CS 6340 – Spring 2013 – Assignment 8 **Name** Sam Britt, Shriram Swaminathan,
Assigned: March 4, 2013
Due: March 13, 2013 **Name** and Sivaramachandran Ganesan

At the beginning of class on the due date, submit your neatly presented solution with this page stapled to the front (100 points).

Part 1

Your new position as Test Manager requires that you establish a set of requirements that developers will use for unit testing of the software that they write. Before you establish these requirements, you want to assess the fault-detection ability, expense, tool availability, etc. of various techniques that have been proposed in the literature. To do this, you will use a program, which we'll call **tritype**, that has the following requirements specification

> **tritype** takes as input three integer values. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.

You are to do the following:

1. Use the specification to develop a set of test cases (a test suite) for **tritype** using two black box testing methods (both described in “EquivalencePartitioningBoundaryValue:”
   - Equivalence Partitioning
   - Boundary Value Analysis

2. Create a file of test cases (reason for test (in quotes), inputs, expected outputs) that consists of one test case per line; the number of the test case will be the line number in the file. For example, suppose I created two test cases:
   Test Case 1: isosceles 2 2 3 isosceles
   Test Case 2: equilateral 4 4 4 equilateral
   The file should contain
   "isosceles" 2 2 3 isosceles
   "equilateral" 4 4 4 equilateral

3. Send the test cases to Sangmin, and he’ll send you the **tritype** program for the second part of the assignment. Let him know whether you want the C version or the Java version.

Sam Britt
Shriram Swaminathan
Sivaramachandran Ganesan

CS 6340
Assignment 8
Mar. 12, 2013

Our test suite, along with the test results, is shown in Table 1 below. We achieved **100 % statement coverage** with this test suite; Table 2 shows which tests cover each statement. Our test suite achieved **65.8 % multiple condition coverage**; the coverage is tabulated in Table 3. When considering just the subset of conditions necessary for MC/DC coverage, our suite achieves **92.6 % MC/DC coverage**. These data are tabulated in Table 4.

The CFG for `tritype.c` can be seen in Figure 1. The cyclomatic complexity is 11, obtained by counting the number of decision statements in the CFG and adding one. We identified ten linearly independent paths from Table 2, resulting in **90.9 % basis path coverage**.

The statement coverage results were obtained by using a debugger to step through an execution of the program for each test case, while marking which statements were executed. For multiple condition and MC/DC coverage, we used a similar procedure: after identifying all the truth vectors for each decision in the program, we stepped through each test case in a debugger and marked which truth vector was executed by each test. Some reasons for the low multiple condition coverage are:

1. The test suite does not have a test case with multiple negative inputs.
2. The test suite does not have as many test cases for scalene output as required. Line 63 has three conditions and eight truth vectors; our test suite covers only two of them.

The program has a binary operator bugs at lines 41 and 63. At line 41, the check should be for `k<=0` and not `k<0`. This bug was identified by test case 9 for which inputs $(\texttt{i}, \texttt{j}, \texttt{k})$ were $(1, 1, 0)$ and expected output was "invalid," but the actual output was found to be "isosceles." Similarly, at line 63, the condition should be `i+k<=j` and not `i+k<j`. However, this bug was not identified by our test suite; this is a result of the poor condition coverage of for the decision on line 63 (see Table 3), as well as not properly testing the boundary conditions for violation of the triangle inequality; e.g., the case where $\texttt{i} + \texttt{k} = \texttt{j}$ for $\texttt{i}, \texttt{j}, \texttt{k} \neq 0$.

The program accepts invalid inputs like character strings instead of strictly integer inputs. Test case 12 expects "invalid" as output, but the execution resulted in "isosceles." This illustrates that input values are not validated properly by the program. However, tests 10 and 11 both passed, even though they should not have. This is because of the unpredictable nature of the bug—the compiler has left some variables undefined, and the result coincidentally caused the tests to pass.

