ECE 322 Lab 1 Report

By Mattheas Jamieson

Introduction (3 marks)

`

The objective of this lab was to apply various methods of Black Box testing across two different programs to discover errors. The purpose of these methods as well as their effectiveness was also discussed. Black Box testing occurs when the internal functionality is hidden from us, as opposed to White or Glass Box testing where the internal implementation is visible to us. The two testing methods used in Part One is Failure(Dirty) testing and Error Guessing. Failure testing is an approach whereas many inputs as possible are given in both expected and unexpected manner to elicit the program to fail. Error Guessing is using your intuition and knowledge of software to test areas that are likely to produce errors. In Part Two the method used is Equivalence Classes. Equivalence classes are a partition of the input domain, in which all the inputs of a domain can be considered indistinguishable from each other, so by testing any one input you have essentially tested them all.

Part One – Calculator Program (32 marks)

Q1 : Provide an explanation of the application under test, they should describe the problem that is being tested :

Answer: The application under test is a simple calculator that performs basic arithmetic (i.e BEDMAS operators) with base 10 numbers, the calculator has no other apparent functionality to be tested (e.g. trigonometric functions, graphing, etc.).

Q2: Explain what testing methods you are using. The testing methods should be explained, what do they test, what are they good for etc ..
Errors in the application should be identified, and some justification given for what may be causing these errors. :

Answer: The two testing methods used in Part One are Failure (Dirty) testing and Error Guessing which are both techniques of Black Box testing. Failure or Dirty testing is an approach of Black Box testing where the tester attempts to bring the software to failure by using any means necessary, i.e. using both the software as it was designed to be used and how it was not designed to be used with a multitude of inputs. I believe Failure/ Dirty testing is a relatively informal approach used to test the software's reliability, correctness, and robustness. It does not require much structure of planning and can be done "on the fly" in a sense if the test cases are systematically recorded. Error Guessing is an approach that uses the intuition and the

experience of the software tester to guess at probable location of errors. Error Guessing can be used to test boundary cases as well as common mistakes (e.g mistake in BEDMAS order). A disadvantage of both Failure testing and Error guessing is that there is likely redundancy in the testing as seen in my Test Cases Table in Q4, and there is also likely functionality that has been overlooked, i.e., not tested.

The first error I identified in the calculator program was the incorrect evaluation of the double negative sign using Error Guessing in test cases **1** & **2**. My assumption is that in a normal calculator the leading negative sign is treated as implicit multiplication meaning that a double negative is implicit multiplication of two negative numbers that yields a positive. I believe that in the tested program the negative sign is always treated as subtraction, meaning after any negative sign a number must follow.

The second error I discovered was the incorrect evaluation of multiple signs with missing values using a Failure test as seen in test cases **3**, **4**, **30** & **31**. I assume that if an addition sign is placed an no number is placed after the addition sign then the program implicitly adds zero. So in test case 3 it would mean that it would actually evaluate like 5+0+0+6. Test cases **30** & **31** display similar erroneous behaviour.

The third error I found was the incorrect order of evaluation for BEDMAS operators using Error Guessing and was discovered with test cases **5**, **7**, **9** & **10**. It is seen that in actuality the order of operations for the program is BDMASE. I believe that the software developer(s) most likely made a mistake when assigning the precedence for the exponential operator.

The fourth error I found is regarding implicit multiplication in Dirty test **28**, where instead of implicit multiplication occurring, the numbers were simply concatenated together like string concatenation. I am unsure as to why this error would occur besides incorrect requirements.

The fifth error I found using test case **11** with a Dirty test, it appears having a number inside two sets of brackets results in an error, my guess is that when a open bracket is encountered the program mistakenly searches for first the next closing bracket and then evaluate the statement, meaning the program tried to evaluate “((5)”, it did not continue to search for other potential open brackets that the first encountered closed bracket could belong to.

The hypothesis regarding first, fourth and fifth error lines up with the results of Error guessing test cases **25** & **27** which is the sixth error I found. My guess is that the program tried to evaluate “(-(-5)” which evaluates to “- -5” which we know the program cannot evaluate.

The seventh error I found through Dirty testing as seen in test cases **14**, **15** & **21**. With a missing value in the expression, it still evaluates to a number which is incorrect, similar to second error I believe that implicitly a zero is inserted into the expression. I believe most likely that in the program there is a variable that is supposed to store the missing value; however it is assigned a value of zero, and then when the variable is not updated due to the missing value and then it incorrectly evaluates the expression with the zero value.

