Team International
Report: Deliverable #5: Fault Injection
Project: Celestia-g2
Date: 11.29.2016
Members: Gui Costa, Megan Landau, Tony Tang

Task: Design and inject 5 faults into the code you are testing that will cause at least 5 tests to fail, but hopefully not all tests to fail. Exercise your framework and analyze the results.

Test Case ID	Method	Input	Expected Output	Actual Output	Pass / Fail
001	square	2	4	4	Pass
002	square	10	100	100	Pass
003	sphereArea	5	314.159	314.159	Pass
004	cube	8	512	512	Pass
005	radToDeg	0	0	0	Pass
006	sphereArea	2.05	52.8102	52.8102	Pass
007	cube	-2	-8	-8	Pass
008	circleArea	-5.0	error	78.5398	Fail
009	radToDeg	5	286.479	286.479	Pass
010	cube	876	672221376	6.72221e+08	Fail
011	circleArea	0.0	0	0	Pass
012	sphereArea	0	0	0	Pass
013	square	-98209	9645007681	1055073089	Fail
014	square	800000	640000000000	49872896	Fail
015	square	0	0	0	Pass
016	cube	-1024	-1073741824	-1.07374e+09	Fail
017	cube	2.1111	9.4088	9.40863	Fail
018	sphereArea	100000	1.25664e+11	1.25664e+11	Pass
019	sphereArea	-300.05	error	1.13135e+06	Fail
020	circleArea	10	314.159	314.159	Pass
021	circleArea	155555555555	7.6e+22	7.60188e+22	Fail
022	radToDeg	1	57.2958	57.2958	Pass
023	radToDeg	-57	-3265.86	-3265.86	Pass
024	circleArea	500	785398	785398	Pass
025	radToDeg	125000000	7161972439.15	7.16197e+09	Fail

Figure 1: Test results before fault injections.

Below are detailed descriptions of what faults were injected, how each method was changed, which test cases were affected by the fault injection, a figure of the Test Results Table after fault injection, and an analysis of the results.

1. **First Fault Injection:**
    - **Description:** Inject a fault into method radToDeg method found in the mathlib.h library. Changed the first multiplication operation in the return statement to division.

    - **Original Method:**

```
template<class T> T radToDeg(T r)
{
      return r * 180 / static_cast<T>(PI);
```

```
}
```

- **Method with fault injection:**

```
template<class T> T radToDeg(T r)
{
      return r / 180 /static_cast<T>(PI);
}
```

- **Test cases failed:** testCase009, testCase022,testCase023, and testCase025

- **Test results table:**

Test Case ID	Method	Input	Expected Output	Actual Output	Pass / Fail
001	square	2	4	4	Pass
002	square	10	100	100	Pass
003	sphereArea	5	314.159	314.159	Pass
004	cube	8	512	512	Pass
005	radToDeg	0	0	0	Pass
006	sphereArea	2.05	52.8102	52.8102	Pass
007	cube	-2	-8	-8	Pass
008	circleArea	-5.0	error	78.5398	Fail
009	radToDeg	5	286.479	0.00884194	Fail
010	cube	876	672221376	6.72221e+08	Fail
011	circleArea	0.0	0	0	Pass
012	sphereArea	0	0	0	Pass
013	square	-98209	9645007681	1055073089	Fail
014	square	800000	640000000000	49872896	Fail
015	square	0	0	0	Pass
016	cube	-1024	-1073741824	-1.07374e+09	Fail
017	cube	2.1111	9.4088	9.40863	Fail
018	sphereArea	100000	1.25664e+11	1.25664e+11	Pass
019	sphereArea	-300.05	error	1.13135e+06	Fail
020	circleArea	10	314.159	314.159	Pass
021	circleArea	155555555555	7.6e+22	7.60188e+22	Fail
022	radToDeg	1	57.2958	0.00176839	Fail
023	radToDeg	-57	-3265.86	-0.100798	Fail
024	circleArea	500	785398	785398	Pass
025	radToDeg	125000000	7161972439.15	221049	Fail

Figure 2: Test results after first fault injection

- **Analysis:** According to the original results test cases 005, 009, 0022, and 0023 passed while test case 0025 failed. After the fault was injected test case 005 was the only test case to pass while test cases 009, 022, 023, and 025 failed. Test cases 009, 022, and 023 were the only test cases to change from a pass to a fail. The result of test case 005 and test case 025 remained the same.

2. **Second Fault Injection:**
   - **Description:** Inject a fault into the square method found in the mathlib.h library. Changed the multiplication operator to a division operator.

