ST 2017 Past Paper

1. Consider the following function:

```
 public int reverse_number(int num)          //Line 1
{
   int reversenum = 0;                       //Line 2
   while( num != 0 )                         //Line 3
      {
          reversenum = reversenum * 10;      //Line 4
          reversenum = reversenum + num%10;  //Line 5
          num = num/10;                      //Line 6
      }
   return reversenum;                        //Line 7
}
```

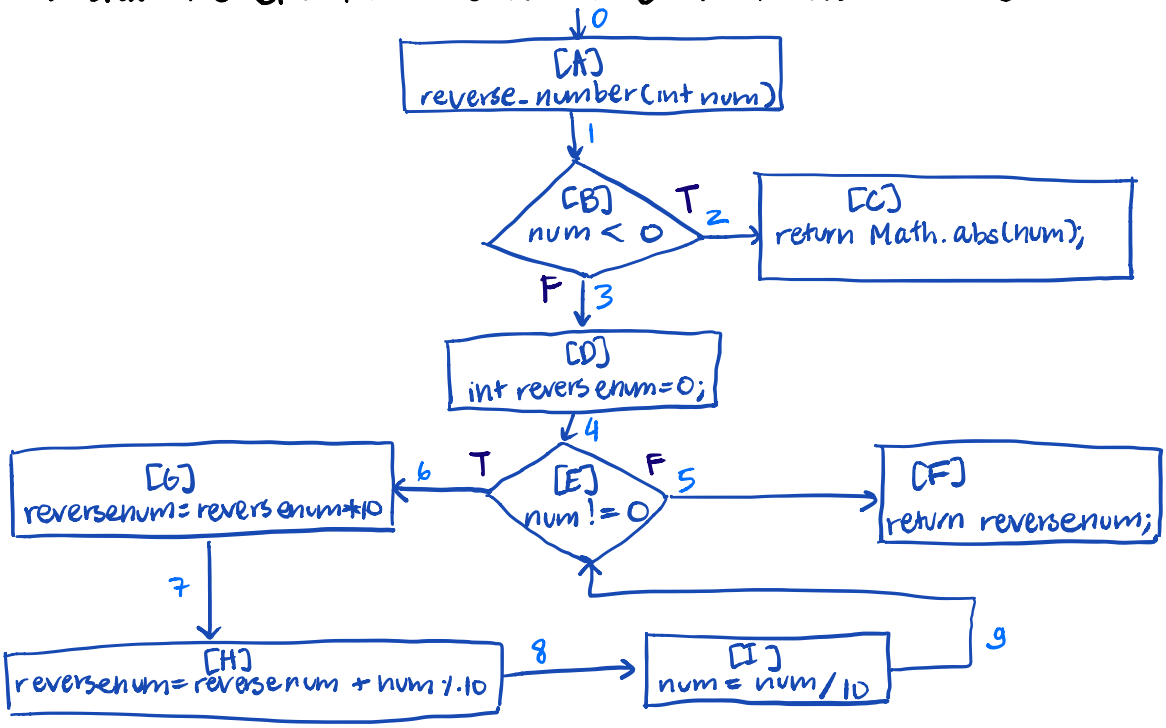
The function takes in the decimal representation of an input number and returns the reversed number if the input number is positive.
e.g. 123 → 321
If the input number is negative, the function will return the absolute value of the input number (without reversing). e.g. -41 → 41

a. Identify the fault and suggest a correct solution

The function does not check if the num is negative and it does not return the absolute value of the number.

CORRECT SOLUTION
Between lines 1 and 2 →
```
if (num<0) {
  return Math.abs(num) // or return (-1*num)
}
```

b. Draw the CFG for the corrected version of reverse_number.



c. Write tests with input, expected output, and sequence of edges traversed.

Test	Input	Expected output	Edges Traversed
a	-1	1	0, 1, 2
b	1	1	0, 1, 3, 4, 6, 7, 8, 9, 5
c	21	12	0, 1, 3, 4, 6, 7, 8, 9, 6, 7, 8, 9, 5
d	40	4	0, 1, 3, 4, 6, 7, 8, 9, 6, 7, 8, 9, 5
e	0	0	0, 1, 3, 4, 5

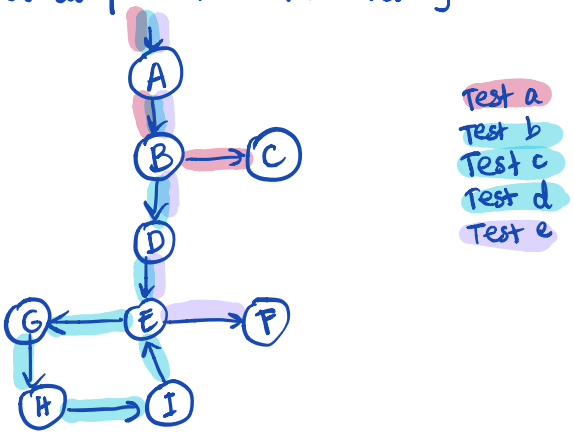
d. Evaluate coverage.

edges = 9/9 = 100%. → No additional tests required.

e. Name at least 2 coverage criteria that are practical for testing paths through loops. For one of these criteria, compute coverage achieved using tests developed in (c) and (d). If full coverage is not achieved, develop tests to achieve full coverage.

Loop boundary adequacy, boundary interior path testing.

