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： 2018/11/8

学生姓名： 张睦婕

学生学号： 71117133

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、 实验环境

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

三、 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。

题目 1: 数据流测试技术实验

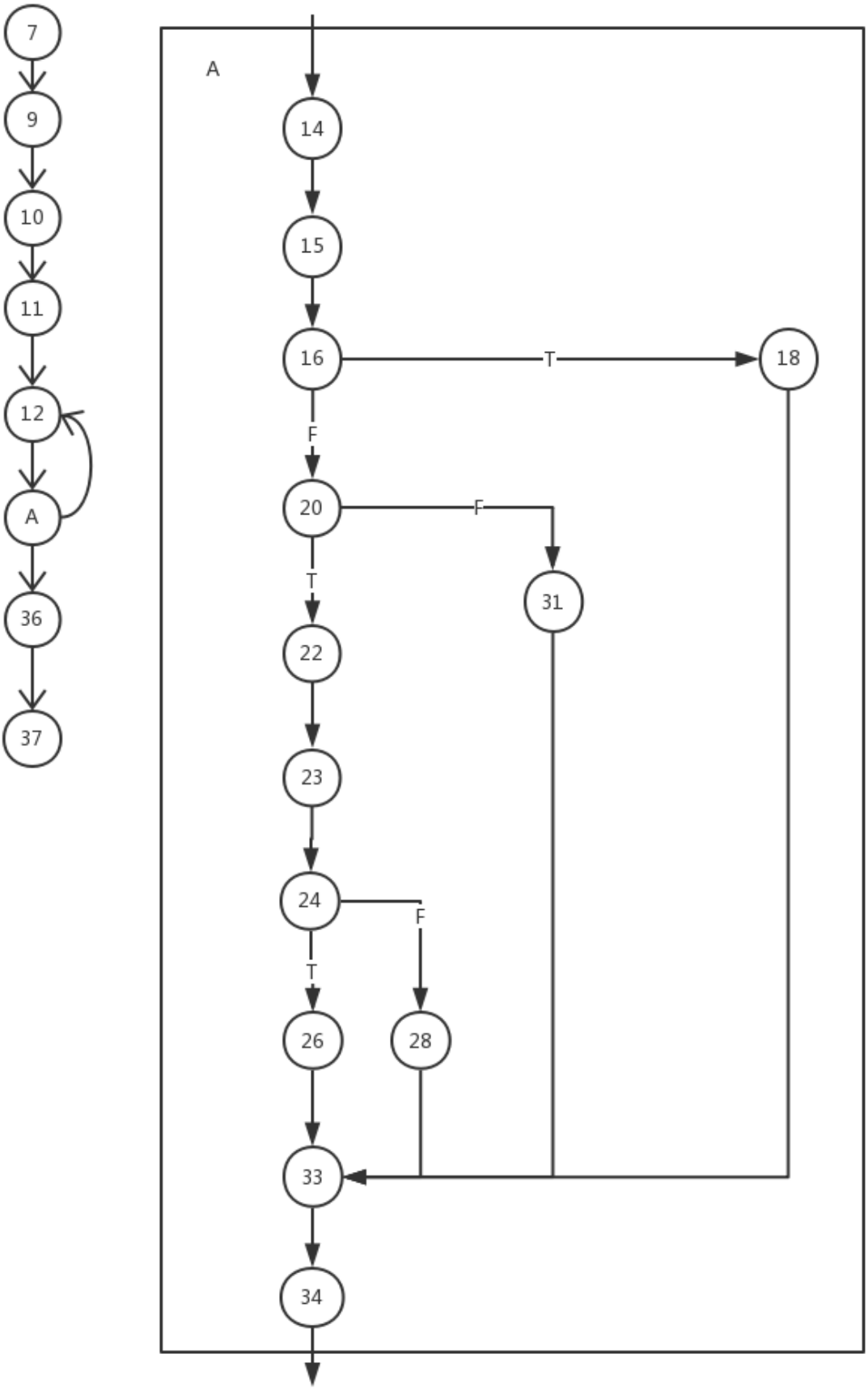
1. 题目

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 || digit_low==-1) {
25                                 /* *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. decoded（）的流程图


7
9
10
11
12
A
36
37
A
14
15
16
T
18
F
20
F
T
31
22
23
24
F
T
26
28
33
34

3. encoded 变量

Node	Type	Code
7	DEF	int decode(char *encoded, char *decoded)
9	USE	char *eptr = encoded;

encoded 的 DU-path:

P1: 7,9

DC-path:

P1: 7,9

4. decoded 变量

Node	Type	Code
7	DEF	int decode(char *encoded, char *decoded)
10	USE	char *dptr = decoded;

decoded 的 DU-path:

P1: 7,9,10

DC-path:

P1: 7,9,10

5. *eptr 变量

Node	Type	Code
9	DEF	char *eptr = encoded;
12	USE	while (*eptr)
15	USE	c = *eptr;
22	DEF	int digit_high = getHexValue(*(++eptr));
22	USE	int digit_high = getHexValue(*(++eptr));
23	DEF	int digit_low = getHexValue(*(++eptr));
23	USE	int digit_low = getHexValue(*(++eptr));
31	USE	*dptr = *eptr;
34	DEF	++eptr;

