ST 2013 Past Paper

1. The filter command takes a pattern and a list of files and searches through the lines of the file copying those that match the pattern to the output. We can think about this as a linux command and write it like this:

> filter <pat> <files>

pat is a sequence of characters with wildcard * symbol which matches any sequence of characters. To include a * symbol in the pattern the sequence * is used, to include a \ the sequence \\ is used.
files is a list of files to be filtered.

The operation of the command is to scan through the lines of the files in the list copying those lines that match the pattern to the output and ignoring the others. Thus the output is the sequence of all the lines that match the pattern in their order of appearance in the list of files.

a. Use a category-partition method to construct a test suite from the specification for filter. You should:
   i. clearly identify each stage in the process
   ii. provide a general description of each stage you identify
   iii. Illustrate your description by constructing a test suite for the filter command and using the category-partition method.
   iv. The given specification for filter may be incomplete/ambiguous. State any assumptions you make in resolving deficiencies in the specification

PARAMETERS

Category	Partition	Properties
pat: length	empty single char many chars very long	
pat: wildcard	no wildcard one wildcard many wildcards	
pat: escape characters	no escaped characters escaped characters escaped wildcard	
filename list	empty one file long more than one file long	property Empty property Single

ENVIRONMENTS		
Category	Partition	Properties
Pattern occurences	Zero occurences	
	occ. in at most one file	if not Empty
	occ in many files	if not Empty or Single

Process taken to create the table
- analyse the specification
- partition the categories into choices
- determine constraints among the choices
- an explanation of how test specifications are generated
  e.g. identify the individually testable features (ITF) and test every possible value for the ITFs.
- generate tests that match the specification

One anomaly from the specification is the omission of what the function should do when a filename does not refer to a file in the environment.
→ e.g. exception: file not found!

b. Consider the following statement: "A category-partition test suite provides good evidence that the specification has been implemented correctly." Write brief notes assessing the strengths and weaknesses of this statement.

- category partition is driven from the specification so it should be useful
- category partition tries to cover all possible different behaviours
- The partitions are hand-generated so they are error-prone
- category partition assumes parsimony in implementation - often programmers are not parsimonious.
- category partition can generate many combinations of partitions and some are omitted.

c. Suppose a user of the filter command had extensive records of the use of the command in operation and offered them to you to help with the test process for a new implementation of the command. How would you make use of such information? What aspects of the filter command could be tested more adequately using such information? Write a short note pointing out the potential uses of such data and what additional confidence it might give you in the filter command. You should also point out characteristics of the filter command whose testing is unlikely to be affected by the availability of such data.

- Reliability testing is the most obvious use - looking for failures will help characterise failure rate.

- using real-world test cases to meet the test case specification developed by category partition.
- This dataset might also be useful in terms of stress / performance testing.
- This is unlikely to help with safety testing (i.e. demonstration of the absence of errors even in quite unusual situations) since any operational dataset is likely to contain unusual situations.

d. "A fault-free system is failure-free but a failure-free system need not to be fault-free." Using the technical sense of fault and failure, explain this statement and give as an assessment of whether it is true or false.

A fault is a static defect in the system, which may or may not cause failure. (white-box notion) → fault: you know the cause

A failure is an external, incorrect behaviour wrt the requirements or other description of the expected behaviour. (black-box notion) → something is wrong but you don't know the cause.

No cause of failure → no failures, but no failure ↛ no faults, as some faults might be benign (e.g. typos).

Quite different measures are likely to be used to establish freedom from faults or failures.

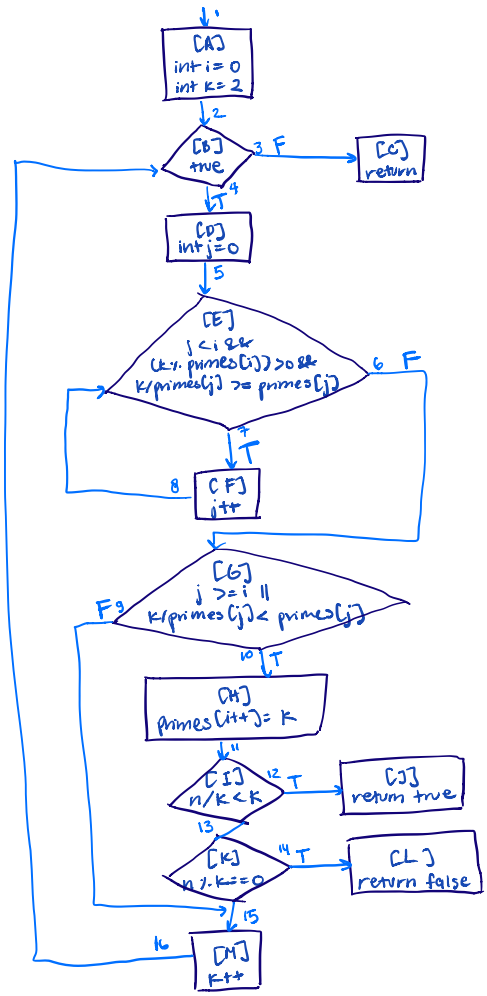
Fault freedom is always relative to a specification so it can be quite difficult to determine if a system is fault-free. this also depends on the development process because, say in Test Driven Design, the code is nearly always fault-free or incomplete but the specification evolves.

Failure freedom is behavioural notion so a system can carry many faults that are never expressed because the system never uses some parts of the functionality. The assumption that failure freedom is reasonably stable notion is often misleading because small changes in the specification can uncover latent faults.