Boundary Interior Path Testing



Loop boundary adequacy criteria
- In at least one test case the loop body is covered zero times
  → Test a, test e
- In at least one test case the loop body is covered exactly once
  → test b
- In at least one test case the loop body is covered more than once
  → test c, test d

f. <D,U> pairs for faulty reverse-number.
Write tests with input and expected output that achieves 'all DU pairs' cov.

Variables	Definitions	Uses	<D,U> pairs
num	1, 6	3, 5, 6	<1,3>, <1,6>, <6,6>, <6,3>, <6,5>
reversenum	2, 4, 5	4, 5, 7	<2,4>, <4,5>, <5,7>, <2,7>, <5,4>

Test	Input	Expected output	<D,U> pairs covered
a	0	0	<1,3>, <2,7> <1,6> <5,4>
b	21	12	<1,3>, <2,4>, <4,5>, <6,6>, <6,3>, <6,5>, <5,7>

The test suite covers all the <D,U> pairs → 100% coverage.

2. Function area(int n) calculates area of a square with side n.

area(int n) { n >= 0 returns area
n < 0 returns -1
returns -1 if the result is out of range

a. Assume area is the ITF: "correctly computes and returns the area of a square with side provided by parameter n, or returns an appropriate error code". What are the characteristics of the input and the environment?

Input: n; n can be positive or negative.
Environment: size of integer; area of the square can be within the integer size or not.

b. Define partitions/value classes for the input and environment based on the characteristics.

Characteristic	Partition	Value classes
Positive or negative n	P1	n >= 0
	P2	n < 0
Area within int size/not	P3	Area within integer size
	P4	Area not within integer size

c. Generate test case specifications for the different value classes, giving expected result.

Test	n	Area inside/outside int range	Expected
1	positive	inside	correct area
2	positive	outside	-1
3	negative	inside	0
4	negative	outside	0

d. Give concrete test examples.

Test	n	Integer size	Expected result
1	-10	2 bytes	0
2	0	2 bytes	0
3	10	2 bytes	100
4	-182	2 bytes	0
5	182	2 bytes	-1
6	-10	4 bytes	0
7	0	4 bytes	0
8	10	4 bytes	100
9	182	4 bytes	182 x 182
10	46341	4 bytes	-1

3. a. Explain the steps in the TDD. What are the benefits of using TDD?

TDD is a software development process that relies on the repetition of the following cycle:

1. Developer writes a failed test that is based on an improvement or a new feature of the system.
2. Developer produces the minimum amount of code that pass the test.
3. Developer refactors both the test and the code.

Benefits - shortens the programming feedback loop
- promotes development of high quality code
- user requirements are more easily understood

(b) For the following specification, illustrate steps involved in TDD. As part of the steps, the code that you write can be in the form of pseudo-code or a programming language of your choice. For the purpose of the exam, do not worry about the syntactical correctness of the code. You only need to focus on program logic and its correspondence to tests developed.

```
Take two integer numbers as inputs,
Spec 1: If both input numbers are positive,
          compute and return the sum.
Spec 2: If both input numbers are negative, return zero.
Spec 3: If one input is positive and the other is negative,
          return positive number.
Spec 4: If both input numbers are zero, return -1.
```

Test 1: Returns correct sum of 2 input numbers.
Input: (2,3) Expected output: 5

Code stage 1:
```
def sum_two(n1, n2):
    return n1 + n2
```

Test 2: If both numbers are negative, the function should return 0.
Input: (-4, -5) Expected output: 0

Code stage 2:
```
def sum_two(n1, n2):
    if n1<0 and n2<0:
        return 0
    return n1 + n2
```

Test 3: If one number is positive and the other one is negative, return the positive number.
Input: (6, -7) Expected output: 6

Code stage 3:
```
def sum_two (n1, n2):
    if n1<0 and n2<0:
        return 0
    elif n1<0 and n2>=0:
        return n2
    elif n1>=0 and n2<0:
        return n1
    return n1+n2
```

Test 4: If both numbers are 0, return -1.
Input: (0,0) Expected output: -1

Code stage 4:
```
def sum_two (n1, n2):
    if n1==0 and n2==0:
        return -1
    elif n1<0 and n2<0:
        return 0
    elif n1<0 and n2>=0:
        return n2
    elif n1>=0 and n2<0:
        return n1
    return n1+n2
```

c. "A fault-free system is failure-free but a failure-free system need not be fault-free." Explain & assess if true or false.

FAULT: A static defect in software.

FAILURE: External, incorrect behaviour wrt. the requirements or other description of the expected behaviour

* Faults cause errors that cause failures.
* this is a repeat question → check 2014 paper!

d. Compute reaching equations for line 7. Iterate once and show the Reach Equations for this eqn. How are Reach Eqns diff. from Avail?

```
public int check_prime(int a)
{
   int c;                              // Line 1
   if ( a ==0 || a ==1)                // Line 2
      return 0;                        // Line 3
   for ( c = 2 ;                       // Line 4
         c <= a - 1 ;                  // Line 5
         c++ ) {                       // Line 6
   {
      if ( a%c == 0 )                  // Line 7
          return 0;                    // Line 8
   }
   if ( (c == a) && (a%10 == 7) )      // Line 9
      return 1;                        // Line 10
   return 0;                           // Line 11
}
```

e. Arrays and pointers introduce uncertainty into dataflow analysis. Explain why.

E.g. a[i] is the same as a[k] if i=k.
a[i] is the same as b[k] if a=b.

Potential definition or use of a whole set of variables/locations could be aliases of each other.

This uncertainty is accommodated depending on the kind of analysis.