DU-path:

P1: 9,10,11,12
P2: 9,10,11,12,14,15
P3: 9,10,11,12,14,15,16,20,22
P4: 9,10,11,12,14,15,16,20,22,23
P5: 9,10,11,12,14,15,16,20,31
P6: 22
P7: 22,23
P8: 23

DC-path:

P1: 9,10,11,12
P2: 9,10,11,12,14,15
P5: 9,10,11,12,14,15,16,20,31
P6: 22
P8: 23

6. eptr 变量

Node	Type	Code
9	DEF	char *eptr = encoded;
12	USE	while (*eptr)
15	USE	c = *eptr;
22	DEF	int digit_high = getHexValue(*(++eptr));
22	USE	int digit_high = getHexValue(*(++eptr));
23	DEF	int digit_low = getHexValue(*(++eptr));
23	USE	int digit_low = getHexValue(*(++eptr));
31	USE	*dptr = *eptr;
34	DEF	++eptr;
34	USE	++eptr;

DU-path:

P1: 9,10,11,12
P2: 9,10,11,12,14,15
P3: 9,10,11,12,14,15,16,20,22
P4: 9,10,11,12,14,15,16,20,22,23
P5: 9,10,11,12,14,15,16,20,31
P6: 9,10,11,12,14,15,16,20,22,23,24,26,33,34
P7: 9,10,11,12,14,15,16,20,22,23,24,28,33,34
P8: 9,10,11,12,14,15,16,18,33,34

P9: 9,10,11,12,14,15,16,20,31,33,34
P10: 22
P11: 22,23
P12: 22,23,24,26,33,34
P13: 22,23,24,28,33,34
P14: 23
P15: 23,24,26,33,34
P16: 23,24,28,33,34
P17: 34

CU-path:

P1: 9,10,11,12
P2: 9,10,11,12,14,15
P5: 9,10,11,12,14,15,16,20,31
P10: 22
P14: 23
P17: 34

7. *dptr 变量

Node	Type	Code
10	DEF	char *dptr = decoded;
18	DEF	*dptr = ' ';
28	DEF	*dptr = 16* digit_high + digit_low;
31	DEF	*dptr = *eptr;
33	DEF	++dptr;
36	DEF	*dptr = '\0';

DU-path 不存在

8. dptr 变量

Node	Type	Code
10	DEF	char *dptr = decoded;
18	USE	*dptr = ' ';
28	USE	*dptr = 16* digit_high + digit_low;
31	USE	*dptr = *eptr;
33	DEF	++dptr;

33	USE	++dptr;
36	USE	*dptr = '\0';

DU-path:

P1: 10,11,12,14,15,16,18
P2: 10,11,12,14,15,16,20,22,23,24,28
P3: 10,11,12,14,15,16,20,31
P4: 10,11,12,14,15,16,20,22,23,24,26,33
P5: 10,11,12,14,15,16,20,22,23,24,28,33
P6: 10,11,12,14,15,16,18,33,34,36
P7: 10,11,12,14,15,16,20,31,33,34,36
P8: 10,11,12,14,15,16,20,22,23,24,26,33,34,36
P9: 10,11,12,14,15,16,20,22,23,24,28,33,34,36
P10: 33
P11: 33,34,36

P8: 10,11,12,14,15,16,20,22,23,24,26,33,34,36
P9: 10,11,12,14,15,16,20,22,23,24,28,33,34,36

CU-path:

P1: 10,11,12,14,15,16,18
P2: 10,11,12,14,15,16,20,22,23,24,28
P3: 10,11,12,14,15,16,20,31
P10: 33
P11: 33,34,36

9. ok 变量

Node	Type	Code
11	DEF	int ok=0;
26	DEF	ok=1; /* Bad return code */
37	USE	return ok;

DU-path:

P1: 11,12,14,15,16,18,33,34,36,37
P2: 11,12,14,15,16,20,31,33,34,36,37
P3: 11,12,14,15,16,20,22,23,24,26,33,34,36,37
P4: 11,12,14,15,16,20,22,23,24,28,33,34,36,37
P5: 26,33,34,36,37

CU-path:

P1: 11,12,14,15,16,18,33,34,36,37

P2: 11,12,14,15,16,20,31,33,34,36,37

P4: 11,12,14,15,16,20,22,23,24,28,33,34,36,37

P5: 26,33,34,36,37

10.c 变量

Node	Type	Code
14	DEF	char c;
15	DEF	c = *eptr;
16	USE	if (c == '+')
20	USE	else if (c == '%')

DU-path:

P1: 14,15,16

P2: 14,15,16,20

P3: 15,16

P4: 15,16,20

CU-path:

P3: 15,16

P4: 15,16,20

11.digit_high 变量

Node	Type	Code
22	DEF	int digit_high = getHexValue(*(++eptr));
24	USE	if (digit_high == -1 || digit_low==-1)
28	USE	*dptr = 16* digit_high + digit_low;

