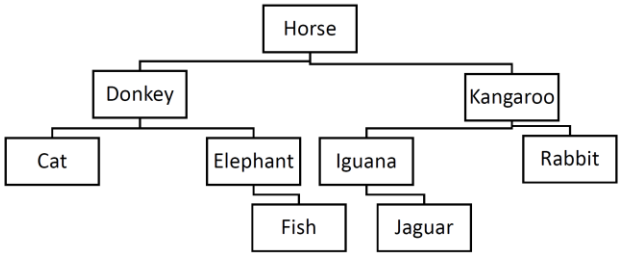
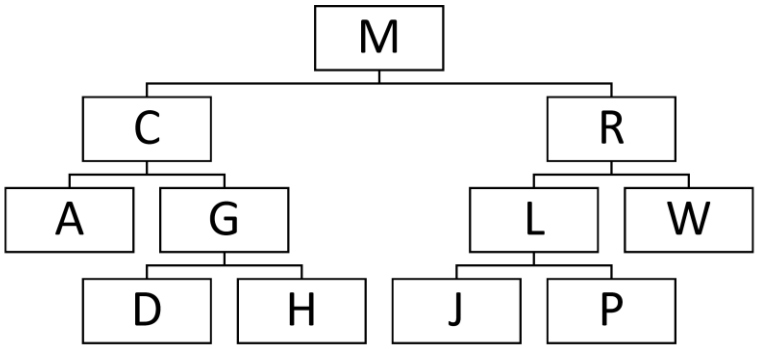
Question No. 1

1(a)	Horse	**1**
1(b)	Cat // Elephant // Kangaroo	**1**
1(c)	1 mark for Iguana and Jaguar in the correct place 1 mark for Rabbit and Fish in the correct place  	**2**
1(d)	1 mark per bullet point. Mark in pairs. • (Compare Elephant to horse) **Elephant/E** is **less** than **Horse/H** so check/go **left** … • … (Compare to Elephant to Donkey) **Elephant/E** is **greater** than **Donkey/D** so check/go **right** (Elephant found) or • Check if **Elephant/E** is **less than** or **greater than root** node … • … check **subtree**/follow pointer to next node to left/right **recursively** until **found** or **leaf**	**2**

Question No. 2

4(a) — **2**

1 mark for adding D and H below G
1 mark for adding J and P below L



4(b)(i) — **4**

1 mark for rootPointer pointing to 0
1 mark for freePointer pointing to 11
1 mark for left and right correctly linked nodes 0 TO 5
1 mark for -1 added as pointer for all remaining null pointers

rootPointer	0
freePointer	11

Index	leftPointer	data	rightPointer
0	1	M	5
1	2	C	4
2	-1	A	-1
3	7	L	9
4	8	G	10
5	3	R	6
6	-1	W	-1
7	-1	J	-1
8	-1	D	-1
9	-1	P	-1
10	-1	H	-1
11	(-1)		(-1)

4(b)(ii) — **2**

1 mark per bullet point
- Defining 1D array with 100 elements
- of type node, with identifier `binaryTree`

Example:
```
DECLARE binaryTree : ARRAY[0:99] OF node
```

(ii) Write **pseudocode** to declare the array `binaryTree` to store up to 100 objects of type `node`.

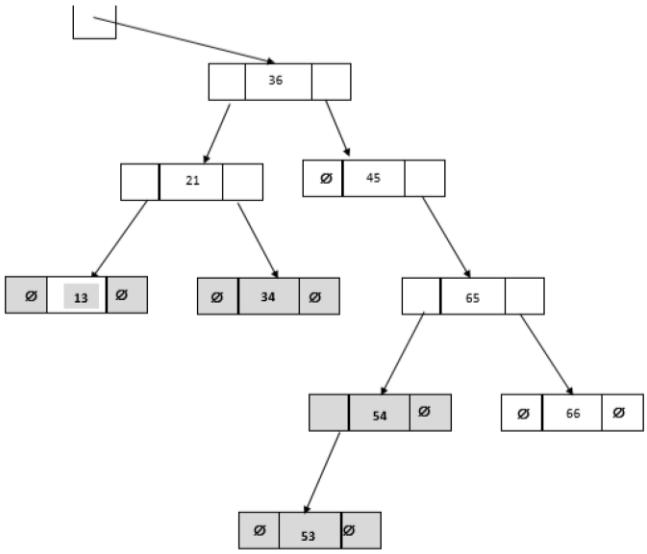
.....

.....

.....

