SFWRENG 3S03 Software Testing
Assignment 3 – due Monday 11 April 2022 by 1159pm (submit on Avenue)
}

1. [10 marks] Suppose you are building a first-person shooter style game, in an object-oriented style, and as part of this you implement a class *Sprite*, which represents any object that can appear in the playfield of the game (e.g., a player character, a block, a cactus). A Sprite object provides fields and methods for keeping track of the location of the sprite on the playfield, records the bitmap associated with the Sprite, the position of the sprite, and provides additional methods for moving the sprite and detecting whether the sprite overlaps with another sprite (i.e., collision detection).

Using a table, specify a *test plan* for the Sprite class. Your test plan should state the following:

- Objectives for the class
- Any requirements for inspection/review of the class.
- Brief statement of objectives for specification-based testing for the class.
- Brief statement of objectives for implementation-based testing for the class.
- Brief statement of objectives for interaction-based testing (i.e., interactions between particular methods of the class)

As a guideline, the test plan should be no more than a page long, and each part should be a short paragraph of text. Remember that you are focusing on objectives, not detailed descriptions of *what* your tests will look like!

2. [25 marks] Based on what was discussed in class, in terms of contracts and contract-based specification, write preconditions and postconditions for each method of a class *Linked_Set<G>,* which represents a generic Set data structure implemented using a linked list. In particular, specify pre and postconditions for each of the following methods. Also, write a class invariant: it specifies (as a boolean expression) properties that must be true for *all* instances of class Linked_Set. Hint: think about constraints on the attributes of Linked_Set and relationships between these attributes. Hint: some of the preconditions and postconditions are *very* simple, others are more complex! You may make any reasonable assumptions about the linked list data structure and how it works (though you will find some hints embedded in the method descriptions below, in particular the linked_set must contain a *cursor* which can be moved to point at an object in the underlying linked list data structure).

- linked_set: the constructor for linked set objects, which creates an empty set
- first: returns the item of type G at the first position
- at(int i): returns the item of type G at the i-th position
- has(G v): returns true if v is contained in the linked set
- item() : returns the current item of type G
- count: returns the number of elements in the linked set
- is_equal(Linked_Set<G> t): returns true if the argument t contains the same elements as the current object

- move_item(G v): move v to the left of the cursor used in the linked set

3. [15 marks] Suppose you are building a fuel tracker web application, which enables a fuel distributor to determine whether a truck of fuel to be received at the depot will fit in a single track. Low volatility fuels can fill a tank completely. High volatility fuels (like propane) require an expansion space to be left for safety reasons (e.g., a propane tank is filled to no more than 80%). The tank capacity is 1200 litres without the expansion space, 800 with it. All loads are considered to the nearest litre. Assume that the web application includes a number of data entry screens with appropriate UI controls.

The application supports the following:

- Check whether a high or low volatility load fits in a tank
- Check whether a high or low volatility load does not fit in a tank
- List tank capacities
- Exit when done
- Identify a user input data error

Write 7 tests for this application. Your tests may be written out in English, pseudocode, or jUnit. Each test case must specify the input to the test case and the expected outputs.