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：白盒测试实验一

实验地点：计算机楼 268 机房

实验日期：2019 年 11 月 6 日

学生姓名：唐珞鑫

学生学号：71117106

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实验测试用例
（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。

实验一：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。

要求：

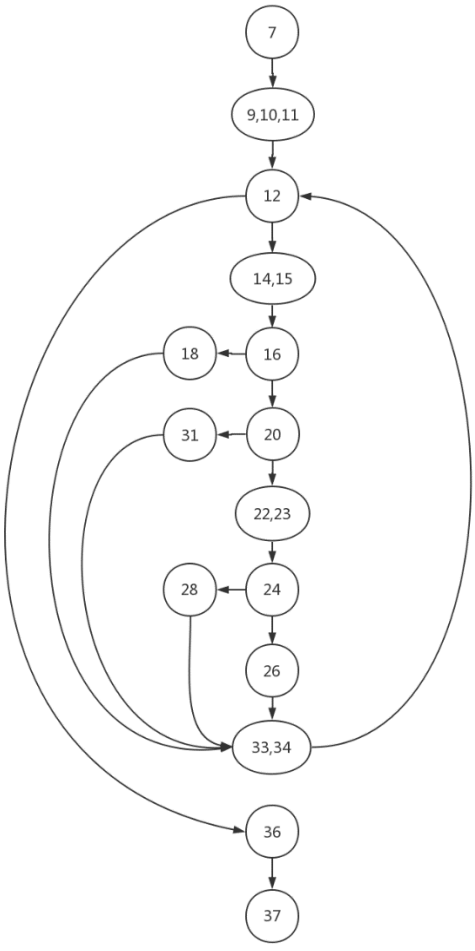
(1) 测试要考虑 decode()中 encoded, decoded, *eptr, eptr, *dptr, dptr, ok, c, digit_high, digit_low 变量；
(2) 给出每个变量对应的 du-path 和 dc-path;
(3) 根据变量的 dc-path 设计测试用例，完成对 decode()的测试；

