NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

CS4218 – Software Testing

(Semester 2: AY2019/20)

Take-at-home Test

Time Allowed: 2 hours

INSTRUCTIONS TO STUDENTS

1. To solve the questions. You can either:

- write your answers on the PDF directly,
- print out, hand-write your answers, and scan into a PDF
- write your answers in the DOCX, and save as PDF
- have any other approach that would results in a PDF file

2. Submit by **Wed, 11 Mar, 4pm** on LumiNUS Files->Student Submissions -> Test a PDF file:

- Name your PDF file using your student number: A1234567Z.pdf
- The format of the answers has to **strictly** follow the format provided in the question.PDF file. For example, if Question 3 point a. (requirement and answer space) is on page 7 in the questions.PDF file, it should be on page 7 in the answers PDF file submitted by you.
- In case you scan your answers, make sure that they can be read once included in the PDF. We will not grade answers that cannot be read.
- When multiple submissions are made, we will grade only your most recent submission.

3. Students are required to answer **ALL** questions within the space in this file.

4. Failing to submit your answers to LumiNUS folder by 4pm on Wed, 11 Mar means you failed your test (0 marks will be awarded). No late submission is allowed.

5. Address any questions you might have to dcscrist@nus.edu.eg or call 65168850.

6. Write your student number below.

STUDENT NO: ____**A0176884J**____

Figure 1 used for Questions 2 and 4.

Consider the function in Figure 1. For input, **0**<=**n**, and there are **n** integer values in array **a**. The function returns the sum of the even numbers on even positions from the array.

```
Line   Code
1.     int even_sum (int n, int a[]) {
2.       sum = 0;
3.       i = 0;
4.       while (i < n) {
5.          if (a[i] % 2 == 0 && i % 2 == 0)
6.                sum = sum + a[i];
7.          i = i + 1;
8.       }
9.     return sum;
10.    }
```

Figure 1: Function even_sum

Question 1. [Total 10 marks] General testing

a. [4 marks] What is the difference between Unit testing and Integration testing? Briefly explain your answer using one example.

Ans)
Unit Testing involves testing a single isolated module. The purpose of unit testing is to validate that each unit/component of the software performs as designed. This helps in isolating errors to a single module. Unit testing is generally the first level of software testing

Integration testing is where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between the units integrated together. It usually occurs after unit testing.

Example: Lets consider that there is a program which has three modules – Module 1, Module 2, Module 3. The modules depend on each other as shown in the following:

Module 1 -> Module 2 -> Module 3

When we do Unit Testing we will individually tests all the components. For example, we will test Module 2 as shown:

Driver(Module 1) -> Module 2 -> Stub(Module 3)

When we do Integration Testing we will test components as a group. For example, we will test Module 2 and Module 3 together as shown:

Driver(Module 1) -> Module 2 -> Module 3

b. [6 marks] Here is a scenario taken from a web-based system which deals with sports events.

Scenario Outline:

The web-based system allows for browsing events with or without media items.
The user is browsing <event without media> and <events with media>.
On the sidebar, there is a <media item> available attached to <event with media>.
Some <events with media> have only certain types of <media items> attached to them.
Specifically, Tennis events can only have Video.

The possibilities for <event without media>, <event with media>, <event without media> cells are:

<event without media>	**<media item>**	**<event with media>**
Football	Image	Football
Basketball	Video	Basketball
Tennis	Music	Tennis

The tester needs to test that the web page will be properly displayed for any type of combination of events and media items. What testing approach should the tester use to reduce (minimize) the number of test cases? Enumerate these test cases.

Ans)

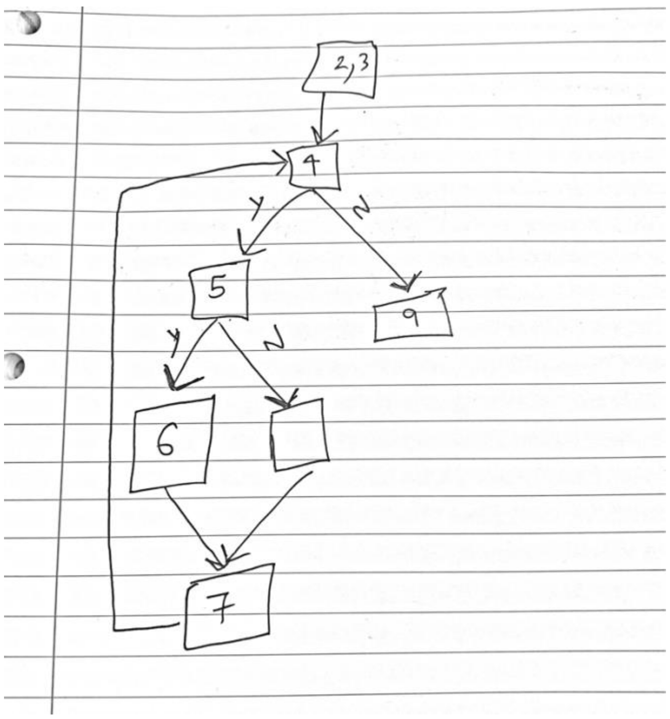
The tester should use the Pairwise Testing approach to minimize the number of test cases.

