SL Unit 4 – Problem Solving

Quiz 1

Question 1			
Objectives:	5.1.6, 5.1.7	Exam Reference:	May-15 13

1. Theo entered a maze (labyrinth) and tries to get to the centre. As soon as he arrived at the first possibility to turn right or left, he started recording each move on his phone so that he could find his way back to the start. He entered the moves as the direction he turned followed by the number of steps taken before the next turn. For example:

   R3 , L5 , L10 , R6 , … , L4

   which indicates "TURN **right**, STEP **3**", and then "TURN **left**, STEP **5**" etc.

   An app on his phone stored the moves in a stack named STK, using 0 for "right" and 1 for "left".

   The above moves were therefore stored as

   0 , 3 , 1 , 5 , 1 , 10 , 0 , 6 , … , 1 , 4.

   (a) Explain why a stack is a suitable structure to hold the data. [2]

Because items/moves are needed in reverse order;
To the order input;

OR

Because it is a LIFO (Last In First Out) data structure;
The items/moves pushed/placed onto the stack;
Will be popped off/taken from it in reverse order (to the order input);

Theo was successful in reaching the centre of the maze and now has to get back to the start.

(b) Construct an algorithm, using appropriate stack access methods, to output the moves needed to return from the centre to the first point where Theo started recording his moves. You can assume that he is **facing** the correct exit when he starts his return journey. [5]

*Award marks as follows, up to **[5 marks max]**.*
*Award **[1 mark]** for checking for empty.*
*Award **[1 mark]** for popping STEP and TURN from stack.*
*Award **[1 mark]** for an output (STEP).*
*Award **[1 mark]** for correct if statement (checking popped TURN).*
*Award **[1 mark]** for an output (TURN).*

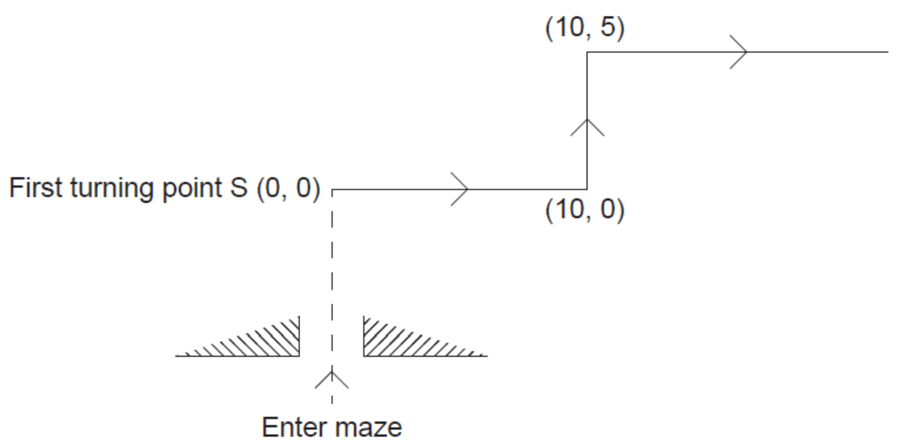
For example:

```
(let STK be the stack)
  while NOT STK.isEmpty() do
    STEP = STK.pop()
    output "Take " + STEP + " steps"
    TURN = STK.pop()
    if TURN = 0 then
      output "Turn left"
    else
      output "Turn right"
    end if
  end while
```

Another app on the phone gives Theo a visual representation of his path through a maze as a map. This app makes use of a procedure *MOVE()*, which outputs the coordinates of Theo's path through a maze, in reference to the point S, where he first turned right or left and which has coordinates (0, 0).

The diagram shows, for a **new maze**, the map from point S given the following moves:

