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：2019 年 11 月 6 日

学生姓名：姜子玥

学生学号：71117201

东南大学 软件学院 制

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

实验背景：

在 Web 服务等应用中，CGI(Common Gateway Interface)是用户访问服务器端 Web 页面内容的一种传输标准。有关 CGI 的文档详见：
http://en.wikipedia.org/wiki/Common_Gateway_Interface
http://tools.ietf.org/html/rfc3875
http://baike.baidu.com/view/32614.htm
在应用程序开发中，常常需要将 CGI 编码的字符串解码为普通的 ASCII 字符串。
本次实验的被测程序 CgiDecode 展示了此功能的 C 语言实现。

（一）题目 1：数据流测试技术实验

实验内容：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。要求：

(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；

Decode()函数语句以及编号以及各节点对应变量类型如下：

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)	ecncoded 定义节点 decoded 定义节点
8	{	
9	char *eptr = encoded;	*eptr 定义节点、eptr 定义节点、encoded 使用节点

10	char *dptr = decoded;	*dptr 定义节点、dptr 定义节点、decoded 使用节点
11	int ok=0;	ok 定义节点
12	while (*eptr)	*eptr 使用节点、eptr 使用节点
13	{	
14	char c;	c 定义节点
15	c = *eptr;	eptr 使用节点、*eptr 使用节点 c 定义节点
16	if (c == '+')	c 使用节点
17	{ /* Case 1: '+' maps to blank */	
18	*dptr = ' ';	dptr 使用节点、*dptr 定义节点
19	}	
20	else if (c == '%')	c 使用节点
21	{ /* Case 2: '%xx' is hex for character xx */	
22	int digit_high = getHexValue(*(++eptr));	digit_high 定义节点 eptr 定义使用节点 *eptr 定义使用节点
23	int digit_low = getHexValue(*(++eptr));	digit_low 定义节点 eptr 使用节点 *eptr 使用节点
24	if (digit_high == -1 || digit_low==-1) {	digit_high 使用节点 digit_low 使用节点
25	/* *dptr='?'; */	
26	ok=1; /* Bad return code */	ok 定义节点
27	} else {	
28	*dptr = 16* digit_high + digit_low;	dptr 使用节点 *dptr 定义节点 digit_high 使用节点 digit_low 使用节点
29	}	
30	} else {/* Case 3: All other characters map to themselves */	
31	*dptr = *eptr;	eptr 使用节点 *eptr 使用节点 dptr 使用节点 *dptr 定义节点
32	}	
33	++dptr;	dptr 定义使用节点、*dptr 定义节点
34	++eptr;	eptr 定义使用节点 、*eptr 定义节点
35	}	
36	*dptr = '\0'; /* Null terminator for string */	dptr 使用节点 *dptr 定义节点
37	return ok;	ok 使用节点
38	}	

总结来说，对各个变量有以下结论：

变量名	定义节点	使用节点
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	24，28

（2）各个变量的 du-path 和 dc-path 如下表所示，其中被线划去的为可约简 path

变量名	du-path	dc-path
encoded	7-9	7-9
decoded	7-10	7-10
*eptr	~~9-10-11-12~~	~~9-10-11-12~~
	~~9-10-11-12-14-15~~	~~9-10-11-12-14-15~~
	~~9-10-11-12-14-15-16-20-22 （无穷多）~~	9-10-11-12-14-15-16-20-22
	9-10-11-12-14-15-16-20-22-23 （无穷条）	~~9-10-11-12-14-15-16-20-22-23~~
	9-10-11-12-14-15-16-20-30-31（无穷条）	9-10-11-12-14-15-16-20-30-31
	~~22~~	22
	~~22-23~~	~~22-23~~
	~~22-23-24-26-33-34-12~~	~~22-23-24-26-33-34-12~~
	22-23-24-26-33-34-12-14-15-16-20-30-31（无穷条）	~~22-23-24-26-33-34-12-14-15-16-20-30-31~~
	~~23~~	23
	~~23-24-26-33-34-12~~	~~23-24-26-33-34-12~~
	~~23-24-28-33-34-12~~	~~23-24-28-33-34-12~~
	~~23-24-26-33-34-12-14-15~~	~~23-24-26-33-34-12-14-15~~
	~~23-24-28-33-34-12-14-15~~	~~23-24-28-33-34-12-14-15~~
	23-24-26-33-34-12-14-15-16-20-22 （无穷条）	~~23-24-26-33-34-12-14-15-16-20-22~~
	23-24-28-33-34-12-14-15-16-20-22-23	~~23-24-28-33-34-12-14-15-16-20-22-23~~
	23-24-26-33-34-12-14-15-16-20-30-31	~~23-24-26-33-34-12-14-15-16-20-30-31~~
	~~34-12~~	~~34-12~~
	~~34-12-14-15~~	~~34-12-14-15~~