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  }
```

（一）对于给定的程序，构造相应的程序图



（二）分析定义结点和使用节点，找出所有变量的定义-使用路径，并设计测试用例

encoded

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

可能的路径条数：1x1=1

P1:7-9

设计测试用例如下:

编号	执行条件	输入	期望输出	实际输出	路径
01	数据流覆盖测试	'+',' '	ok=0	ok=0	7-9

decoded

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

可能的路径条数：1x1=1

P1:7-9-10

设计测试用例如下:

编号	执行条件	输入	期望输出	实际输出	路径
01	数据流覆盖测试	'+','='	ok=0	ok=0	7-9-10

*eptr

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

可能的路径数：4x5=20

P1:9-10-11-12 (DC)

P2:9-10-11-12-14-15 (DC)

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 (DC)
P6:22-23-24-26-33-34-12 或 22-23-24-28-33-34-12
P7:22-23-24-26-33-34-12-14-15 或 22-23-24-28-33-34-12-14-15
P8:22 (DC)
P9:22-23
P10:22-23-24-26-33-34-12-14-15-16-20-31 或 22-23-24-28-33-34-12-14-15-16-20-31
P11:23-24-26-33-34-12 或 23-24-28-33-34-12
P12: 23-24-26-33-34-12-14-15 或 23-24-26-33-34-12-14-15
P13:23-24-26-33-34-12-14-15-16-20-22 或 23-24-28-33-34-12-14-15-16-20-22
P14:23 (DC)
P15:23-24-26-33-34-12-14-15-16-20-31 或 23-24-28-33-34-12-14-15-16-20-31
P16:34-12 (DC)
P17:34-12-14-15 (DC)
P18:34-12-14-15-16-20-22
P19: 34-12-14-15-16-20-23
P20:34-12-14-15-16-20-31 (DC)

约简得到(根据实验要求选择 DC 路径)：
P1: 9-10-11-12-14-15-16-20-31
P2: 22
P3: 23
P4: 34-12-14-15-16-20-31

设计测试用例如下:

编号	执行条件	输入	期望输出	实际输出	路径
01	数据流覆盖测试	’$$’,’=’	ok=0	ok=0	9-10-11-12-14-15-16-20-31
02	数据流覆盖测试	无符合条件的测试用例			22
03	数据流覆盖测试	无符合条件的测试用例			23
04	数据流覆盖测试	无符合条件的测试用例			34-12-14-15-16-20-31

eptr

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

31	USE	`*dptr = *eptr;`
34	DEF，USE	`++eptr;`

可能的路径数：4x6=24

P1:9-10-11-12 (DC)

P2:9-10-11-12-14-15 (DC)

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 (DC)

P6:9-10-11-12-14-15-16-18-33-34 或 9-10-11-12-14-15-16-20-22-23-24-26-33-34 或 9-10-11-12-14-15-16-20-22-23-24-28-33-34 或 9-10-11-12-14-15-16-20-31-33-34

P7:22-23-24-26-33-34-12 或 22-23-24-28-33-34-12

P8:22-23-24-26-33-34-12-14-15 或 22-23-24-28-33-34-12-14-15

P9:22 (DC)

P10:22-23

P11:22-23-24-26-33-34-12-14-15-16-20-31 或 22-23-24-28-33-34-12-14-15-16-20-31

P12:22-23-24-26-33-34 或 22-23-24-28-33-34

P13:23-24-26-33-34-12 或 23-24-28-33-34-12

P14:23-24-26-33-34-12-14-15 或 23-24-28-33-34-12-14-15

P15:23-24-26-33-34-12-14-15-16-20-22 或 23-24-28-33-34-12-14-15-16-20-22

P16:23 (DC)

P17:23-24-26-33-34-12-14-15-16-20-31 或 23-24-28-33-34-12-14-15-16-20-31

P18:23-24-26-33-34 或 23-24-28-33-34

P19:34-12 (DC)

P20:34-12-14-15 (DC)

P21:34-12-14-15-16-20-22

P22:34-12-14-15-16-20-22-23

P23:34-12-14-15-16-20-31 (DC)

P24:34 (DC)

约简得到(根据实验要求选择 DC 路径)：

P1: 9-10-11-12-14-15-16-20-31

P2:22

P3:23

P4: 34-12-14-15-16-20-31

设计测试用例如下:

编号	执行条件	输入	期望输出	实际输出	路径
01	数据流覆盖测试	’$$’,’=’	ok=0	ok=0	9-10-11-12-14-15-16-20-31
02	数据流覆盖测试	无符合条件的测试用例			22
03	数据流覆盖测试	无符合条件的测试用例			23

04	数据流覆盖测试	无符合条件的测试用例			34-12-14-15-16-20-31

*dptr

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';`

可能的路径数：0

测试用例：无

dptr

