**Exercise 2-1**

**RunMultiplier.groovy**

**def** processList = [ **new** Producer ( outChannel: connect1.out() ),

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

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

outChannel: connect2.out(), factor: 4 ),

**new** Consumer ( inChannel: connect2.in() )

]

**Multiplier.groovy**

// write i \* factor to outChannel

outChannel.write (i \* factor)

// read in the next value of i

i = inChannel.read()

**Consumer.groovy**

//insert a modified println statement

println "Next integer multiplied by 4 is: ${i}"

i = inChannel.read()

**Output**

next: 2

next: Next integer multiplied by 4 is: 8

3

next: Next integer multiplied by 4 is: 12

4

next: Next integer multiplied by 4 is: 16

5

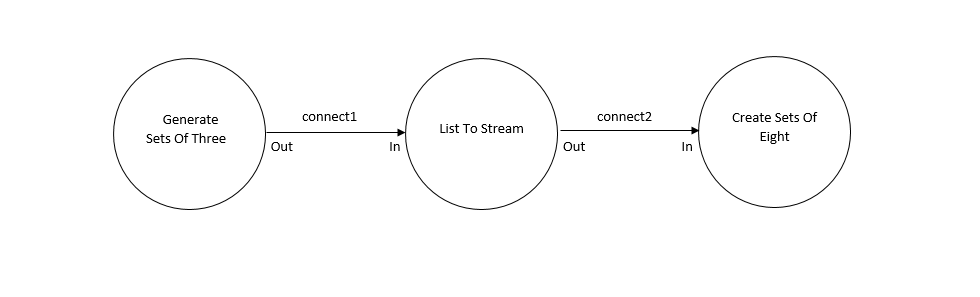
next: Next integer multiplied by 4 is: 20

2

next: Next integer multiplied by 4 is: 8

0

Finished

**Exercise 2-2**

**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]

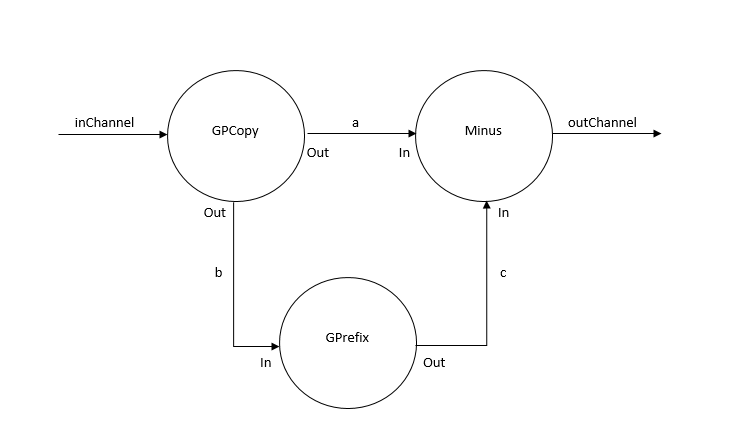
Eight Object is [17, 18, 19, 20, 21, 22, 23, 24]

Finished

**Questions**

1. To output 6 integers, change the line **for(i in 0 ..7)** to **for(i in 0 ..5)** in CreateSetsOfEight.groovy
2. Create a variable **size** that can be adjusted to any number
3. The process will just get stuck, it will eventually read the termination number of -1 but then will have no number to read and will be stuck.

**Exercise 3-1**

 Differentiate

**Minus.groovy**

parRead2.run()

// 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**

**def** differentiateList = [ **new** GPrefix ( prefixValue: 0,

inChannel: b.in(),

outChannel: c.out() ),

**new** GPCopy ( inChannel: inChannel,

outChannel0: a.out(),

outChannel1: b.out() ),

// insert a constructor for Minus

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

inChannel1: c.in(),

outChannel: outChannel )

]

**Output - Minus**

Differentiated Numbers

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

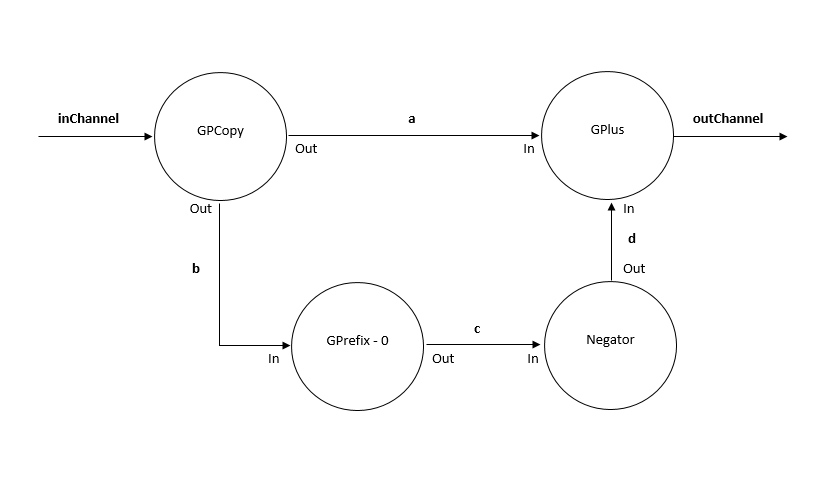
16

17

18

19

20

 DifferentiateNeg

**Negator.groovy**

//output the negative of the input value

outChannel.write(-inChannel.read())

**DifferentiateNeg.groovy**

**def** differentiateList = [ **new** GPrefix ( prefixValue: 0,

inChannel: b.in(),

outChannel: c.out() ),

**new** GPCopy ( inChannel: inChannel,

outChannel0: a.out(),

outChannel1: b.out() ),

//insert a constructor for Negator

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

outChannel: d.out() ),

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

inChannel1: d.in(),

outChannel: outChannel )

]

**Output - negator**

Differentiated Numbers - Negator

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

**Question**

I believe the Minus version is much simpler to understand and the Minus process will probably be reused way more often as its pretty much the same as the Plus process, which is reused all the time. The only problem is that if you want to insert the Minus process, you must be careful about the order of subtraction.

**Exercise 3-2**

**GSCopy.groovy**

**def** i = inChannel.read()

// output the input value in sequence to each output channel

outChannel0.write(i)

outChannel1.write(i)

**GSquares.groovy Version A**

**def** testList = [ **new** GNumbers ( outChannel: N2I.out() ),

**new** GIntegrate ( inChannel: N2I.in(),

outChannel: I2P.out() ),

// you will need to modify this twice

//first modification is to insert a constructor for GSPairsA

// then run the network using TestGSCopy

**new** GSPairsA ( inChannel: I2P.in() )

//second modification replace the constructor for GSPairsA with GSPairsB

// then run the network again using TestGSCopy

// you will then be able to compare the behaviour and to

// explain why this happens!

]

**GSquares.groovy Version B**

**def** testList = [ **new** GNumbers ( outChannel: N2I.out() ),

**new** GIntegrate ( inChannel: N2I.in(),

outChannel: I2P.out() ),

// you will need to modify this twice

//first modification is to insert a constructor for GSPairsA

// then run the network using TestGSCopy

**new** GSPairsB ( inChannel: I2P.in() )

//second modification replace the constructor for GSPairsA with GSPairsB

// then run the network again using TestGSCopy

// you will then be able to compare the behaviour and to

// explain why this happens!

]