东南大学国家示范性软件学院
College of Software Engineering
Southeast University

软件测试基础与实践

实验报告

实验名称：白盒测试实验三

实验地点：计算机楼 268

实验日期：2018 年 11 月 8 日星期四

学生姓名：白丰硕

学生学号：71116233

东南大学 软件学院 制

目录

一、实验目的

（1） 巩固白盒测试知识， 能应用数据流覆盖方法设计测试用例；

（2）学习测试用例的书写。

二、实验内容

硬件环境： PC 机一台

软件环境： Java 编程环境： Java SDK + Eclipse

C/C++编程环境： Visual Studio

程序流图绘制： Visio

待测程序： CgiDecode

实验指导书、 Eclipse 和待测程序可从 FTP://223.3.68.135 或课程主页下载：http://cse.seu.edu.cn/PersonalPage/pwang/course/st.html

三、实验内容

实验 1: 数据流测试技术实验

运用数据流测试方法，对用 C/C++语言实现的 CgiDecode 程序中的 decode()方法进行测试。

(1) 测试要考虑 decode()中 encoded, decoded, *eptr, eptr, *dptr, dptr, ok, c, digit_high, digit_low 变量；

(2) 给出每个变量对应的 du-path 和 dc-path;

(3) 根据变量的 dc-path 设计测试用例，完成对 decode()的测试；

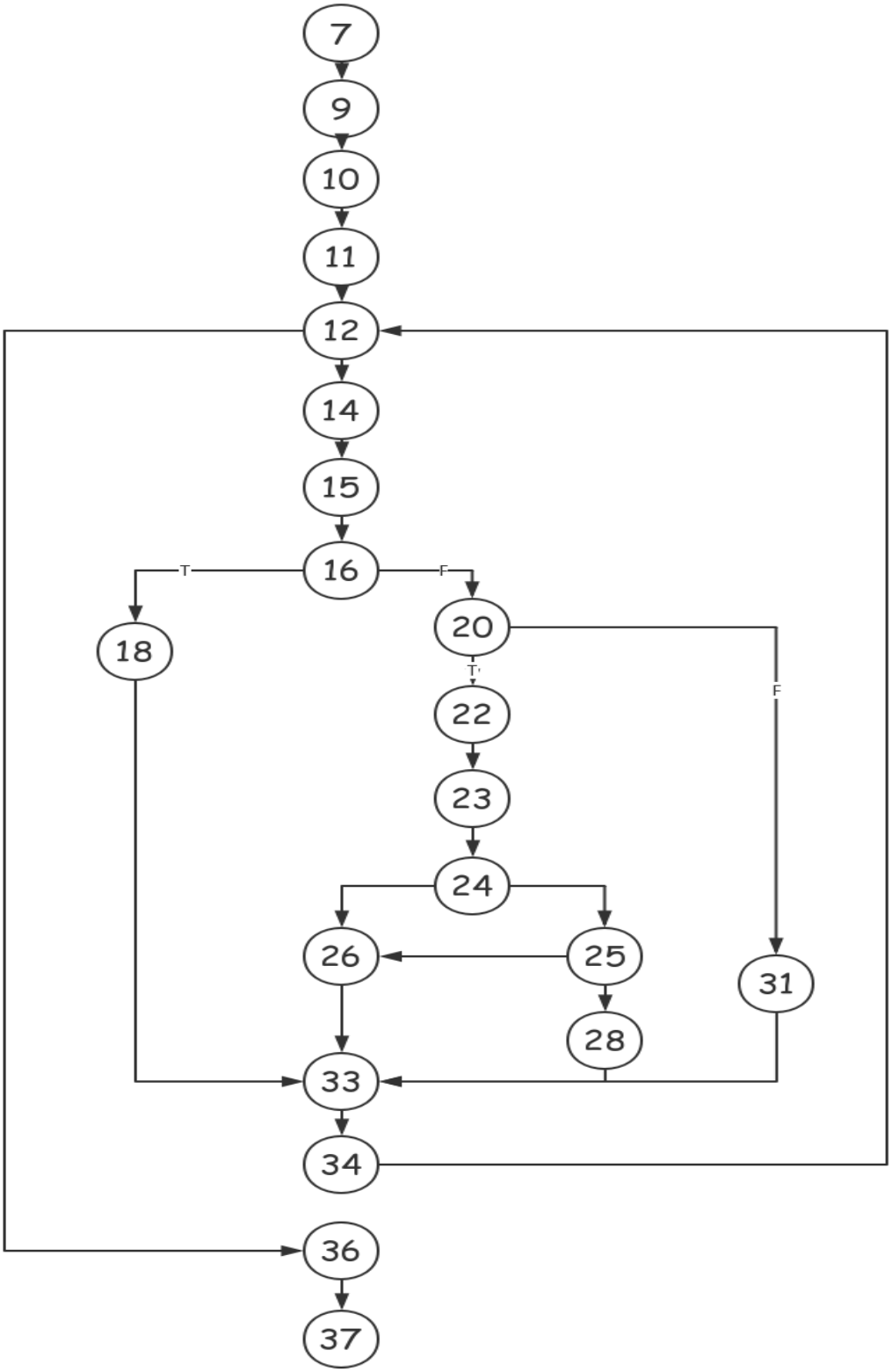
对 decode()的测试：

1. decode()函数的语句及其编号如下：

```
1  /** Translate a string from the CGI encoding to plain ascii text.
2  * '+' becomes space, %xx becomes byte with hex value xx,
3  * other alphanumeric characters map to themselves.
4  * Returns 0 for success, positive for erroneous input
5  * 1 = bad hexadecimal digit
6  */
7  int decode(char *encoded, char *decoded)
8  {
9  char *eptr = encoded;
10 char *dptr = decoded;
11 int ok=0;
12 while (*eptr)
13 {
14 char c;
15 c = *eptr;
16 if (c == '+')
17 { /* Case 1: '+' maps to blank */
18 *dptr =' ';
19 }
20 else if (c == '%')
21 { /* Case 2: '%xx' is hex for character xx */
22 int digit_high = getHexValue(*(++eptr));
23 int digit_low = getHexValue(*(++eptr));
24 if ( digit_high == -1 ||
25 digit_low==-1) {/* *dptr='?'; */
26 ok=1; /* Bad return code */
27 } else {
28 *dptr = 16* digit_high + digit_low;
29 }
30 } else {/* Case 3: All other characters map to themselves */
31 *dptr = *eptr;
32 }
33 ++dptr;
34 ++eptr;
35 }
36 *dptr ='\0'; /* Null terminator for string */
```

```
37  return ok;
38  }
```

2. decode()的程序流图



3. decode()中的变量的定义使用节点

变量名	DEF	USE
encoded	7	9
decoded	7	10
***eptr**	9,22,23,34	12,15,22,23,31
eptr	9,22,23,34	12,15,22,23,31,34
***dptr**	10,18,28,31,33,36	
dptr	10,33	18,28,31,33,36
ok	11,26	37
c	14,15	16,20
digit_high	22	24,28
digit_low	23	25,28

4. decode()变量的 du-path 和 dc-path

变量名	Du-path
encoded	7->9
decoded	7->10
