《软件测试技术》

实验指导书

**王勇、刘进**

安徽工程大学计算机与信息学院

2018年09月

# 前言

本实验指导书是《软件测试》课程的实验指导书，目的是要求学生通过上机实践，能够初步具备对小型软件系统进行测试的能力，包括掌握基本的白盒测试技术和黑盒测试技术，学会编写测试用例，养成良好的代码编写风格，初步了解缺陷报告的撰写。

本实验设计了**5个设计性实验和1个综合性实验**，实验包括图覆盖测试、数据流覆盖测试、谓词覆盖测试、输入域划分、因果图法和一个综合实验，共计六个。针对学生的实际情况，实验内容和实验题目的设计做到难易适中，验证性实验、设计性实验和综合性实验分别在实验要求上分成不同的层次，力争让学生经过一定的努力，都能够完成相应的题目，从而激发学生的学习兴趣和积极性。

本指导书主要用于《软件测试》课程的实验指导，可供计算机科学与技术专业用。

# 目 录

[前言 2](#_Toc516044390)

[目 录 3](#_Toc516044391)

[第一章 实验基础知识 5](#_Toc516044392)

[1.1 软件测试概述 5](#_Toc516044393)

[1.1.1 软件测试 5](#_Toc516044394)

[1.1.2 测试过程 5](#_Toc516044395)

[1.1.3 测试方法 5](#_Toc516044396)

[1.1.4 测试工具 5](#_Toc516044397)

[1.2 测试用例的编写 5](#_Toc516044398)

[1.2.1 测试用例 5](#_Toc516044399)

[1.2.2 软件缺陷分类 6](#_Toc516044400)

[1.2.3 测试用例的选择 6](#_Toc516044401)

[第二章 实验要求 7](#_Toc516044402)

[2.1实验总体要求 7](#_Toc516044403)

[2.1.1 实验目的和要求 7](#_Toc516044404)

[2.1.2 实验内容安排 7](#_Toc516044405)

[2.1.3 实验过程要求 7](#_Toc516044406)

[2.2 实验报告要求 7](#_Toc516044407)

[2.3 实验成绩评价 8](#_Toc516044408)

[2.3.1 实验成绩结构和比例 8](#_Toc516044409)

[2.3.2 考核方式 8](#_Toc516044410)

[第三章 实验内容和指导 9](#_Toc516044411)

[3.1 图覆盖测试 9](#_Toc516044412)

[3.1.1.实验类型 9](#_Toc516044413)

[3.1.2 实验目的 9](#_Toc516044414)

[3.1.3 实验内容 9](#_Toc516044415)

[3.1.4 实验步骤 10](#_Toc516044416)

[3.1.5 实验要求 10](#_Toc516044417)

[3.1.6 实验思考 10](#_Toc516044418)

[3.2 数据流覆盖测试 15](#_Toc516044419)

[3.2.1 实验类型 15](#_Toc516044420)

[3.2.2 实验目的 15](#_Toc516044421)

[3.2.3 实验内容 15](#_Toc516044422)

[3.2.4 实验步骤 16](#_Toc516044423)

[3.2.5 实验要求 16](#_Toc516044424)

[3.2.6 实验思考 16](#_Toc516044425)

[3.3 谓词覆盖测试 20](#_Toc516044426)

[3.3.1 实验类型 20](#_Toc516044427)

[3.3.2 实验目的 20](#_Toc516044428)

[3.3.3 实验内容 20](#_Toc516044429)

[3.3.4 实验步骤 21](#_Toc516044430)

[3.3.5 实验要求 24](#_Toc516044431)

[3.3.6 实验思考 24](#_Toc516044432)

[3.4 输入域划分 26](#_Toc516044433)

[3.4.1 实验类型 26](#_Toc516044434)

[3.4.2 实验目的 26](#_Toc516044435)

[3.4.3 实验内容 26](#_Toc516044436)

[3.4.4 实验步骤 26](#_Toc516044437)

[3.4.5 实验要求 29](#_Toc516044438)

[3.4.6 实验思考 29](#_Toc516044439)

[3.5 因果图法 32](#_Toc516044440)

[3.5.1 实验类型 32](#_Toc516044441)

[3.5.2 实验目的 32](#_Toc516044442)

[3.5.3 实验内容 32](#_Toc516044443)

[3.5.4 实验步骤 32](#_Toc516044444)

[3.5.5 实验要求 34](#_Toc516044445)

[3.5.6 实验思考 34](#_Toc516044446)

[3.6 综合实验 40](#_Toc516044447)

[3.6.1 实验类型 40](#_Toc516044448)

[3.6.2 实验目的 40](#_Toc516044449)

[3.6.3 实验内容 40](#_Toc516044450)

[3.6.4 实验步骤 41](#_Toc516044451)

[3.6.5 实验要求 42](#_Toc516044452)

[3.6.6 实验思考 42](#_Toc516044453)

[附录A：通用代码审查清单 43](#_Toc516044454)

[附录B：程序代码审查表 44](#_Toc516044455)

[附录C：测试用例设计表 45](#_Toc516044456)

[附录D：缺陷报告 45](#_Toc516044457)

# 第一章 实验基础知识

## 1.1 软件测试概述

### 1.1.1 软件测试

软件测试就是在软件投入运行前，对软件需求分析、设计规格说明和编码的最终复审，是软件质量保证的关键步骤。 软件测试是为了发现错误而执行程序的过程。或者说，软件测试是根据软件开发各阶段的规格说明和程序的内部结构而精心设计一批测试用例（即输入数据及其预期的输出结果），并利用这些测试用例去运行程序，以发现程序错误的过程。

### 1.1.2 测试过程

为了保证测试的质量，将测试过程分成几个阶段，即：代码审查、单元测试、集成测试、系统测试和验收测试。代码会审由一组人通过阅读、讨论和争议对程序进行静态分析的过程。单元测试集中在检查软件设计的最小单位——模块上，通过测试发现实现该模块的实际功能与定义该模块的功能说明不符合的情况，以及编码的错误。集成测试是将模块按照设计要求组装起来同时进行测试，主要目标是发现与接口有关的问题。系统测试是测试整个系统，以证实它满足“需求规格说明书”所规定的功能、质量和性能等方面的特性。验收测试的目的是向未来的用户表明系统能够像预定要求那样工作。与系统测试非常相似，主要区别是测试人员不同，验收测试由用户执行。

### 1.1.3 测试方法

软件测试的方法分为功能性测试和结构性测试。功能测试是指在对程序进行功能抽象的基础上，将程序划分成功能单元，然后在数据抽象的基础上，对每个功能单元生成测试数据进行测试。进行功能测试时，被测程序被当作打不开的黑盒，因而无法了解其内部构造，因此又称为黑盒测试。

结构性测试是知道产品内部工作过程，检测产品内部动作是否按照规格说明书的规定正常进行。它允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。此方法把测试对象看作一个透明的盒子，又叫白盒测试。

### 1.1.4 测试工具

软件测试的工作量很大（据统计，会用到40% 的开发时间；一些可靠性要求非常高的软件，测试时间甚至占到总开发时间的60% ），但测试却是在整个软件过程中极有可能应用计算机进行自动化的工作，原因是测试的许多操作是重复性的、非智力创造性的、需求细致注意力的工作。测试工具的应用已经成为了普遍的趋势。测试工具一般可分为白盒测试工具、黑盒测试工具、性能测试工具，另外还有用于测试管理（包括测试流程管理、缺陷跟踪管理、测试用例管理）的工具。

## 1.2 测试用例的编写

### 1.2.1 测试用例

软件测试的本质是针对要测试的内容确定一组测试用例。测试用例是为实施一次测试而向被测系统提供的输入数据、操作或各种环境设置。

测试用例应该包含基本的内容有输入和预期输出，输入实际有两种类型：前提（在测试用例执行前已经存在的环境）和由某种测试方法所标识的实际输入。预期输出也有两类：后果和实际输出。测试活动要建立必要的前提条件，提供测试用例输入、观测输出、然后将这些输出与预期输出进行比较，以确定该测试是否通过。

有两种基本方法可以用来标识测试用例，即功能性测试和结构性测试。功能性测试的基本观点是，任何程序都可以看作是将从定义域取值映射到输出值域的函数。这种观点常常在工程中使用，将系统看作是黑盒。采用功能性方法标识测试用例，所使用的唯一信息就是软件的规格说明。功能性测试用例具有两个显著的优点：（1）功能性测试与软件如何实现无关，所以如果实现发生变化，测试用例仍然有用；（2）测试用例开发可以与实现并行进行，因此可缩短总的开发时间。在缺点方面，功能性测试用例也常常带来两个问题：测试用例可能存在严重的冗余，此外可能还会有未测试的软件漏洞。功能性测试的主流方法主要有：边界值分析、健壮性分析、最坏情况分析、特殊值测试、输入等价类、输出等价类和基于决策树的测试。

