ST 2014 Past Paper

1. check-delivery function.

> check-delivery <post-time> <delivery-time> <service>

- <post-time>: date and time item posted
- <delivery-time>: date and time item arrived
- <service>: Type of service: normal (if before noon, arr. next day)
  express (arrive before noon next day)

The command returns 0 if the posting and delivery meets the service specification. Otherwise, it returns the no. of minutes late.

a. Use category-partition to create a test suite.

PARAMETERS		
Category	Partition	Properties
post-time: feasibility	posted before delivery posted after delivery	Property Feasible Property Infeasible
post-time: period	before noon after noon any time	Property Bnoon if CN & Feasible) Property Anoon if CN & Feasible) if X and Feasible
delivery-time: feasibility	posted before delivery posted after delivery	if Feasible if Infeasible
delivery-time: period	before noon day after after noon day after day after two days after more than two days later ...	if X and Feasible if X and Feasible if N and Bnoon and Feasible if N and Anoon and Feasible if N and Anoon and Feasible
service	normal express	property N property X
ENVIRONMENTS		
Category	Partition	Properties
None	None	None

Process taken to arrive at table
- analyse specification
- partition the categories into choices

b. "Constructing a CP test suite provides a good method for identifying omissions in the specification of the system under test."
Assess the ⊕ and ⊖ of this statement.

- CP is driven from the specification so it should be useful.

- CP tries to cover all possible different behaviours so it may be possible to identify some partitions for which the output of the function is not specified.

- The partitions are hand generated so they are error-prone.

- CP assumes parsimony in implementation - often programmers are not parsimonious.

- CP tends to generate a large number of partitions and we choose to omit some from consideration, this may conceal omitted cases

c. How to check the frequency of failure for check-delivery? What information would you require?

- The organisation is interested in testing the reliability of the function so we anticipate they are looking for reliability testing.

- Reliability testing needs an operational profile so we would need to collect data on calls of the function to understand the distribution of calls.

- We probably need to tighten up the specification so that the behaviour is fully specified for various error conditions.
  → e.g. delivery before posting

- We will need to run a large number of tests with 'same' distribution as the operational profiles so we need a way of generating that.

- We probably need some sort of oracle to tell us if the answer is correct.

d. "For every failure, if we find the fault and fix it then eventually the code will be fault free." Explain, assess if TRUE or FALSE.

FAULT - a static defect in software (cause)
→ white-box notion

FAILURE - external, incorrect behaviour wrt the requirements or other description of the expected behaviour.
→ black-box notion

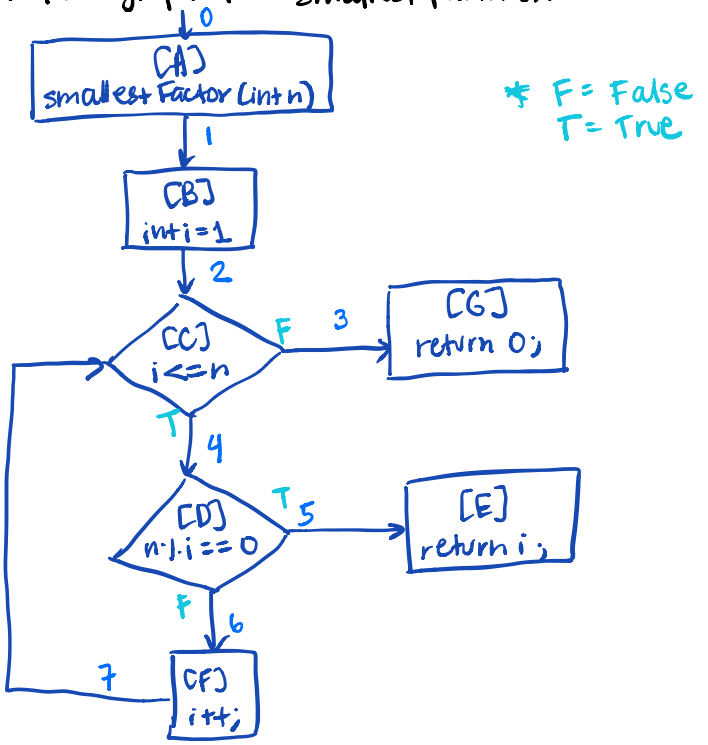
* not all faults will manifest as a failure and some may take a very long time to manifest

* we may not observe all the possible failures eventually give rise to a failure we may not uncover all faults.

2. Consider the following method:

```
1   public int smallestFactor(int n){ // n > 1
2       int i=1;
3       while (i<=n){
4           if (n%i == 0){ // recall % is mod
5               return i;
6           }
7           i++;
8       }
9       return 0;
10  }
```

a. Draw control flow graph for smallestFactor().