2. Consider the Java method below. The method tests whether a number is prime.

```
1  public boolean isPrime(int n){ /* n >=0  */
2    int [] primes = new int[1000];
3    int i = 0; /* number of primes in array */
4    int k = 2; /* Candidate next prime */
5    while (true){
6    /* Check if k is divisible by any prime in primes */
7    /* Recall,  % is the modulus operator, so 7%3 ==1 */
8    int j = 0;
9    while (j < i && (k%primes[j])>0 && k/primes[j] >= primes[j]){
10     j++;
11     }
12   if (j >= i || k/primes[j] < primes[j]){
13      primes[i++] = k; /* k is the next prime - add it to primes */
14     /* return true if n is not divisible by any prime p < sqrt(n) */
15     /* return false if we have found a prime that divides n */
16     if (n/k < k) {return(true);}
17     if (n%k == 0) {return(false);}
18     }
19   k++;
20   }
21 }
```

a. Draw the control flow graph diagram for isPrime(). Label each of the edges in the flow graph with a number and each blocks with a letter.



b. Consider the following test suite

Test	Input	Expected result
a	1	false
b	2	true
c	3	true

i. Using the labels on the edges in your flowgraph write down the sequence of edges traversed by each of the tests in the test suite. If you find bugs in the code while doing this please describe them in your answer.

Test a: 1 2 4 5 6 10 11 12 → fails
Test b: 1 2 4 5 6 10 11 12
Test c: 1 2 4 5 6 10 11 12

ii. Use this data to evaluate the fraction of (COVERED/TOTAL) of edges and blocks covered by the test suite.

We traversed 8 edges so $^{8}/_{16}$ → coverage is 50%.
We visited 8 blocks so $^{8}/_{13}$ → block coverage is 61.5%.

iii. Identify two blocks or two edges that are not covered by this test suite and create new tests that cover your chosen blocks or edges.

* Any test of a non-prime greater than 2 will test the false return. (edge 14, block L)

* Any test of a number bigger than 4 will test the main loop (edge 7, block F)

c. Consider the condition on line 9 of the method. What measure of adequacy would provide a stringent level of test condition? Construct a test suite that provides that level of adequacy.

Basic condition coverage
Test set: 2, 4 (any non-prime > 2), 5 (any prime > 4)

d. Identify two def-clear paths for the variable j defined on line 8. Are both paths covered by the test suite in Q(2b)? If not, suggest tests that, added to the above would cover both paths.

5 and 5,7 are both def-clear (j is changed after edge 7). Edge 7 is not included in the test suite above so we would need to include a number bigger than 4 in the test set.

e. The code contains at least one more error which causes it to fail. Can you identify the error and provide a description of a test that will uncover it. You do not need to give a specific number to use in the test - a description of the number is both adequate and easier to specify than writing the number down explicitly.

? The error is caused by a table overflow — to achieve this we would need to supply a number with more than 1000 prime factors that are smaller than its square root.

3. You have been hired to work as the test manager in a company that develops engine control software for cars. Your job is to develop a testing strategy to ensure that the testing process generates sufficient evidence to justify using the software in consumer products, i.e. cars. The software is both safety-critical and highly business critical because consumers will not purchase cars they think are dangerous. You are just at the start of planning the strategy and you are trying to identify key issues and sections in the strategy.

a. Write down three key areas of test activity. These should be quite broad areas that are important to making the decision that the software could be deployed. For each:

i. Provide a justification that this area is of importance in deciding whether the software is fit for purpose.
ii. Outline the types of testing that are relevant and how they contribute to assuming requirements in the chosen area.

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

reliability - here we want to provide some estimate of time we can expect to operate without encountering a failure.
- this should be an extended series of tests in different modes gathering the response of the system.
- this should probably be a black box test driven by a simulator that takes into account the operational profile and perhaps an oracle to test the response of the system is appropriate.

performance - here we want to gain confidence that the system will respond to inputs within given timescales.
- we will be interested in worst case execution times for most combinations of inputs.
- this will include elements of stress testing since we are interested in how the system responds to multiple stimuli appearing over a short period of time.

b. On the basis of your previous answers suggest a suitable lifecycle for developing and maintaining your engine control software and illustrate how this ensures adequate testing of the system.

There needs to be substantial testing of components as the system is developed that are not necessarily directed at fault finding so a more 'classical' approach is probably the best in this case.

e.g. V-model or the spiral model
→ there is sufficient structure to support this kind of test activity.

c. Your manager wants you to adopt whose goal is to create 'fault-free' software. Write short notes outlining the arguments in favour and against this approach.

IN FAVOUR - there is only a tenuous connection between faults and failures and faults can be tolerated and managed in a variety of ways.

AGAINST - faults can only be measured against some specification for the component containing the fault. These are often incomplete and evolving during development so the notion of fault need not be particularly stable. This is particularly true when we consider how to resolve problems arising from interactions between components.
- chasing and killing bugs can be very time consuming and often they have no bearing on the final operation of the code. Excessive emphasis on bug killing can distract from getting correct operation overall.

d. Mutation Testing might help provide an improved estimate of the residual defect density in code. Explain strengths and weaknesses of this approach in a safety critical environment.

⊕ we'd expect the process in a safety critical design environment to come up with good RDD estimates.
⊖ the mutation approach has very low level model of errors - basically those that arise from slips. This is not a typical defect we are interested in high quality code.
⊖ we would expect that the test suites to be of very high quality and consequently to kill most mutants which is good but also having a very low RDD means the improvement we gain from using Mutation Testing to improve estimate is quite small.