结构性测试有时也叫白盒测试。结构性测试是知道软件产品内部工作过程，检测软件产品内部动作是否按照规格说明书的规定正常进行。结构性测试需要全面了解程序内部逻辑结构、对所有逻辑路径进行测试。结构性测试是穷举路径测试，并力求提高测试覆盖率。结构性测试的主要方法有：逻辑覆盖测试、基路径测试、数据流测试等方法。在实际应用中，为全面的测试软件产品，一般将结构性测试和功能性测试结合起来使用。

### 1.2.2 软件缺陷分类

有多种方法可以对缺陷分类：以出现相应错误的开发阶段来划分、以相应失效产生的后果来划分、以解决难度来划分、以不解决难度会产生的风险来划分等等。在日常的软件测试中，通常给出的缺陷是根据缺陷后果的严重程度来进行划分，在实际应用中可以根据具体情况对严重程度来划分不同的等级。

### 1.2.3 测试用例的选择

选择测试用例是软件测试员最重要的一项任务，不正确的选择可能导致测试量过大或过小，甚至测试目标不对。从工程实践的角度讲，测试用例有几条基本准则：

（1）测试用例的代表性：能够代表各种合理和不合理的、合法的和非法的、边界和越界的,以及 极限的输入数据、操作和环境设置等；

（2）测试结果的可判定性：即测试执行结果的正确性是可判定的或可评估的；

（3）测试结果的可再现性：即对同样的测试用例,系统的执行结果应当是相同的。

# 第二章 实验要求

## 2.1实验总体要求

### 2.1.1 实验目的和要求

本实验为配合课程《软件测试》的理论而开设，其中包括验证型、设计型和综合型实验。本实验着眼于理论与应用的结合，注重培养学生软件测试的实际动手能力，增强软件工程项目的质量管理意识。通过实验教学，使学生掌握软件测试的主要方法和技术。

本实验课程要求学生完成适当的上机实验，并编写相应的实验报告。验证型题目侧重于静态测试，使学生理解清晰完整的项目文档以及逻辑严谨的程序代码对于软件质量的重要性。设计型题目使学生掌握软件测试的基本理论和方法，培养基本的应用能力。综合型题目在于提高学生分析问题、解决问题的能力，培养工程项目的测试能力和管理能力。

### 2.1.2 实验内容安排

实验内容安排如表2-1所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验编号** | **实验性质** | **实验要求** | **实验项目名称** | **学时** | **备注** |
| 01 | 设计型 | 必修 | 图覆盖测试 | 4 |  |
| 02 | 设计型 | 必修 | 数据流覆盖测试 | 2 |  |
| 03 | 设计型 | 必修 | 谓词覆盖测试 | 4 |  |
| 04 | 设计型 | 必修 | 输入域划分 | 4 |  |
| 05 | 设计型 | 必修 | 因果图法 | 4 |  |
| 06 | 综合性 | 必修 | 综合实验 | 6 |  |

### 2.1.3 实验过程要求

实验前要充分做好准备工作：

1. 复习和掌握与本实验有关的知识内容。
2. 预习、思考实验内容。
3. 对实验内容进行分析和设计。

实验过程中，实验者必须服从指导教师和实验室工作人员的安排，遵守纪律和实验制度，爱护设备和卫生，在指定的实验时间内，必须到实验室做实验。

对于上机过程中出现的问题，尽量先独立思考和解决；对于难以解决的问题可以和同学交流或询问老师；对于同一实验题目，可以考虑多种方法实现，然后比较并选择一种较为有效的方法来实验。

对于每个实验，要求学生在实验前组成测试小组，每个小组最多不超过6人，由组长来组织小组成员，共同完成实验内容。

## 2.2 实验报告要求

实验后，应及时整理出实验报告，实验报告提交纸质文档，[并在规定的时间内提交到指导教师处。](mailto:并在规定的时间前发送至指定的电子邮箱testreport_bistu@163.com)每个实验报告的具体内容见各个实验的实验要求。按照学校统一格式编写实验报告。

## 2.3 实验成绩评价

### 2.3.1 实验成绩结构和比例

实验课程的总成绩由实验考勤和实验报告成绩组成，其中实验考勤占10%，实验过程占50%，实验报告占40%。

每一个实验的分值分配为：实验一15分，实验二10分，实验三15分，实验四20分，实验五20分，实验六20分。

### 2.3.2 考核方式

实验考核主要是三方面：

1．上机实验考勤。每次上机应主动到指导教师处签到，冒签等不诚信行为一旦被发现，取消该次实验成绩。累计缺席实验课4次(含4次)以上者（共6次），期末实验成绩为0分。

如果学生需要请假，必须提前出具正式假条（需要班主任或辅导员签字），不接受事后假条（如有特殊情况，需要有辅导员或班主任的情况说明）。

2．实验过程考核。每次实验课，指导教师根据实验过程情况，随机抽查5~10名同学，进行当场实验答辩。

3．实验报告提交。应根据实验指导教师要求按时提交每次实验的数据。如果发生学生抄袭、伪造实验数据，或实验报告和设计报告抄袭、雷同，则涉及的所有学生的该课程实验成绩记为0分。

# 第三章 实验内容和指导

## 3.1 图覆盖测试

### 3.1.1.实验类型

实验类型为设计型，4个学时。

### 3.1.2 实验目的

（1）熟练掌握白盒法测试的逻辑覆盖法，并能进行实际程序测试；

（2）掌握分支覆盖和测试流的测试方法，并能进行实际程序测试；

（3）对测试用例进行优化。

（4）熟悉各种图覆盖标准的测试方法

### 3.1.3 实验内容

**题目一：已知计算数组中0的个数的源程序如下：**

public static int numZero (int[] x)

{

int count = 0;

for (int i = 1; i < x.length; i++)

{

if (x[i] == 0) count++;

}

return count;

}

1. 利用语句覆盖准则，设计测试用例进行测试；
2. 利用分支覆盖准则，设计测试用例进行测试；
3. 利用基本路径覆盖准则，设计测试用例进行测试；
4. 计算各自覆盖充分性，并讨论bug。

**题目二 折半查找函数**

该函数能在元素呈升序排列的数组中查找值为key的元素。

**题目三、计算生日是星期几**

已知公元1年1日是星期一。编写程序，只要输入年月日，就能回答那天是星期几。

1. 编写程序；
2. 画出流程图及流图；
3. 找出所有的主路径；
4. 覆盖所有路径，设计测试用例。

### 3.1.4 实验步骤

（1）对“题目一”，在小组中进行代码走查，每个小组成员分别介绍自己的程序代码，小组成员依据《通用代码检查清单》（参见附录A）检查小组成员的代码，对所发现的代码问题填写《程序代码审查表》（参见附录B）。

（2）根据白盒测试技术设计“题目二、三”的测试用例，主要考虑图覆盖（比如节点覆盖，边覆盖，边对覆盖，主路径覆盖，基本路径覆盖）。

（3）编写实验报告。

### 3.1.5 实验要求

（1）根据“题目一”的要求找出小组成员程序代码的问题，编写《程序代码审查表》。

（2）根据“题目二、三”的要求编写测试用例（参考附件C）。

（3）编写实验报告。实验内容要求对每个题目分别说明。

* 对于“题目一”，应针对每个成员的程序分别进行报告（每个小组成员依据代码走查的结果负责编写另一个成员的《程序代码审查表》，而不是写自己程序代码的《程序代码审查表》）。实验内容的“测试用例编写”部分换成《程序代码审查表》，“测试结果分析”部分应说明依据《通用代码检查清单》所发现的各类问题的数量，并将审查的程序代码作为实验报告的附件一起提交。如果未发现“题目一”中的任何问题，可省略《程序代码审查表》部分，直接在“测试结果分析”部分给予说明。
* 对于“题目二、三”，实验内容的“测试用例编写”部分换为《测试用例设计表》，“测试结果分析”可省略。

### 3.1.6 实验思考

**（1）在实际的工作中，程序开发人员是否需要对自己的程序进行测试？作为程序开发人员，如何使自己的程序代码更易于理解、可读性强？**

**答**：程序员需要对自己的程序进行测试。对于如何使自己的程序代码更易于理解、可读性强，我认为程序员在编写程序时，应使程序具有良好的风格，力图从编码的角度提高程序的可读性，改善程序的质量。而程序设计风格包括4个方面：

***a.*** 源程序文档化。在源程序可包含一些内部文档，以帮助阅读和理解程序。在源程序的内部文档中包括：标识符的命名、注解和程序的视觉组织。

1、命名标识符时，要注意以下问题：

* 这些名字应能反映他所代表的实体，应有一定的实际意义；
* 应当选择精炼的意义明确的名字；
* 必要时可使用缩写名字，但缩写规则要一致，并且要给每一个缩写名字加注释；
* 不用关键字做标识符；
* 同一个名字不要有多个含义；
* 不用相似的名字，相似的名字易混淆，不易发现错误；
* 名字中避免使用易混淆的字符。