R10 , L5 , …

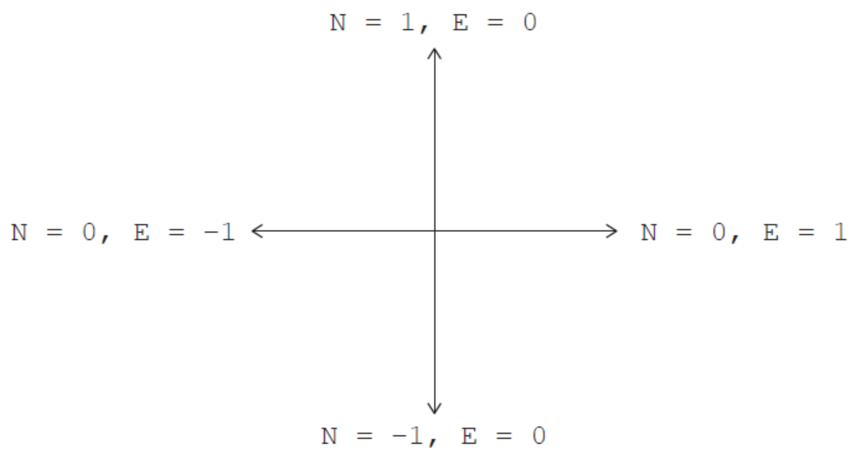


The third move is R8.

(c) State the coordinates on the map after this third move. [1]

(18, 5)

At each point, the direction in which Theo is facing is given by the variables *N* and *E*.
Note that before the very first turn is made, *N = 1* and *E = 0*.



The following table shows **part** of the trace of *MOVE()* according to the *TURN* and *STEP* values: R10 , L5 , R8 , R2 , L3 , R0.

The last move, with a *STEP* value of 0, indicates that there are no more moves and that the stack is empty.

(d) Complete the table by tracing the algorithm on the following page.

Move		**Coordinates**		**Direction facing**	
TURN	STEP	X	Y	N	E
		0	0	1	0
0	10	10	0	0	1
1	5	10	5	1	0
0	8				
0	2				
1	3				
0	0				

```
X, Y = 0                          // Initial coordinates
N = 1, E = 0                      // Direction facing at first turn
output (X, Y, N, E)               // Outputs starting point to the table
MOVE(X,Y,N,E)                     // Procedure to move on
    input (TURN, STEP)
    loop while STEP ≠ 0           // No more moves when STEP = 0
        if TURN = 0               // Right move
            X = X + N*STEP
            Y = Y - E*STEP
            if N = 0
                N = -E
                E = 0
            else
                E = N
                N = 0
            end if
        end if

        if TURN = 1               // Left move
            X = X - N*STEP
            Y = Y + E*STEP
            if N = 0
                N = E
                E = 0
            else
                E = -N
                N = 0
            end if
        end if
        output (X, Y, N, E)
        MOVE(X, Y, N, E)
    end loop
end MOVE
```

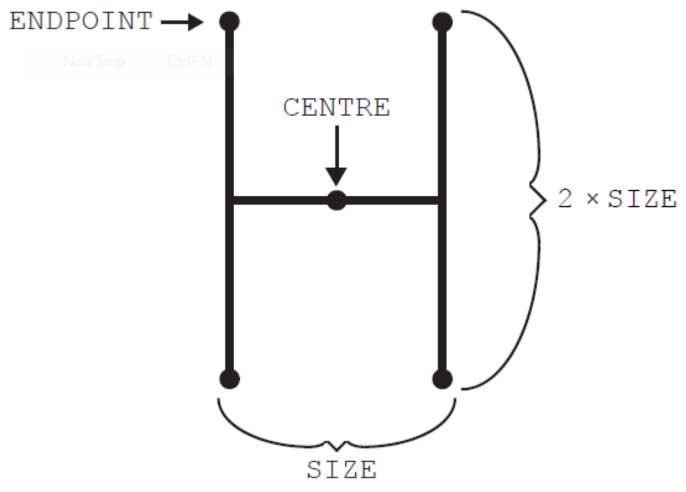
[6]

*Award marks as follows up to **[6 marks max]**.*
*Award **[1 mark]** for each correct pair X and Y (coordinates), **x3**.*
*Award **[1 mark]** for each correct change of direction facing (correct E and N), **x3**.*

Move		Coordinates		Direction facing	
TURN	STEP	X	Y	N	E
		0	0	1	0
0	10	10	0	0	1
1	5	10	5	1	0
0	8	18	5	0	1
0	2	18	3	-1	0
1	3	21	3	0	1
0	0				