Table 1: Test Suite

Test ID	Reason for test	Input: (i, j, k)	Expected Output	Test Result
1	Violation of triangle inequality	$(10, 1, 1)$	invalid	PASS
2	Violation of triangle inequality	$(1, 10, 1)$	invalid	PASS
3	Violation of triangle inequality	$(1, 1, 10)$	invalid	PASS
4	Negative input	$(-1, 1, 1)$	invalid	PASS
5	Negative input	$(1, -1, 1)$	invalid	PASS
6	Negative input	$(1, 1, -1)$	invalid	PASS
7	Zero input	$(0, 1, 1)$	invalid	PASS
8	Zero input	$(1, 0, 1)$	invalid	PASS
9	Zero input	$(1, 1, 0)$	invalid	FAIL
10	Invalid input	$(\texttt{w}, 1, 1)$	invalid	PASS
11	Invalid input	$(1, \texttt{w}, 1)$	invalid	PASS
12	Invalid input	$(1, 1, \texttt{w})$	invalid	FAIL
13	Equilateral	$(1, 1, 1)$	equilateral	PASS
14	Scalene	$(4, 3, 2)$	scalene	PASS
15	Isosceles	$(2, 2, 3)$	isosceles	PASS
16	Isosceles	$(2, 3, 2)$	isosceles	PASS
17	Isosceles	$(3, 2, 2)$	isosceles	PASS

Table 2: Statement Coverage per Test

	Statement line number																							
Test ID	33	41	43	47	48	49	50	51	52	53	55	63	64	66	74	75	76	77	78	79	80	81	83	86
1	✓	✓		✓	✓		✓		✓	✓	✓				✓		✓		✓		✓		✓	✓
2	✓	✓		✓	✓		✓	✓	✓		✓				✓		✓		✓		✓		✓	✓
3	✓	✓		✓	✓	✓	✓		✓		✓				✓		✓		✓		✓		✓	✓
4	✓	✓	✓																					✓
5	✓	✓	✓																					✓
6	✓	✓	✓																					✓
7	✓	✓	✓																					✓
8	✓	✓	✓																					✓
9	✓	✓		✓	✓	✓	✓		✓		✓				✓		✓	✓						✓
10	✓	✓	✓																					✓
11	✓	✓		✓	✓		✓		✓		✓	✓	✓											✓
12	✓	✓		✓	✓	✓	✓		✓		✓				✓		✓	✓						✓
13	✓	✓		✓	✓		✓	✓	✓	✓	✓				✓	✓								✓
14	✓	✓		✓	✓		✓		✓		✓	✓		✓										✓
15	✓	✓		✓	✓	✓	✓		✓		✓				✓		✓	✓						✓
16	✓	✓		✓	✓		✓	✓	✓		✓				✓		✓		✓	✓				✓
17	✓	✓		✓	✓		✓		✓	✓	✓				✓		✓		✓		✓	✓		✓

Table 3: Multiple Condition Coverage, per Condition

Decision (Result)	Coverage	Conditions (Possible Evaluations)		
Line 41	Covered?	`i<=0`	`j<=0`	`k<0`
F	✓	F	F	F
T	✓	F	F	T
T	✓	F	T	F
T		F	T	T
T	✓	T	F	F
T		T	F	T
T		T	T	F
T		T	T	T
Line 48	Covered?	`i == j`		
T	✓	T		
F	✓	F		
Line 50	Covered?	`i==k`		
T	✓	T		
F	✓	F		
Line 52	Covered?	`j==k`		
T	✓	T		
F	✓	F		
Line 55	Covered?	`triang==0`		
T	✓	T		
F	✓	F		

Decision (Result)	Coverage	Conditions (Possible Evaluations)		
Line 63	Covered?	`i+j<=k`	`j+k<=i`	`i+k<j`
F	✓	F	F	F
T	✓	F	F	T
T		F	T	F
T		F	T	T
T		T	F	F
T		T	F	T
T		T	T	F
T		T	T	T
Line 74	Covered?	`triang>3`		
T	✓	T		
F	✓	F		
Line 76	Covered?	`triang==1`	`i+j>k`	
F		F	F	
F	✓	F	T	
F	✓	T	F	
T	✓	T	T	
Line 78	Covered?	`triang==2`	`i+k>j`	
F		F	F	
F	✓	F	T	
F	✓	T	F	
T	✓	T	T	
Line 80	Covered?	`triang==3`	`j+k>i`	
F		F	F	
F	✓	F	T	
F	✓	T	F	
T	✓	T	T	