***eptr**	~~9->10->11->12~~
	~~9->10->11->12->14->15~~
	~~9->10->11->12->14->15->16->20->22~~（无穷条）
	9->10->11->12->14->15->16->20->22->23（无穷条）
	9->10->11->12->14->15->16->20->31（无穷条）
	~~22~~
	~~22->23~~
	~~22->23->24->26->33->34->12~~ ~~22->23->24->25->26->33->34->12~~ ~~22->23->24->25->28->33->34->12~~
	~~22->23->24->26->33->34->12->14->15~~ 22->23->24->25->26->33->34->12->14->15 22->23->24->25->28->33->34->12->14->15
	22->23->24->26->33->34->12->14->15->16->20->31（无穷条）
	~~23~~
	~~23->24->26->33->34->12~~ ~~23->24->25->26->33->34->12~~ ~~23->24->25->28->33->34->12~~
	~~23->24->26->33->34->12->14->15~~ 23->24->25->26->33->34->12->14->15 23->24->25->28->33->34->12->14->15
	~~23->24->26->33->34->12->14->15->16->20->31~~（无穷条）
	23->24->26->33->34->12->14->15->16->20->22（无穷条）

	~~34->12~~
	~~34->12->14->15~~
	~~34->12->14->15->16->20->22~~
	34->12->14->15->16->20->22->23
	~~34->12->14->15->16->20->31~~
eptr	~~9->10->11->12~~
	~~9->10->11->12->14->15~~
	~~9->10->11->12->14->15->16->20->22~~（无穷条）
	~~9->10->11->12->14->15->16->20->22->23~~（无穷条）
	~~9->10->11->12->14->15->16->20->31~~（无穷条）
	9->10->11->12->14->15->16->20->22->23->24->26->33->34 9->10->11->12->14->15->16->20->22->23->24->25->26->33->34 9->10->11->12->14->15->16->20->22->23->24->25->28->33->34 9->10->11->12->14->15->16->20->31->33->34 9->10->11->12->14->15->16->18->33->34
	~~22~~
	~~22->23~~
	~~22->23->24->26->33->34~~ ~~22->23->24->25->26->33->34~~ ~~22->23->24->25->28->33->34~~
	~~22->23->24->26->33->34->12~~ ~~22->23->24->25->26->33->34->12~~ ~~22->23->24->25->28->33->34->12~~
	~~22->23->24->26->33->34->12->14->15~~ 22->23->24->25->26->33->34->12->14->15 22->23->24->25->28->33->34->12->14->15
	22->23->24->26->33->34->12->14->15->16->20->31（无穷条）
	~~23~~
	~~23->24->26->33->34~~ ~~23->24->25->26->33->34~~ ~~23->24->25->28->33->34~~
	~~23->24->26->33->34->12~~ ~~23->24->25->26->33->34->12~~ ~~23->24->25->28->33->34->12~~
	~~23->24->26->33->34->12->14->15~~ 23->24->25->26->33->34->12->14->15 23->24->25->28->33->34->12->14->15
	~~23->24->26->33->34->12->14->15->16->20->31~~（无穷条）
	23->24->26->33->34->12->14->15->16->20->22（无穷条）
	~~34~~
	~~34->12~~
	~~34->12->14->15~~
	~~34->12->14->15->16->20->22~~