DU-path:

P1: 22,23,24

P2: 22,23,24,28

CU-path:

P1: 22,23,24

P2: 22,23,24,28

12.digit_low 变量

Node	Type	Code
23	DEF	int digit_low = getHexValue(*(++eptr));
24	USE	if (digit_high == -1 || digit_low==-1) {
28	USE	*dptr = 16* digit_high + digit_low;

DU-path:

P1: 23,24

P2: 23,24,28

CU-path:

P1: 23,24

P2: 23,24,28

13.设计测试用例

以上变量的定义清除路径以及相应的测试用例如下（黑色的路径是被其他路径包含的）：

encoded

P1: 7,9

编号	变量名	测试用例	预期输出		实际输出		dc-path
		Encoded 串	ok	decoded 串	ok	decoded 串	
EN1	encoded	%61	0	a	0	a	7,9

decoded

P1: 7,9,10

编号	变量名	测试用例	预期输出		实际输出		dc-path
		Encoded 串	ok	decoded 串	ok	decoded 串	
DE1	decoded	%61	0	a	0	a	7,9,10

***eptr**

P1: 9,10,11,12

P2: 9,10,11,12,14,15

P5: 9,10,11,12,14,15,16,20,31

P6: 22

P8: 23

编号	变量名	测试用例	预期输出		实际输出		dc-path
		Encoded 串	ok	decoded 串	ok	decoded 串	
PEP1	*eptr	1	0	1	0	1	9,10,11,12,14,15,16,20,31
PEP2	*eptr	%411	0	A1	0	A1	22
PEP3	*eptr	%61%31	0	a1	0	a1	23

eptr

P1: 9,10,11,12

P2: 9,10,11,12,14,15

P5: 9,10,11,12,14,15,16,20,31

P10: 22

P14: 23

P17: 34

编号	变量名	测试用例	预期输出		实际输出		dc-path
		Encoded 串	ok	decoded 串	ok	decoded 串	
EP1	eptr	1	0	1	0	1	9,10,11,12,14,15,16,20,31
EP2	eptr	%411	0	A1	0	A1	22
EP3	eptr	%61%31	0	a1	0	a1	23
EP4	eptr	%41+	0	A	0	A	34

dptr

P1: 10,11,12,14,15,16,18

P2: 10,11,12,14,15,16,20,22,23,24,28

P3: 10,11,12,14,15,16,20,31

P10: 33

P11: 33,34,36

编号	变量名	测试用例	预期输出		实际输出		dc-path
		Encoded 串	ok	decoded 串	ok	decoded 串	
DP1	dptr	+	0	_	0	_	10,11,12,14,15,16,18
DP2	dptr	%61	0	a	0	a	10,11,12,14,15,16,20,22,23,24,28
DP3	dptr	1	0	1	0	1	10,11,12,14,15,16,20,31,32,33,34,35,36

ok

P1: 11,12,14,15,16,18,33,34,36,37

P2: 11,12,14,15,16,20,31,33,34,36,37

P4: 11,12,14,15,16,20,22,23,24,28,33,34,36,37

P5: 26,33,34,36,37

编号	变量名	测试用例	预期输出		实际输出		dc-path
		Encoded 串	ok	decoded 串	ok	decoded 串	
OK1	ok	+	0	_	0	_	11,12,14,15,16,18,33,34,36,37
OK2	ok	1	0	1	0	1	11,12,14,15,16,20,31,33,34,36,37
OK3	ok	%61	0	a	0	a	11,12,14,15,16,20,22,23,24,28,33,34,36,37
OK4	ok	%%%	1		1		26,33,34,36,37

c

P3: 15,16

P4: 15,16,20

编号	变量名	测试用例	预期输出		实际输出		dc-path
		Encoded 串	ok	decoded 串	ok	decoded 串	
C1	c	%61%31	0	a1	0	a1	15,16,20

digit_high

P1: 22,23,24

P2: 22,23,24,28

编号	变量名	测试用例	预期输出		实际输出		dc-path
		Encoded 串	ok	decoded 串	ok	decoded 串	
DH1	digit_high	%61%31	0	a1	0	a1	22,23,24,28

digit_low

P1: 23,24

P2: 23,24,28

编号	变量名	测试用例	预期输出		实际输出		dc-path
		Encoded 串	ok	decoded 串	ok	decoded 串	
DL1	digit_low	%61%31	0	a1	0	a1	23,24,28

四、 实验体会

1. 数据流测试会出现很多包含与被包含的路径，可以约简；对于不同的数据也存在重复的测试用例，因此找到关键有效的用例可以有效控制测试用例的数目。
2. 数据流测试在变量较多的情况下就会变得很复杂，不适用；但也存在着对密集的程序的更强的测试能力。
3. 对于指针，在数据流测试中存在以下容易混淆的情况：

	*p	p
int *p	DEF	DEF
p++	DEF	DEF/USE
get(*(++p))	DEF/USE	USE/DEF
c=*p	USE	USE
*p=” ”	DEF	USE

这些细小的差别会使 p 和*p 的路径有很大的差别