CS 6340 – Spring 2013 – Assignment 8
Assigned: March 27, 2013
Due: April 3, 2013

Name Sam Britt, Shriram Swaminathan,

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: Random Test-data Generation

a. Rewrite the **tritype** program so that it contains only single conditions (no multiple conditions) but has the same functionality as the original program. Call this new program **tritype_single_conditions.**
b. Write (implement) a **random_test_data_generator** that will generate sets of three integers (the inputs to **tritype_single_conditions**).
c. Using your **random_test_data_generator,** generate 15 test cases for **tritype-_single_conditions.** Present these test cases in the same format you used for Problem Set 6—e.g., if you have 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
d. Determine the branch coverage adequacy of the test suite you generated in (c).

For Part 1, submit

- the listing (typed) of **tritype_single_conditions** (from a)
- the listing of your **random_test_data_generator** (from b)
- your test suite (typed) (from c)
- branch coverage adequacy of your test suite, along with how you got it (from d).

Part 2: Goal-oriented or Path-oriented Test-data Generation

Select one of the test-data generation methods we discussed (symbolic execution, concolic execution, search-based (genetic algorithms) generation)—call it **my_test_data_generation,** and use it for the following:

a. Select one of the uncovered branches in **tritype_single_conditions** that requires the traversal of at least three branches to reach it—this will be the **target_branch** for your test-data generation.
b. Apply **my_test_data_generation** to generate test input to cover **target_branch.** Although there may be tools available for **my_test_data_generation,** you are not to use them. Instead do the work yourself, and show all steps, so that it will be easy for the reader to understand your approach.

Sam Britt
Shriram Swaminathan
Sivaramachandran Ganesan

CS 6340
Assignment 9
April 3, 2013

Random Test-data Generation

Our `tritype_single_conditions.c` is in Listing 1 below. Our random test generation program is in Listing 2, and the test suite it produced is in Table 1. We determined our branch coverage be 41.18 %. This value was obtained by using a debugger to step through an execution of the program for each test case, while marking which branches were taken. This number was then divided by the total number of branches in the program (34) to obtain the coverage. The coverage of each branch evaluation, broken down by test case, is shown in Table 2, where a "✓" symbol indicates that a particular evaluation of a branch statement was covered by the test, and a "–" (or a blank) indicates that it the branch was not covered.

Path-oriented Test-data Generation

We decided to target the FALSE evaluation of the branch on line 21 using symbolic execution. We see that the execution of our target line 21 is conditionally dependent on the branches on lines 6, 8, and 10 evaluating to FALSE. Therefore, in order for execution to reach line 13, the input must satisfy

$$(\texttt{i} > 0) \wedge (\texttt{j} > 0) \wedge (\texttt{k} \geq 0). \tag{1}$$

Continuing, we see that `triang` is set to 0 on line 13, and there are branches on lines 14, 16, and 18. All three of these branches will be executed before execution reaches the target, which means there are eight possible paths from line 13 to the target on line 21, one for each combination of possible evaluations of the branches. Symbolically, we evaluated each of these eight branches and determined the value of the `triang` variable at the time execution reaches line 21. (Because the value of `triang` does not depend directly on the values of the inputs aside from its conditional dependence, we are able to evaluate `triang` concretely.) We find that, if one or more of the three branches evaluates to TRUE, the value of `triang` is different from 0 at the execution of the target branch at line 21; that is, the only way the target evaluates to TRUE is if all three branches evaluate to FALSE. Therefore, our target will be satisfied if any condition is TRUE—we take the branch at line 18 arbitrarily. Combining with the condition in Eqn. (1), we must choose inputs that satisfy

$$(\texttt{i} > 0) \wedge (\texttt{j} > 0) \wedge (\texttt{k} \geq 0) \wedge (\texttt{i} \neq \texttt{j}) \wedge (\texttt{i} \neq \texttt{k}) \wedge (\texttt{j} = \texttt{k}).$$

Solving this constraint is straightforward; we took $(\texttt{i}, \texttt{j}, \texttt{k}) = (1, 2, 2)$.

