Calculator-CLI 测试报告

一、关于 Calculator-CLI

Calculator-CLI 是我们基于大二数据结构课程的 lab 修改而来的命令行工具，使用 C++编写，它的功能是输入一段算术表达式，计算出表达式的结果。

比如：输入：“(1+2)*(4-3)/(6%5);(ANS+3)!;ANS+5*2;”将得到输出“730”。

二、测试环境

gtest、gcov、lcov、cmake、vscode，从零搭建。

三、基于路径的测试

3.1.测试结果：

gcov 输出：

```
[100%] Built target test-calculator-cli
Running tests...
Test project /home/richard/Desktop/Software_Testing/test/build
    Start 1: output_test
1/1 Test #1: output_test .....................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 1
```

lcov 输出：

LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - /home/richard/Desktop/Software_Testing/src	Lines:	197	200	98.5 %
Test:	main_coverage.info	Functions:	27	27	100.0 %
Date:	2020-05-17 21:26:42				

Filename	Line Coverage		Functions	
calculator-cli.cpp	98.4 %	188 / 191	100.0 %	14 / 14
calculator-cli.hpp	100.0 %	9 / 9	100.0 %	13 / 13

Generated by: LCOV version 1.13

复现指南-编译运行 Calculator-CLI：

mkdir build && cd build

cmake ..

make

./calculator-cli

复现指南-编译测试：

cd test && mkdir build && cd build

cmake ..