Question 2			
Objectives:	5.1.7	Exam Reference:	May-14 14

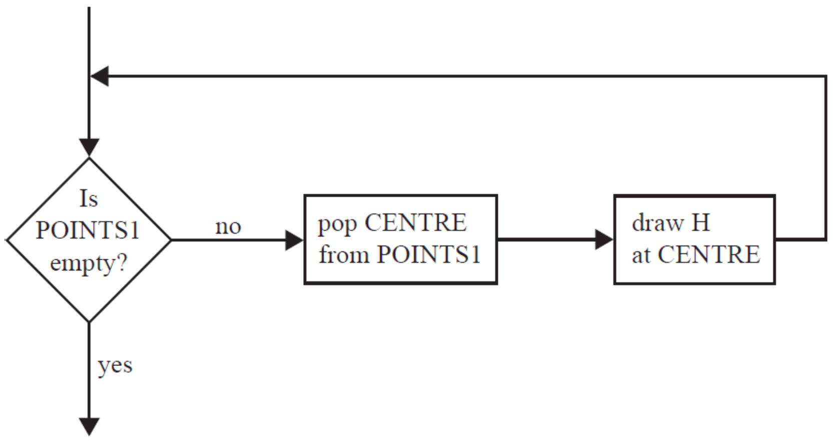
Consider the following diagram and pseudocode for drawing on a display screen.



```
ENDPOINTS = drawH(CENTRE, SIZE)
```

The method *drawH(CENTRE, SIZE)* will draw an "H" located at *CENTRE* with width of *SIZE* and height of *2 × SIZE*, as shown. It returns an array containing the four **endpoints** of the vertical lines.

In the following flowchart, *POINTS1* is a stack.



(a) Construct pseudocode corresponding to the flowchart. [3]

*Award up to **[3 marks max]**.*
*Award **[1 mark]** for a loop (while).*
*Award **[1 mark]** for correct condition/logical expression.*
*Award **[1 mark]** for correct operations.*

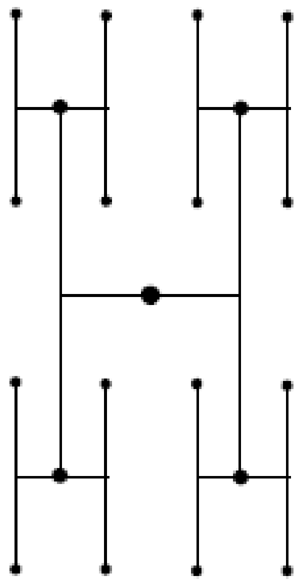
```
loop while NOT POINTS1.isEmpty()
    CENTRE = POINTS1.pop()
    drawH(CENTRE, SIZE)
end loop
```

(b) Construct the drawing that would be produced by the flowchart on page 6 if it is preceded
by the following steps.

```
SIZE = 20
CENTRE = the middle of the user's display
POINTS1 is a stack, initially empty
ENDPOINTS = drawH(CENTRE, SIZE)
loop COUNT from 0 to 3
  POINTS1.push(ENDPOINTS[COUNT])
end loop
SIZE = SIZE / 2
```

[2]

Award [1 mark] for correct number and placement of H's;
Award [1 mark] for reasonable attempt to show reduced size of successive H's;



The pattern of drawing a new set of H's, which have a *SIZE* value that is half the *SIZE* value of the previous H, can be repeated. Each set of H's of the same size is called a generation.

(c) Construct an algorithm that will draw an initial H in the centre of the display and three generations after that. [6]

*Award up to **[6 marks max]** as follows:*
***[1 mark]** for the idea that there needs to be a loop that executes once for each generation, **with an additional [1 mark]** if it is implemented correctly, for **[2 marks max]**.*

***[1 mark]** for the idea that the endpoints returned while drawing one generation must be stored for use in drawing the subsequent generation, **with an additional [1 mark]** if correctly implemented, for **[2 marks max]**. **Note:** Any valid storage mechanism is permissible ie it does not have to be a stack.*

***[1 mark]** for correctly drawing a generation of H's.*

***[1 mark]** for correctly scaling the size for each generation.*

Example answer:
```
/////////////////////
SIZE = 20                             //do not award marks
CENTRE = the middle of the user's display
POINTS1 is a stack, initially empty   //for this part of the
ENDPOINTS = drawH( CENTRE, SIZE )     // of the algorithm
loop COUNT from 0 to 3
     POINTS1.push( ENDPOINTS[COUNT] )  // it is given
end loop
SIZE = SIZE / 2                       // in the question paper
//////////////////////

POINTS2 is a stack, initially empty
loop GENERATION from 1 to 3
     loop while NOT POINTS1.isEmpty()
          CENTRE = POINTS1.pop()
          ENDPOINTS = drawH(CENTRE, SIZE)
          loop COUNT from 0 to 3
               POINTS2.push( ENDPOINTS[COUNT] )
          end loop
     end loop
     POINTS1 = POINTS2
     empty the POINTS2 stack
     SIZE = SIZE / 2
```