The eighth error I found was using Error guessing as seen in test case **29** where there is an incorrect order of operations. Since division and multiplication are of the same precedence the expression should simply be evaluated from left to right but instead the multiplication is evaluated first. I believe the error is that there were unclear requirements, or the developer simply did not know that even though division comes first in the BEDMAS expression it still has the same level of precedence as multiplication.

The ninth error I found was in test case **32** using dirty testing. The program evaluates the expression simply from left to right which was probably a lack of understanding on the developer's part about exponent laws.

The tenth error I found was a rounding error in test case **34** which was Error guessing. This probably has to do with the level of precision that is used in the program to store the result, e.g. floating point instead of some higher level like double floating-point precision.

The eleventh error I found using a dirty test (as seen in test case **38**) but is closely related to the fifth error found. I believe the program searched for a corresponding closing bracket to the initial open one, and when it could not find one the program threw an exception or a function that was called returned null, and the way it was handled in the program was simply to return NaN.

Q3: Discuss the effectiveness of the testing methods used

Answer: I believe using Dirty testing and Error guessing together is an effective approach if you wish to perform informal and quick testing. They complement each other well. Dirty testing finds errors that are unexpected like "*5". This would most likely not be found with error guessing since you would reasonably expect the program to behave correctly with such basic functionality. Then using Error guessing, errors are found at likely places, i.e rounding error, but this is an error you may likely not discover with Dirty testing.

Q4 : Test Cases Table:
Answer: Tests 1-39

ID	Inputs	Description	Expected Result	Actual Result
1	1--1	Test to see if double negative sign turns into a addition sign	2	NaN
2	--1+1	Test to see if double negative sign turns into a positive sign	2	NaN

3	5+++6	Test to see if more than one addition sign of two +ive numbers results in error	NaN	11
4	5++-6	Test to see if more than one addition sign of +ive & -ive numbers results in error	NaN	-1
5	1+2^2	Check to see if exponent operation has higher precedence than addition	5	9
6	(1+2)^2	Check to see if bracket has higher precedence than exponential operation	9	9
7	5-2^2	Check to see if exponent operation has higher precedence than subtraction	1	9
8	5-(2^2)	Checks to see if brackets have higher precedence than exponent	1	1
9	2*3^2	Check to see if exponent operation has higher precedence than multiplication	36	18
10	2/3^2	Check to see if exponent operation has higher precedence than division	2/9	4/9 (0.44444 actual)
11	((5))-1	Check to see if double bracket evaluates	4	NaN
12	(5^)	Evaluate exponent with missing values inside brackets	NaN	NaN
13	5/	Evaluate division with missing values	NaN	NaN
14	/5	Evaluate division with missing values	NaN	0.0
15	*5	Evaluate multiplication with missing values	NaN	0.0
16	5*	Evaluate multiplication with missing values	NaN	NaN
17	+5	Test to see if a leading addition sign turns into a positive sign	5	5
18	5+	Evaluate addition with missing values	NaN	Nan
19	-5	Test to see if a leading subtraction sign turns into a negative sign	-5	-5.0

20	5-	Evaluate subtraction with missing values	NaN	NaN
21	^5	Evaluate exponential with missing values	NaN	0.0
22	5^	Evaluate exponential with missing values	NaN	NaN
23	(5))	Evaluate expression with missing bracket	NaN	NaN
24	(-(5)^2)	Check to see if even exponential with implicit negative multiplication evaluates correctly	-25	25.0
25	(-(-5)^2)	Check to see if even negative exponential with implicit negative multiplication evaluates correctly	-25	NaN
26	(-(5)^3)	Check to see if odd exponential with implicit negative multiplication evaluates correctly	-125	-125
27	(-(-5)^3)	Check to see if odd negative exponential with implicit negative multiplication evaluates correctly	125	NaN
28	-5(8)	Evaluate implicit multiplication with negative number and bracketed value	-40	-58
29	5/5*2	Check if BEDMAS applies with multiplication and division	2	0.5
30	5*****2	Test to see if more than one multiplication sign results in error	NaN	0.0
31	2^^2	Check to see if missing value in double exponential results in error	NaN	1.0
32	2^3^2	Check if a double exponential evaluates correctly	512	64
33	02-1	Evaluate subtraction with a value that has a leading zero	1	1
34	0.000001-1	Check to look at rounding error	-0.999999	-1
35	0/5	Check zero divided by nonzero	0	0
36	0/0	Check zero divided by zero	NaN	NaN

