DD2459 Software Reliability 2020
Take-home Exam
Wednesday 4 March, 15.00 – Friday 6 March, 12.00

Instructions.
Please read the following instructions carefully.

(1) Clearly mark at the top of each sheet you use: (a) your name, (b) the page number.

(2) On your front page indicate: (a) how many pages are contained in your work in total, (b) your name (c) your personal or KTH number, (d) your e-mail address (in case I need to contact you)

(3) There are two ways to submit your work. (1) Your work may be handed in personally to EECS studentexpedition, E building, level 4, Lindstedsvägen 3, no later than Friday 6 March 2020 at 12.00 midday. After this time it will be marked as late, and marks will be subtracted. (2) If you are unable to reach the studentexpedition yourself (for example if you are at work) you may post your manuscript to: Karl Meinke, EECS School, KTH Kungliga Tekniska Högskolan, 100 44 Stockholm, Sweden. The date on the postmark will be taken as the date of your submission. The deadline of March 6, 2020 applies to all manuscripts submitted by post.

(4) If manuscripts are submitted in any other place or by any other means than those described in Part (3) then the examiner and EECS School cannot be held responsible in the case that manuscripts are lost. In the case of postal submission it is strongly recommend that you keep a digital copy or photocopy in case of postal loss. KTH cannot be held responsible for loss in any national postal system.

(5) If you have any questions about the exam (for example, if you do not understand a question) you may call the examiner on 08 790 6337. Please do not call before 10.00 am or after 5.00pm! You can also e-mail the examiner at karlm@kth.se. I will publish any typographical errors and corrections that become known during the exam on the course web page and notify you with canvas.

(6) You may use your course notes, books and the internet. However, all material you submit must be your own. (a) You are not allowed to discuss your answers with anyone else until you have submitted your manuscript and the exam is finished. (b) You are not allowed to copy anyone else's work. (c) *By handing in your manuscript you are declaring that you have read and abide by all the rules on this cover page*. (d) In the case that cheating is suspected, disciplinary action will be taken.

(7) Write clearly. No marks will be awarded for work that I cannot read. You can write your answers in Swedish or in English. You can submit a hand-written manuscript or a typed manuscript or any combination of both.

Please turn over.

NOTE: For a full score of 65 points you should answer all 3 questions below.

Question 1. **(Total 20 points)**
Write appropriate pre- and post-conditions for the following mathematical operations using the JML specification language (i.e. write *requires-ensures* conditions).
You should try to avoid under-specification, i.e. write all the constraints necessary for each condition and not omit any relevant constraints.
If you are unsure how any operation/method is defined you may look it up online or in a book.
You do not have to provide Java code for any of the operations, only JML.

(i) **(5 points)** A *substring detection method:*

```
boolean mySubstring(char[] substr, char[] superstr)
```

Input: Two one-dimensional arrays `substr, superstr` of `char`.
Output: a Boolean value which is true if, and only if, `substr` matches some substring of `superstr`.

(ii) **(5 points)** A *suffix detection method*:

```
boolean mySuffix(char[] mainstr, char[] suffixstr)
```

Input: Two one-dimensional arrays `mainstr, suffixstr` of `char`.
Output: a Boolean value which is true if, and only if, `mainstr` has `suffixstr` as one of its suffixes.

(iii) **(5 points)** Recall that a set of X can be implemented as an array of X without repetition of elements. Give pre- and postconditions for an *element addition method:*

```
char[] myElemAdd(char[] myCharSet, char myChar)
```

Input: A one-dimensional array `myCharSet` of `char` implementing sets as arrays without repetition and a `char` value `myChar`.

Output: A one-dimensional array of `char` implementing the set that results from adding `myChar` to `myCharSet`.

(iv) **(3 points)** Show how you could use your JML specification of the element addition method in part (iii) to construct a *black-box test script* that implements both *pairwise test case generation* and *verdict construction* (the test oracle), to call and test a Java method

```
char[] myElemAdd(char[] myCharSet, char myChar)
```

Important Note: Your answer should only implement the pairwise testing and JML requirements of part (iii). You do not have to write any code for the `myElemAdd()` method itself.

(v) **(2 points)** What is the best way to view the *dimensionality* of the system under test (SUT) in the testing problem of part (iv) above? Motivate your answer. Give a precise expression for the dimension value *n* in terms of the SUT API features given in part (iv).

Question 2. **(Total 15 points)** Give *propositional linear temporal logic* (PLTL) formulas that capture the following informal statements about temporal events. (**Hint**: it may help to identify the basic propositional variables first.)

(a) (**2 points)** *If you always save money then sooner or later you'll be rich.*

(b) (**2 points**) *Applying the brake always brings the car to a stop in two or three time units, and the car stays stopped until the brake is released*.

(c) **(2 points)** *A lamp is always in one of two distinct states – green or red. Initially it is red. Whenever you press the button two times in a row then the lamp changes color.*

(d) **(4 points)** Write two different test cases to test two different aspects of the functional requirement given by your answer to (c). Are there any aspects of (c) which cannot be tested? Motivate your answer.

(e) (**5 points**) Draw a state machine that models the usual sequence of the **seven days of the week**. Formalise in PLTL using *future time operators* only (such as G, F, X and U) the false statement

"*If tomorrow is Sunday then the day before yesterday was Wednesday*"

Use your state machine to construct a counterexample to your false PLTL temporal formula.

Question 3. **(Total 30 points)** Consider the following seven lines of pseudo-code:

```
1. x' = x;
2. y' = y;
2. while (x' != 0 & y' != 0) do {
3.    if (x' < 0) then x' = x' + 1 else x' = x' - 1;
4.    if (y' < 0) then y' = y' + 1 else y' = y' - 1;
6. }
7. if (x' == 0) then return y else return x;
```

You can assume that `x:integer` and `y:integer` are the only integer input variables to the above pseudo-code. The operator `&` is the normal Boolean conjunction operator and is not lazy.

a) (**2 points**) Draw a condensation graph for the above pseudo-code. Clearly number all the nodes in your graph.

b) (**3 points**) Write out the full set ER of all possible *edge requirements* for your condensation graph. How many edge requirements are there in ER?

c) (**3 points**) Eliminate *redundancy* in ER by writing out the smallest subset RER of *reduced edge requirements* such that every requirement of ER is covered by a requirement of RER. Motivate your answer. How many requirements are there in RER?

d) (**2 points**) Explain in detail why RER is a better requirement set for testing than ER.

e) (**3 points**) Use your answer to RER from (d) to write a minimized set TC of test cases for your condensation graph that give 100% edge coverage.

f) (**3 points**) Write out the set of E^2R all possible *edge-pair requirements* for your condensation graph. How many test requirements are there in E^2R?

g) (**6 points**) Eliminate *redundancy* in E^2R by writing out the smallest subset RE^2R of *reduced edge-pair requirements* such that every test requirement of E^2R is covered by a test requirement of RE^2R. Motivate your answer. How many requirements are there in RE^2R ?

h) (**3 points**) Explain in detail by means of requirements and/or test cases why RE^2R is a *more effective* set of test requirements than RER.

i) (**2 points**) Define the *mathematical function*

$$f(x,y) : \mathbf{Z} * \mathbf{Z} \rightarrow \mathbf{Z}$$

that describes the *black-box behavior* of the pseudo-code above in terms of its two integer input variables and one integer output variable. (Here **Z** denotes the set of all integers.)

(j) (**3 points**) For each of your test cases in TC in your answer to (e) above, extend the test case by giving a *predicted output* using your answer to (i) above.