make init

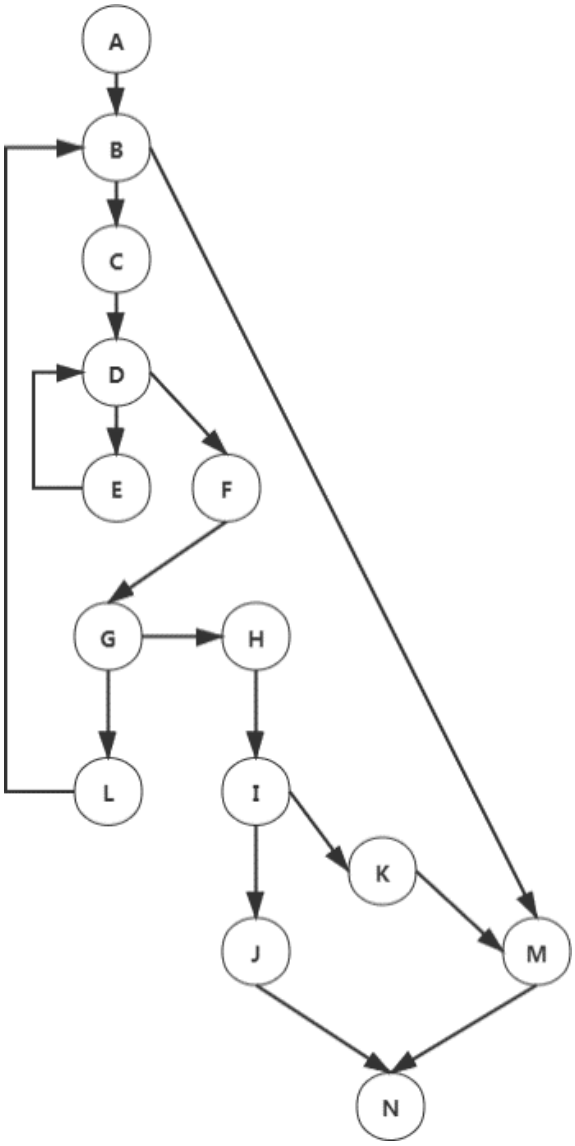
make gcov

make lcov

生成的网页位于：./test/build/lcoverage/src/index.html

3.2.测试分析：

3.2.1.Calculate：



V(G) = E – N + 2 = 17 – 14 + 2 = 5，5 条独立路径

节点和对应条件：

B	字节流为空
D	Token 类型为 print
G	Token 类型为右括号
I	字节流不为空

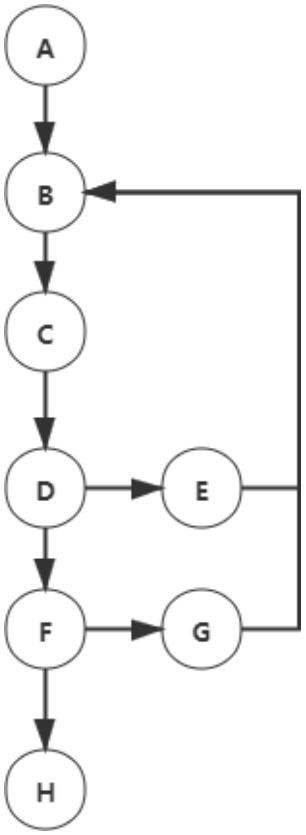
独立路径集合为：

1	A-B-M-N
2	A-B-C-D-F-G-L-B-M-N
3	A-B-C-D-F-G-H-I-J-N
4	A-B-C-D-F-G-H-I-K-M-N
5	A-B-C-D-E-D-F-G-L-B-M-N
6	A-B-C-D-E-D-F-G-H-I-J-N
7	A-B-C-D-E-D-F-G-H-I-K-M-N

测试用例设计：

测试用例	预期路径编号	预期输出	实际输出	匹配
（空）	1	（空）	（空）	是
0;	2	0	0	是
?	3	error	error	是
??	4	error	error	是
10;	5	10	10	是
0;?	6	error	error	是
0;??	7	error	error	是