2、程序的注释在编写程序过程中程序员应当适当给予注释。注释是用来帮助人们理解程序，绝不是可与可无。注释分为序言性注释和功能性注释，序言性注释常置于每个程序模块的开头部分，而功能性注释则通常嵌在程序体内，主要描述程序段的功能。

3、视觉组织。通过在程序中使用空格、空行和缩进等技巧，帮助人们从视觉上看清程序的结构。

***b.*** 数据说明。为了使程序中的数据说明更易于理解和维护，可采用以下风格:显示说明一切变量；数据说明的次序规范化；说明语句中变量安排有序化；使用注释说明复杂的数据结构。

***c.*** 语句结构。程序编写做到清晰第一，效率第二。对此有以下几条常见规则：

1、一行内只写一条语句；

2、首先考虑清晰性；

3、直截了当说明程序员的用意；

4、让编译程序做简单的优化；

5、尽可能使用库函数；

6、避免不必要的转移；

7、尽量只采用3种基本控制结构来编写程序。

***d.*** 输入和输出。输入输出的方式和格式应该尽可能方便用户的使用

**（2）在进行测试设计时，如何考虑测试用例的充分性和减少测试用例的冗余性。**

**答：**在编写测试用例的时候，要充分熟悉需求，并且了解软件的架构设计，各个模块的设计。精心选择测试数据来达到对需求的一个较好的覆盖率。在实际测试过程中，会因软件的特性，可能各模块之间有所关联，此时可以安排好测试用例的执行顺序，以达到测试数据的重用。

**附源程序和测试框架：**

**程序1源代码：**

public class NumZero

{

/\*\*

\* Counts zeroes in an array

\*

\* @param x array to count zeroes in

\* @return number of occurrences of 0 in x

\* @throws NullPointerException if x is null

\*/

public static int numZero (int[] x)

{

int count = 0;

// As example in the book points out, this loop should start at 0.

// Better yet, is should be a foreach loop,

// which eliminates the possibility of the fencepost fault:

// for (int i:x) { if (x==0) count++; }

for (int i = 1; i < x.length; i++)

{

if (x[i] == 0) count++;

}

return count;

}

public static void main (String []argv)

{ // Driver method for numZero

// Read an array from standard input, call numZero()

int []inArr = new int [argv.length];

if (argv.length == 0)

{

System.out.println ("Usage: java NumZero v1 [v2] [v3] ... ");

return;

}

for (int i = 0; i< argv.length; i++)

{

try

{

inArr [i] = Integer.parseInt (argv[i]);

}

catch (NumberFormatException e)

{

System.out.println ("Entry must be a integer, using 1.");

inArr [i] = 1;

}

}

System.out.println ("Number of zeros is: " + numZero (inArr));

}

}

**程序1测试框架：**

import static org.junit.Assert.\*;

import org.junit.\*;

import java.util.\*;

public class NumZeroTest

{

// this test passes

@Test public void nonZeroFirstElement()

{

int arr[] = {2, 7, 0};

assertEquals("Nonzero in first element", 1, NumZero.numZero(arr));

}

// this test fails!

@Test public void zeroFirstElement()

{

int arr[] = {0, 7, 2};

assertEquals("Zero in first element", 1, NumZero.numZero(arr));

}

}

**程序2源程序**

public class BinarySearch {

static public int binarySearch(int a[],int goal){

int high=a.length-1;

int low=0;

while (low<=high){

int middle=(low+high)/2;

if(a[middle]==goal){

return middle;

}

else if(a[middle]>goal){

high=middle-1;

}

else{

low=middle+1;

}

}

return -1;

}

}

**程序2测试框架：**

import static org.junit.Assert.\*;

import org.junit.Test;

public class BinarySearchTest {

@Test

public void testBinarySearch1() {

int input[]={1,2,5,3,6,9};

int goal = 3;

int expected = 3;

assertTrue("error",expected==BinarySearch.binarySearch(input, goal));

}

@Test

public void testBinarySearch2() {

int input[]={};

int goal = 3;

int expected = -1;

assertTrue("error",expected==BinarySearch.binarySearch(input, goal));

}

@Test

public void testBinarySearch3() {

int input[]={3,6};

int goal = 6;

int expected = 1;

assertTrue("error",expected==BinarySearch.binarySearch(input, goal));

}

}

## 3.2 数据流覆盖测试

### 3.2.1 实验类型

实验类型为设计性，2个学时。

### 3.2.2 实验目的

（1）掌握常见的数据流覆盖标准

（2）熟悉各种数据流测试方法

### 3.2.3 实验内容

**读懂如下程序：**

public static int patternIndex (String subject, String pattern)

{

final int NOTFOUND = -1;

int iSub = 0, rtnIndex = NOTFOUND;

boolean isPat = false;

int subjectLen = subject.length();

int patternLen = pattern.length();

while (isPat == false && iSub + patternLen - 1 < subjectLen)

{

if (subject.charAt(iSub) == pattern.charAt(0))

{

rtnIndex = iSub; // Starting at zero

isPat = true;

for (int iPat = 1; iPat < patternLen; iPat ++)

{

if (subject.charAt(iSub + iPat) != pattern.charAt(iPat))

{

rtnIndex = NOTFOUND;

isPat = false;

break; // out of for loop

}

}

}

iSub ++;

}

return (rtnIndex);

}

1. 画出程序流程图
2. 给出程序中的def和use集合
3. 利用all-Defs Coverage、all-Uses Coverage和all-du-Path coverage覆盖准则设计测试用例。
4. 注入相关的bug，给出测试结果。

### 3.2.4 实验步骤

（1）在小组中进行代码走查，画出程序流程图；

（2）根据白盒测试技术设计测试用例，主要考虑数据流覆盖测试；

（3）利用不同的覆盖准则设计测试用例；

（4）编写实验报告。

### 3.2.5 实验要求

（1）掌握数据流测试方法，并能进行实际程序测试。

（2）根据所设计的测试用例进行实际的测试。

（3）采用小组成员互测方式，即每个小组成员对小组其他成员的程序进行报告。

（4）编写实验报告。

* 上述题目实验内容的“测试用例编写”为测试小组其他成员程序的《测试用例设计表》，“测试结果分析”部分应换成《缺陷报告》。

### 3.2.6 实验思考

（1）在实际的测试中，如何利用DD路径表述定义使用路径的静态与动态二义性？

（2）数据流覆盖准则有哪些？

ADC：全定义覆盖，所有定义的地方都覆盖过

AUC： 全引用覆盖，所有引用的地方都覆盖过，包括所有的定义也都覆盖过

ADUPC：全部定义引用覆盖，所有的定义和引用之间的路径都覆盖过

**附源程序和测试框架：**

**源程序**

public class PatternIndex

{

public static void main (String[] argv)

{

if (argv.length != 2)

{

System.out.println

("java PatternIndex Subject Pattern");

return;

}

String subject = argv[0];

String pattern = argv[1];

int n = 0;

if ((n = patternIndex (subject, pattern)) == -1)

System.out.println

("Pattern string is not a substring of the subject string");

else

System.out.println

("Pattern string begins at character " + n);

}

/\*\*

\* Find index of pattern in subject string

\*

\* @param subject String to search

\* @param pattern String to find

\* @return index (zero-based) of first occurrence of pattern in subject; -1 if not found

\* @throws NullPointerException if subject or pattern is null

\*/

public static int patternIndex (String subject, String pattern)

{

final int NOTFOUND = -1;

int iSub = 0, rtnIndex = NOTFOUND;

boolean isPat = false;

int subjectLen = subject.length();

int patternLen = pattern.length();

while (isPat == false && iSub + patternLen - 1 < subjectLen)

{

if (subject.charAt(iSub) == pattern.charAt(0))

{

rtnIndex = iSub; // Starting at zero

isPat = true;

for (int iPat = 1; iPat < patternLen; iPat ++)

{

if (subject.charAt(iSub + iPat) != pattern.charAt(iPat))

{

rtnIndex = NOTFOUND;

isPat = false;

break; // out of for loop

}

}

}

iSub ++;

}

return (rtnIndex);

}

}

**测试框架**

import org.junit.\*;

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

import org.junit.runners.Parameterized.Parameters;

import static org.junit.Assert.\*;

import java.util.\*;

@RunWith (Parameterized.class)

public class DataDrivenPatternIndexTest

{

private String subject;

private String pattern;

private int expected;

public DataDrivenPatternIndexTest (String subject, String pattern, int expected)

{

this.subject = subject;

this.pattern = pattern;

this.expected = expected;

}

// The following values are from the 1st edition, Table 2.4, page 59

// Need an update to the new edition table number and page number.