   - **Original method:**

```
template<class T> T square(T x)
{
      return x * x;
}
```

   - **Method with fault injection:**

```
template<class T> T square(T x)
{
      return x / x;
}
```

   - **Test cases failed:** testCase001, testCase002, testCase013, testCase014, testCase015

   - **Test results table:**

Test Case ID	Method	Input	Expected Output	Actual Output	Pass / Fail
001	square	2	4	1	Fail
002	square	10	100	1	Fail
003	sphereArea	5	314.159	314.159	Pass
004	cube	8	512	512	Pass
005	radToDeg	0	0	0	Pass
006	sphereArea	2.05	52.8102	52.8102	Pass
007	cube	-2	-8	-8	Pass
008	circleArea	-5.0	error	78.5398	Fail
009	radToDeg	5	286.479	286.479	Pass
010	cube	876	672221376	6.72221e+08	Fail
011	circleArea	0.0	0	0	Pass
012	sphereArea	0	0	0	Pass
013	square	-98209	9645007681	1	Fail
014	square	800000	640000000000	1	Fail
015	square	0	0		Fail
016	cube	-1024	-1073741824	-1.07374e+09	Fail
017	cube	2.1111	9.4088	9.40863	Fail
018	sphereArea	100000	1.25664e+11	1.25664e+11	Pass
019	sphereArea	-300.05	error	1.13135e+06	Fail
020	circleArea	10	314.159	314.159	Pass
021	circleArea	155555555555	7.6e+22	7.60188e+22	Fail
022	radToDeg	1	57.2958	57.2958	Pass
023	radToDeg	-57	-3265.86	-3265.86	Pass
024	circleArea	500	785398	785398	Pass
025	radToDeg	125000000	7161972439.15	7.16197e+09	Fail

Figure 3: Test results after second fault injection

   - **Analysis:** Before the fault was injected test cases 001, 002, and 015 passed while test cases 013 and 014 failed. After the fault was injected all test cases which

includes 001, 002, 013, 014, and 015 failed. Test cases 013 and 014 resulted in a fail before the fault was injected and after, meaning the fault had no effect on the outcome of the test. In other words, test cases 013 and 014 failed both times. Test cases 001, 002, 013, 014, and 015 all failed after the fault was injected because the division operator was dividing the inputted variable by itself. This resulted in an actual output of 1 for test cases 001,002,013, and 014. Test case 015 resulted in a fail because the actual output could not be calculated due to division by 0.

3. **Third Fault Injection:**
    - **Description:** Inject a fault into sphereArea method found in the mathlib.h library. Changed the multiplication operator to a division operator for the first multiplication operator only.

    - **Original method:**

```
template<class T> T sphereArea(T r)
{
      return 4 * (T) PI * r * r;
}
```

    - **New method:**

```
//inject a fault into sphereArea method
template<class T> T sphereArea(T r)
{
      return 4 / (T) PI * r * r;
}
```

    - **Test cases failed:** testCase003, testCase006, testCase018, testCase019

- **Test results table:**

Test Case ID	Method	Input	Expected Output	Actual Output	Pass / Fail
001	square	2	4	4	Pass
002	square	10	100	100	Pass
003	sphereArea	5	314.159	31.831	Fail
004	cube	8	512	512	Pass
005	radToDeg	0	0	0	Pass
006	sphereArea	2.05	52.8102	5.35079	Fail
007	cube	-2	-8	-8	Pass
008	circleArea	-5.0	error	78.5398	Fail
009	radToDeg	5	286.479	286.479	Pass
010	cube	876	672221376	6.72221e+08	Fail
011	circleArea	0.0	0	0	Pass
012	sphereArea	0	0	0	Pass
013	square	-98209	9645007681	1055073089	Fail
014	square	800000	640000000000	49872896	Fail
015	square	0	0	0	Pass
016	cube	-1024	-1073741824	-1.07374e+09	Fail
017	cube	2.1111	9.4088	9.40863	Fail
018	sphereArea	100000	1.25664e+11	1.27324e+10	Fail
019	sphereArea	-300.05	error	114630	Fail
020	circleArea	10	314.159	314.159	Pass
021	circleArea	155555555555	7.6e+22	7.60188e+22	Fail
022	radToDeg	1	57.2958	57.2958	Pass
023	radToDeg	-57	-3265.86	-3265.86	Pass
024	circleArea	500	785398	785398	Pass
025	radToDeg	125000000	7161972439.15	7.16197e+09	Fail

Figure 4: Test results after third fault injection

- **Analysis:** Before the fault was injected into the sphereArea method, test cases 003, 006, 012, and 018 passed while test case 019 failed. After the fault was injected into the sphereArea method, test cases 003, 006, 018, and 019 failed while only test case 012 passed. Test cases 012 and 019 did not change the outcome of pass or fail due to the fault injection. Test case 012 resulted in a pass outcome before the fault injection and after the fault injection. Test case 019 resulted in a fail outcome before the fault injection and after the fault injection. Test case 012 passed after the fault injection because the input variable was 0. It is interesting to note that there was no empty value for the actual output for test case 012 and it did not result in a division by zero error.

4. **Fourth Fault Injection:**
   - **Description:** Inject a fault into circleArea method found in the mathlib.h library. Changed all multiplication operators to division operators.