3.2.2.Expression：



V(G) = E – N + 2 = 9 – 8 + 2 = 3，3 条独立路径

节点和对应条件：

D	Token 类型为加号
F	Token 类型为减号

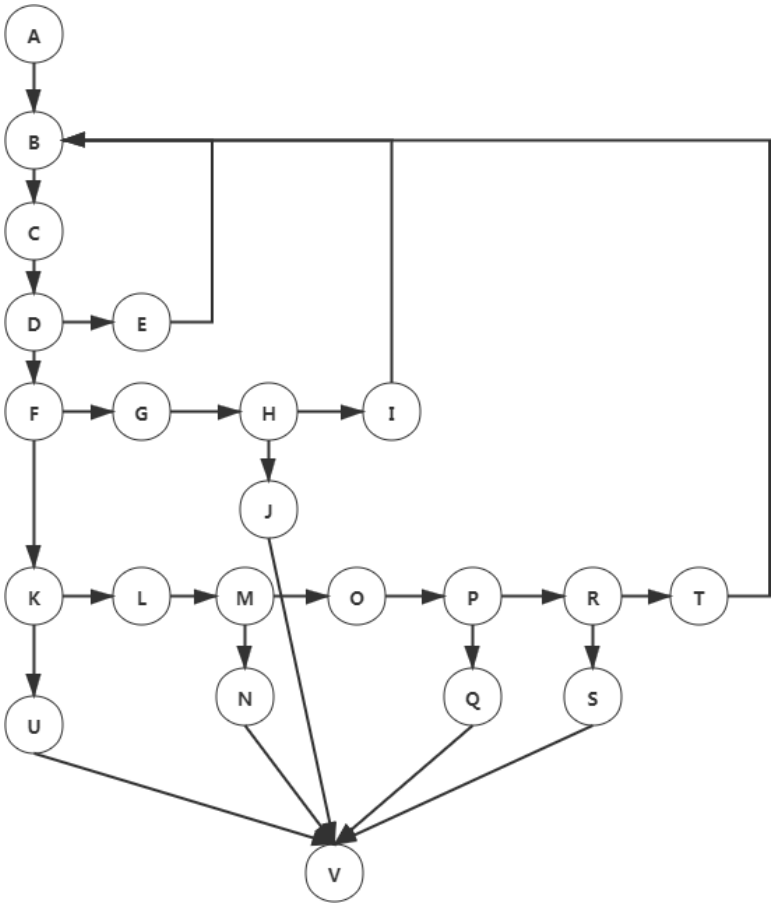
独立路径集合为：

1	A-B-C-D-F-H
2	A-B-C-D-E-B-C-D-F-H
3	A-B-C-D-F-G-B-C-D-F-H

测试用例设计：

测试用例	预期路径编号	预期输出	实际输出	匹配
1+1;	1	2	2	是
1-1;	2	0	0	是
1;	3	1	1	是

3.2.3.Term



V(G) = E – N + 2 = 28 – 22 + 2 =8，8 条独立路径

节点和对应条件：

D	Token 类型为乘号
F	Token 类型为除号
H	除数为 0
K	Token 类型为取模符号
M	取模符号的左边不为整数
P	取模符号的右边不为整数
R	取模符号的右边为 0

独立路径集合为：

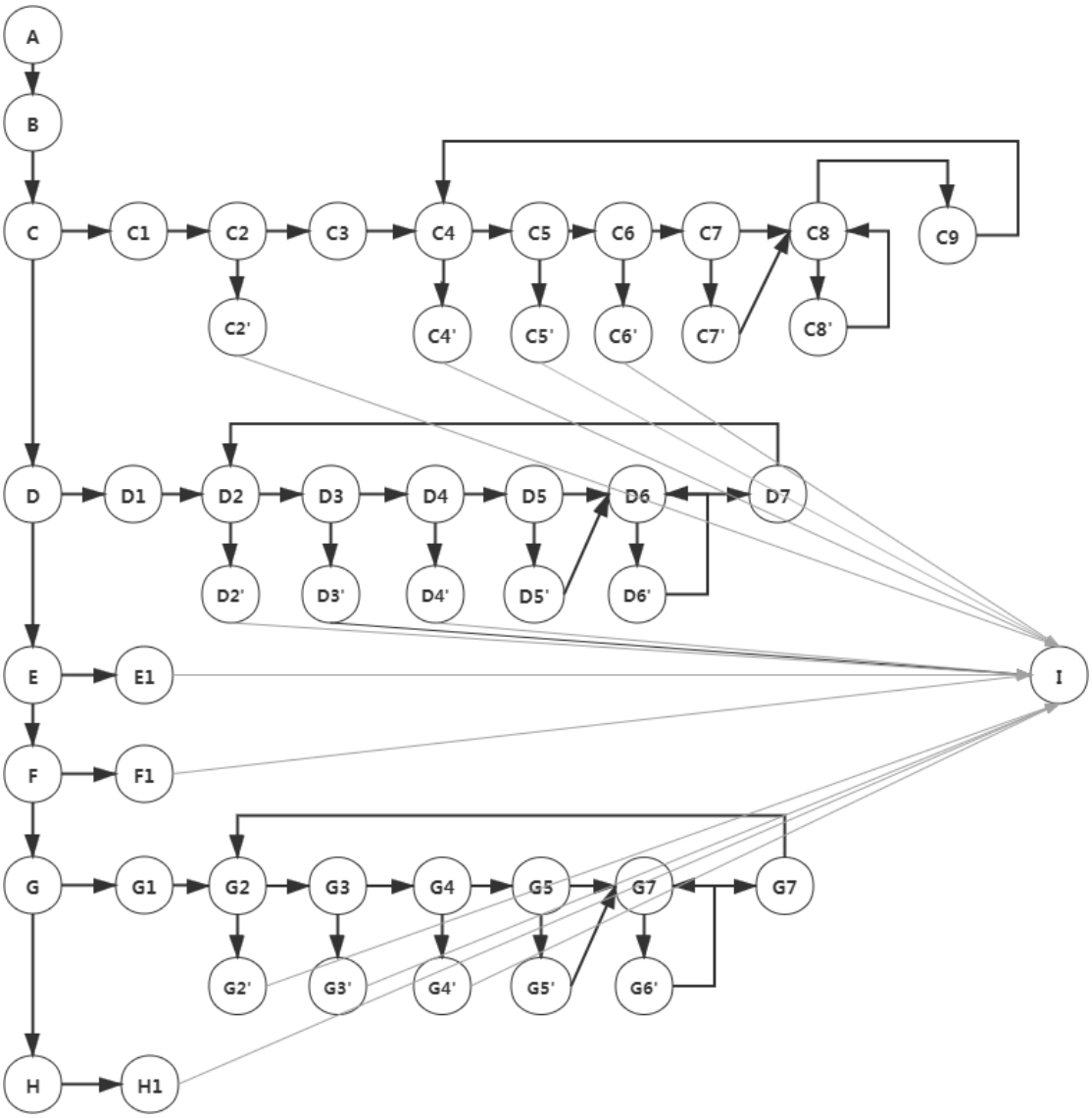
1	A-B-C-D-F-K-U-V
2	A-B-C-D-E-B-C-D-F-K-U-V
3	A-B-C-D-F-G-H-I-B-C-D-F-K-U-V
4	A-B-C-D-F-G-H-J-V
5	A-B-C-D-F-K-L-M-N-V
6	A-B-C-D-F-K-L-M-O-P-Q-V
7	A-B-C-D-F-K-L-M-O-P-R-S-V
8	A-B-C-D-F-K-L-M-O-P-R-T-B-C-D-F-K-U-V