@Parameters

public static Collection<Object[]> parameters()

{

return Arrays.asList (new Object [][] {

{"a", "bc", -1},

{"ab", "a", 0},

{"ab", "ab", 0},

{"ab", "ac", -1},

{"ab", "b", 1},

{"ab", "c", -1},

{"abc", "abc", 0},

{"abc", "abd", -1},

{"abc", "ac", -1},

{"abc", "ba", -1},

{"abc", "bc", 1}

});

}

@Test

public void dataFlowTests()

{

assertEquals ("Data flow PatternIndex test",

PatternIndex.patternIndex (subject, pattern), expected);

}

}

## 3.3 谓词覆盖测试

### 3.3.1 实验类型

实验类型为设计型，4个学时。

### 3.3.2 实验目的

（1）掌握常见的图逻辑覆盖标准（谓词覆盖、子句覆盖、组合覆盖及相关的有效子句覆盖等）

（2）掌握逻辑覆盖常见方法和步骤

### 3.3.3 实验内容

**题目一：使用逻辑覆盖测试以下程序段**

public static int doWork(int x, int y, int z)

{

float k=0;

int j=0;

if(x>3&&z<10)

{

k=x\*y-1;

j = (int)(Math.sqrt(k));

}

if(x==4||y>5)

{

j=x\*y+10;

j=j%3;

}

return j;

}

1. 画出程序的流程图及流图；
2. 分别注入bug1和bug2；
3. 分别以谓词覆盖、子句覆盖、组合覆盖及相关的有效子句覆盖等方法设计测试用例，并写出每个测试用例的测试路径。

**实验1：bug1谓词测试**

对于包含bug1的程序存在谓词：

P1 = x>4&&z<10

P2 = x==4||y>5

使用谓词覆盖：

测试需求：（1）P1=T；（2）P1=F；（3）P2=T；（4）P2=F

选择输入实参x，y，z的值，覆盖以上四个测试需求：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | 测试数据 | 覆盖测试需求 | 预期结果 | 实际结果 | 测试结果 |
| 1 | x=5，y=10，z=5 | （1），（3） |  |  | pass |
| 2 | x=3，y=4，z=5 | （2），（4） |  |  | pass |

思考为什么？

如何才能测试出bug？

**实验2：bug2谓词测试**

对于包含bug1的程序存在谓词：

P1 = x>3&&z<10

P2 = x!=4||y>5

使用谓词覆盖：

测试需求：（1）P1=T；（2）P1=F；（3）P2=T；（4）P2=F

选择输入实参x，y，z的值，覆盖以上四个测试需求：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | 测试数据 | 覆盖测试需求 | 预期结果 | 实际结果 | 测试结果 |
| 1 | x=5，y=10，z=5 | （1），（3） |  |  | pass |
| 2 | x=3，y=3，z=5 | （2），（3） |  |  | fail |
| 3 | x=4，y=4,z=1 | （1），（4） |  |  | fail |

思考为什么？

### 3.3.4 实验步骤

（1）画出函数doWork的程序流程图，分析该段代码包含的基本逻辑判定条件和执行路径。

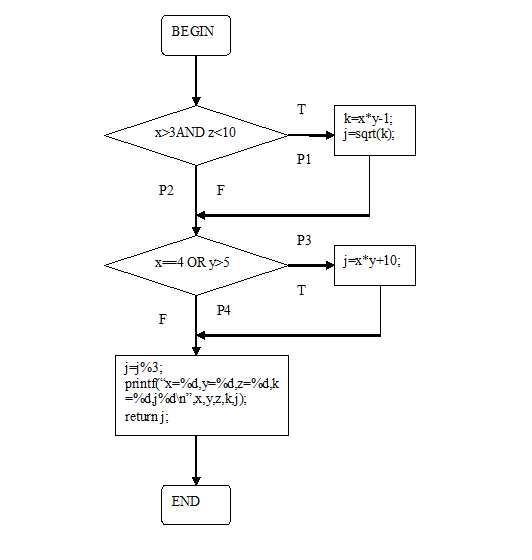


图3.3.1 程序流程图

（2）根据白盒测试技术设计测试用例，主要考虑逻辑覆盖测试（语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、修正的判定/条件覆盖），计算测试用例的语句覆盖率、判定覆盖率和条件覆盖率等测试管理指标。

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **覆盖条件取值** | **判定条件取值** | **具体条件取值** |
| 1 | T1, T2 | M取Y | x>3,z<10 |
| 2 | T1, F2 | M取N | x>3,z>=10 |
| 3 | F1, T2 | M取N | x<=3,z<10 |
| 4 | F1, F2 | M取N | x<=3,z>=10 |
| 5 | T3, T4 | Q取Y | x==4,y>5 |
| 6 | T3, F4 | Q取Y | x==4,y<=5 |
| 7 | F3, T4 | Q取Y | x!=4,y>5 |
| 8 | F3, F4 | Q取N | x!=4,y<=5 |
| 注： M: x>3 AND z<10 | | N: x==4 OR y>5 | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 测试用例 ID | 输入条件 | | | | | 预期输出 | 通过路径 |
| x | y | z | k | j | j |  |
| 1 | 4 | 6 | 9 | 0 | 0 | 1 | L13 |
| 2 | 3 | 5 | 11 | 0 | 0 | 2 | L24 |
| 3 | 4 | 4 | 11 | 0 | 0 | 2 | L23 |
| 4 | 3 | 6 | 9 | 0 | 0 | 1 | L23 |

备注：1.2判定覆盖

语句覆盖：1； 判定覆盖：1，2； 判定/条件覆盖：1，2，3，4； 修正覆盖：

（3）编写测试程序,运行测试程序并记录测试结果。（给出运行结果界面）

程序代码：

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// 定义结构来获取测试用例的输入

struct strInput{

int x;

int y;

int z;

int k;

int j;

}strIn;

int DoWork (int x,int y,int z,int k,int j)

{

if ( (x>3)&&(z<10) ){

k=x\*y-1;

j=sqrt(k);

}

if((x==4)||(y>5))

j=x\*y+10;

j=j%3;

printf("x=%d,y=%d,z=%d,k=%d,j=%d\n",x,y,z,k,j);

return j;

}

void Driveroffunc()

{

// 设置局部变量

int tcPassNum = 0, tcFailNum = 0; // 存储通过和失败的测试用例总数

int i;

printf( "这是对DoWork的测试\n" );

// 读取测试用例的所有输入数据

struct strInput tcInput[] = { {4, 6, 9, 0, 0},

{3, 5, 11, 0, 0},

{4, 4, 11, 0, 0},

{3, 6, 9, 0, 0}

};

int tcOutcome[] = { 1, 0, 2, 1 }; // 读取测试用例的预期输出

int actualOutcome = 0; // 存储测试用例的实际执行结果

for(i = 0; i < sizeof(tcOutcome) / sizeof(tcOutcome[0]); i++)

{

printf( "\n第%d个测试用例，输入为x=%d, y=%d, z=%d, k=%d, j=%d 预期输出x=%d\n",

i+1,

tcInput[i].x,

tcInput[i].y,

tcInput[i].z,

tcInput[i].k,

tcInput[i].j,

tcOutcome[i]

);

printf( "实际执行情况如下:\n" );

actualOutcome = DoWork( tcInput[i].x,

tcInput[i].y,

tcInput[i].z,

tcInput[i].k,

tcInput[i].j

);

printf("实际: %d, 预期： %d", actualOutcome, tcOutcome[i]);

if( actualOutcome == tcOutcome[i] ) {

tcPassNum ++; // 记录通过的测试用例总数

printf( " [Pass]\n" );

} else {

tcFailNum ++; // 记录失败的测试用例总数

printf( " [Fail]\n" );

}

printf("\n");

}

// 显示统计结果

printf( "共执行10个测试用例，其中%d个通过，%d个失败\n", tcPassNum, tcFailNum );

}

int main()

{

Driveroffunc();

system("pause");

return 0;