   - **Original method:**

```
template<class T> T circleArea(T r)
{
      return (T) PI * r * r;
}
```

- **New method:**

```
//inject a fault into circleArea method
template<class T> T circleArea(T r)
{
      return (T) PI / r / r;
}
```

- **Test cases failed:** testCase008, testCase011, testCase020, testCase021, testCase024

- **After fault injection:**

Test Case ID	Method	Input	Expected Output	Actual Output	Pass / Fail
001	square	2	4	4	Pass
002	square	10	100	100	Pass
003	sphereArea	5	314.159	314.159	Pass
004	cube	8	512	512	Pass
005	radToDeg	0	0	0	Pass
006	sphereArea	2.05	52.8102	52.8102	Pass
007	cube	-2	-8	-8	Pass
008	circleArea	-5.0	error	0.125664	Fail
009	radToDeg	5	286.479	286.479	Pass
010	cube	876	672221376	6.72221e+08	Fail
011	circleArea	0.0	0	inf	Fail
012	sphereArea	0	0	0	Pass
013	square	-98209	9645007681	1055073089	Fail
014	square	800000	640000000000	49872896	Fail
015	square	0	0	0	Pass
016	cube	-1024	-1073741824	-1.07374e+09	Fail
017	cube	2.1111	9.4088	9.40863	Fail
018	sphereArea	100000	1.25664e+11	1.25664e+11	Pass
019	sphereArea	-300.05	error	1.13135e+06	Fail
020	circleArea	10	314.159	0.0314159	Fail
021	circleArea	155555555555	7.6e+22	1.29831e-22	Fail
022	radToDeg	1	57.2958	57.2958	Pass
023	radToDeg	-57	-3265.86	-3265.86	Pass
024	circleArea	500	785398	1.25664e-05	Fail
025	radToDeg	125000000	7161972439.15	7.16197e+09	Fail

Figure 5: Test results after fourth fault injection

- **Analysis:** Before the fault was injected into the circleArea method test cases 011, 020, and 024 passed while test cases 008 and 021 failed. After the fault was injected, test cases 008, 011, 020, 021, and 024. All test cases that tested the circleArea method failed after changing all the multiplication operators to division operators. Also, on testCase 011 after the fault injection, the actual output was found to be inf, shown on the table above. When searching google, this means infinity in C++ and the test case resulted in fail because 0 does not equal infinity.

5. **Fifth Fault Injection:**
   - **Description:** Inject a fault into cube method in the mathlib.h library. Changed all of the multiplication operators to subtraction operators.

   - **Original method:**

```
template<class T> T cube(T x)
{
      return x * x * x;
}
```

   - **New Method:**

```
template<class T> T cube(T x)
{
      return x - x - x;
}
```

   - **Test cases failed:** testCase004, testCase007, testCase010, testCase016, testCase017

   - **Test results table:**

Test Case ID	Method	Input	Expected Output	Actual Output	Pass / Fail
001	square	2	4	4	Pass
002	square	10	100	100	Pass
003	sphereArea	5	314.159	314.159	Pass
004	cube	8	512	-8	Fail
005	radToDeg	0	0	0	Pass
006	sphereArea	2.05	52.8102	52.8102	Pass
007	cube	-2	-8	2	Fail
008	circleArea	-5.0	error	78.5398	Fail
009	radToDeg	5	286.479	286.479	Pass
010	cube	876	672221376	-876	Fail
011	circleArea	0.0	0	0	Pass
012	sphereArea	0	0	0	Pass
013	square	-98209	9645007681	1055073089	Fail
014	square	800000	640000000000	49872896	Fail
015	square	0	0	0	Pass
016	cube	-1024	-1073741824	1024	Fail
017	cube	2.1111	9.4088	-2.1111	Fail
018	sphereArea	100000	1.25664e+11	1.25664e+11	Pass
019	sphereArea	-300.05	error	1.13135e+06	Fail
020	circleArea	10	314.159	314.159	Pass
021	circleArea	155555555555	7.6e+22	7.60188e+22	Fail
022	radToDeg	1	57.2958	57.2958	Pass
023	radToDeg	-57	-3265.86	-3265.86	Pass
024	circleArea	500	785398	785398	Pass
025	radToDeg	125000000	7161972439.15	7.16197e+09	Fail

Figure 6: Test results after fifth fault injection

- **Analysis:** Before the fault injection of the cube method, test cases 004 and 007 passed while test cases 010, 016, and 017 failed. After the fault injection, test cases 004, 007, 010, 016, and 017 all failed due to changing all of the multiplication operators to subtraction operators. When changing these operators, all of the test cases would fail because the actual output is vastly different from the expected output. For example, look at test case 004, the expected outcome was 512 but after the fault injection, the actual output was -8. These numbers are really spread apart from being the same value. Test cases 010, 016, and 017 resulted in a fail status before and after the fault injection. Test cases 004 and 007 changed from pass to fail after the fault injection.