COSC 3P95- Software Analysis & Testing

Assignment 1

Due date: Monday, Oct 16th, 2023, at **23:59** (11:59 pm)

Delivery method: This is an individual assignment. Each student should submit one PDF through Brightspace.

Attention: This assignment is worth 10% of the course grade. Please also check the Late Assignment Policy.

Name: Athavan Jesunesan **Student ID: 6705271**

Questions:

1- **Explain the difference between "sound" and "complete" analysis in software analysis. Then, define what true positive, true negative, false positive, and false negative mean. How would these terms change if the goal of the analysis changes, particularly when "positive" means finding a bug, and then when "positive" means not finding a bug. (10 pts)**

Sound analysis prioritises accurate identifications and minimizes false positives. It will ensure that what is reported is accurate but does not guarantee that it will find everything. On the other hand, complete aims to ensure that all the bugs are found but will sometimes report false positives.

True positive is the case when a bug is correctly identifies a bug. This result indicates an issue with the software, and it turns out the issue truly exists.

True Negative is the case when the analysis says there is no bug in a particular portion and there is truly no bug in that portion. It accurately identifies that there is no concern in a particular section of the code.

False Positive is when the analysis returns that it has identified a problem in the program however upon further inspection it is deduced that is no actual problem in that region. The analysis does not accurately identify the issues and states there is a problem in the program where there is not.

False Negative is case where the analysis indicates the section of the program is issue-free, where in reality it has missed an issue in that region of the code. The analysis returns that there is nothing to worry about in that region however, there is an issue, and it has missed it.

In the case that a positive is a bug then the definitions above hold. The premise of my definitions above are under the assumption that a positive is the analysis returning an issue/bug in the code.

In the case where positive means not finding a bug then;

True positive is when it correctly reports that this section of the program is error free.

True Negative is when it correctly reports that this section of the program contains a bug.

False Positive is when it identifies that there are no issues; however, it has missed something and there is indeed an issue in that section.

False Negative is when it identifies that there are is an issue; however, it has incorrectly identified an issue and there was no bugs in that section.

2- **Using your preferred programming language, implement a random test case generator for a sorting algorithm program that sorts integers in ascending order. The test case generator should be designed to produce arrays of integers with random lengths, and values for each sorting method.**

A) Your submission should consist of:

a. **Source code files for the sorting algorithm and the random test case generator.**
    i. Included in the folder named 'Q2.py'.

b. **Explanation of how your method/approach works and a discussion of the results (for example, if and how the method was able to generate or find any bugs, etc.). You can also include bugs in your code and show your method is able to find the input values causing that.**
    i. My program generates various arrays of varying lengths, order, and integers. From the tests that I have run there have been no errors so I created a situation where if the array was specifically of length 12 the program would throw an exception and it provides the array which caused the errors as well as the name of the error (in this case Value Error). Considering my tests have never thrown an error other than what I implemented for the purpose of this assignment I can say that this analysis method is sound and not complete. I only ran into the error I created, however when the error is thrown it is a true positive. The bug is found when the test case happens to generate an array of size 12 which can happen if enough tests are run. For marking purposes, I included a situation where it occurs at the end of the code.

c. **Comments within the code for better understanding of the code.**

d. **Instructions for compiling and running your code.**
    i. Enter the correct folder and run 'python Q2.py' I used python3 to develop this code.

e. **Logs generated by the print statements, capturing both input array, output arrays for each run of the program.**
    i. Included in folder labeled 'Q2 Example output.png'.

f. **Logs for the random test executions, showing if the test was a pass and the output of the execution (e.g., exception, bug message, etc.).**
    i. Included in folder labeled 'Q2 Example output.png'.

B) Provide a context-free grammar to generate all the possible test-cases. (18 + 8 = 26 pts)

Array -> "[" List "]"
List -> Integer | List "," Integer
Integer -> "-MAXINT" | … | "-1" | "0" | "1" | … | "MAXINT"

3- A) For the following code, manually draw a control flow graph to represent its logic and structure.