}

### 3.3.5 实验要求

（1）根据白盒测试方法分别对上面的题目设计测试用例

（2）采用小组成员互测方式，即每个小组成员对小组其他成员的程序进行报告。

（3）编写实验报告。

* 上述题目实验内容的“测试用例编写”为测试小组其他成员程序的《测试用例设计表》，“测试结果分析”部分应换成《缺陷报告》。

### 3.3.6 实验思考

（1）在实际的测试中，如何设计测试用例才能达到用最少的测试用例检测出最多的缺陷。

（2）在进行测试设计时，如何考虑测试用例的充分性和减少测试用例的冗余性。

**附源程序和测试框架：**

**测试框架：**

import static org.junit.Assert.\*;

import org.junit.Test;

public class logicexampleTest {

//predicate testing

@Test

public void testDoWorkbug1\_1() {

assertTrue("logic error",logicexample.doWork(5,10,5)==logicexample.doWorkbug1(5,10,5));

}

@Test

public void testDoWorkbug1\_2() {

assertTrue("logic error",logicexample.doWork(3,4,5)==logicexample.doWorkbug1(3,4,5));

}

@Test

public void testDoWorkbug2\_1() {

assertTrue("logic error",logicexample.doWork(5,10,5)==logicexample.doWorkbug2(5,10,5));

}

@Test

public void testDoWorkbug2\_2() {

assertTrue("logic error",logicexample.doWork(3,3,5)==logicexample.doWorkbug2(3,3,5));

}

@Test

public void testDoWorkbug2\_3() {

assertTrue("logic error",logicexample.doWork(4,4,1)==logicexample.doWorkbug2(4,4,1));

}

}

## 3.4 输入域划分

### 3.4.1 实验类型

实验类型为设计型，4个学时。

### 3.4.2 实验目的

（1）掌握输入域建模的过程；

（2）熟悉各种输入域划分标准的测试方法与步骤。

### 3.4.3 实验内容

**题目一：编写程序判定三角形的类型**

根据下面给出的规格说明，利用等价类划分的方法，给出足够的测试用例。

“一个程序读入三个整数。把此三个数值看成是一个三角形的三个边。这个程序要打印出信息，说明这个三角形是三边不等的、是等腰的、还是等边的。”

**题目二：编写程序实现简单的ATM机的找零钱最佳组合功能**

假设商店货品价格(**R)**皆不大于100元（且为整数），若顾客付款在100元内 (**P)** ， 求找给顾客最少货币个（**张）**数？（货币面值50元10 元，5 元，1元四 种 ）

（1）编写程序；

（2）使用等价类划分等方法划分测试、按成对覆盖、基本选择覆盖对该程序进行测试。

### 3.4.4 实验步骤

**题目一：**

（1）先用等价类和边界值方法设计测试用例，然后用白盒法进行检验与补充。

（2）判断三角形问题的程序流程图和程序流图如图1和图2所示。用你熟悉的语言编写源程序。

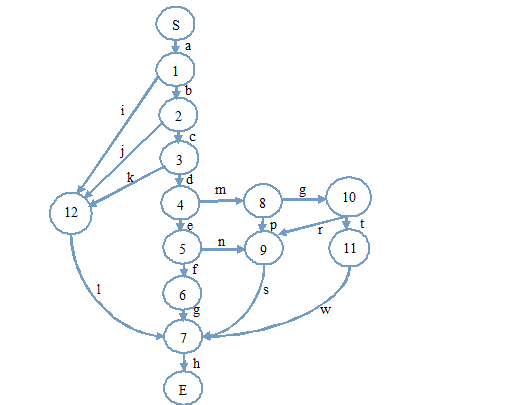
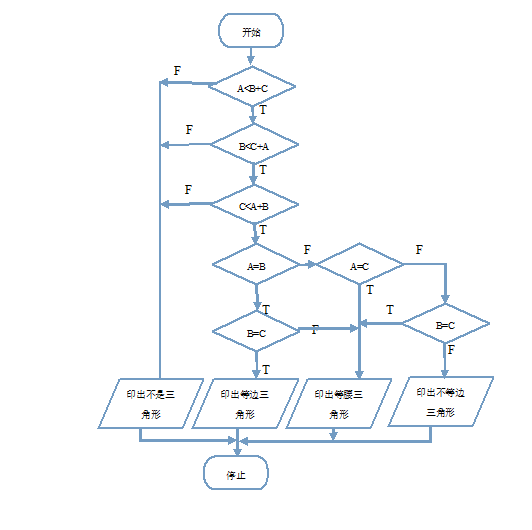


图3.4.1 程序流程图 图3.4.2程序流图

（3）使用等价类方法设计测试用例，并填写完成表2和表3。

a.等价类划分如下：

|  |  |  |
| --- | --- | --- |
| 输入数据 | 有效等价类 | 无效等价类 |
| 三角形边a、b、c | 1. 三边均大于0  2. 任意三边之和大于第三边是三角形  3. 三边相等是等边三角形  4. 任意三边之和大于第三边且其中任意两边相等是等腰三角形 | 5. 含边为0的三个数  6. 含两数之和小于或等于第三个数 |

b. 为等价类设计测试用例

|  |  |  |
| --- | --- | --- |
| 测试数据：a、b、c | 期望结果 | 覆盖范围 |
| 3,4,5 | 显示有效输入，是不等边三角形 | 1、2 |
| 10,10,10 | 显示有效输入，是等边三角形 | 1、3 |
| 6,6,7 | 显示有效输入，是等腰三角形 | 1、4 |
| 12,0,13 | 显示无效输入 | 5 |
| 12,7,4 | 显示无效输入 | 6 |

表2. 等价类划分表

|  |  |  |
| --- | --- | --- |
| 输入条件 | 有效等价类 | 无效等价类 |
| 是否构成三角形 |  |  |
| 是否等腰三角形 |  |  |
| 是否等边三角形 |  |  |

表3. 测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 用例编号 | 测试数据(A,B,C) | 等价类覆盖情况 | 输出 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

（4）输入设计好的测试用例，执行源程序，记录输出结果。

**题目二：**

（1）分析输入的情形

货品价格:

R > 100

0 < R < = 100  
R <= 0  
顾客付款:

P > 100  
R<= P <= 100  
P < R

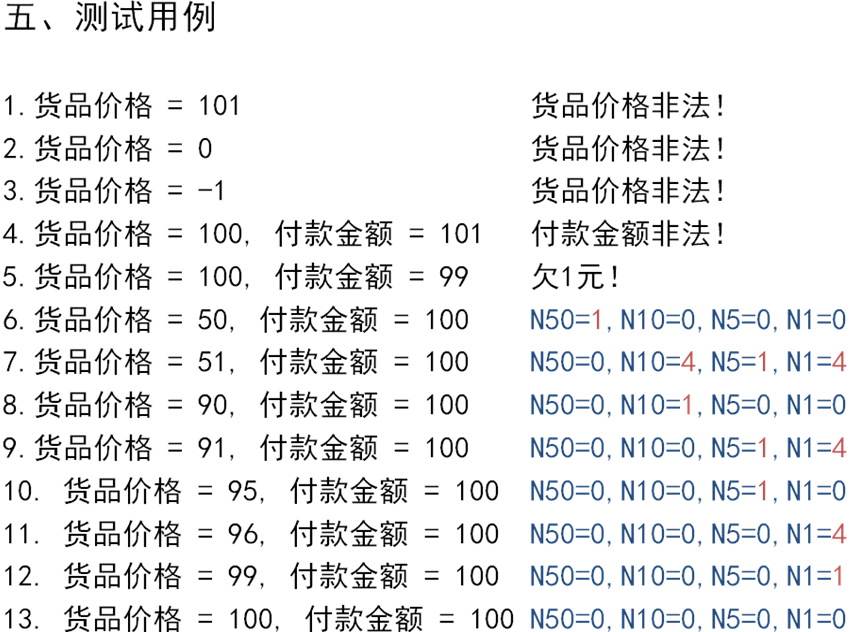
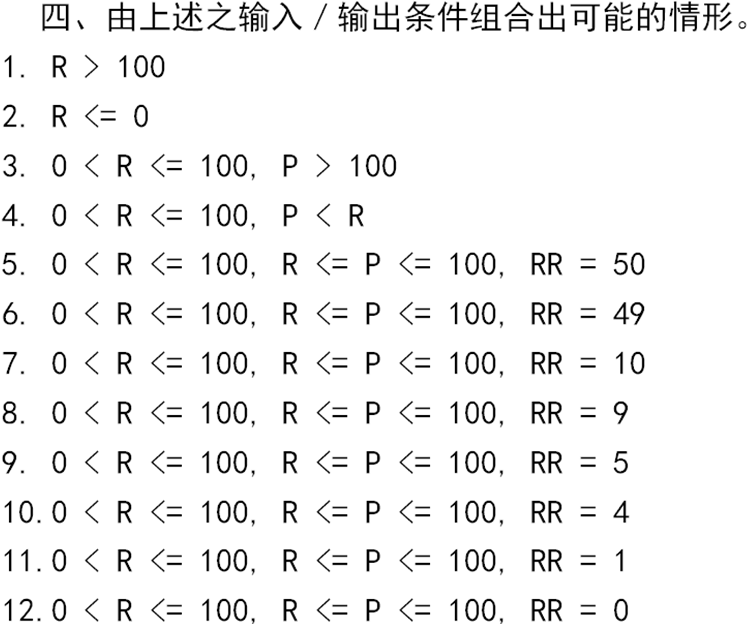
（2）分析输出情形

