(ECE453/CS447/CS647/SE465) Software Testing, Quality Assurance & Maintenance: Assignment 1 (100 Points)

Instructor: Lin Tan

Due: 5:00 PM, Thursday, Feb. 3rd, 2011
Submit: both an electronic copy on UWACE
and
a hardcopy to Masoomeh in DC 2546

Please download a1.tar from the course website to get the necessary source code and test cases needed for finishing this assignment/lab.

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 10 points.

Electronic submission: Go to "Lessons" $->$ "Submit A1" on UWACE. Submit only one file in .tar or .zip format. Please name your file

<FirstName>-<LastName>-<StudentID>.[tar/zip]

For example, use John-Smith-12345678.tar or John-Smith-12345678.zip if you are John Smith with student ID 12345678. Please include your file name as part of your submission title. The .tar or .zip file should contain the following items:

- a single pdf file "a1_sub.pdf"
- a directory "q1" that contains your code (source code only; no binaries or object files) for Question 1.
- a directory "q4" that contains your test cases for Question 4.
- "CFGTest.java" and "CFG.java" for Question 6.

You can submit up to 5 times.

Hardcopy submission: Please print a1_sub.pdf and submit it to Masoomeh in DC 2546. No need to print your code or test cases.

Question 1 (10 points)

Write one simple program that prints out the value of -3%2 in at least 4 programming languages (including scripting languages), e.g., C/C++, Java, Perl, and Python (You will write 4 programs in total). Report what you have found, and discuss at least 2 strategies to cope with such a problem.

Question 2 (10 points)

Below is a faulty **Java** program, which includes a test case that results in failure. Answer the following questions for this program.

(a) Identify the fault, and fix the fault.

(b) If possible, identify a test case that does not execute the fault.

(c) If possible, identify a test case that executes the fault, but does not result in an error state.

(d) If possible identify a test case that results in an error, but not a failure. Hint: Don't forget about the program counter.

(e) For the given test case, identify the first error state. Be sure to describe the complete state.

```
public static int oddOrPos(int[] x)
{
//Effects: if x==null throw NullPointerException
// else return the number of elements in x that
//      are either odd or positive (or both)
   int count = 0;
   for (int i = 0; i < x.length; i++)
   {
      if (x[i]%2 == 1 || x[i] > 0)
      {
         count++;
      }
   }
   return count;
}
   // test:  x=[-3, -2, 0, 1, 4]
   //        Expected = 3
```

Question 3 (20 points)

Learn about quality assurance in some other domain of engineering. Write a short essay (1 page) comparing software quality assurance with the domain of your choice. Include about 2 references[1]. CS647 students: I expect at least one of the references to be more specialized than a textbook.

[1] Wikipedia is not an acceptable reference.

Question 4 (20 points)

Read the faulty a1/print_token2 program. Some test cases (inputs and outputs) are provided. Generate more tests to find failure(s) caused by the fault, and then identify the fault. Describe the failure(s), the fault and how you find them through testing and debugging. Be specific in describing how you design the tests to find the failure(s), and how you debug to find the fault. If you simply say what the fault is, you will receive at most 5 points. Submit a short report of 1 page and all test cases you create.

Question 5 (10 points)

(Original version by Patrick Lam, with modifications by Lin Tan.) Consider the following (contrived) program.

```
class M {
  public static void main(String[] argv) {
    M obj = new M();
    if (argv.length > 0)
      obj.m(argv[0], argv.length);
  }

  public void m(String arg, int i) {
    int q = 1;
    A o = null;

    if (i == 0)
      q = 4;
    q++;
    switch (arg.length()) {
      case 0: q /= 2; break;
      case 1: o = new A(); q = 10; break;
      case 2: o = new B(); q = 5; break;
      case 3: o = new A(); new B(); q = 25; break;
      default: o = new B(); break;
    }
    if (arg.length() > 0) {
      o.m();
    } else {
      System.out.println("zero");
    }
  }
}

class A {
  public void m() {
    System.out.println("a");
  }
}

class B extends A {
  public void m() {
    System.out.println("b");
  }
}
```

Write JUnit test sets (i.e. a JUnit test class) which achieves, for method `M.m()`, (1) node coverage; (2) edge coverage; and (3) edge-pair coverage.