37	1/0	Check to see if division by zero evaluates correctly	NaN	NaN
38	(7/5	Evaluate division with missing bracket	1.4	NaN
39	0.00001-1	Check to look at rounding error	-0.99999	-0.99999

Part 2 – Triangle Classification Program (32 marks)

Q1: explanation of the application under test, and a description of the problem

Answer: The Triangle Classification program takes 3 positive integers as inputs, these inputs correspond to the 3 side lengths of a triangle, if the numbers satisfy the condition for a valid triangle the program informs the user of the type of triangle these values form. If a valid triangle is not formed the program informs the user of that. The problem with testing this application is we are tasked with developing equivalence classes. For this we must remind ourselves of the rule for that defines a triangle which is that any two side lengths added together must be strictly greater than the third in length. With this rule in mind the creation of the equivalence classes was straightforward.

Q2: List all the Partition classes
Answer:

Input Condition	Valid Equivalence Classes	Invalid Equivalence Classes
3 numbers on the same line separated by a space	X Y Z (1)	(<3 inputs) (2) (>3 inputs) (3)
3 positive numbers	+ + + (4) Yes	(+ + -, …. < - - -) (5) No
3 integers	(6) Yes	(7) No
X + Y > Z X + Z > Y Y + Z > X	(8) Yes Valid Triangle	(9) No

Q3: Identification of error(s) in the program. Must be identified and justified

Answer: The only error I found in program was in test case number **5**. As seen in the inputs the program incorrectly identifies 3 side lengths as a triangle. From test cases 6 through 8 we can see that the program can in some cases correctly identify the three triangle types. Using this information and looking at the inputs of test case number 5 it can be concluded that the error happens at the boundary and that the program concludes if three side lengths are in fact a triangle if any two sides are greater or equal to the third length. From this I conclude that the developer incorrectly wrote compared values with a ">=" instead of the required ">" sign.

Q4: Discuss the effectiveness of the testing methods used

Answer: While equivalence classes are an effective way to feasibly test the "entire" input domain I believe that it is not complete without also doing boundary value analysis, which is exactly where I found an error in the program. If a boundary value analysis is also completed I believe it is a very formal and complete testing method which makes testing the "entire" input domain feasible.

Q5 : Test Cases Table :
Answer:

ID	Inputs a b c	Description	Expected Result	Actual Result
1	1 2	Tests equivalence class 2, i.e <3 inputs	ERROR: Not enough arguments	ERROR: Not enough arguments
2	1 2 3 4	Tests equivalence class 3, i.e >3 inputs	ERROR: Too many arguments	ERROR: Too many arguments
3	1 2 -5.5	Tests equivalence class 1,5,7 i.e 3 inputs	ERROR: Invalid argument – non integer	ERROR: Invalid argument – non integer

4	1 2 5.5	Tests equivalence class 4 i.e 3 positive inputs	ERROR: Invalid argument – non integer	ERROR: Invalid argument – non integer
5	1 2 3	Tests equivalence class 6,9 i.e 3 positive integer inputs	ERROR: Invalid triangle	Scalene
6	3 3 3	Tests equivalence class 8 i.e 3 positive integer inputs Note: Equilateral Triangle	Equilateral	Equilateral
7	3 4 4	Tests equivalence class 1,4,6,8 i.e 3 positive integer inputs Note: Isosceles Triangle	Isosceles	Isosceles
8	5 6 7	Tests equivalence class 1,4,6,8 i.e 3 positive integer inputs Note: Scalene Triangle	Scalene	Scalene

Conclusion (3 marks)

The purpose of the lab was to experiment with different methods of Black Box testing on two programs to find errors. The use of both Failure(Dirty) testing and Error Guessing resulted in a informal but thorough testing of the Calculator program where about 10 errors were identified. The use of Equivalence class testing in the Part Two Triangle Classification program resulted in a more formal but slightly less complete testing initiative where only one error was identified. I think that the testing in Part Two could have been improved by actively consider Boundary Value Analysis in addition to the Equivalence classes.