MIDTERM W2016

Student ID: **Student Name:**

Signature: **Course Number:**

Department of Electrical and Computer Engineering
SE465/ECE453/CS447/ECE653/CS647 Software Testing, Quality Assurance & Maintenance
Instructor: Lin Tan
Examination Date and Time: Monday February 22, 2016, 7:00 PM - 8:20 PM

Rooms:	ECE 453	MC-4063	
	ECE 653	MC-4058	**20174109–20625436**
		MC-4060	**20625610–20648131**
	CS 647	MC-4060	
	CS 447	MC-4041	**20218863–20431753**
		MC-4042	**20432543–20647635**
	SE 465 - 001	MC-1056	**20412954–20508706**
		MC-2034	**20508921–20515947**
		MC-2035	**20515993–20529147**
		MC-4064	**20529417–20540967**
	SE 465 - 002	MC-2038	**20335244–20426751**
		MC-4040	**20427598–20643546**

Instructions:

- You have **80 minutes** to complete the exam.
- You can bring **printed or handwritten** material, e.g., books, slides, notes, etc.
- If you separate the pages, make sure your names and student IDs are on the top of every page.
- Unauthorized duplication or archival of these questions is not permitted. To be returned with the exam booklet after the completion of the exam.
- If information appears to be missing from a question, make a reasonable assumption, state your assumption, and proceed. Do not simplify the question.
- You must answer questions in the space provided.
- **Illegible answers receive NO point.**

Question	Mark	Points
Q1		**15**
Q2		**20**
Q3		**15**
Q4		**15**
Q5		**35**
Total:		**100**

Question 1 (15 points)

The following C program is expected to determine if the given number `num_to_find` exists in the given array `list`. It expects that the length of the array is `list_length`.

```
1 int contains_num (int list[], int num_to_find, int list_length) {
2     for (int i = 0; i <= list_length; i++) {
3         if (list[i] == num_to_find) {
4             return 1;
5         }
6     }
7     return -1;
8 }
```

(a) Identify the fault in the program and propose a fix. [5 points]

(b) If possible, give a test case that does **not** execute the fault. If no such test case exists, explain why. [5 points]

(c) If possible, give a test case that executes the fault and does **not** result in a failure. If no such test case exists, explain why. [5 points]

Question 2 (20 points)

(a) **Coverity.** Provide one concrete example of a bug that we can infer based on a MUST belief and one concrete example of a bug that we can infer based on a MAY belief. For each example, you must use a valid C/C++ or Java code snippet. [6 points]

(b) **Tools.** One of these tools is not like the others. Which one, and why? Coverity Static Analyzer, Valgrind, EvoSuite, Randoop, and American fuzzy lop. [4 points]

(c) **Security bugs.** What common security bugs are associated with library functions `strcpy()`, `strcat()`, and `sprintf()`? [3 points]

(d) **Concurrency bugs.** What is a benign data race? [2 points] Point out the fault in the Java code snippet and propose a fix. [5 points]

```
1  String sCurrentLine = "";
2  try {
3      lockForFile();
4      String scopeName = null;
5      while ((sCurrentLine = br.readLine()) != null) {
6          if (sCurrentLine.startsWith("Call")) {
7              scopeName = sCurrentLine.substring(sCurrentLine.indexOf("\'") + 1,sCurrentLine.lastIndexOf("\'"));
8          }
9          else if (sCurrentLine.contains("calls function")) {
10             String funcName = sCurrentLine.substring(sCurrentLine.indexOf("\'") + 1,sCurrentLine.lastIndexOf("\'"));
11             if (map_Scope_Funcs.get(hashfunc(scopeName)) == null) {
12                 HashSet<Integer> temp = new HashSet<Integer>();
13                 int functionHashCode = hashfunc(funcName);
14                 temp.add(functionHashCode);
15                 map_Scope_Funcs.put(hashfunc(scopeName), temp);
16             } else {
17                 map_Scope_Funcs.get(hashfunc(scopeName)).add(hashfunc(funcName));
18             }
19         }
20     }
21     unlockForFile();
22 }
23 catch (IOException e) {
24     e.printStackTrace();
25 }
```

Question 3 (15 points)

Propose two distinct non-stillborn and non-equivalent mutants for the following method. Say which mutation operator you used and where you apply it. Show that you can kill the mutants, demonstrating that they are non-equivalent mutants, by writing down test cases and relevant outputs for each of these mutants and the original method.

```
1  //For a >=0 and b >= 0, this method calculates the greatest common divisor of a and b.
2  public static int gcd(int a, int b) {
3      if (a == 0) {
4        return b;
5      }
6
7      while (b != 0) {
8        if (a > b)
9          a = a - b;
10       else
11         b = b - a;
12     }
13     return a;
14 }
```

Question 4 (15 points)

In this question, we will work with Prime Path Coverage (PPC), All-du-Paths Coverage (ADUPC), and two new coverage criteria: Bridge Coverage (BC) and Simple Test Path Coverate (STPC). If removing an edge adds unreachable nodes to the graph, then this edge is a bridge. The set of test requirements for BC contains all bridges. The set of test requirements for STPC contains all simple test paths. A simple test path is a test path that is also simple. For this problem, assume that a graph contains at least two nodes, and all nodes in a graph are reachable from the initial nodes.

(a) Does BC subsume STPC? If yes, justify your answer. If no, give a counterexample. Simply saying yes or no receives no points. [8 points]

(b) Does PPC subsume ADUPC? If yes, justify your answer. If no, give a counterexample. Simply saying yes or no receives no points. [7 points]

Question 5 (35 points)

The following function `pow()` takes two integers `x` and `y` as input, and computes `x` to the power of `y`.

```
1  void pow (int x, int y) {
2      double z;
3      int p;
4      if(y<0)
5          p = 0 - y;
6      else
7          p = y;
8      z = 1.0;
9      do {
10         z = z*x;
11         p = p -1;
12     } while(p!=0)
13     if(y<0)
14        z =1.0/z;
15     print(z);
16 }
```

(a) Draw the minimal node (hint: 8) control flow graph (CFG) for the function `pow()`. You must use only the line numbers provided above as the content of each node (e.g., 20–25, if lines 20–25 belong to a basic block), and label the nodes using only circled lowercase letters (i.e., nodes (a), (b), (c), (d), (e), (f), (g), and (h)). Include line 1 as part of (a). Ensure the graph is clear and legible. Unclear graphs will not receive full points. [5 points]

(b) Create the Def-Use CFG for variables `z`, `p`, and `y` in `pow()` by reproducing the graph from (a), filling the nodes with def-set(s) and use-set(s). Include the implicit definitions at the beginning of the function as part of (a). Ensure the graph is clear and legible. Unclear graphs will not receive full points. [6 points]

(c) List test requirements for Prime Path Coverage (PPC) for `pow()`. [7 points]

(d) Enumerate def-pair sets and list the test requirements for All-Uses Coverage (AUC) for the variable `z` in `pow()`. [7 points]

(e) There is at least one fault in the program. Identify one fault [5 points], find a test case that executes the fault [3 points], and fix the fault [2 points].

(This page is intentionally left blank.)

(This page is intentionally left blank.)