Department of Computing

Curtin University

Software Engineering Testing (SET)

Week 4 Laboratory/Tutorial

The following exercises are intended to be done in a laboratory/tutorial session with a teaching assistant or instructor present. The exercises have been designed to reinforce concepts taught in SET.

Below are four faulty programs. Each includes a test case that results in failure. Answer the following questions about each program.

```
public int findLast (int[] x, int y)
{
//Effects: If x==null throw NullPointerException
//   else return the index of the last element
//   in x that equals y.
//   If no such element exists, return -1
   for (int i=x.length-1; i > 0; i--)
   {
      if (x[i] == y)
      {
         return i;
      }
   }
   return -1;
}
   // test:  x=[2, 3, 5]; y = 2
   //        Expected = 0
```

```
public static int lastZero (int[] x)
{
//Effects: if x==null throw NullPointerException
//   else return the index of the LAST 0 in x.
//   Return -1 if 0 does not occur in x

   for (int i = 0; i < x.length; i++)
   {
      if (x[i] == 0)
      {
         return i;
      }
   }
   return -1;
}
   // test:  x=[0, 1, 0]
   //        Expected = 2
```

```
public int countPositive (int[] x)
{
//Effects: If x==null throw NullPointerException
//   else return the number of
//        positive elements in x.
   int count = 0;
   for (int i=0; i < x.length; i++)
   {
      if (x[i] >= 0)
      {
         count++;
      }
   }
   return count;
}
   // test:  x=[-4, 2, 0, 2]
   //        Expected = 2
```

```
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
```

a) Identify 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.

f) Fix the fault and verify that the given test now produces the expected output.