	~~34-12-14-15-16-20-22~~	34-12-14-15-16-20-22
	34-12-14-15-16-20-22-23	~~34-12-14-15-16-20-22-23~~
	~~34-12-14-15-16-20-31~~	34-12-14-15-16-20-31
eptr	~~9-10-11-12~~	~~9-10-11-12~~
	~~9-10-11-12-14-15~~	~~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
	~~9-10-11-12-14-15-16-20-22-23（无穷条）~~	~~9-10-11-12-14-15-16-20-22-23~~
	~~9-10-11-12-14-15-16-20-31（无穷条）~~	~~9-10-11-12-14-15-16-20-31~~
	9-10-11-12-14-15-16-20-22-23-24-26-33-34	~~9-10-11-12-14-15-16-20-22-23-24-26-33-34~~
	9-10-11-12-14-15-16-20-30-31-33-34	9-10-11-12-14-15-16-20-30-31-33-34
	9-10-11-12-14-15-16-18-33-34	9-10-11-12-14-15-16-18-33-34
	~~22~~	~~22~~
	~~22-23~~	22-23
	~~22-23-24-26-33-34~~	~~22-23-24-26-33-34~~
	~~22-23-24-26-33-34-12~~	~~22-23-24-26-33-34-12~~
	~~22-23-24-26-33-34-12-14-15~~	~~22-23-24-26-33-34-12-14-15~~
	22-23-24-26-33-34-12-14-15-16-20-31（无穷条）	~~22-23-24-26-33-34-12-14-15-16-20-31~~
	~~22-23-24-28-33-34-12-14-15-16-20-31~~	~~22-23-24-28-33-34-12-14-15-16-20-31~~
	~~23~~	~~23~~
	~~23-24-26-33-34~~	23-24-26-33-34
	~~23-24-26-33-34-12~~	~~23-24-26-33-34-12~~
	~~23-24-26-33-34-12-14-15~~	~~23-24-26-33-34-12-14-15~~
	~~23-24-28-33-34-12-14-15~~	~~23-24-28-33-34-12-14-15~~
	23-24-26-33-34-12-14-15-16-20-31（无穷条）	~~23-24-26-33-34-12-14-15-16-20-31~~
		~~23-24-28-33-34-12-14-15-16-20-31~~
	~~34~~	~~34~~
	~~34-12~~	~~34-12~~
	~~34-12-14-15~~	~~34-12-14-15~~
	34-12-14-15-16-20-30-31	34-12-14-15-16-20-30-31
	34-12-14-15-16-20-22-23	~~34-12-14-15-16-20-22-23~~
*dptr	无	无
dptr	10-11-12-36	10-11-12-36
		10-11-12-14-15-16-18
	~~10-11-12-14-15-16-20-31~~	~~10-11-12-14-15-16-20-31~~
	10-11-12-14-15-16-20-31（无穷条）	10-11-12-14-15-16-20-31
	~~10-11-12-14-15-16-20-22-23-24-28~~	~~10-11-12-14-15-16-20-22-23-24-28~~
	10-11-12-14-15-16-18（无穷条）	
	10-11-12-14-15-16-20-22-23-24-28（无穷条）	10-11-12-14-15-16-20-22-23-24-28
	10-11-12-14-15-16-20-22-23-24-26	10-11-12-14-15-16-20-22-23-24-26
	~~33~~	~~33~~