Question 6 (30 points)

(Original version by Sarfraz Khurshid and Patrick Lam, with modifications by Lin Tan.)

You are to construct a partial[2] control-flow graph from the bytecode[3] of a given Java class using the ASM framework, version 3.2[4].

To illustrate, consider the following class C:

```
public class C {
     int max(int x, int y) {
          if (x < y) {
              return y;
          } else return x;
     }
}
```

which can be represented in bytecode as (e.g. the output of `javap -c`):

```
Compiled from "C.java"
public class ee379k.pset1.C extends java.lang.Object{
public C();
   Code:
     0:   aload_0
     1:   invokespecial     #8; //Method java/lang/Object."<init>":()V
     4:   return

int max(int, int);
   Code:
     0:   iload_1
     1:   iload_2
     2:   if_icmpge         7
     5:   iload_2
     6:   ireturn
     7:   iload_1
     8:   ireturn
}
```

You can easily draw the corresponding control-flow graph.

Graph representation of control-flow. Implement the class `CFG` to model control-flow in a Java bytecode program. A `CFG` object has a set of nodes that represent bytecode statements and a set of edges that represent the flow of control (branches) among statements. Each node contains:

- an integer that represents the position (bytecode line number) of the statement in the method.
- a reference to the method (an object of `org.objectweb.asm.tree.MethodNode`) containing the bytecode statement; and
- a reference to the class (an object of class `org.objectweb.asm.tree.ClassNode`) that defines the method.

Represent the set of nodes using a `java.util.HashSet` object, and the set of edges using a `java.util.HashMap` object, which maps a node to the set of its neighbors. Ensure the sets of nodes and edges have values that are consistent, i.e., for any edge, say from node a to node b, both a and b are in the set of nodes. Moreover, ensure that

[2]This assignment ignores the labels that are traditionally annotated on nodes and edges, as well as the edges that correspond to method invocations or `jsr[ w]` bytecodes.

[3]`http://java.sun.com/docs/books/jvms/secondedition/html/VMSpecTOC.doc.html`

[4]`http://asm.ow2.org`

for any node, say, n, the map maps n to a non-null set, which is empty if the node has no neighbours.

You can find a partial implementation of the `CFG` class at

`a1/CFG_partial.java`;

you'll need to rename it to `CFG.java` to continue. Your first task is to fill in the implementation of this class.

Adding a node (2 points). Implement the method `CFG.addNode` such that it creates a new node with the given values and adds it to nodes as well as initializes edges to map the node to an empty set. If the graph already contains a node that is `.equals` to the new node, `addNode` does not modify the graph.

Adding an edge (2 points). Implement the method `CFG.addEdge` such that it adds an edge from the node `(p1, m1, c1)` to the node `(p2, m2, c2)`. Your implementation should update nodes to maintain its consistency with edges as needed.

Deleting a node (4 points). Implement the method `CFG.deleteNode` such that it deletes a node with the given values, and all edges connected to the node, if any. If the graph does not contain a node that is `.equals` to the new node, `deleteNode` does not modify the graph. Your implementation should update edges to maintain its consistency with nodes as needed.

Deleting an edge (4 points). Implement the method `CFG.deleteEdge` such that it deletes an edge from the node `(p1, m1, c1)` to the node `(p2, m2, c2)`.

Reachability (4 points). Implement the method `CFG.isReachable` such that it traverses the control-flow graph starting at the node represented by `(p1, m1, c1)` and ending at the node represented by `(p2, m2, c2)` to determine if there exists any path from the given start node to the given end node. If the start node or the end node are not in the graph, the method returns false.

Testing (14 points). I've also provided a class `CFGTest`, at

`a1/CFGTest.java`

which exercises your `CFG` class. Examine this test class. Which coverage criteria does it satisfy? (Justify your answer.) If it doesn't satisfy any interesting coverage criteria, write additional JUnit test cases which make it satisfy some coverage criterion.