(ECE453/CS447/ECE653/CS647/SE465) Software Testing, Quality Assurance & Maintenance: Assignment/Lab 2 (70 Points), Version 2

Instructor: Lin Tan
Release Date: February 11, 2013
Revision Date: February 12, 2013

Due: 5:00 PM, Thursday, February 28th, 2013
Submit: An electronic copy on LEARN

Please download "sll_buggy.c" for Q2 from LEARN.

I expect each of you to do the assignment independently. I will follow UW's Policy 71 if I discover any cases of plagiarism.

Submission Instructions:

Please read the following instructions carefully. **If you do not follow the instructions, you may be penalized up to 5 points.** Illegible answers receive no point.

Electronic submission: Go to "Dropbox" –⟩ "Submit A2" on LEARN. Submit only one file in .tar.gz format. Please name your file

<FirstName>-<LastName>-<StudentNo>.tar.gz

For example, use John-Smith-12345678.tar.gz if you are John Smith with student No 12345678. *You must include your file name as part of your submission title.* The .tar.gz file should contain the following items:

- a single pdf file "*a2_sub.pdf*".
  You must include a cover page that contains your full name, 8-digit student No, the class number (one of ECE453, CS447, ECE653, CS647, & SE465), and your uwaterloo email address.
- "*sll_fixed.c*" for Question 2

You can submit multiple times. After submission, you can view your submissions to make sure you have uploaded the right files/versions.

Question	TA in Charge
1	Tian
2	Pei
3	Akramul
4	Jeff

Question 1 (10 points)

For an acyclic graph, does Prime Path Coverage (PPC) subsume Complete Path Coverage (CPC)? Answer yes or no explicitly and then justify your answer.

Question 2 (20 points)

Compile program *sll_buggy.c* with -g option, execute it with Valgrind to find some bugs, and answer the following questions:

(a) Run TC1 and report the memory problem you find. Briefly explain the problem. *Paste the Valgrind outputs.* Fix the bug, and submit your code "*sll_fixed.c*". Using comments, e.g., "Fix for Q2 (a)", to mark the bug fixes in your program, and explain how you fix the bugs here in the report. (7 Points)

(b) Run TC2 and report the memory problem you find. Briefly explain the problem. Fix the bug, and submit your code "*sll_fixed.c*". Using comments, e.g., "Fix for Q2 (b)", to mark the bug fixes in your program, and explain how you fix the bug here in the report. (6 Points)

(c) Generate a test case that would cause a new bug, e.g., a buffer overflow bug. Attach the test case, and briefly explain the problem. Fix the bug, and submit your code "*sll_fixed.c*". Using comments, e.g., "Fix for Q2 (c)", to mark the bug fixes in your program, and explain how you fix the bug here in the report. (7 Points)

Note that you only submit one *sll_fixed.c* file with all the fixes. For instructions on downloading Valgrind and installing it, please refer to the tutorial slides. You should do this exercise in Unix/Linux. You may find 'gdb' very useful. Our ECE Linux systems (ecelinux.uwaterloo.ca) should have Valgrind installed. If you have any questions regarding the access to these machines, or the installation of Valgrind, please contact Bernie.

TC 1

```
insert>insert>delete>exit

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>100
enter the name:>Tom

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>111
enter the name:>Mary

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:d
enter the tel:>111

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:x
```

TC 2

```
insert>insert>insert>edit>delete all>exit

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>100
enter the name:>Tom

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>111
enter the name:>Mary

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:i
enter the tel:>112
enter the name:>John

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:e
enter the old tel :>111
enter the new tel :>111
enter the new name:>Mary

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:a

[(i)nsert,(d)elet,delete (a)ll,d(u)plicate,(e)dit,(p)rint,e(x)it]:x
```

Question 3 (10 points)

Consider the following binary search function from en.literateprograms.org. Propose two non-stillborn and non-equivalent mutants of this function. Write down test inputs which strongly kill these mutants (syntax doesn't matter) and the expected output of your test cases on the original code and on the mutant.

```
int binary_search(int a[], int low, int high, int target) {
    // binary search for target in a[low, high]
    while (low <= high) {
        int middle = low + (high - low)/2;
        if (target < a[middle])
            high = middle - 1;
        else if (target > a[middle])
            low = middle + 1;
        else
            return middle;
    }
    return -1; // return -1 if target is not found in a[low, high]
}
```

Question 4 (30 points)

(Adapted from the original version by Patrick Lam.)

We will identify test requirements and augment a test suite to achieve prime path coverage of a specific method in the free Java application SweetHome3D, available at

`http://www.sweethome3d.eu.`

Download the source code from LEARN (version 3.7). We will work with `computeIntersection()` in class `PlanController`.

(a) Creating a CFG (6 points). Draw the control-flow graph for the method `computeIntersection()`. It will be unmanageable unless you group together statements into basic blocks. Then draw the CFG again by replacing the content in the nodes with defs and uses.

(b) Identifying requirements for PPC (4 points). Identify the test requirements for prime path coverage in `computeIntersection()`.

(c) Identifying requirements for ADC, AUC, ADUPC (7 points). Identify the test requirements for All-Defs Coverage, All-Uses Coverage, and All-du-paths Coverage in `computeIntersection()`. Don't forget about the paths where the defs and uses of the same variable are in the same node.

(d) Comparing coverage criteria (4 points). Compare the sets of test requirements you've collected. Point out *strengths and weaknesses* of these criteria, for instance by giving scenarios where one criterion is going to help you find something that another criterion wouldn't; discuss whether the additional number of test requirements is worth it in this case. Simply saying one criterion subsumes another is not enough.

(e) Creating tests (9 points). Create tests which achieve prime path coverage in `computeIntersection()`. Justify your answer.