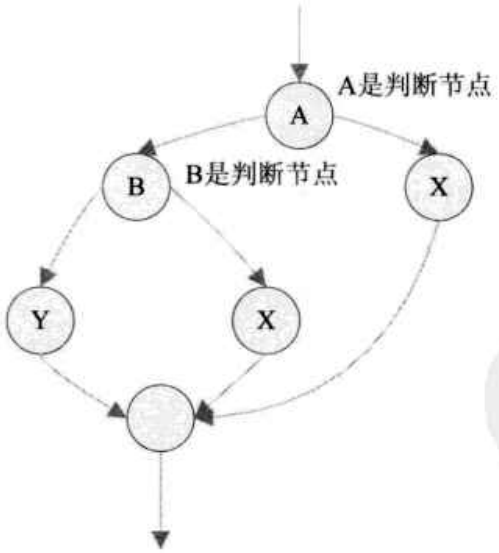
圆圈称为控制流图的一个节点，表示一个或多个无分支的语句或源程序语句，如下图所示。



例如:

```
1 if A or B
2 X
3 else
4 Y
```

对应的逻辑如下图所示。

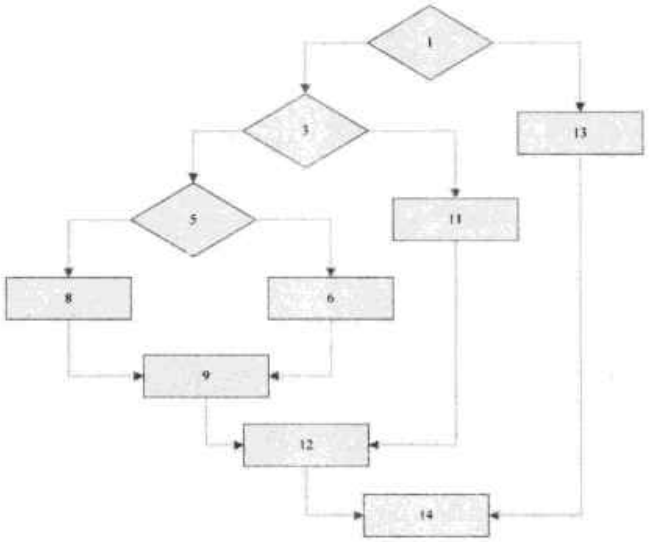


基本路径测试法的步骤:

第 1 步: 画出控制流图

流程图用来描述程序控制结构。可将流程图映射到一个相应的流图（假设流程图的菱形决定框中不包含复合条件）。在流图中，每一个圆，称为流图的节点，代表一个或多个语句。

一个处理方框序列和一个菱形决定框可被映射为一个节点，流图中的箭头，称为边或连接，代表控制流，类似于流程图中的箭头。一条边必须终止于一个节点，即使该节点并不代表任何语句（例如: if-else-then 结构）。由边和节点限定的范围称为区域。计算区域时应包括图外部的范围，如下图所示。



第 2 步: 计算圈复杂度

圈复杂度是一种为程序逻辑复杂性提供定量测度的软件度量，将该度量用于计算程序的基本的独立路径数目，为确保所有语句至少执行一次的测试数量的上界。独立路径必须包含一条在定义之前不曾用到的边。

有以下 3 种方法计算圈复杂度，

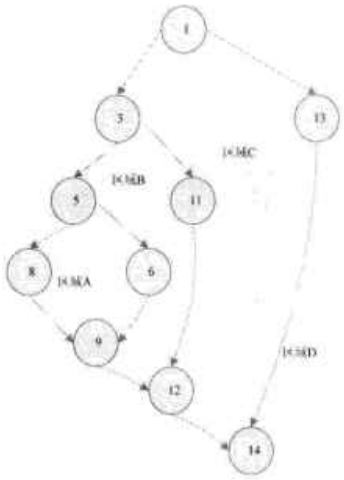
- 流图中区域的数量对应于环型的复杂性。
- 给定流图 G 的圈复杂度 V(G)，定义为 V(G)=E-N+2，E 是流图中边的数量，N 是流图中节点的数量。
- 给定流图 G 的圈复杂度 V(G)，定义为 V(G)=P+1，P 是流图 G 中判定节点的数量。

对应本题图中的圈复杂度，如下图所示，计算如下:

a. 流图中有 4 个区域。

b. V(G)=12 条边-10 节点+2=4。

c. V(G)=3 个判定节点+1=4。



第 3 步：导出测试用例

根据上面的计算方法，可得出 4 个独立的路径（(一条独立路径是指和其他的独立路径相比，至少引入一个新处理语句或一个新判断的程序通路。V(G)值正好等于该程序的独立路径的条数)。

路径 1：1-13-14

路径 2：1-3-11-12-14

路径 3：1-3-5-6-9-12-14

路径 4：1-3-5-8-9-12-14

根据上面的独立路径，设计输入数据，使程序分别执行到上面 4 条路径。[中国某互联网公司 2010 年 6 月笔试题]

答案：该程序的控制流圆圈复杂度 V(G)是 4，独立线性路径数是 4。

为了确保基本路径集中的每一条路径的执行，根据判断节点给出的条件，选择适当的数据以保证某一条路径可以被测试到，满足上面例子基本路径集的测试用例如下：

```
路径 1：1-13-14
输入数据：1001
预期结果：leap=0;

路径 2：1-3-11-12-14
输入数据：1004
预期结果：leap=1;

路径 3：1-3-5-6-9-12-14
输入数据：1100
预期结果：leap=0;

路径 4：1-3-5-8-9-12-14
输入数据：1600
预期结果：leap=1;
```