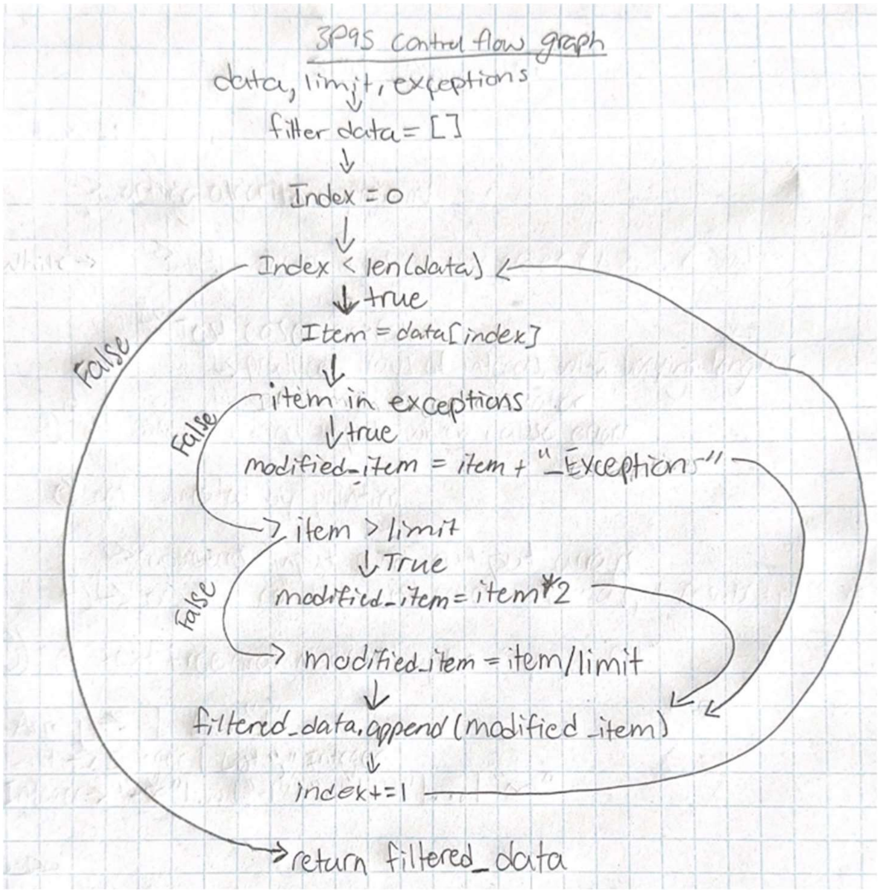
```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

The code is supposed to perform the followings:

a. **If an item is in the exceptions list, the function appends "_EXCEPTION" to the item.**
b. **If an item is greater than a given limit, the function doubles the item.**
c. **Otherwise, the function divides the item by 2.**



B) Explain and provide detailed steps for "random testing" the above code. No need to run any code, just present the coding strategy or describe your testing method in detail. (8 + 8 = 16 pts)

For this program I would implement a random testing method with as much code coverage as possible. This way I can see what occurs when the code executes each of the if statements and all the possible inputs that can go into each of those if statements. The random testing would generate many test cases of data values with varying size, and values. I would aim for a more complete analysis to ensure that the errors in input are found. I would want my test cases to be able to cast a wide net so that each of the statements are executed and contain a variety of test case situations. I would ensure that by creating many different data, limits, and exceptions and testing them all with a variety of overlap.

4- A) Develop 4 distinct test cases to test the above code, with code coverage ranging from 30% to 100%. For each test-case calculate and mention its code coverage.

Test Suite 1:	{Limit(5), Data[2, 4, 6, 8], Exceptions(4)}
Test Suite 2:	{Limit(30), Data[2, 4, 6, 8], Exceptions(4)}
Test Suite 3:	{Limit(30), Data[2, 4, 6, 8], Exceptions(3)}
Test Suite 4:	{Limit(30), Data[3, 3, 3, 3], Exceptions(3)}

B) Generate 6 modified (mutated) versions of the above code.

	Test Suite 1	Test Suite 2	Test Suite 3	Test Suite 4
Statement Coverage	100%	92%	83%	67%
Branch Coverage	100%	67%	33%	33%

C) Assess the effectiveness of the test cases from part A by using mutation analysis in conjunction with the mutated codes from part B. Rank the test-cases and explain your answer.
Mutation 1: item * limit
Mutation 2: item/limit
Mutation 3: elif item<limit
Mutation 4: if item not in exceptions
Mutation 5: index += 2
Mutation 6: filter_data.append(item)

	Test Suite 1	Test Suite 2	Test Suite 3	Test Suite 4
Mutation 1: item * limit	pass	fail	fail	fail
Mutation 2: item/limit	pass	fail	fail	fail
Mutation 3: elif item<limit	pass	pass	pass	fail
Mutation 4: if item not in exceptions	pass	pass	pass	pass
Mutation 5: index += 2	pass	pass	pass	pass
Mutation 6: filter_data.append(item)	pass	pass	pass	pass
Score:	100%	67%	67%	50%

The clear best test suite here is Test Suite 1. It was the most reactive for all the mutations and was able to kill them all. The test cases are order from left to right in order of effectiveness.

D) Discuss how you would use path, branch, and statement static analysis to evaluate/analyse the above code. **(4 * 8 = 32 pts)**

The branch coverage has clearly triumphed in terms of its relationship to proper test suites. If an analysis with high branch coverage was better at killing mutations. Therefore, I would use the coverages as a guide to how effective the analyses are.

5- The code snippet below aims to switch uppercase characters to their lowercase counterparts and vice versa. Numeric characters are supposed to remain unchanged. The function contains at least one known bug that results in incorrect output for specific inputs.

