# SET09109 Exercises to be included in the Lab Book

**Exercise 2-1 (2 marks)**

Using Listing 2-7 as a basic design implement and test a new process called `Multiply` that is inserted into the network between the `Producer` and `Consumer` processes which takes an input value from the `Producer` process and multiplies it by some constant `factor` before outputting it to the `Consumer` process. The multiplication `factor` should be one of the properties of the `Multiply` process. To make the output more meaningful you may want to change the output text in the `Consumer` process. You should reuse the `Producer` process from the `ChapterExamples` project in `src` package `c2`.

**Exercise 2-2 (2 marks)**

A system inputs data objects that contain three integers; it is required to output the data in objects that contain eight integers. Write and test a system that undertakes this operation. The process `ChapterExercises/src/c2.GenerateSetsOfThree` outputs a sequence of `List`s, each of which contains three positive integers. The sequence is to be terminated by a `List` comprising [-1, -1, -1].

Then answer the following questions

> What change is required to output objects containing six integers?
> How could you parameterise this in the system to output objects that contain any number of integers (e.g. 2, 4, 8, 12) ?
> What happens if the number of integers required in the output stream is not a factor of the total number of integers in the input stream (e.g. 5 or 7) ?

**Exercise 3-1 (3 marks)**

Write a process that undoes the effect of `GIntegrate`. This can be achieved in two ways, first, by writing a `Minus` process that subtracts pairs of numbers read in parallel similar to `GPlus` or by implementing a `Negator` process and inserting it before a `GPlus` process. Implement both approaches and test them. Which is the more pleasing solution? Why?

**Exercise 3-2 (3 marks)**

Write a sequential version of `GPCopy`, called `GSCopy` that has the same properties as `GPCopy`. Make a copy of Listing 3-13 replacing `GPCopy` by your `GSCopy` and call it `GSPairsA`. Create another version, called `GSPairsB` in which the output channels `outChannel0` and `outChannel1` are assigned to the other actual channel, that is `a.out`() is assigned to `outChannel1` and `b.out()` is assigned to `outChannel1.` Take Listing 3-14 as the basis and replace `GPairs` by `GSPairsA` or `GSPairsB` and determine the effect of the change. Why does this happen? The accompanying web site contains the basis for this exercise apart from the body of `GSCopy`. Hint: read Section 3.7.2 that describes the operation of `GTail`.

**Exercise 3-3 (2 marks)**

Why was it considered easier to build `GParPrint` as a new process rather than using multiple instances of `GPrint` to output the table of results?

**Exercise 4-1 (2 marks)**

What happens if line {25} of `ResetPrefix` Listing 4-1 is commented out? Why?

Explore what happens if you try to send several reset values hence, explain what happens and provide a reason for this.

**Exercise 4-2 (2 marks**

Construct a different formulation of `ResetNumbers` that connects the reset channel to the `GSuccessor` process instead of `GPrefix`. You will have to write a `ResetSuccssor` process. Does it overcome the problem identified in Exercise 1? If not, why not?

**Exercise 5-1 (2 marks)**

The accompanying web site contains a script, called RunQueue, in package `ChapterExercises/src/c5` to run the queue network. The delays associated with `QProducer` and `QConsumer` can be modified. By varying the delay times demonstrate that the system works in the manner expected. Correct operation can be determined by the QConsumer process outputting the messages "`QConsumer has read 1`" to "`QConsumer has read 50`" in sequence. What do you conclude from these experiments?

**Exercise 5-2 (4 marks)**

Reformulate the scaling device so that it uses pre-conditions rather than nested alternatives. Which is the more elegant formulation? Why?

**Exercise 6-1 (5 marks)**

Construct a Test Case for the Three-To-Eight system constructed in the exercise for Chapter 2.

**Exercise 7-1 (5 marks)**

By placing print statements in the coding for the Server and Client processes see if you can determine the precise nature of the deadlock in the Client Server system. You will probably find it useful to add a property to the Server process by which you can identify each Server.

**Exercise 8-1 ( 4 marks)**

Modify the `Client` process `c07.Client` so that it can ensure that the values returned from the `Server` arrive in the order expected according to their `selectList` property. It should print a suitable message that the test has been undertaken and whether it passed or failed. You are **not** to use the `GroovyTestCase` mechanism because this would require that the `CSMux` and `Server` processes would have to terminate, which would require a lot of unnecessary programming.

**Exercise 9-1 ( 5 marks)**

Using the suggestion (Section 9.4.4) made earlier in the chapter, construct an additional process for the event handling system that ensures that the number of missed events is

correct. The additional process should be added to the network of processes. You may need to modify the EventData class (Section 9.2.4) to facilitate this.

**Exercise 9-2 (5.marks)**

The accompanying exercise package contains a version of the event handling system, `RunMultiStream`, which allows the creation of 1 to 9 event streams. By modifying the times associated with each event generation stream and also of the processing system explore the performance of the system. What do you conclude?

**Exercise 9-3 (6 marks)**

The process `EventProcessing` has three versions of multiplexer defined within it, two of which are commented out. By choosing each of the options in turn, comment upon the effect that each multiplexer variation has on overall system performance.

**Exercise 11-3 (8 marks)**

The Control process in the Scaling system currently updates the scaling factor according to an automatic system. Replace this with a user interface that issues the suspend communication, obtains the current scaling factor and then asks the user for the new scaling factor that is then injected into the `Scaler`. The original and scaled values should also be output to the user interface.

**Total Marks (60)**