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,USE	`++dptr;`
36	USE	`*dptr = '\0';`

可能的路径数：2x5=10

P1:10-11-12-14-15-16-18 (DC)

P2:10-11-12-14-15-16-20-22-23-24-28 (DC)

P3:10-11-12-14-15-16-20-31 (DC)

P4:10-11-12-14-15-16-20-31-33 或 10-11-12-14-15-16-20-22-23-24-26-33 或 10-11-12-14-15-16-20-22-23-24-28-33

P5:10-11-12-36 (DC)或 10-11-12-14-15-16-18-33-34-12-36 或 10-11-12-14-15-16-20-22-23-24-26-33-34-12-36 或 10-11-12-14-15-16-20-22-23-24-28-33-34-12-36 或 10-11-12-14-15-16-20-31-33-34-12-36 因为有循环的存在，存在多条路径

P6:33-34-12-14-15-16-18 (DC)

P7:33-34-12-14-15-16-20-22-23-24-28 (DC)

P8:33-34-12-14-15-16-20-31 (DC)

P9:33 (DC)

P10:33-34-12-36 (DC)或 33-34-12-14-15-16-18-33-34-12-36 或 33-34-12-14-15-16-20-22-23-24-26-33-34-12-36 或 33-34-12-14-15-16-20-22-23-24-28-33-34-12-36 或 33-34-12-14-15-16-20-31-33-34-12-36 因为有循环存在，存在多条路径

约简得到(根据实验要求选择 DC 路径)：

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

P4:10-11-12-36

P5:33-34-12-14-15-16-18

P6:33-34-12-14-15-16-20-22-23-24-28

P7:33-34-12-14-15-16-20-31

设计测试用例如下:

编号	执行条件	输入	期望输出	实际输出	路径
01	数据流覆盖测试	'+','='	ok=0	ok=0	10-11-12-14-15-16-18
02	数据流覆盖测试	'%3','='	ok=0	ok=0	10-11-12-14-15-16-20-22-23-24-28
03	数据流覆盖测试	无符合条件的测试用例			10-11-12-14-15-16-20-31
04	数据流覆盖测试	' ', '='	ok=0	ok=0	10-11-12-36
05	数据流覆盖测试	'3+','='	ok=0	ok=0	33-34-12-14-15-16-18
06	数据流覆盖测试	'+%3','='	ok=0	ok=0	33-34-12-14-15-16-20-22-23-24-28
07	数据流覆盖测试	'+$','='	ok=0	ok=0	33-34-12-14-15-16-20-31

ok

Node	Type	Code
11	DEF	`int ok=0;`
26	DEF	`ok=1;`
37	USE	`return ok;`

可能的路径数：1x2=2

P1:11-12-36-37 (DC)或 11-12-14-15-16-18-33-34-12-36-37 (DC)或 11-12-14-15-16-20-22-23-24-26-33-34-12-36-37 或 11-12-14-15-16-20-22-23-24-28-33-34-12-36-37 (DC)或 11-12-14-15-16-20-31-33-34-12-36-37 (DC)因为循环的存在，存在多条路径

P2: 26-33-34-12-36-37 (DC)

约简得到(根据实验要求选择 DC 路径)：

P1: 11-12-36-37

P2: 26-33-34-12-36-37

设计测试用例如下:

编号	执行条件	输入	期望输出	实际输出	路径
01	数据流覆盖测试	' ', '='	ok=0	ok=0	11-12-36-37
02	数据流覆盖测试	'%-3', '='	ok=1	ok=1	26-33-34-12-36-37

c

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

可能的路径数：2x2=4

P1:14-15-16

P2:14-15-16-20

P3:15-16 (DC)

P4:15-16-20 (DC)

约简得到(根据实验要求选择 DC 路径)：

P1: 15-16-20

设计测试用例如下:

编号	执行条件	输入	期望输出	实际输出	路径
01	数据流覆盖测试	无符合条件的测试用例			15-16-20

digit_high

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;`

可能的路径数：1x2=2

P1:22-23-24 (DC)

P2:22-23-24-28 (DC)

约简得到(根据实验要求选择 DC 路径)：

P1: 22-23-24-28

设计测试用例如下:

编号	执行条件	输入	期望输出	实际输出	路径
01	数据流覆盖测试	'%3', '='	ok=0	ok=0	22-23-24-28

digit_low

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;`

可能的路径数：1x2=2

P1:23-24 (DC)

P2:23-24-28 (DC)

约简得到(根据实验要求选择 DC 路径)：

P1:23-24-28

设计测试用例如下:

编号	执行条件	输入	期望输出	实际输出	路径
01	数据流覆盖测试	'%3', '='	ok=0	ok=0	23-24-28

三、实验思考

(1)控制流测试是一种在考虑测试对象的控制流情况下导出测试用例的测试方法，并且借助于控制流图能评估测试的完整性（覆盖率）。数据流测试是针对程序对数据的加工处理过程进行测试，对程序的测试可从数据处理流程的角度进行考虑。数据的处理过程对应一定的控制流路径。

(2)代替手工的白盒测试工具，在设计时分为两种类型应该具有以下的功能：静态测试功能（复杂度分析、编程规范、代码优化重构、不可达路径判断、内存泄漏判断、数组越界判断），动态测试功能（模块运行时间、动态白盒测试方法、自动生成测试用例）等功能。