ECE 322
Lab Report 1

Arun Woosaree
XXXXXXX

September 27, 2019

Introduction

The purpose of this lab was to serve as a practical introduction to rudimentary black-box testing techniques. The testing methods introduced were dirty testing, error guessing, and partition-based testing. It should be noted that numerous other black-box testing methods exist The idea of black-box testing is that tests are carried out with no knowledge of how the software internally works. In other words, the implementation details are a "black box" as the name would suggest. Dirty testing and error guessing involves using creativity to come up with test cases, and also using past experiences to come up with test cases to find faults in the program. THe purpose of partition-based testing is to categorize possible test cases in 'equivalence' classes, and to test as many valid equivalence classes with as few test cases, and to come up with a test case for each invalid equivalence class. The goal for partition-based testing is to lower the number of test cases.

Part 1 - Failure/Dirty Testing, Error Guessing

For task one in this lab, we had to be creative, as is the nature of Failure/Dirty testing, and error guessing. The purpose was to test the functionality of a calculator program, which was written in Java. A table of test cases was produced, checking for basic functionality, common errors. A few test cases were also made based on previous experience, which is also known as error guessing. Altogether, the test cases check for the following functionality:

1. whether the calculator buttons work
2. non-numerical input
3. mismatched brackets
4. order or operations (BEDMAS/PEMDAS)

5. large numbers
6. small numbers
7. incorrect syntax (e.g. 2++2)

The full list of test cases, along with the inputs and expected versus actual outputs can be found in Appendix A. The test cases where the expected result does not match the actual result are highlighted in red. From experience, these test failures could be due to the following:

- numbers like 2^{512} are ridiculously big. It's likely that that the implementation was not designed to handle such large numbers.
- similarly, numbers like 2^{-512} are so small that the calculator interprets it as being the number 0
- entering nothing results in an output of 0. It's likely that a lone space is interpreted as a 0 by the calculator
- if there's a space between a number and the minus sign after it, the calculator outputs NaN. It could be that when parsing the input, the calculator may be interpreting it as two numbers with no operator in between, so it does not know what to do and returns an error (NaN)
- the calculator does not follow the correct order of operations. If there's a division and multiplication. (e.g. $80/4 * 5$), it appears to be doing the multiplication before the division, even though the multiplication should be done afterwards, since it is on the right hand side of the previous division operation.
- when multiple exponent (^) operators are used in between two operands, the result is always one, instead of showing an error. This might be because the right half is evaluated as 0 by the calculator, and anything to the power of 0 is 1.
- when the first operand is missing for + the calculator seems to ignore it and treat it like a positive number. This is why the calculator says $2 + + + +2 = 4$
- when the first operand is missing for $*$, /, or ^, the calculator evaluates it as being equal to 0 instead of returning an error.
- with the exponent operator, the calculator tries to evaluate everything after the exponent. however, this violates the BEDMAS rule in cases where the exponent should be done before multiplication, division, addition, or subtraction. This results in errors like $2^1 + 1$ resulting in 4 instead of 3. This also compounds with the previous errors in some cases where an expression like $2^3 + 2^3$ is incorrectly evaluated as 2^5, since $3 + 2$ is evaluated first, which results in ^3 being evaluated as 1, and the expression is reduced to 2^5, resulting in the incorrect response.

- when numbers are only separated by a space, the calculator concatenates the digits. i.e. 1 2 becomes 12, instead of showing an error. when these numbers are wrapped by brackets, the calculator still concatenates them instead of multiplying. e.g. (2)(2) becomes 22 instead of 4. However, multiplying negative numbers using brackets e.g. (2)(-2) instead of the * operator results in a NaN error.
- the calculator has some rounding issues, which result in a loss of precision in the answer. Adding numbers like 2.000001+2.000002 result in a rounded answer instead of the exact answer.

fix the table and highlightthe failed test case

Part 2 - Partition Testing

Task two of this lab involved partition-based testing of a triangle application. The purpose of this application is to take 3 space separated positive integers, each representing sides of a triangle, and the program is expected to tell the user whether the triangle is a scalene, isosceles, or equilateral triangle. The following equivalence classes were decided on for creating the test cases.