	33-34-12-36	33-34-12-36
	33-34-12-14-15-16-18	33-34-12-14-15-16-18
	33-34-12-14-15-16-20-30-31	33-34-12-14-15-16-20-30-31
	33-34-12-14-15-16-20-22-23-24-28	33-34-12-14-15-16-20-22-23-24-28
ok	11-12-36-37（无穷条）	11-12-36-37（无穷条）
	26-33-34-12-36-37（无穷条）	26-33-34-12-36-37（无穷条）
c	14-15-16-20	~~14-15-16-20~~
	~~14-15-16~~	~~14-15-16~~
	~~15-16-20~~	15-16-20
digit_high	22-23-24-28	22-23-24-28
	22-23-24	~~22-23-24~~
digit_low	23-24-28	23-24-28
	23-24	~~23-24~~

(3)根据 dc-path 对 decoded（）设计测试用例如下：

变量	dc-path	输入	期望输出	实际输出
encode	7-9	“abc”	0	0
decoded	7-10	“zmn”	0	0
*eptr	9-10-11-12-14-15-16-20-22	“%741”	0	0
	9-10-11-12-14-15-16-20-31	“12”	0	0
	22	“%12”	0	0
	23	“%34”	0	0
	34-12-14-15-16-20-22	“%333”	0	0
	34-12-14-15-16-20-31	“12”	0	0
eptr	9-10-11-12-14-15-16-20-22	“%111”	0	0
	9-10-11-12-14-15-16-20-30-31-33-34	“qaz”	0	0
	9-10-11-12-14-15-16-18-33-34	“+7891”	0	0
	22-23	“%135”	0	0
	23-24-26-33-34	“%741”	0	0
	34-12-14-15-16-20-31	“qaz”	0	0
dptr	10-11-12-36	“”	0	0
	10-11-12-14-15-16-18	“+7891”	0	0
	10-11-12-14-15-16-20-31	“74185”	0	0
	10-11-12-14-15-16-20-22-23-24-28	“%7ax5”	0	0
	10-11-12-14-15-16-20-22-23-24-26	“%x”	1	1
	33-34-12-36	“1”	0	0
	33-34-12-14-15-16-18	“+45”	0	0
	33-34-12-14-15-16-20-31	“746232”	0	0
	33-34-12-14-15-16-20-22-23-24-28	“%92d03”	0	0
ok	11-12-36-37	“”	0	0

	26-33-34-12-36-37	“%x”	1	1
digit_high	22-23-24-28	“%a123”	0	0
digit_low	23-24-28	“%b123”	0	0

三、实验体会

（1） 通过测试，是否发现程序中存在的缺陷？

通过测试，没有发现程序中存在的缺陷

（2） 谈谈数据流测试和控制流测试的区别和联系。

数据流测试主要关注程序中一些变量的定义、使用（如变量的声明、调用、空间的生气等），在测试中利用观察程序中出现的变量是否按照预期的情况在变化来发现程序中是否存在缺陷；

控制流测试主要关注于程序中的逻辑，这些逻辑由一些判定以及组成条件构成，这些判定影响程序的走向，控制流测试着眼于这些判定，在这些节点处是否有正确的走向来判断程序中是否存在缺陷

联系：程序整体逻辑是一个大的框架，有程序中各个判定以及组成条件是构成的节点，节点间的通路构成数据走向；数据从输入到输出是在这个框架中流动。因此这两种测试方法，虽然测试角度不同，但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代码有深刻的理解才能进行测试。

（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？

我认为应该如同编译器一样设计，理解词法、句法、语法，这样才能识别程序的整体逻辑、整体框架，然后才能针对不同的测试准则设计不同的测试用例。

难点在于：程序中不可避免的存在复杂的逻辑结构、多个循环、判断的嵌套，同时互相之间可能还存在关联。因此，充分理解程序整体逻辑以及数据的流向首先最大的问题，其次在理解后的基础上，根据不同的准则如何规范化设计出针对的测试用例更是一大难关。