	34->12->14->15->16->20->22->23
	~~34->12->14->15->16->20->31~~
***dptr**	None
dptr	10->11->12->36
	~~10->11->12->14->15->16->18~~（无穷条）
	~~10->11->12->14->15->16->20->31~~（无穷条）
	10->11->12->14->15->16->20->31->33 10->11->12->14->15->16->18->33 10->11->12->14->15->16->20->22->23->24->26->33 10->11->12->14->15->16->20->22->23->24->25->26->33 10->11->12->14->15->16->20->22->23->24->25->28->33
	~~10->11->12->14->15->16->20->22->23->24->25->28~~（无穷条）
	~~33~~
	33->34->12->36
	33->34->12->14->15->16->18
	33->34->12->14->15->16->20->31
	33->34->12->14->15->16->20->22->23->24->25->28
ok	11->12->36->37（无穷条）
	26->33->34->12->36->37（无穷条）
c	~~14->15->16~~
	14->15->16->20
	~~15->16~~
	~~15->16->20~~
digit_high	~~22->23->24~~
	22->23->24->25->28
digit_low	~~23->24~~
	23->24->25->28

变量名	dc-path	测试用例	预期输出	实际输出
encoded	7->9	“abc”	0	0
decoded	7->10	“zaq”	0	0
*eptr	~~9->10->11->12~~			
	~~9->10->11->12->14->15~~			
	9->10->11->12->14->15->16->20->31	“12”	0	0
	22	“%123”	0	0
	23	“%456”	0	0
	~~34->12~~			
	~~34->12->14->15~~			
	34->12->14->15->16->20->31	“encode”	0	0
eptr	~~9->10->11->12~~			
	~~9->10->11->12->14->15~~			

	9->10->11->12->14->15->16->20->31	“789”	0	0
	22	“%741”	0	0
	23	“%135”	0	0
	~~34~~			
	~~34->12~~			
	~~34->12->14->15~~			
	34->12->14->15->16->20->31	“qaz”	0	0
*dptr	None			
dptr	10->11->12->14->15->16->18	“+7891”	0	0
	10->11->12->14->15->16->20->31	“74185”	0	0
	10->11->12->14->15->16->20->22->23->24->25->28	“%7axbv15”	0	0
	10->11->12->36	“”	0	0
	~~33~~			
	33->34->12->14->15->16->18	“+45”	0	0
	33->34->12->14->15->16->20->31	“7465”	0	0
	33->34->12->14->15->16->20->22->23->24->25->28	“%92d03”	0	0
	33->34->12->36	“1”	0	0
ok	11->12->36->37(无穷条)	“”	0	0
	26->33->34->12->36->37(无穷条)	“%x”	1	1
c	~~15->16~~			
	15->16->20	“%4152”	0	0
digit_high	~~22->23->24~~			
	22->23->24->25->28	“%a123”	0	0
digit_low	~~23->24~~			
	23->24->25->28	“%b123”	0	0

四、实验体会

（1）通过测试，是否发现程序中存在的缺陷？

没有发现缺陷。

（2）谈谈数据流测试和控制流测试的区别和联系。

数据流测试：数据流测试主要关注点在于程序中的一些变量的定义（变量的声明，空间的申请等），以及一些程序变量值的改变。在程序中，数据是占有很大的作用的，往往是将一些数据输入到程序中，希望程序可以处理成为我们想要的数据。基于数据流的测试就是在测试中着重关心这些程序中出现的变量是否在按照预期的情况在变化，以此来发现程序中的 bug。

控制流测试：控制流测试主要关注点是程序中的逻辑，程序中的逻辑是由一些判定以及它的组成条件构成的，控制流测试着眼于这些判定的，这些判定影响着程序的走向，来测试在这些节点处是否可以有正确的走向，以此来发现是否存在 bug。并且控制流测试有根据不同的粒度，不同的覆盖效率，不同的思路分为不同的控制流测试。