<event without media>	**<media item>**	**<event with media>**
Basketball	Music	Football
Basketball	Video	Tennis
Basketball	Image	Basketball
Tennis	Music	Football
Tennis	Video	Tennis
Tennis	Image	Basketball
Football	Music	Basketball
Football	Video	Tennis
Football	Image	Football
-	Video	Football
-	Video	Basketball

Question 2. [Total 10 marks] Structural testing

Answer the following questions using function even_sum in Figure 1.

a. [2 marks] Draw the control flow graph for function even_sum in Figure 1.



b. [3 marks] Give a test suite with one test case that satisfies the basic condition adequacy criterion without satisfying the branch adequacy criterion for function even_sum in Figure 1. Explain your answer in one-two sentences. What are the basic condition coverage and the branch coverage for the chosen test case?

Ans)

Test Case: n = 2 , a = [1, 4]

This test case makes sure that both parts of the “if” condition are true and false once each without the entire condition becoming true even once and thus we achieve the required.

Basic condition coverage: 100%
Branch coverage: 75%

c. [3 marks] Give a test suite with one test case that satisfies the statement adequacy criterion without satisfying the branch adequacy criterion for function `even_sum` in Figure 1. Explain your answer in one-two sentences. What are the statement coverage and the branch coverage for the chosen test case?

Ans)

Test Case: n = 1, a = [4]

Since there is no “else” statement for the “if” condition at Line 5, this test case gets 100% statement coverage without covering the “else” part (or branch) of the “if” condition”. Thus resulting in satisfying the statement adequacy criterion but not the branch adequacy criterion.

Statement Coverage = 100%
Branch Coverage = 75%

d. [2 marks] Give a test suite with one test case that achieves 100% path coverage for function `even_sum` in Figure 1. Explain your answer in one-two sentences.

Ans)

Test Case: n=4, a=[4,2,3,3]

This test case will ensure that all the paths covered for the function even_sum. This will evaluate the “if” condition with two conditions in the following order -> (true,true), (true,false), (false,true) and (false, false)

Question 3. [Total 10 marks] Condition coverage

Consider the following branch condition ((a || b) || (c && d)), where a, b, c, d take Boolean values.

a. [2 marks] Give a test suite that achieves **basic condition adequacy** for condition ((a || b) || (c && d)). Explain why each test case has been added to the suite.

Ans)

a	b	c	d	Outcome
T	F	T	T	T
F	T	F	F	F

Each test case makes sure that all the basic conditions a,b,c,d have been evaluated to both true and false which fulfills the basic condition frequency.

b. [3 marks] Give a test suite that achieves **compound condition adequacy** for condition ((a || b) || (c && d)). You may apply short-circuit evaluation. Explain why each test case has been added to the suite.

Ans)

a	b	c	d	Outcome
T	-	-	-	T
F	T	-	-	T
F	F	T	F	F
F	F	F	-	F
F	F	T	T	T

Each test case makes sure that the compound condition adequacy is fulfilled to some extent and has therefore been added in the case.

c. [3 marks] Give a test suite that achiever **MC/DC adequacy** for condition ((a || b) || (c && d)). Explain why each test case has been added to the suite.

Ans)

a	b	c	d	Outcome
T	-	-	-	T
F	T	-	-	T
F	F	T	F	F
F	F	F	-	F
F	F	T	T	T

Each test case makes sure that every point of entry and exit in the program has been invoked at least once, and every decision in the program has taken on all possible outcomes at least once and that is why they have been added. For example, 1 and 3 make the above sure for a.

d. [2 marks] If you are to test code that contains condition ((a || b) || (c && d)), which coverage criteria would you choose to use and why?

Ans)

I would select MC/DC as my coverage criteria because this criterion requires that every point of entry and exit in the program has been invoked at least once, and every decision in the program has taken on all possible outcomes at least once. This is done in a systematic manner and this reduces the test cases to a large extent despite covering all entry and exits and all decisions.

Question 4. [Total 10 marks] Feasible paths and symbolic execution

a. [5 marks] Assume that n is replaced with 100 in line 4. of Figure 1. How many paths for function `even_sum` in Figure 1? How many of these are feasible, and how many are infeasible? Explain your answer.

Ans)

b. [5 marks] Consider the following program with line numbers shown. Illustrate your approach to use symbolic execution to systematically generate various tests which reaches line 7 and prints “How to get here?”. Show your test cases and how the test cases have been added to the test suite.

```
Line    Code
1.      input x, y, z;
2.      if (y > 0){
3.           z = y * 2;
4.           x = y - 2;
5.           x = x - 2; }
6.      if (z == x){
7.           print ("How to get here?");
```

Ans)

a) Let’s consider a random concrete case to begin with where **initial values are x = 0, y =2 , z = 0**

On running this code with the values we get,
Path condition: (y>0) ^ not(y*2 == y-4)
This means this path condition doesn’t execute/reach line 7.

b) Now lets minimally modify the last path condition to get a new path. Therefore, our new path comes to,
Path condition: (y>0) ^ (y*2==y-4)
This means this path condition will reach/ execute line 7. But we realize **this is an infeasible path** as its not possible for the given path condition to be true.

c) Now lets minimally modify the last path condition to get a new path. Therefore, our new path comes to,
Path condition: not (y>0) ^ (z==x)
This means this path condition will reach/ execute line 7.

d) Now lets minimally modify the last path condition to get a new path. Therefore, our new path comes to,
Path condition: not(y>0) ^ not (z==x)
This means this path condition will not reach/ execute line 7.

As a result of the 4 path conditions obtained above, we can generate 1 test which will reach line 7 and print “How to get here?” and generate 2 tests which will not reach line 7.

Test Cases which will reach line 7:
c) x = 2, y = -4 , z = 2

Test Cases which will not reach line 7:
a) x = 0, y = 2 , z = 0
d) x = 3,y = -2, z = 4

END of PAPER