**Lab Book**

**Ex 2-1**



**Consumer.groovy**

// insert a modified println statement

*println* "the result was: ${i}"

**Multiplier.groovy**

// write i \* factor to outChannel

outChannel.write(i \* factor)

// read in the next value of i

i = inChannel.read()

**RunMultiplier.groovy**

//insert here an instance of multiplier with a multiplication factor of 4

**new** Multiplier (inChannel: connect1.in(),

outChannel: connect2.out(),

factor: 4)

**Output**

next: 1

next: the result was: 4

2

next: the result was: 8

4

next: the result was: 16

3

next: the result was: 12

0

Finished

**Ex 2-2C:\Users\Beej\AppData\Local\Microsoft\Windows\INetCacheContent.Word\ex2.2_diagram.png**

**GenerateSetsOfThree.groovy**

//write the terminating List as per exercise definition

outChannel.write([-1,-1,-1])

**ListToStream.groovy**

// hint: output list elements as single integers

**for** ( i **in** 0 ..< inList.size)outChannel.write(inList[i])

inList = inChannel.read()

**CreateSetsOfEight.groovy**

// put v into outList and read next input

outList[i] = v

v = inChannel.read()

**Output**

Eight Object is [1, 2, 3, 4, 5, 6, 7, 8]

Eight Object is [9, 10, 11, 12, 13, 14, 15, 16]

Eight Object is [17, 18, 19, 20, 21, 22, 23, 24]

Finished

**Questions**

What change is required to output objects containing six integers?

**for** ( i **in** 0 .. 7 ) becomes **for** ( i **in** 0 .. 5 ) in CreateSetsOfEight.groovy

How could you parameterise this in the system to output objects that contain any number of integers (e.g. 2, 4, 8, 12) ?

Have the number the for loop be a variable that is read in from the console and can be decided by the user by writing it in the console.

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)

Numbers are left out and the process does not terminate because ListToStream.groovy can’t finish its for loop, and doesn’t send ‘-1’ to CreateSetsofEight.groovy

**Ex 3-1**

**Process Network Diagram for Differentiate using Minus**



**Minus.groovy**

// output one value subtracted from the other

// be certain you know which way round you are doing the subtraction!!

outChannel.write(read0.value - read1.value)

**Differentiate.groovy**

// insert a constructor for Minus

**new** Minus ( inChannel0: a.in(),

inChannel1: c.in(),

outChannel: outChannel)

**Output**

Differentiated Numbers

0

1

2

3

4

5

6

7

8

9

10

11

12

13

**Process Network Diagram for Differentiate using Negator**



**Negator.groovy**

//output the negative of the input value

outChannel.write(-(inChannel.read()))

**Differentiate.groovy**

//insert a constructor for Negator

**new** Negator ( inChannel: c.in(),

outChannel: d.out() )

**Output**

Differentiated Numbers

0

1

2

3

4

5

6

7

8

9

10

11

**Questions**

Which is the more pleasing solution? Why?

I prefer the Minus solution. It seems to be simpler, especially when considering the network process diagram. It directly undoes the integrate step as Minus.groovy is the oppositve to GPlus.groovy. The negator adds another process before the GPlus that I feel overcomplicates it.