联系：在程序中，程序的整体逻辑就像一个框架，由程序中的各个判定及其组成它的的条件构成节点，节点和节点之间的通路构成数据的走向。而数据从输入到输出，就是在这个框架中流动。所以基于数据流和基于控制流的测试方法，虽然是以不同的角度来进行测试的，但是都是关注程序的结构、细节内容以及运作情况，都是需要测试人员对程序的代码有一定的理解才可以进行测试的。

（2） 如果用工具来替代手工的白盒测试，你觉得这样的工具应该如何设计？设计的技术中可能的技术难点在哪里？

我认为白盒测试的工具设计的思路应该是先要识别程序的整体逻辑，先形成程序的整体框架，然后针对不同的测试准则，来生成不同的测试用例。因为对于白盒测试，基于控制流的测试重于逻辑结构，基于数据流的测试重于数据的改变。生成程序的整体框架，相当于构建出数据流动的通路或者走向，检测流动的走向是否正确就是需要检查逻辑的正确，检测数据变化是否正常就是需要检查数据流的变化。

主要的困难，对于程序中难免出现一些复杂的逻辑结构，多个循环的结构，甚至之间还有嵌套；判定条件的互相有关联；测试单个函数时，函数之间有调用的需求或者跨文件之间需要有函数或者变量的使用；诸如此类的复杂情况下，保证测试工具可以正确，完整的分析出来全部的情况，生成出合理合适的测试用例。

实验 2：白盒测试工具的使用

由于所在地区不支持下载，在官网上没有下载到 Parasoft Jtest 白盒测试工具。所以就查了一些 Jtest 的使用方法、优点等资料。

1. 工具简介

Jtest 是第一个自动化 Java 单元测试工具。Jtest 自动测试任何 Java 类或部件，而不需要写一个测试用例、驱动程序或桩函数。只要点击一个按钮，Jtest 自动测试代码构造（白盒测试）、测试代码功能性（黑盒测试）、维护代码完整性（回归测试）和静态分析（编程标准执行和指标度量）。无需复杂的设置，Jtest 能够立即使用并指出问题。如果使用“按合同设计”技术在代码中加入描述信息，Jtest 能够自动建立和执行测试用例验证一个类的功能是否符合其功能描述。

2. Jtest 的白盒测试

Jtest 通过自动生成和执行能够全面测试类代码的测试用例，使白盒测试完全自动化。Jtest 使用一个符号化的虚拟机执行类，并搜寻未捕获的运行时异常。对于检测到的每个未捕获的运行时异常，Jtest 报告一个错误，并提供导致错误的栈轨迹和调用序列。Jtest 的先进技术保证它能够自动测试类的所有代码分支，从而彻底检查被测类的结构。

3. Jtest 的报告

Jtest 报告下列未捕获的运行时异常：

1. 行为错误的方法：这些方法对于某些特定输入不会产生异常，必须修改这些代码。
2. 非预期参数：这一问题出现在当某方法遇到非预期的输入（不知任何处理）而产生一个异常。这些问题的修正可以通过检查输入并产生一个 IllegalArgumentException （假如该输入是非法的），改正这类问题可以使代码更清晰更易维护。
3. 行为正确的方法：这时，方法的正确输出是产生一个异常。在这种情形下，建议开发人员修改代码，将这类异常的产生置于方法的 throw 子句中。这会得到更清晰的代码并易于维护。
4. 仅为开发人员使用的方法：在这种情况下，这些方法"不被假设"成处理 Jtest 生成的输入，开发人员是这些方法的唯一使用者，并且不传递这些输入参数。最好的办法是修改这些代码，让它产生一个 IAE。这将带来额外的好处，使代码更易阅读。

过执行自动白盒测试，并提示上述类型的问题，Jtest 能够为开发人员节省大量的时间并防止了错误，同时也提高了代码的可维护性。由于能够自动执行白盒测试的各个步骤，Jtest 对开发人员来说是非常实用的，为了保证质量可以经常执行这一综合性测试。更进一步，使用测试生成系统技术产生的测试输入，Jtest 使得白盒测试比手工测试更精确更有效。白盒（构造）测试验证对一个类的非预期输入不会导致程序的崩溃。虽然白盒测试是保证类和应用质量的一个关键步骤，但手工执行的难度通常会使开发人员望而却步或草草了事。有效地执行白盒测试需要我们能够确定要完全检查被测类那些测试用例是必需的，这对于手工测试来说是太难了。Jtest 使用独特的技术完全自动化白盒测试过程。Jtest 分析每个被测类的内部结构，自动设计和执行能够完全测试类的结构的测试用例，然后确定每个测试输入是否会产生一个未捕获的运行时异常。对于检测到的每个异常报告一个错误信息并提供一个导致错误的栈轨迹和调用序列。

这只是 Jtest 的白盒测试功能，它还有静态代码分析，黑盒测试，回归测试等功能。