# SYSC 4101

# Laboratory 3

## What to hand-in:

A Word (or PDF) document with your answers to the questions.
**Do not forget to provide your name and student ID in that document.**

## Exercise 1 (4+4+4 = 12 pts)

In a health-care application, an *age* input is processed to identify different kinds of coverage a person is entitled to. Up to 16 years of age (value 16 included), coverage service X is available, then, up to 25 (value 25 included), coverage Y is available, then up to 55 years of age (value 55 excluded), coverage Z is available. Then, below (strictly) age 65, coverage service T is available, while coverage service U is available to only someone 65 years of age, and above (strictly) 65 years of age, coverage service S is available.

**Question 1:** Design equivalence classes for the *age* input (without specifically focusing on equivalence class boundaries).
How many test inputs values would this lead to?
<u>Succinctly</u> justify your answer.

**Question 2:** Add boundary values to your equivalence classes for the *age* input.
How many test input values would this lead to?
<u>Succinctly</u> justify your answer.

**Question 3:** Add values around boundaries to your answer to Question 2.
How many test input values would this lead to?
<u>Succinctly</u> justify your answer.

## Exercise 2 (2+2+2 = 6 pts)

Suppose the social security function discussed previously in Exercise 1 has a Graphical User Interface front-end that ensures the *age* input provided by the user is strictly greater than (or equal to) 0 and at most equal to 120 before being passed to the code that determines the level of coverage service.

**Question 1:** Design equivalence classes for the *age* input of the code that determines the level of coverage service (without focusing on boundaries).
How many inputs will you need?
How does this differ from your answer to Question 1 of Exercise 1?
<u>Succinctly</u> justify your answer.

**Question 2:** How would this change your answer to Question 2 of Exercise 1?
How many inputs will you need?
<u>Succinctly</u> justify your answer.

**Question 3:** How would this change your answer to Question 3 of Exercise 1?
How many inputs will you need?
<u>Succinctly</u> justify your answer.

## Exercise 3 (6 pts)

A function you must test takes an integer as input and behaves differently, i.e., provides different outputs, depending on whether the input value is strictly below -5, equal to -5, smaller or equal to 30 but strictly greater than -5, and (strictly) greater than 30.

**Question:** Which of the following options (A, B, C, D, E, F, G, H or I) is **a** good selection of input values to exercise partitions (equivalence classes) of the input domain?
(You are not asked to specifically focus on boundaries, simply consider equivalence classes.)
Succinctly justify your answer, i.e., justify *why your selection is **the** good selection* and *why each of the other ones is not a good selection*.

| | | | | | | |
|---|---|---|---|---|---|---|
| A | -12 | 0 | 17 | 30 | 55 | |
| B | -1 | 10 | 20 | 29 | 50 | |
| C | -2 | 15 | 30 | 60 | | |
| D | -5 | 0 | 10 | 30 | 31 | 100 |
| E | -15 | 9 | 30 | | | |
| F | 0 | 2 | 50 | | | |
| G | -20 | -5 | 0 | 12 | 30 | |
| H | -100 | -20 | -5 | 1 | 30 | 31 |
| I | -150 | -5 | 30 | 45 | | |

## Exercise 4 (2+2+2+2=8 pts)

Given the score of a player and the number of remaining lives of the player, a function does the following:

- If the player's score is below 50 (strictly), then it always adds 50 points on top of the current points.
- If the player's score is greater than or equals to 50, then:
  - if the number of remaining lives is greater than or equal to 3, it triples the score of the player.
  - otherwise, it adds 30 points on top of the current points.

**Question 1:** Identify equivalence classes for the "score" parameter. Label them S1, S2 …
**Question 2:** Identify equivalence classes for the "lives" parameter. Label them L1, L2, …
**Question 3:** What are all the possible combinations of equivalence classes (i.e., blocks)?
**Question 4:** Identify values for "score" and "lives" for the combinations you identified.

## Exercise 5 (4 pts)

A function takes two parameters as inputs: an integer representing a month; an integer representing a day in the month. (We simplify the problem by not considering leap years, so the February month always has 28 days.)
An engineer defined the following characteristics along with some blocks for the two parameters.
Characteristic for parameter Month: month-of-the-year

Month A: <0
Month B: >12
Month C: ≥0 and <12

Characteristic of parameter Day: day-of-the-month

Day 1: ≤0
Day 2: 1
Day 3: ≥1 and ≤31

**Question:** There are several issues with this specification of blocks. Can you spot them? Succinctly justify. (Recall the two properties that equivalence classes should satisfy.)

## Exercise 6 (15 pts)

Suppose you have to test a function isValideDate() that takes three integers as input: a value for the day, a value for the month, a value for the year. The function returns true or false, depending on whether the combination of values for the day, the month and the year is valid.
Create equivalence classes for the day parameter, for the month parameter, for the year parameter.
Notes:

- recall that different months have different days.
- recall that in a leap year, the month of February has 29 days. A leap year is a year with one additional day.

Succinctly justify your decisions.