Assignment #3

Turn in your work as **a single Word or PDF file**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Problem 1** | | | | | **Problem 2** | **Total** |
| **1.1** | **1.2** | **1.3** | **1.4** | **1.5** |
| 20 pts | 10 pts | 10 pts | 15 pts | 15 pts | 30 pts | 100 pts |

1. The following method, void isPrimeNumber(int number), reports whether a given integer is a prime number.

|  |  |
| --- | --- |
| **Node no** | **Code** |
| start  1  2  3  4  5  6  7  8  9  end | public static void isPrimeNumber(int number) {  if (number > 1) {  boolean isPrimeNumber = true;  int i = 2;  while (isPrimeNumber && i < number) {  if (number % i == 0)  isPrimeNumber = false;  else  i++;  }  if (isPrimeNumber)  System.out.println(number + " is a prime number.");  else  System.out.println(number + " is not a prime number.");  }  else {  System.out.println("Please input a number greater than 1.");  }  } |

**If you choose to use a different language, it is your responsibility to translate the code correctly.**

* 1. Use a computer-aided tool (e.g., MS Office or Visio) to draw the control flow graph of isPrimeNumber with the given node numbers. Handwritten answer will not be accepted.



* 1. Enumerate all the independent paths.

Cyclomatic Complexity = 14 Edges, 11 Nodes -> 14-11+2 = 5

So, there should be 5 independent paths

1 - (Start, 1, 9, End)

2 - (Start, 1, 2, 6, 8, End)

3 - (Start, 1, 2, 6, 7, End)

4 - (Start, 1, 2, 3, 5, 2, 6, 7, End)

5 - (Start, 1, 2, 3, 4, 2, 6, 8, End)

* 1. Create a test case (test input, no oracle value) for each independent path (only if feasible) and write a JUnit class, PrimeNumberTest.java, in the “tests” package to implement all the test cases. Here is a sample test (“13” may not be needed in your solution):

@Test

public void test3() {

PrimeNumber.isPrimeNumber(13);

}

Provide in this document the source code of PrimeNumberTest.java, a screenshot of the test execution results in the console window, and a screenshot of the code coverage for isPrimeNumber.

PATH 2 WAS INFEASIBLE AS THE WHILE LOOP MUST BE ENTERED TO CHANGE THE BOOLEAN VALUE OF IsPrimeNumber

package programs;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

class PrimeNumberTest {

@Test

// Covers path 1

public void test1() {

PrimeNumber.isPrimeNumber(1);

}

@Test

// Covers path 3

public void test2() {

PrimeNumber.isPrimeNumber(2);

}

@Test

// Covers path 4

public void test3() {

PrimeNumber.isPrimeNumber(3);

}

@Test

// Covers path 5

public void test4() {

PrimeNumber.isPrimeNumber(4);

}

}

A picture containing text, indoor, monitor, screenshot

Description automatically generated

A picture containing text, screenshot, computer

Description automatically generated

**1.4** Complete the following Definition/Use table for isPrimeNumber (D, C-Use, or P-Use).

|  |  |  |  |
| --- | --- | --- | --- |
| **Node number** | **Variable** | | |
| number | isPrimeNumber | i |
| start | D |  |  |
| 1 | P-Use |  |  |
| 2 | P-Use | D/P-Use | D/P-Use |
| 3 | P-Use |  | P-Use |
| 4 |  | D |  |
| 5 |  |  | D |
| 6 |  | P-Use |  |
| 7 | C-Use |  |  |
| 8 | C-Use |  |  |
| 9 |  |  |  |

I made isPrimeNumber and i in node 2 D/P-Use because node 2 holds more info than it maybe should. According to the nodes we were given to use in node 2 we have 2 declarations and a while loop all in one and the while loop uses isPrimeNumber and i which are both being declared in node 2 as well.

