# SENG3320 Assignment 2: Automated Test Data Generation

|  |  |
| --- | --- |
| **Group 10** | |
| Kyle Beattie | C3303374 |
| Ni Zeng | C3238805 |
| Brandon Allen | C3279505 |
| Austin Baxter | C3356468 |

**Table of Contents**

[SENG3320 Assignment 2: Automated Test Data Generation 1](#_Toc103869436)

[1. Question 1: Fuzz Testing 2](#_Toc103869437)

[1.1. Test Tool Design 2](#_Toc103869438)

[1.2. Test Environment 2](#_Toc103869439)

[1.3. Test Cases 2](#_Toc103869440)

[1.4. Summary 2](#_Toc103869441)

[2. Question 2: Automated Testing Techniques 2](#_Toc103869442)

[2.1. Symbolic Execution 2](#_Toc103869443)

[2.2. Fuzz Testing 2](#_Toc103869444)

[2.3. Mutation Testing 2](#_Toc103869445)

[2.4. Comparison 2](#_Toc103869446)

## Question 1: Fuzz Testing

### Test Tool Design

### Test Environment

### Test Cases

### Summary

## Question 2: Automated Testing Techniques

### Control Flow Analysis



### Symbolic Execution

KLEE was used in this section through a browser interface. The function was small enough that a main function could be placed below it. This function setup and ran KLEE providing an output.

#### KLEE Output

non-triangle.

non-triangle.

non-triangle.

triangle.

equilateral triangle .

isosceles triangle.

isosceles triangle.

KLEE: done: total instructions = 136

KLEE: done: completed paths = 8

KLEE: done: generated tests = 8

#### Interpretation of Results

To give greater context a missing output exists in the function. In the deepest nested if statement, no statement is presented if only a==b is satisfied. A print statement is placed here (else printf(“missed area.\n”);). This reveals the eighth test condition. This results in the output sequence:

non-triangle.

non-triangle.

non-triangle.

triangle.

equilateral triangle .

missed area.

isosceles triangle.

isosceles triangle.

First the run fails each of the first decision’s conditions. That is (a+b<=c) then (a+c<=b) then (b+c<=a). This results in the three “non-triangle.” Outputs.

The rest of the run only concerns the equality of the different values. The following table may be used.

|  |  |  |
| --- | --- | --- |
| a == b | a == c | b == c |
| false | false | false |
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | true |

The first-row results in the “triangle.” Output, no two values are equal. The second-row results in the “equilateral triangle.” Result, all values are equal. The third through fifth results are each condition being true one at a time. The first condition being true triggers the missed area statement that should result in an isosceles. The last two correctly result in isosceles outputs.

#### Control flow analysis:

##### Decision Coverage:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | c | Output | Valid |
| 1 | 1 | 5 | Non-triangle | True |
| 2 | 3 | 4 | Triangle | True |
| 2 | 2 | 2 | Equilateral | True |
| 2 | 2 | 3 | No output | False, should output isosceles. |
| 2 | 3 | 2 | Isosceles | True |

##### Condition Coverage:

Conditions:

|  |  |
| --- | --- |
| Dentoted by | Condition |
| C1 | a+b>c |
| C2 | a+c>b |
| C3 | b+c>a |
| C4 | a==b |
| C5 | a==c |
| C6 | b==c |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | c | conditions | output | valid |
| 1 | 1 | 5 | C2,C3,C4 | Non-triangle | True |
| 1 | 5 | 1 | C1,C3,C5 | Non-triangle | True |
| 5 | 1 | 1 | C1,C2,C6 | Non-triangle | True |

##### Condition / Decision Coverage:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | c | conditions | output | valid |
| 1 | 1 | 5 | C2,C3,C4 | Non-triangle | True |
| 1 | 5 | 1 | C1,C3,C5 | Non-triangle | True |
| 5 | 1 | 1 | C1,C2,C6 | Non-triangle | True |
| 2 | 3 | 4 | C1,C2,C3 | Triangle | True |
| 2 | 2 | 2 | C1,C2,C3,C4,C5,C6 | Equilateral | True |
| 2 | 2 | 3 | C1,C2,C3,C4 | No output | False, should output isosceles. |
| 2 | 3 | 2 | C1,C2,C3,C5 | Isosceles | True |

##### Multiple Condition Coverage:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | c | conditions | output | valid |
| 5 | 1 | 2 | C1,C2 | Non-triangle | True |
| 5 | 1 | 1 | C1,C2,C6 | Non-triangle | True |
| 1 | 2 | 5 | C2,C3 | Non-triangle | True |
| 1 | 1 | 5 | C2,C3,C4 | Non-triangle | True |
| 1 | 5 | 2 | C1,C3 | Non-triangle | True |
| 1 | 5 | 1 | C1,C3,C5 | Non-triangle | True |
| 2 | 3 | 4 | C1,C2,C3 | Triangle | True |
| 2 | 2 | 3 | C1,C2,C3,C4 | No output | False, should output isosceles. |
| 2 | 3 | 2 | C1,C2,C3,C5 | Isosceles | True |
| 3 | 2 | 2 | C1,C2,C3,C6 | Isosceles | True |
| 2 | 2 | 2 | C1,C2,C3,C4,C5,C6 | Equilateral | True |

### Fuzz Testing

The idea of Fuzz Testing on this question is to apply random integers for variables a, b, and c to examine the outcome of the triangle (int a, int b, int c).  
  
Fuzz Testing structure:  
>FuzzTesting (Q2 Fuzz Testing task folder)

FuzzInput\_Output.txt (contain the input & output test case result)

FuzzTesting.c (Fuzz test case generator)

FuzzTesting.exe (Fuzz test case generator executor)

triangle. c (Given c program for Fuzz Testing)  
  
When run the FuzzTesting.exe, the Fuzz generator will first ask for user input for the number of test cases that need to be generated. After the input, the generator will generate 3 random integer numbers from the range 0 to 9. The generated number will execute using triangle.c and record the input and output result in a text file Name ‘Fuzzinput\_Output.txt’.  
  
Compare the test results of Fuzz testing and symbolic execution:

|  |  |
| --- | --- |
| **Control-flow coverage achieved:** | |
| Fuzz Testing: | Symbolic testing: |
| **Time spent:** | |
| Fuzz Testing: | Symbolic testing: |

### Mutation Testing

### Comparison