Triangle Equivalence Classes

Input Condition	Valid Input Classes	Invalid Input Classes
number of input arguments	3 input arguments (1)	< 3 input arguments (9) > 3 input arguments (10)
space between arguments	one space (2)	more than one space (11) non space character separating arguments (12)
argument type	positive integer (3)	negative integer (13) zero (14) decimal (15)
triangle type	equilateral (4) isosceles (5) scalene (6) $a + b > c$ (7)	$a + b = c$ (16) $a + b < c$ (17)
key pressed after inputting arguments	Enter key pressed (8)	Enter key not pressed (18)

Table 1: Valid and Invalid equivalence classes for the triangle program

From these equivalence classes, the following test cases were created:

Test cases for valid inputs

- 3 3 3 covers (1, 2, 3, 4, 7, 8)

- 4 4 5 covers (1, 2, 3, 5, 7, 8)
- 6 7 8 covers (1, 2, 3, 6, 7, 8)

Test cases for invalid inputs

- 1 2 covers (9)
- 3 4 5 6 covers (10)
- 7 8 9 covers (11)
- 8_7_6 covers (12)
- 1 −2 3 covers (13)
- 5 0 4 covers (14)
- 3 2 0.1 covers (15)
- 2 2 4 covers (16)
- 3 4 9 covers (17)
- not pressing Enter covers (18)

The results of these test cases, including expected versus actual output can be found in Appendix B. Failed test cases are highlighted in red. There was one test case which failed. For the input where we have the case $a+b=c$, (2 2 4) in this case, the program tells us that it is a isosceles triangle instead of an invalid one. This is likely because the implementation checks validity by looking for $a+b>c$ instead of $a+b\geq c$.

Conclusion

In this lab, we were introduced to dirty testing. ... While equivalence testing allowed us to write significantly fewer test cases, it does not catch some unique cases like testing inputs like 1234567890 1234567890 1, which returns ERROR: Invalid triangle instead of saying that it's an isosceles one.

Appendix

A Calculator Test Cases

Testid	description	Expected	Actual
1	1+1	2	2
2	0+1	1	1
3	9223372036854775807 + 9223372036854775807	18446744073709551614	1.84E+19
4	9 + 10	19	
5	4294967295 + 4294967295	8589934590	8.59E+09
6	1-1	0	0
7	-1	-1	-1
8	$	NaN	NaN
9	2^4	16	16
10	2^512	13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649006084096	NaN
11	NaN + 2	NaN	NaN
12	entering nothing		0
13	60 - 0 (with a space between 60 and -)	60	NaN
14	60 * 0	0	0
15	5 - 2	3	NaN
16	Robert'); DROP TABLE STUDENTS; --	NaN	NaN
17	80/4*5	100	4
18	(80/4)*5	100	100.0
19	5*80/4	100	100.0
20	5*(80/4)	100	100.0
21	80/(4*5)	4	4.0
22	2&1	NaN	NaN
23	16 ^^ 2	NaN	1.0
24	3443 ^^^^ 23	NaN	1.0
25	1/0	NaN	NaN
26	0/1	0	0.0
27	0.1 + 0.2 (checking for ieee 754 floating point error)	0.3	0.3
28	1+	NaN	NaN
29	/1	NaN	0.0
30	1/	NaN	NaN
31	1*	NaN	NaN
32	*1	NaN	0.0
33	((((1+1))))	2	2.0
34	5--2	7	NaN
35	2^3 + 2	10	32.0
36	2^1 + 2 + 3	7	64.0
37	2^(3) + 1	9	16.0
38	(2^3) + 1	9	9.0
39	+ 1	NaN	1.0
40	(2^3)-3	5	NaN
41	(2^3)+3*(8-6)	14	14.0
42	(2^3)+3(8-6)	14	40
43	3(2)	6	32.0
44	(1)(1)	1	11.0
45	2^2^2	16	16.0
46	2^(2-3)	0.5	0.5
47	1-2^2+3	0	1.0
48	2^-512	7.458340731200206743290965315462933837376471534600406894271518333206278385070118304936174890400427803361511603255836101453412728095225302660486164829592084691481260792318781377495204074266435262941446554365063914765414217260588507120031686823003222742297563699265350215337206058336516628646003612927433551846968657326499008153319891789578832685947418212890625 × 10^-155	0
49	1.0 + 2	3	3.0
50	(-1)^(0.5)	NaN or i	NaN
51	2**2	NaN	0.0
52	2++++++++2	NaN	4.0
53	()	NaN	NaN
54	(1+2 missing bracket	NaN	NaN
55	1+()	NaN	NaN
56	1(-1)	-1	NaN
57	2-(-2)	4	NaN
58	2--2	4	4
59	2//2	NaN	NaN
60	(1 + (2 + 3))	6	6.0
61	3+*3	NaN	3
62	2^3 + 2^3	16	32768.0
63	2^(3) + 2^(3)	16	32768
64	2(^3)	NaN	20
65	(^0)	NaN	1