**1.5** Complete the following Definition/Use Associations table for isPrimeNumber. Add rows as needed.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Def node** | **Def-C-Uses** | **Def-P-Uses** |
| number | Start | (Start 7), (Start,8) | (Start,1,t), (Start,1,f), (Start,2,t), (Start,2,f), (Start,3,t), (Start,3,f) |
| isPrimeNumber | 2 |  | (2,2,t), (2,2,f), (2,6,t), (2,6,f) |
| isPrimeNumber | 4 |  | (4,2,t), (4,2,f), (4,6,t), (4,6,f) |
| i | 2 |  | (2,2,t), (2,2,f), (2,3,t), (2,3,f) |
| i | 5 |  | (5,2,t), (5,2,f), (5,3,t), (5,3,f) |

1. Given the following code and a test case foo(0, 0), i.e., a=0 and b=0.

1 int foo(int a, int b) {

2 int x=0, y=0;

3 if (a>0)

4 x=b-5;

5 else

6 x=b+5;

7 if (b >0) {

8 if (x>0) {

9 y=x;

10 }

11 y=y+1;

12 }

13 return y;

14 }

Use dynamic symbolic execution to generate a new test suite. Provide the concrete state, the symbolic state, and the path condition for the execution of each line of code (refer to the example in the lecture notes).

1. Execution of given test a=0, b=0

Line 1

CS = {a=0, b=0}

SS = {a=A, b=B}

PC = {}

Line 2

CS = {a=0, b=0, x=0, y=0}

SS = {a=A, b=B, x=X, y=Y}

PC = {}

Line 3

CS = {a=0, b=0, x=0, y=0}

SS = {a=A, b=B, x=X, y=Y}

PC = {A!>0} (A is not greater than 0)

New path condition: A>0

New test from constraint solving: a=1, b=0

1. Execution of the new test: a=1, b=0 from (1)

Line 1

CS = {a=1, b=0}

SS = {a=A, b=B}

PC = {}

Line 2

CS = {a=1, b=0, x=0, y=0}

SS = {a=A, b=B, x=X, y=Y}

PC = {}

Line 3

CS = {a=1, b=0, x=0, y=0}

SS = {a=A, b=B, x=X, y=Y}

PC = {A>0}

Line 4

CS = {a=1, b=0, x=-5, y=0}

SS = {a=A, b=B, x=B-5, y=Y}

PC = {A>0}

Line 7

CS = {a=1, b=0, x=-5, y=0}

SS = {a=A, b=B, x=B-5, y=Y}

PC = {A>0, B!>0} (B is not greater than 0)

New path condition: B>0

New test from constraint solving: a=1, b=1

1. Execution of the new test: a=1, b=1 from (2)

Line 1

CS = {a=1, b=1}

SS = {a=A, b=B}

PC = {}

Line 2

CS = {a=1, b=1, x=0, y=0}

SS = {a=A, b=B, x=X, y=Y}

PC = {}

Line 3

CS = {a=1, b=1, x=0, y=0}

SS = {a=A, b=B, x=X, y=Y}

PC = {A>0}

Line 4

CS = {a=1, b=1, x=-4, y=0}

SS = {a=A, b=B, x=B-5, y=Y}

PC = {A>0}

Line 7

CS = {a=1, b=1, x=-4, y=0}

SS = {a=A, b=B, x=B-5, y=Y}

PC = {A>0, B>0}

Line 8

CS = {a=1, b=1, x=-4, y=0}

SS = {a=A, b=B, x=B-5, y=Y}

PC = {A>0, B>2, B-5<0}

New path condition: B-5>0

New test from constraint solving: a=1, b=6

1. Execution of the new test: a=1, b=6 from (3)

Line 1

CS = {a=1, b=6}

SS = {a=A, b=B}

PC = {}

Line 2

CS = {a=1, b=6, x=0, y=0}

SS = {a=A, b=B, x=X, y=Y}

PC = {}

Line 3

CS = {a=1, b=6, x=0, y=0}

SS = {a=A, b=B, x=X, y=Y}

PC = {A>0}

Line 4

CS = {a=1, b=6, x=1, y=0}

SS = {a=A, b=B, x=B-5, y=Y}

PC = {A>0}

Line 7

CS = {a=1, b=6, x=1, y=0}

SS = {a=A, b=B, x=B-5, y=Y}

PC = {A>0, B>2}

Line 8

CS = {a=1, b=6, x=1, y=0}

SS = {a=A, b=B, x=B-5, y=Y}

PC = {A>0, B>2, B-5<0}

Line 9

CS = {a=1, b=6, x=1, y=1}

SS = {a=A, b=B, x=B-5, y=X}

PC = {A>0, B>2, B-5<0}

Line 13

CS = {a=1, b=6, x=1, y=1}

SS = {a=A, b=B, x=B-5, y=Y}

PC = {A>0, B>2, B-5<0}

Y is returned