测试用例设计：

测试用例	预期路径编号	预期输出	实际输出	匹配
4321;	1	4321	4321	是

11*11;	2	121	121	是
1/0;	3	error	error	是
3/2;	4	1.5	1.5	是
3.5%1;	5	error	error	是
3%1.5;	6	error	error	是
3%0;	7	error	error	是
5%2;	8	1	1	是

3.2.4.Primary



V(G)= E – N + 2 = 71 – 51 + 2 = 22，22 条独立路径

节点和对应条件：

C	Token 类型为左括号
C2	Token 类型不为右括号
C4	Token 类型等于阶乘符号
C5	表达式结果为整数
C6	表达式结果小于 0

C7	表达式结果等于 0
C8	阶乘处理完毕
D	Token 类型为数字
D2	Token 类型为阶乘符号
D3	Token 数字不为整数
D4	Token 数字小于 0
D5	Token 数字等于 0
D6	阶乘处理完毕
E	Token 类型为减号
F	Token 类型为加号
G	Token 类型为变量名
G2	Token 类型为阶乘符号
G3	Token 数字不为整数
G4	Token 数字小于 0
G5	Token 数字等于 0
G7	阶乘处理完毕

独立路径集合为：

1	A-B-C-C1-C2-C2'-I
2	A-B-C-C1-C2-C3-C4-C4'-I
3	A-B-C-C1-C2-C3-C4-C5-C5'-I
4	A-B-C-C1-C2-C3-C4-C5-C6-C6'-I
5	A-B-C-C1-C2-C3-C4-C5-C6-C7-C7'-I
6	A-B-C-C1-C2-C3-C4-C5-C6-C7-C8-C8'-C8-C9-C4-C4'-I
7	A-B-C-C1-C2-C3-C4-C5-C6-C7-C8-C9-C4-C4'-I
8	A-B-C-D-D1-D2-D2'-I
9	A-B-C-D-D1-D2-D3-D3'-I
10	A-B-C-D-D1-D2-D3-D4-D4'-I
11	A-B-C-D-D1-D2-D3-D4-D5-D5'-D6-D6'-D7-D2-D2'-I
12	A-B-C-D-D1-D2-D3-D4-D5-D6-D6'-D7-D2-D2'-I
13	A-B-C-D-D1-D2-D3-D4-D5-D5'-D6-D7-D2-D2'-I
14	A-B-C-D-D1-D2-D3-D4-D5-D6-D7-D2-D2'-I
15	A-B-C-D-E-E1-I
16	A-B-C-D-E-F-F1-I
17	A-B-C-D-E-F-G-G1-G2-G2'-I
18	A-B-C-D-E-F-G-G1-G2-G3-G3'-I
19	A-B-C-D-E-F-G-G1-G2-G3-G4-G4'-I
20	A-B-C-D-E-F-G-G1-G2-G3-G4-G5-G5'-I
21	A-B-C-D-E-F-G-G1-G2-G3-G4-G5-G6-G6'-G6-G7-G2-G2'-I
22	A-B-C-D-E-F-G-H-H1-I