* F = False
T = True

b. Consider the following test suite:

Test	Input	Expected result	Sequence of edges
a	4	2	0, 1, 2, 4, 5
b	15	3	0, 1, 2, 4, 6, 7, 4, 5
c	25	5	0, 1, 2, 4, 6, 7, 4, 6, 7, 4, 6, 7, 4, 5

i. There is a bug in the code. Find bug and correct it. Explain.

Block B should be changed to int i=2, because 1 is always a factor for ALL numbers. the fact that all our tests are non-primes.
Example: input=7 Expected=7 Actual=1

ii. Sequence of edges for tests.

Beside table.

iii. Evaluate coverage of edges and blocks.

edges = b/t blocks = b/t

iv. Identify code not covered by test suite.

Edge 3 is not covered. 2 0

New test suite → input = 0 (i<=n will be false)

c. change code on lines 3-9 into the following:

```
1   while (n%.1 != 0 && i<=n) {
2         i++;
3   }
4   return i;
```

What measure of adequacy would provide a stringent level of test on the condition on the while loop? Is it possible to construct a test suite that provides that level of adequacy?

Basic condition coverage - each basic condition must be executed at least once.

We will not be able to achieve this because it is impossible to reach the situation where the first condition is true and the second one is false.

c. How many def-free paths are there for variable i beginning at line 2? Diswss the adequacy measure used.

There is an infinite collection of paths.

We need to cut this down → choosing paths with different nodesets.

3. You have been hired as the software test manager in a company that is developing a highly novel powertrain for vehicles. It comprises a normal engine plus a large battery and other ancilliary equipment. It works as follows:

   - when the driver presses the accelerator: energy is transferred to the wheels in proportion to how much the accelerator is pressed where energy is supplied from the engine only to "top up" the battery if it cannot supply energy at the required rate.
   - when the driver presses the brake: braking is supplied in proportion to how far the brake pedal is pressed, braking is supplied by using a generator that slows the car by charging the battery, additional braking is supplied by conventional braking only if the generator cannot supply enough braking.

   The company is attempting to provide proof of concept by using the powertrain in motor racing. Races last at most two hours, cars need to be closely matched to the capacities and expectations of the drivers and it is important that the powertrain does not have unanticipated behaviour.

a. Write down three key areas of test activity.
broad areas that are important in ensuring the software is fit for purpose.

i. Provide justification that this area is important for deciding whether or not the software is fit for purpose.

ii. Outline the types of testing that are relevant and how they contribute to assuring requirements in the chosen area?

SAFETY - capacity to deliver the specification and not to deliver any other functionality.
- we might want to consider testing for resilience to failure and operation in degraded modes to respond to sensor or actuator failure.
- some kind of black-box testing is probably most appropriate since that focuses on the behaviour.
- perhaps model-based testing might help in looking for correct behaviour.
- in very high critical code some kind of verification may be appropriate.

RELIABILITY - estimate of time we can expect to operate without encountering a failure.
- time-limited issue; we want to insure the MTTF is higher than two hours with reasonably high probability.
- using some sort of simulation / possibly collected data from test running or data collected from older generations of powertrain would be appropriate.

USABILITY - system responds in the way the driver expects.
- the hybridity of the power system is not evident in variability of response or uncharacteristic and unpredictable behaviour from a driver perspective → may involve some sort of simulator with drivers being asked to experience the simulated behaviour.

b. Compare V-model with XP for this software.

- hybrid approach → use XP at the start of the process where it is important to try to uncover the required functionality
  → then switch to a V-model later when the requirements are decided on

- the novelty of the system favours XP while the safety aspect favours the V model.

c. Regression testing pros and cons for this software.

- Regression testing depends on the stability of the tests, early in the process this is not the case.

- For some of the sorts of tests we are looking at the cost of doing regression testing could be high (getting drivers into a simulator) so it may not be useful to regression test in this context.

- Focusing regression testing on reasonably stable safety properties may be a useful thing to do.

- Phasing the introduction of regression testing to include tests as they become stable may be a useful approach.

d. Suitable measure of quality of code for the powertrain system.

- code quality may be captured by some of the test results e.g. the reliability estimates generated by testing.

- in terms of defects, some measure of the quality of the test set (using mutation as a test adequacy measure) might be useful if the system passes all the tests.

- other approaches might also be useful such as code inspections, compliance to coding standards, etc.

- proofs of some properties of the code may also be useful but we would need to be aware of the assumptions used in proofs.