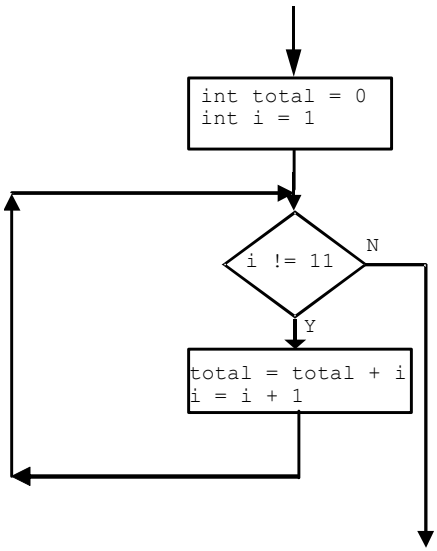
University of Stirling
Computing Science and Mathematics
CSCU9A2 Tutorial 2 – Sample solutions Spring 2017
(for week starting 6 February)

1. (a) It is calculating 1+2+3+…+10

   (b) Flow diagram:



   (c) Brookshear assembly code:

   Need to allocate memory byte addresses for variables total and i, and chose a start address: Say 40, 41 and 20. Register allocation is arbitrary. Variants in the sequence are OK. Could also use labels instead of memory addresses for total, i, loop test, exit.

   Now code:

```
Address    Assembly code
20         MOV 00 -> R1        )
22         MOV R1 -> [40]      ) total = 0

24         MOV 01 -> R1        )
26         MOV R1 -> [41]      ) i = 1, and note i is in R1

28         MOV 0B -> R0        11 decimal is 0B hex
2A         JMPEQ 3E, R1        Exit if i == 11

2C         MOV [40] -> R2      )
2E         MOV [41] -> R3      )
30         ADDI R2, R3 -> R2   )
32         MOV R2 -> [40]      ) total = total + i

34         MOV 01 -> R2        )
36         MOV [41] -> R1      )
38         ADDI R1, R2 -> R1   )
3A         MOV R1 -> [41]      ) i = i+1, and i remains in R1

3C         JMPEQ 28, R0        Uncond jump to loop test
3E         ...
```

   This could have some optimizations:

   - For the loop test, i has previously been loaded into R1 so do not need to load it again;
   - 11 for the loop test is kept undisturbed in R0, so the uncond jump back could be to 2A, not 28.

   Other similar optimizations are possible (basically by keeping total and i in registers rather than memory).

   (d) Machine code:

```
Address    Code
20         2100
22         3140
24         2101
26         3141
28         200B
2A         B13E
2C         1240
2E         1341
30         5223
32         3240
34         2201
36         1141
38         5112
3A         3141
3C         B028
3E         ...
```

2.

(a) Detailed tracing should look at how the value of loop variable `i` is used and changed, how the `if` test looks up two elements of the array, what those values are on each repetition, how the actual swapping of values within the array occurs by "shuffling" via `temp`. Partial details are given below (labelled with the line number involved):

1) Arrival at loop: `i: 0  list: 88, 12, 6, 99, 4, 5`

1) Loop test is `true` (`i` is less than `list.length` (6)), so execute loop body

3) `i` is 0, so compare `list[0]` with `list[0+1]`, that is `88 > 12 ? : true`, so carry out block

5) `i` is 0, so assign `list[0]` (that is `88`) to `temp`:
`i: 0  temp: 88  list: 88, 12, 6, 99, 4, 5`

6) Then assign `list[0+1]` (that is `12`) to `list[0]`:
`i: 0  temp: 88  list: 12, 12, 6, 99, 4, 5`

7) Then assign `temp` (that is `88`) to `list[0+1]`: (completing the swap)
`i: 0  temp: 88  list: 12, 88, 6, 99, 4, 5`

8) The block, `if` and loop body are complete, back to loop control

1) Increment `i` to 1, loop test is still `true`, so execute loop body

3) `i` is 1, so compare `list[1]` with `list[1+1]`, that is 88 > 6 ? : `true`, so carry out block

5,6,7) Swap elements 1 and 2 via `temp`:
`i: 1  temp: 88  list: 12, 6, 88, 99, 4, 5`

8) The block, `if` and loop body are complete, back to loop control

1–8) When i=2 there is no swap, leaving list: `12, 6, 88, 99, 4, 5`

1–8) When i=3 it swaps (99 via temp), to get list: `12, 6, 88, 4, 99, 5`

1–8) When i=4 it swaps (99 via temp), to get list: `12, 6, 88, 4, 5, 99`

1) When i=5 the loop exits

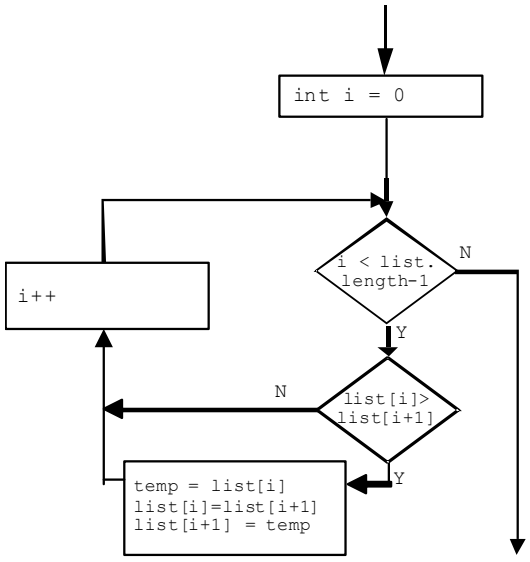
Finally: list: `12, 6, 88, 4, 5, 99`

If the elements are written out from left-to-right, the algorithm "bubbles" high numbers from left to right, exchanging out-of-order pairs: successively. The overall effect is for the largest number to be "dragged" to the highest index element of the array.

(b) Because the indices in `list` are `0..(list.length-1)`, and `i` must not get as high as `(list.length-1)` otherwise `list[i+1]` would be off the end of the array. If the code counted up towards `list.length` then we would get an `ArrayIndexOutOfBoundsException`

(c) Each sweep would move list closer to being sorted in ascending order, so after some number of repetitions, the array would be completely sorted. [The Java fragment is one "sweep" of the array from a sorting algorithm known as "bubble sort".]

(d) Flow diagram:



SBJ February 2017