* N50 = 1
* N50 = 0
* 1<= N10 <=4   
  N10 = 0
* N5 = 1  
  N5 = 0
* 1<= N1 <=4   
  N1 = 0

（3）分析规格中每一决策点之情形，以RR1, RR2, RR3 表示计算要找50, 10, 5元货币数时之剩余金额。

R > 100

R <= 0  
P > 100  
P < R  
RR1 >= 50  
RR2 >= 10  
RR3 >= 5



* 根据动态黑盒测试方法分别对上面的题目设计测试用例，主要考虑等价类划分、边界值以及错误猜测法。
* 根据所设计的测试用例进行实际的测试。

### 3.4.5 实验要求

（1）根据题目的要求设计测试用例。

（2）采用小组成员互测方式，即每个小组成员对小组其他成员的程序进行报告。

（2）编写实验报告。

* 上述题目实验内容的“测试用例编写”为测试小组其他成员程序的《测试用例设计表》，“测试结果分析”部分应换成《缺陷报告》。

### 3.4.6 实验思考

（1）在实际的测试中，输入域建模的方法有哪些？分别包含哪些步骤？。

（2）在进行测试设计时，对于结构化的输入域，要选择每个成员的输入点的组合，这个过程可能会产生大量的数据。若考虑输入域之间内部联系有选择的进行组合，可以一定程度上减轻这个问题，如何理解这一策略。

**附源程序和测试框架：**

**源代码：**

**public** **enum** Triangle {

***SCALENE***, ***ISOSCELES***, ***EQUILATERAL***, ***INVALID***

}

**public** **class** TriangleType

{

**public** **static** Triangle triangle (**int** s1, **int** s2, **int** s3)

{

// Reject non-positive sides

**if** (s1 <= 0 || s2 <= 0 || s3 <= 0)

**return** (Triangle.***INVALID***);

// Check triangle inequality

**if** (s1+s2 <= s3 || s2+s3 <= s1 || s1+s3 <= s2)

**return** (Triangle.***INVALID***);

// Identify equilateral triangles

**if** ((s1 == s2) && (s2 == s3))

**return** Triangle.***EQUILATERAL***;

// Identify isosceles triangles

**if** ((s1 == s2) || (s2 == s3) || (s1 == s3))

**return** Triangle.***ISOSCELES***;

**return** (Triangle.***SCALENE***);

}

}

**测试框架：**

import org.junit.\*;

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

import org.junit.runners.Parameterized.Parameters;

import static org.junit.Assert.\*;

import java.util.\*;

@RunWith(Parameterized.class)

public class TriangleTypeTest {

public int x;

public int y;

public int z;

public Triangle t;

public TriangleTypeTest (int x, int y, int z, Triangle t) {

this.x = x; this.y = y; this.z = z; this.t = t;

}

@Parameters public static Collection<Object[]> parameters() {

List<Object[]> list = new ArrayList<Object[]>();

list.add(new Object[] {1, 2, 0, Triangle.INVALID});

list.add(new Object[] {0, 1, 2, Triangle.INVALID});

list.add(new Object[] {1, 0, 2, Triangle.INVALID});

list.add(new Object[] {1, 2, 3, Triangle.INVALID});

list.add(new Object[] {1, 3, 2, Triangle.INVALID});

list.add(new Object[] {3, 1, 2, Triangle.INVALID});

list.add(new Object[] {2, 2, 4, Triangle.INVALID});

list.add(new Object[] {2, 4, 2, Triangle.INVALID});

list.add(new Object[] {4, 2, 2, Triangle.INVALID});

list.add(new Object[] {2, 2, 3, Triangle.ISOSCELES});

list.add(new Object[] {3, 2, 2, Triangle.ISOSCELES});

list.add(new Object[] {2, 3, 2, Triangle.ISOSCELES});

list.add(new Object[] {2, 2, 2, Triangle.EQUILATERAL});

list.add(new Object[] {2, 3, 4, Triangle.SCALENE});

list.add(new Object[] {4, 3, 2, Triangle.SCALENE});

list.add(new Object[] {3, 4, 2, Triangle.SCALENE});

return list;

}

@Test public void triangleTest() {

assertEquals(TriangleType.triangle(x, y, z), t);

}

}

## 3.5 因果图法

### 3.5.1 实验类型

实验类型为设计型，4个学时。

### 3.5.2 实验目的

（1）掌握因果图进行输入建模方法与过程；

（2）熟悉由因果图生成软件测试用例的方法。

### 3.5.3 实验内容

有一个处理单价为5角钱的饮料的自动售货机软件测试用例的设计。其规格说明如下：若投入5角钱或1元钱的硬币，按下〖橙汁〗或〖啤酒〗的按钮，则相应的饮料就送出来。若售货机没有零钱找，则一个显示〖零钱找完〗的红灯亮，这时在投入1元硬币并按下按钮后，饮料不送出来而且1元硬币也退出来；若有零钱找，则显示〖零钱找完〗的红灯灭，在送出饮料的同时退还5角硬币。

（1） 根据需求画出因果图；

（2） 将因果图转化为决策表；

（3） 编写代码实现自动售货机，并设计测试用例进行测试。

### 3.5.4 实验步骤

**1.分析这一段说明，列出原因和结果**

*原因：*

1.售货机有零钱找

2.投入1元硬币

3.投入5角硬币

4.押下橙汁按钮

5.押下啤酒按钮

*结果：*

21.售货机〖零钱找完〗灯亮

22.退还1元硬币

23.退还5角硬币

24.送出橙汁饮料

25.送出啤酒饮料

**2.画出因果图**

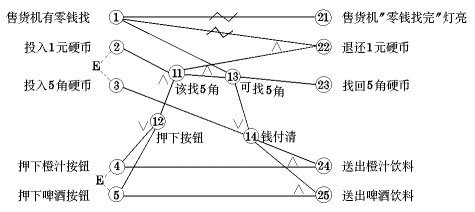
如图所示。所有原因结点列在左边，所有结果结点列在右边。建立中间结点，表示处理的中间状态。中间结点：

11. 投入1元硬币且押下饮料按钮

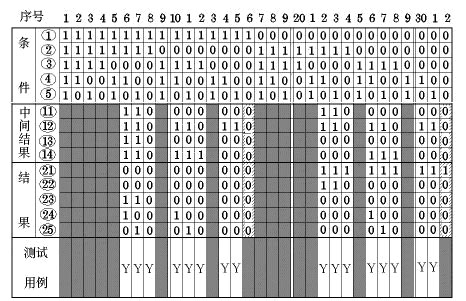
12. 押下〖橙汁〗或〖啤酒〗的按钮

13. 应当找5角零钱并且售货机有零钱找

14. 钱已付清



**3.转换成判定表：**



**4.设计测试用例**

1. 在售货机有零钱找的情况下，投入1元硬币，押下橙汁按钮，找回5角硬币并送出橙汁饮料。
2. 在售货机有零钱找的情况下，投入1元硬币，押下啤酒按钮，找回5角硬币并送出啤酒饮料。
3. 在售货机有零钱找的情况下，投入1元硬币，系统不做任何处理。
4. 在售货机有零钱找的情况下，投入5角硬币，押下橙汁按钮，送出橙汁饮料。
5. 在售货机有零钱找的情况下，投入5角硬币，押下啤酒按钮，送出啤酒饮料。
6. 在售货机有零钱找的情况下，投入5角硬币，系统不做任何处理。
7. 在售货机有零钱找的情况下，押下橙汁按钮，系统不做任何处理。
8. 在售货机有零钱找的情况下，押下啤酒按钮，系统不做任何处理。
9. 在售货机没有零钱找的情况下，投入1元硬币，押下橙汁按钮，售货机“零钱找完”灯亮，并退还1元硬币。
10. 在售货机没有零钱找的情况下，投入1元硬币，押下啤酒按钮，售货机“零钱找完”灯亮，并退还1元硬币。
11. 在售货机没有零钱找的情况下，投入1元硬币，售货机“零钱找完”灯亮。
12. 在售货机没有零钱找的情况下，投入5角硬币，押下橙汁按钮，售货机“零钱找完”灯亮，并送出橙汁饮料。
13. 在售货机没有零钱找的情况下，投入5角硬币，押下啤酒按钮，售货机“零钱找完”灯亮，并送出啤酒饮料。
14. 在售货机没有零钱找的情况下，投入5角硬币，售货机“零钱找完”灯亮。
15. 在售货机没有零钱找的情况下，押下橙汁按钮，售货机“零钱找完”灯亮。
16. 在售货机没有零钱找的情况下，押下啤酒按钮，售货机“零钱找完”灯亮。