66	0^0	1	1
67	(^0)^2	NaN	1
68	(+ 1 2)	NaN	12
69	1 2 3	NaN	123
70	(+* 1 2)	NaN	0.0
71	2.000001+2.000002	4.000003	4.0
72	8/-2	-4	-4.0
73	testing the buttons - delete results in a error stack trace when the input is already empty	don't do anything	Exception in thread "AWT-EventQueue-0" java.lang.StringIndexOutOfBoundsException: begin 0, end -1, length 0 at java.base/java.lang.String.checkBoundsBeginEnd(String.java:3410) at java.base/java.lang.String.substring(String.java:1883) at MainFrame$22.actionPerformed(MainFrame.java:245) at java.desktop/javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:1967) at java.desktop/javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:2308) at java.desktop/javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:405) at java.desktop/javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:262) at java.desktop/javax.swing.plaf.basic.BasicButtonListener.mouseReleased(BasicButtonListener.java:279) at java.desktop/java.awt.Component.processMouseEvent(Component.java:6632) at java.desktop/javax.swing.JComponent.processMouseEvent(JComponent.java:3342) at java.desktop/java.awt.Component.processEvent(Component.java:6397) at java.desktop/java.awt.Container.processEvent(Container.java:2263) at java.desktop/java.awt.Component.dispatchEventImpl(Component.java:5008) at java.desktop/java.awt.Container.dispatchEventImpl(Container.java:2321) at java.desktop/java.awt.Component.dispatchEvent(Component.java:4840) at java.desktop/java.awt.LightweightDispatcher.retargetMouseEvent(Container.java:4918) at java.desktop/java.awt.LightweightDispatcher.processMouseEvent(Container.java:4547) at java.desktop/java.awt.LightweightDispatcher.dispatchEvent(Container.java:4488) at java.desktop/java.awt.Container.dispatchEventImpl(Container.java:2307) at java.desktop/java.awt.Window.dispatchEventImpl(Window.java:2762) at java.desktop/java.awt.Component.dispatchEvent(Component.java:4840) at java.desktop/java.awt.EventQueue.dispatchEventImpl(EventQueue.java:772) at java.desktop/java.awt.EventQueue$4.run(EventQueue.java:721) at java.desktop/java.awt.EventQueue$4.run(EventQueue.java:715) at java.base/java.security.AccessController.doPrivileged(AccessController.java:389) at java.base/java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:85) at java.base/java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:95) at java.desktop/java.awt.EventQueue$5.run(EventQueue.java:745) at java.desktop/java.awt.EventQueue$5.run(EventQueue.java:743) at java.base/java.security.AccessController.doPrivileged(AccessController.java:389) at java.base/java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:85) at java.desktop/java.awt.EventQueue.dispatchEvent(EventQueue.java:742) at java.desktop/java.awt.EventDispatchThread.pumpOneEventForFilters(EventDispatchThread.java:203) at java.desktop/java.awt.EventDispatchThread.pumpEventsForFilter(EventDispatchThread.java:124) at java.desktop/java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:113) at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:109) at java.desktop/java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101) at java.desktop/java.awt.EventDispatchThread.run(EventDispatchThread.java:90)

Table 1: Test cases carried out against the calculator program. Failed test cases are highlighted in red.

B Triangle Test Cases

Test id	Description	a	b	c	Expected	Actual
1	valid equilateral	3	3	3	Equilateral	Equilateral
2	valid isoceles	4	4	5	Isoceles	Isoceles
3	valid scalene	6	7	8	Scalene	Scalene
4	too few arguments	1	2		ERROR: Not enough arguments	ERROR: Not enough arguments
5	too many arguments	3	4	5 6	ERROR: Too many arguments	ERROR: Too many arguments
6	more than one space in between arguments	7	8	9	ERROR: Too many spaces	ERROR: Too many arguments
7	non space separated arguments	8_7_6			ERROR: Not enough arguments	ERROR: Not enough arguments
8	argument with negative	1	-2	3	ERROR: Invalid argument - non positive value	ERROR: Invalid argument - non positive value
9	argument with zero	5	0	4	ERROR: Invalid argument - non positive value	ERROR: Invalid argument - non positive value
10	argument with decimal	3	2	0.1	ERROR: Invalid argument - non integer	ERROR: Invalid argument - non integer
11	a + b = c	2	2	4	ERROR: Invalid triangle	Isosceles
12	a + b < c	3	4	9	ERROR: Invalid triangle	ERROR: Invalid triangle
13	not pressing enter after 3 valid arguments	3	4	5	no output or ERROR	(no output)

Table 2: Test cases covering the equivalence classes identified in Part 2 for the Triangle program. Failed test cases are highlighted in red.