Table 4: MC/DC Coverage, per Condition

Decision (Result)	Coverage	Conditions (Possible Evaluations)			Decision (Result)	Coverage	Conditions (Possible Evaluations)		
Line 41	Covered?	`i<=0`	`j<=0`	`k<0`	Line 63	Covered?	`i+j<=k`	`j+k<=i`	`i+k<j`
F	✓	F	F	F	F	✓	F	F	F
T	✓	F	F	T	T	✓	F	F	T
T	✓	F	T	F	T		F	T	F
T	✓	T	F	F	T		T	F	F
Line 48	Covered?	`i == j`			Line 74	Covered?	`triang>3`		
T	✓	T			T	✓	T		
F	✓	F			F	✓	F		
Line 50	Covered?	`i==k`			Line 76	Covered?	`triang==1`	`i+j>k`	
T	✓	T			F	✓	F	T	
F	✓	F			F	✓	T	F	
					T	✓	T	T	
Line 52	Covered?	`j==k`			Line 78	Covered?	`triang==2`	`i+k>j`	
T	✓	T			F	✓	F	T	
F	✓	F			F	✓	T	F	
					T	✓	T	T	
Line 55	Covered?	`triang==0`			Line 80	Covered?	`triang==3`	`j+k>i`	
T	✓	T			F	✓	F	T	
F	✓	F			F	✓	T	F	
					T	✓	T	T	

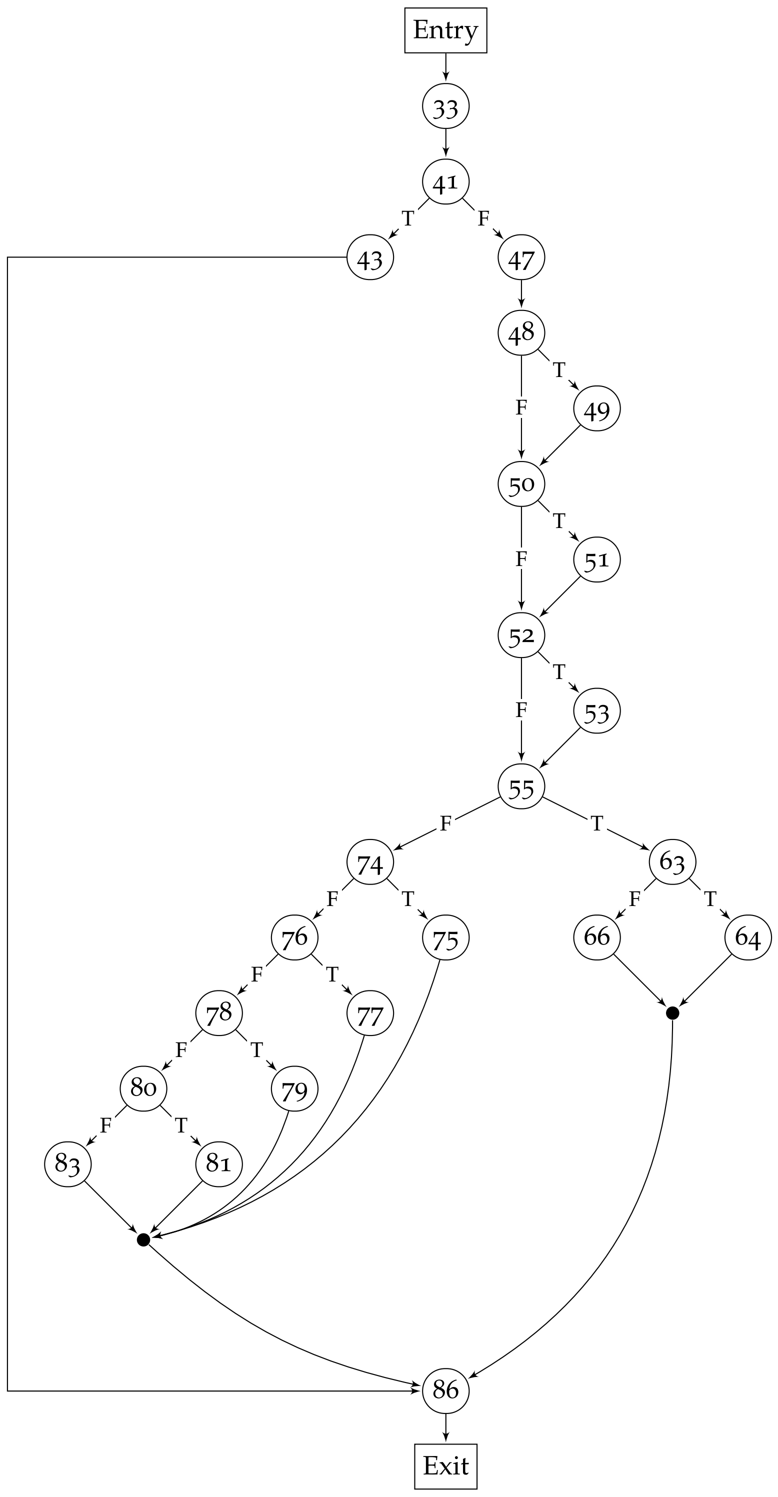

Figure 1: CFG for `tritype.c`