```
def processString(input_str):
    output_str = ""
    for char in input_str:
        if char.isupper():
            output_str += char.lower()
        elif char.isnumeric():
            output_str += char * 2
        else:
            output_str += char.upper()

    return output_str
```

In this assignment, your tasks are:

a. Identify the bug(s) in the code. You can either manually review the code (a form of static analysis) or run it with diverse input values (a form of manual random testing). If you are unable to pinpoint the bug using these methods, you may utilize a random testing tool or implement random test case generator in code. Provide a detailed explanation of the bug, identify the line of code causing it, and describe your strategy for finding it.

   I tried a form of manual random testing and noticed that the program does not properly handle the situations where a number is present. The program aims to leave numeric values unchanged, but it actually duplicates the character. Line 7 of the code is incorrect, and the line should be changed to output_str += char. The strategy I used was aimed at branch coverage and by ensure that my test suites were able to cover every branch and thereby statement, I was able to swiftly discover the bug.

b. Implement Delta Debugging, in your preferred programming language to minimize the input string that reveals the bug. Test your Delta Debugging code for the following input values provided.
   i. “abcdefG1”
   ii. “CCDDEExy”
   iii. “1234567b”
   iv. “8665”

Briefly explain your delta-debugging algorithm and its implementation and provide the source code in/with your assignment. **(4 + 12 = 16 pts)**

This delta debugging algorithm splits the strings down into 2 equal portions to begin keeping only what contains an issue. It then proceeds to split the remainders further until it finds the error.

6- Extra Credit Assignment: Create a GitHub repository to host all the elements of this assignment. This includes source codes, test data, and any screenshots or logs you have generated. Submit the GitHub link along with your main submission through Brightspace. **(5 pts)**

Link: https://github.com/bit-yottabyte/COSC-3P95

Marking Scheme:

Marks will be awarded for completeness and demonstration of understanding of the material. It is important that you fully show your knowledge when providing solutions in a concise manner. Quality and conciseness of solutions are considered when awarding marks. Lack of clarity may lead you to lose marks, so keep it simple and clear.

Submission:

The submission is expected to contain a sole word-processed document. The document can be in either ***DOC or PDF*** *format; it should be a single column, at least single-spaced, and at least in font 11. It is strongly recommended to use the assignment questions to facilitate marking: answer the questions just below them for easier future reference.*

Late Assignment Policy:

A one-time penalty of 25% will be applied on late assignments. Late assignments are accepted until the Late Assignment Date, four days after the Assignment Due Date. No excuses are accepted for missing deadlines. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor's discretion.

Plagiarism:

Students are expected to respect academic integrity and deliver evaluation materials that are only produced by themselves. Any copy of content, text or code, from other students, books, web, or any other source is not tolerated. If there is any indication that an activity contains any part copied from any source, a case will be open and brought to a plagiarism committee's attention. In case plagiarism is determined, the activity will be canceled, and the author(s) will be subject to university regulations. For further information on this sensitive subject, please refer to the document below: ***https://brocku.ca/node/10909***