### 3.5.5 实验要求

（1）根据题目的要求绘制因果图。

（1）根据题目的要求设计测试用例。

（2）采用小组成员互测方式，即每个小组成员对小组其他成员的程序进行报告。

（2）编写实验报告。

* 上述题目实验内容的“测试用例编写”为测试小组其他成员程序的《测试用例设计表》，“测试结果分析”部分应换成《缺陷报告》。

### 3.5.6 实验思考

（1）因果图有四种关系和五种约束。

（2）在进行测试设计时，如何考虑测试用例的充分性和减少测试用例的冗余性。 因果图设计图从自然语言书写的程序规格说明的描述中，找出因（输入条件）和果（输出或程序状态的改变），通过因果图转换为判定表，其步骤有哪些？

（3）因果图法优缺点

**答：**

优点：

①  考虑到了输入情况的各种组合以及各个输入情况之间的相互制约关系。

②  能够帮助测试人员按照一定的步骤，高效率的开发测试用例。

③  因果图法是将自然语言规格说明转化成形式语言规格说明的一种严格的方法，可以指出规格说明存在的不完整性和二义性。

 缺点：

①  因果图来设计测试用例时，作为输入条件的原因与输出结果之间的因果关系，有时很难从软件需求规格说明中得到。

②  而且往往因果关系非常庞大，以至于据此因果图而得到的测试用例数目多的惊人，给软件测试，特别是手工测试带来沉重的负担，为了有效地，合理地减少测试的工时与费用，可利用正交实验设计方法进行测试用例的设计。

**附源程序和测试框架：**

**源代码(包含bug)：**

public class SaleMachine {

/\*\*

\* @param args

\*/

private int milkCount,juiceCount,fiveCents,oneDollar;

private String[] goodsType={"milk","orange"};

private String result;

public SaleMachine(){

initial();

}

private void initial(){

milkCount=6;

juiceCount=6;

fiveCents=6;

oneDollar=6;

}

public SaleMachine(int fiveCents,int oneDollar,int beerCount,int juiceCount){

this.fiveCents=fiveCents;

this.oneDollar=oneDollar;

this.milkCount=beerCount;

this.juiceCount=juiceCount;

}

public String currentState(){

String state ="当前数量\n"+"牛奶:"+milkCount+"\n"

+"橙汁:"+juiceCount+"\n"+"5角:"+fiveCents+"\n"+"1元:"+oneDollar+"\n";

return state;

}

/\*

\* @param type:用户选择的产品

\* @param money:用户投币种类

\* @return 结果信息

\*/

public String operation(String type,String money) {

if(money.equalsIgnoreCase("5")){

if(type.equals(goodsType[0])){

if(milkCount>0){

milkCount++;

fiveCents++;

result="输入信息\n"

+"类型：牛奶; 钱：5角； 找零：0\n" + currentState();

}

else{

result ="失败信息："+"没有牛奶\n";

}

}else if(type.equals(goodsType[1])){

if(juiceCount>0){

juiceCount--;

fiveCents++;

result="输入信息\n"

+"类型：牛奶; 钱：1元； 找零：5角\n"

+currentState();

}else{

result="失败信息："+"没有橙汁\n";

}

}else{

result="失败信息："+"类型错误\n";

}

}else if(money.equalsIgnoreCase("10")){

if(fiveCents>0){

if(type.equals(goodsType[0])&&milkCount>0){

milkCount--;

fiveCents--;

oneDollar++;

result="输入信息\n"

+"类型：牛奶； 钱：1元 ； 找零：5角\n"

+currentState();

}else if(type.equals(goodsType[1])&&juiceCount>0){

juiceCount--;

fiveCents--;

oneDollar++;

result="输入信息\n"

+"类型：牛奶； 钱：1元 ； 找零：5角\n"

+currentState();

}else{

if(type.equals(goodsType[0])&&milkCount<=0){

result="失败信息："+"没有牛奶\n";

}else if(type.equals(goodsType[1])&&juiceCount<=0){

result="失败信息："+"没有橙汁\n";

}else{

result="失败信息："+"类型错误\n";

}

}

}else {

result="失败信息："+"没有零钱\n";

}

}else{

result="失败信息："+"钱的输入错误\n";

}

return result;

}

}

**测试框架:**

import static org.junit.Assert.\*;

import org.junit.Before;

import org.junit.Test;

public class SaleMachineTest extends SaleMachine {

SaleMachine saleMachine=null;

@Before

public void setUp() throws Exception {

saleMachine=new SaleMachine();

}

@Test

public void testOperation1() {

String expected="输入信息\n"+"类型：牛奶; 钱：5角； 找零：0\n"

+"当前数量\n"

+"牛奶:5\n"

+"橙汁:6\n"

+"5角:7\n"

+"1元:6\n" ;

assertEquals(expected, saleMachine.operation("milk","5"));

}

@Test

public void testOperation2() {

String expected="输入信息\n"+"类型：橙汁; 钱：5角； 找零：0\n"

+"当前数量\n"

+"牛奶:6\n"

+"橙汁:5\n"

+"5角:7\n"

+"1元:6\n" ;

assertEquals(expected, saleMachine.operation("orange","5"));

}

@Test

public void testOperation3() {

String expected="输入信息\n"+"类型：牛奶; 钱：1元； 找零：5角\n"

+"当前数量\n"

+"牛奶:5\n"

+"橙汁:6\n"

+"5角:5\n"

+"1元:7\n" ;

assertEquals(expected, saleMachine.operation("milk","10"));

}

@Test

public void testOperation4() {

String expected="输入信息\n"+"类型：橙汁; 钱：1元； 找零：5角\n"

+"当前数量\n"

+"牛奶:6\n"

+"橙汁:5\n"

+"5角:5\n"

+"1元:7\n" ;

assertEquals(expected, saleMachine.operation("orange","10"));

}

@Test

public void testOperation5(){

SaleMachine saleMachine=new SaleMachine(0,6,6,6);

String expected="失败信息：没有零钱\n";

assertEquals(expected,saleMachine.operation("milk", "10"));

}

@Test

public void testOperation6(){

SaleMachine saleMachine=new SaleMachine(0,6,6,6);

String expected="失败信息：没有零钱\n";

assertEquals(expected,saleMachine.operation("orange", "10"));

}

@Test

public void testOperation7(){

SaleMachine saleMachine=new SaleMachine(6,6,0,6);

String expected="失败信息：没有牛奶\n";

assertEquals(expected,saleMachine.operation("milk", "10"));

}

@Test

public void testOperation8(){

SaleMachine saleMachine=new SaleMachine(6,6,6,0);

String expected="失败信息：没有橙汁\n";

assertEquals(expected,saleMachine.operation("orange", "5"));

}

@Test

public void testOperation9(){

//SaleMachine saleMachine=new SaleMachine(0,6,6,6);

String expected="失败信息：投币错误\n";

assertEquals(expected,saleMachine.operation("milk", "100"));

}

@Test

public void testOperation10(){

//SaleMachine saleMachine=new SaleMachine(0,6,6,6);

String expected="失败信息：类型错误\n";

assertEquals(expected,saleMachine.operation("coca-cola", "10"));

}

@Test

public void testOperation11(){

SaleMachine saleMachine=new SaleMachine(6,6,0,6);

String expected="失败信息：没有牛奶\n";

assertEquals(expected,saleMachine.operation("milk", "5"));

}

@Test

public void testOperation12(){

//SaleMachine saleMachine=new SaleMachine(6,6,0,6);

String expected="失败信息：类型错误\n";

assertEquals(expected,saleMachine.operation("coca-cola", "10"));

}

@Test

public void testOperation13(){

SaleMachine saleMachine=new SaleMachine(6,6,6,0);

String expected="失败信息：没有橙汁\n";

assertEquals(expected,saleMachine.operation("orange", "10"));

}

}

## 3.6 综合实验

### 3.6.1 实验类型

实验类型为综合型性，6个学时。

### 3.6.2 实验目的

（1）能够运用黑盒测试方法设计测试用例。

（2）对测试用例进行优化。

### 3.6.3 实验内容

**题目一：**

假设某单位内部电话号码由三部分组成，分别是：分机号+前缀+后缀。其中，分机号为空白或一位数字；前缀为非‘0’开头的二位数字；后缀为非全0的3位数字。假定被测程序能接受一切符合上述规定的电话号码，拒绝所有不符合规定的电话号码。根据问题的规格说明，从输入条件划分等价类。