测试用例设计：

测试用例	预期路径编号	预期输出	实际输出	匹配
(1+2*3-4/5;	1	error	error	是
(1+2*3-4/4);	2	6	6	是

(10.5)!;	3	error	error	是
(-10)!;	4	error	error	是
(1-1)!;	5	1	1	是
(1-1)!;	6	1	1	是
(1+1)!;	7	2	2	是
4321+1234;	8	5555	5555	是
10.5!;	9	error	error	是
-20!;	10	error	error	是
0!;	11	1	1	是
0!;	12	1	1	是
5!;	13	120	120	是
3!;	14	6	6	是
-1;	15	-1	-1	是
+1;	16	1	1	是
1+2;ANS+3;	17	6	6	是
3/2;ANS!;	18	error	error	是
2-9;ANS!;	19	error	error	是
1-1;ANS!;	20	1	1	是
-1+1;ANS!;	21	1	1	是
2+3;ANS!;	22	120	120	是

四、基于数据流的测试

函数	变量	定义节点	使用节点
set_value	s	25	29
	d	25	31
	i	27	27
get_value	s	38	42
	i	40	40
define_name	var	50	52
	val	50	52, 53
Token_stream::ignore	c	80	82, 91
Token_stream::putback	t	100	104
Token_stream::get	ch	116	117, 119, 130, 143, 150, 153, 154, 156, 158
	val	144	145, 146
	s	152, 153, 156	153, 156, 159, 161
primary	t	175, 181, 230	176, 182, 184, 216, 220, 224, 226, 228, 230, 235, 270
	d	180, 204	190, 194, 198, 200, 202, 204, 209
	n	187, 206, 213, 232,	188, 208, 214, 234, 245,

		244, 263	265
	tmp	243, 261	247, 251, 255, 257, 259, 261, 266
term	left	278, 293, 314	285, 293, 300, 301, 321
	t	279, 286, 294, 315	282, 320
	d	290, 299	291, 293, 305, 306
	i1	300	301, 314
	i2	305	306, 310 314
expression	left	329, 341	337, 341, 346
	t	330, 338, 342	334, 345
calculate	input	356	358
	tmp	359, 370	381
	t	364, 367, 371	365, 369, 372, 374, 379
	retval	392, 395	396
	ss	393, 394	395
main	input	402, 403	404
	answer	404	405

五、测试结论

从测试结果来看，我们的 lab 写得蛮好的，改得也蛮好的，各种用例在单独测试的情况下都能得到预期的结果。但是在测试的过程中也的确找到了一些问题（现已修复，所以可能数据流测试的行号不一定对应得上），比如在改成 command line interface 后，输入从读取一个文件变成了读取一个输入的字符串，并且在反复调用的过程中程序不会退出，而一开始的版本中我们抛出异常的代码写在了脏数据处理的前面，而且 calculate 函数在开始前也没有对初始数据进行清空操作，导致上一次的测试影响到了下一次的测试出现单独测试可以通过，一起测试就不通过的情况。另外在测试的过程中我们也学到了如何在没有 IDE 的帮助下手动搭建 C++程序的测试环境，本次大作业受益匪浅。

六、错误修正

对通过测试被发现的两处隐藏错误的修正如下，修改后能够正常通过连续的测试：

3 White-Box Testing/src/calculator-cli.cpp

```
@@ -348,6 +348,9 @@ void clean_up_mess()

 string Calculator::calculate(string input)
 {
+    var_table.clear();
+    ipt.clear();
+
     define_name("ANS", 0);
     ipt << input;
```

2 White-Box Testing/src/calculator-cli.cpp

```
@@ -380,11 +380,11 @@ string Calculator::calculate(string input)
380 380             }
381 381             catch (exception &e)
382 382             {
    383 +               clean_up_mess();
383 384               if (!ipt.eof())
384 385               {
385 386                   return e.what();
386 387               }
387     -               clean_up_mess();
388 388             }
389 389         }
390 390         string retval;
```