Table 1: Test Suite, generated randomly

Test ID	Reason for test	Generated Input			Expected Output
		i	j	k	
1	Random	−1 122 822 091	−1 226 680 922	−193 961 734	Invalid
2	Random	−1 182 307 965	−154 439 951	−1 240 139 973	Invalid
3	Random	1 402 051 776	914 798 430	1 612 220 316	Scalene
4	Random	−496 854 896	−449 175 289	−1 180 638 653	Invalid
5	Random	−1 987 922 192	−1 484 115 204	−266 467 487	Invalid
6	Random	883 411 989	305 256 757	945 915 839	Scalene
7	Random	−653 520 881	−95 223 433	−599 427 703	Invalid
8	Random	1 167 814 945	678 101 484	−1 475 198 278	Invalid
9	Random	−737 454 116	1 299 871 708	1 347 433 141	Invalid
10	Random	1 443 280 107	2 141 951 683	−562 629 834	Invalid
11	Random	688 264 798	1 650 225 101	1 408 651 086	Scalene
12	Random	−1 492 740 046	429 397 449	−722 649 152	Invalid
13	Random	775 843 237	882 820 983	1 825 593 824	Invalid
14	Random	1 596 782 873	35 105 140	1 602 193 698	Scalene
15	Random	−1 082 197 646	422 926 318	−1 677 901 821	Invalid

Table 2: Branch Coverage by Test

	Branch Statement Line Number																
	6	8	10	14	16	18	21	23	25	27	33	35	36	38	39	41	42
Test ID	T F	T F	T F	T F	T F	T F	T F	T F	T F	T F	T F	T F	T F	T F	T F	T F	T F
1	✓ –																
2	✓ –																
3	– ✓	– ✓	– ✓	– ✓	– ✓	– ✓	✓ –	✓ –									
4	✓ –																
5	✓ –																
6	– ✓	– ✓	– ✓	– ✓	– ✓	– ✓	✓ –	– ✓	– ✓	– ✓							
7	✓ –																
8	– ✓	– ✓	✓ –														
9	✓ –																
10	– ✓	– ✓	✓ –														
11	– ✓	– ✓	– ✓	– ✓	– ✓	– ✓	✓ –	✓ –									
12	✓ –																
13	– ✓	– ✓	– ✓	– ✓	– ✓	– ✓	✓ –	✓ –									
14	– ✓	– ✓	– ✓	– ✓	– ✓	– ✓	✓ –	– ✓	– ✓	✓ –							
15	✓ –																
Total	✓ ✓	– ✓	✓ ✓	– ✓	– ✓	– ✓	✓ –	✓ ✓	– ✓	✓ ✓	– –	– –	– –	– –	– –	– –	– –

Listing 1: Tritype, altered to execute only single conditions.

```
#include <stdio.h>
main() {
    int i, j, k, triang;
    scanf("%d %d %d", &i, &j, &k);

    if (i <= 0) {
        triang = 4;
    } else if (j <= 0) {
        triang = 4;
    } else if (k < 0) {
        triang = 4;
    } else {
        triang = 0;
        if (i == j)
            triang += 1;
        if (i == k)
            triang += 2;
        if (j == k)
            triang += 3;

        if (triang == 0) {
            /* Confirm it's a legal triangle before declaring it to be scalene */
            if ( i+j <= k )
                triang = 4;
            else if (j+k <= i)
                triang = 4;
            else if (i+k < j)
                triang = 4;
            else
                triang = 1;
        } else {
            /* Confirm it's a legal triangle before declaring it to be isosceles or equilateral */
            if (triang > 3) {
                triang = 3;
            } else if (triang == 1) {
                if (i+j > k)
                    triang  = 2;
            } else if (triang == 2) {
                if (i+k > j)
                    triang = 2;
            } else if (triang == 3) {
                if (j+k > i)
                    triang = 2;
            } else {
                triang = 4;
            }
        }
    }
    printf("triang = %d\n", triang);
}
```

Listing 2: The random test generation program

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_TESTS 15

/** Returns 1 or -1 picked at random */
int rand_sign() {
    // return 1 if odd, -1 if even
    return rand() % 2 ? 1 : -1;
}

/** Returns a random number between -RAND_MAX and RAND_MAX */
int random_int() {
    int r = rand();
    int sign = rand_sign();
    return sign * r;
}

int main(int argc, char const *argv[]) {
    srand(time(NULL));
    int i, j, k, t;
    for (t = 0; t < MAX_TESTS; ++t) {
        i = random_int();
        j = random_int();
        k = random_int();
        printf("\"Test %2d:\" %13d %13d %13d\n", t+1, i, j, k);
    }
}
```