(d) State how many endpoints there will be after the initial H and three generations have been drawn, without any consideration of the size. [1]

Accept any expression appearing in the following equivalences
$4*4*4*4 = 4^4 = 256$;

(e) Suggest how drawing this pattern of H's could be done recursively. [3]

Award up to ***[3 marks max]*** *for:*
identifying that the recursive algorithm proceeds downwards (showing on one parameter will suffice);
base case of recursion;
recursive call;

The recursive algorithm would use the parameters centre, size, generation counting down (not up);
When the generation reached is 0 no drawing is done;
Otherwise draws an H and calls recursively itself on the endpoints of a lower generation;

Question 3

Objectives:	5.1.9	Exam Reference:	Ma -14 13

The faceplate of a car stereo has six buttons for selecting one of six preferred radio stations. As part of the internal representation of a microprocessor there is an array with six positions, carrying the information about the radio frequencies, as follows.

Radio

[0]	[1]	[2]	[3]	[4]	[5]
100.4	88.7	90.2	104.5	93.8	106.2

(a) State the information at *Radio[2]*. [1]

90.2

(b) Outline how a numerical frequency could be stored in a fixed-length string. [2]

Frequencies less than 100 take a 0 on the left (*eg* 88.7 becomes 088.7);
Convert each digit into a char to get a string;
Allow the "dot" to be omitted in the interpretation. There is always only one

(c) Construct an algorithm in pseudocode that calculates the range of frequencies (*ie* the difference between the highest and lowest frequencies) of any set of six selected radio stations. [6]

Award up to [6 marks max].

***Example answer** (searches for the min and max, and then the range is calculated)*
Award [1 mark] for each of the following
Initialization;
Loop;
Correct if statement (min);
Correct if statement (max);
Compute the range;
Output the range;

```
MIN = Radio[0]
MAX = Radio[0]
K=1
loop while K<=5
      if Radio[K]<MIN then
            MIN=Radio[K]
      else if Radio[K]>MAX then
            MAX=Radio[K]
      endif
      K=K+1
endloop
RANGE=MAX-MIN
output RANGE
```

***Example answer** (sorts the array Radio, and then the range is calculated, any sorting algorithm is acceptable)*
Award [1 mark] for each of the following
Idea of nested loops;
Correct loops;
Correct comparison;
Correct exchange;
Compute the range;
Output the range;

```
loop for  K=0 to 4
     loop for  J=0 to 4
      if Radio[J]> Radio[J+1]
       then
          swap Radio[J]and Radio[J+1]
      endif
     endloop
endloop
RANGE= Radio[5]- Radio[0]
output RANGE
```

The two-dimensional array *Stats* provides an indication of how often a specific station is listened to by the user. For each button in the faceplate it records how often it has been clicked in the last 48 hours. *Stats* is ordered by the second column.

`Stats`

5	13
4	9
0	8
3	4
1	3
2	2

Both *Radio* and *Stats* are used by a procedure that allows the user to access the radio frequencies that are listened to most often, as recorded in *Stats*, by flicking a lever on the steering wheel. The frequencies are accessed cyclically, *ie* after the least used frequency the procedure returns to the most used. For this reason a queue **Q** is used.

(d) Construct an algorithm in pseudocode that, by using the structures *Radio* and *Stats*, performs the following steps:

- it inserts the radio frequencies in the queue **Q**, following the actual order of preference;
  and then

- it uses the queue **Q**, cyclically, to output an element each time the lever is flicked. [6]

Award up to [6 marks max].
Possible solution below, marks to be awarded following the comments in code (there are 7 marks at least).

```
// Q is given and initially empty
I=0
loop while I<6                    //across all rows
     ITEM=Stats[I,0]             //retrieve the button
     Q.enqueue(Radio(ITEM))      //enqueue the frequency
     I=I+1
endloop

if input(Flick) then             //if a flicking occurs
     TEMP=Q.dequeue()            //dequeue an element from Q
     output(TEMP)                //to output it and
     Q.enqueue(TEMP)             //circular use of the queue
endif
```