输入等价类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 输入 | ID | 有效等价类 | ID | 无效等价类 |
| 分机号 | 1 | 1位数字 | 5 | 多于1位数字 |
|  | 2 | 空白 | 6 | 1位非数字 |
| 前缀 | 3 | 01-99 | 7 | 少于2位数字 |
|  |  |  | 8 | 多于2位数字 |
|  |  |  | 9 | 含有非数字 |
|  |  |  | 10 | 起始位为‘0’ |
| 后缀 | 4 | 001-999 | 11 | 少于3位数字 |
|  |  |  | 12 | 多于3位数字 |
|  |  |  | 13 | 含有非数字 |
|  |  |  | 14 | 全‘0’ |

**编程实现：**

（1） 对每一个有效等价类，至少设计一个测试用例。

输入格式：输入（分机号-前缀-后缀）：

输出格式：输出（接受Y/拒绝N）：

覆盖等价类（ID类型）：

例如：输入（分机号-前缀-后缀）： 3,12,001 ↙（回车）

输出（接受Y/拒绝N）：Y

覆盖等价类（ID类型）：1,3,4

（2）对每一个无效的输入，设计一个无效等价类，并设计测试用例。

例如：输入（分机号-前缀-后缀）： #,12,111 ↙（回车）

输出（接受Y/拒绝N）：N

覆盖等价类（ID类型）：6

**题目二：阅读下面的一段程序，**

void Test(int X,int Y,int Z)

1 {

2 int k=0,j=0;

3 if((X>3)&&(Z<10))

4 {

5 k=X\*Y-1;

6 j=sqrt(k);

7 }

8 if((X==4)||(Y>5))

9 j=X\*Y+10;

10 j=j%3;

11 }

根据所给定的程序，该程序的可能路径如下：

路径1：1→2→3→5→6→8→9→10→11

路径2：1→2→3→8→10→11

路径3：1→2→3→8→9→10→11

路径4：1→2→3→5→6→8→10→11

编程实现：

（1）设计测试用例,实现语句覆盖，并在每条语句后插入一条显示输出语句（printf/cout），输出语句标号。

输入格式：输入（X, Y, Z）：

输出格式：输出（k, j）：k= j=

路径ID：

例如：输入（X,Y, Z）： 4,5,5 ↙（回车）

输出（k, j）：k=19 j=0

路径ID：路径1：1→2→3→5→6→8→9→10→11

执行语句2

执行语句3

执行语句5

执行语句6

执行语句8

执行语句9

执行语句10

（2）设计测试用例,实现条件覆盖，使得每个判定中的每个条件至少执行一次。

输入格式：输入（X, Y, Z）：

输出格式：输出（k, j）：k= j=

路径ID：

例如：输入（X, Y, Z）： 2, 5, 15 ↙（回车）

输出（k, j）：k=0 j=0

路径ID：路径2：1→2→3→8→10→11

（3）分别计算语句覆盖、条件覆盖及判定覆盖的测试充分度。

### 3.6.4 实验步骤

（1）根据黑盒测试方法设计测试用例，主要考虑等价类和边界值方法。

（2）根据设计的测试用例进行测试。

### 3.6.5 实验要求

（1）根据题目要求设计测试用例（参考附件C）。

（2）编写实验报告。

### 3.6.6 实验思考

（1）在实际的测试中，系统测试应包括哪些类型的测试？

（2）在进行测试设计时，如何考虑多人对话的测试？

（2）根据软件测试课程，讨论软件测试停止的条件。

# 附录A：通用代码审查清单

1. 数据引用错误

指使用未经正确初始化用法和引用方式的变量、常量、数组、字符串或记录而导致的软件缺陷。

* + 是否引用了未初始化变量？查找遗漏之处与查找错误同等重要。
  + 数组和字符串的下标是整数值吗？下标总是在数组和字符串大小范围之内吗？
  + 常量和变量的使用 。是否在应该使用常量的地方使用了变量？
  + 为引用的指针分配内存了吗？
  + 一个数据结构是否在多个函数或子程序中引用，在每一个引用中明确定义结构了吗？

2． 数据声明错误

指不正确地声明或使用变量和常量。

* + 所有变量都赋予正确的长度、类型和存储类了吗？例如本应声明为字符串的变量声明为字符数组了？
  + 变量是否在声明的同时进行了初始化？是否正确初始化并与其类型一致？
  + 变量有相似的名称吗？这基本上不算软件缺陷，但有可能是程序中其他地方出现名称混淆的信息。
  + 存在声明过、但从未引用或者引用过一次的变量吗？
  + 在特定模块中所有变量都显式声明了吗？如果没有，是否可以理解为该变量与更高级别的模块共享？

3． 计算错误

指基本的数学逻辑问题，计算无法得到与其结果。

* + 计算中是否使用了不同数据类型的变量，例如将整数与浮点数相加？
  + 计算中是否使用了不同数据类型相同但不同长度的变量——例如，将字节与字相加？
  + 计算时是否了解和考虑到编译器对类型或长度不一致的变量的转换规则？
  + 赋值的目的变量是否小于赋值表达式的值？
  + 在数值计算过程中是否可能出现溢出？
  + 除数/模是否可能为零？
  + 对于整型算术运算，某些计算，特别是除法的代码处理是否会丢失精度？
  + 变量的值是否超过有意义的范围？例如，可能性的计算机结果是否小于0%或者大于100%。
  + 对于包含多个操作数的表达式，求值的次序是否混乱，运算优先级对吗？需要加括号使其清楚吗？

4． 比较错误

指大于、小于、等于、不等于、真、假。比较和判断错误很可能是边界条件问题。

* + 比较得正确吗？
  + 存在分数或浮点值之间的比较？如果有，精度问题会影响比较吗？1.00000001和1.00000002极其接近，它们相等吗？
  + 每一个逻辑表达式都正确地表达了吗？逻辑计算如期进行了吗？求值次序有疑问吗？
  + 逻辑表达式的操作数是逻辑值吗？例如，是否包含数值的整型变量用于逻辑计算中？

5． 控制流程错误

指编程语言中循环等控制结构未按预期方式工作。它们通常是由计算或者比较错误直接或间接造成。

* + 如果程序包含begin…end和do…while等语句组，end是否对应？
  + 程序、模块、子程序和循环能否终止？如果不能，可以接受吗？
  + 可能存在永远不停的循环吗？
  + 循环可能从不执行吗？如果是这样，可以接受吗？
  + 如果程序包括像switch…case语句这样的多个分支，索引变量能否超出可能的分支数目吗？如果超出，该情况能正确处理吗？
  + 是否存在“丢掉一个”的错误，导致意外进入循环？

6． 子程序参数错误

* + 子程序接受的参数类型和大小与调用代码发送的匹配吗？次序正确吗？
  + 如果子程序有多个入口点，引用的参数是否与当前入口点没有关联？
  + 常量是否当作形参传递，意外在子程序中改动？
  + 子程序是更改了仅作为输入值的参数？
  + 每一个参数的单位是否与相关的形参匹配
  + 如果存在全局变量，在所有引用子程序中是否有相似的定义和属性？

7． 输入/输出错误

* + 软件是否严格遵守外部设备读写数据的专用格式？
  + 文件或者外设不存在或者未准备好的错误情况有处理吗？
  + 软件是否处理外部设备未连接、不可用，或者读写过程中存储空间占满等情况？
  + 软件以与其方式处理预计的错误？
  + 检查错误提示信息的准确性、正确性、语法和拼写了吗？

# 附录B：程序代码审查表

|  |  |  |  |
| --- | --- | --- | --- |
| 问题编号 | 涉及的代码行 | 问题类型 | 拟建议的修改 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 附录C：测试用例设计表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例 | | | | 测试记录 | |
| 用例编号 | 测试目的/对应需求 | 输入/预置条件 | 预期结果 | 操作过程 | 结果 |
|  |  |  |  |  | 功能正确 |
|  |  |  |  |  | 功能不正确 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# 附录D：缺陷报告

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 缺陷编号 | 简要描述 | 位置 | 严重程度 | 缺陷类型 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

注：严重程度定义

（1）系统崩溃、数据丢失、数据毁坏，安全性被破坏。

（2）操作性错误、结果错误、功能遗漏。

（3）小问题、拼写错误、UI布局、罕见故障。

（4）建议

缺陷类型：

* 1. 输入/输出错误
  2. 逻辑错误
  3. 设计错误
  4. 需求错误