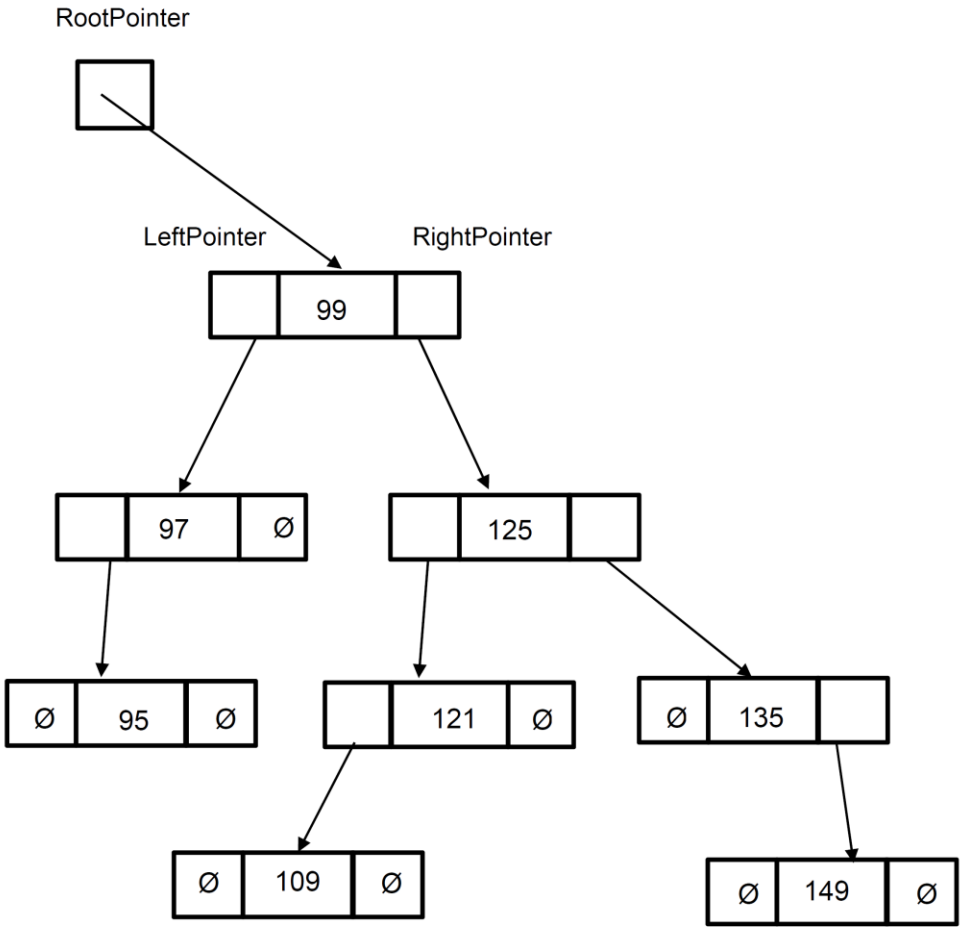
..... [2]

Question No. 3

1(a)(i)	1 mark per bullet point • 13 to left of 21 • 34 to right of 21 • 54 to left of 65 • 53 to left of 54 • Null and other pointers on all boxes written **and** no other pointers filled in  	5

1(a)(ii)	1 mark per bullet point • FreePointer • 36 • 45 • 21 and 65 • 66 and 13 • 54, 53 and 34 RootPointer: 0 FreePointer: 9 Index / LeftPointer / Data / RightPointer [0] 2 36 1 [1] null 45 3 [2] 5 21 8 [3] 6 65 4 [4] null 66 null [5] null 13 null [6] 7 54 null [7] null 53 null [8] null 34 null [9]	6

Question No. 4

2(a)(i)	1 mark per bullet point • 95 to left of 97 • 109 to left of 121 • 135 to right of 125 • 149 to right of 135 • Null points in all places and no inappropriate pointers  	5

2(a)(ii) — 1 mark per bullet point — **6**

- FreePointer as 8
- 99
- 125
- 121 and 97
- 109 and 95
- 135 and 149

RootPointer
0

FreePointer
8

Index	LeftPointer	Data	RightPointer
[0]	3	99	1
[1]	2	125	6
[2]	4	121	null
[3]	5	97	null
[4]	null	109	null
[5]	null	95	null
[6]	null	135	7
[7]	null	149	null
[8]			

2(b) — 1 mark for each completed section — **6**

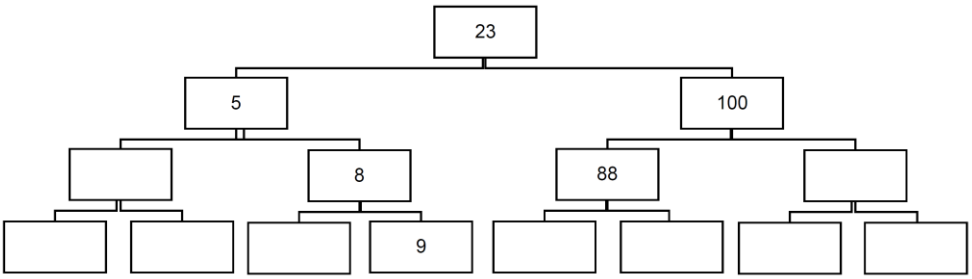
```
FUNCTION FindElement(Item : INTEGER) RETURNS INTEGER
   CurrentPointer ← RootPointer
   WHILE CurrentPointer <> NullPointer
      IF List[CurrentPointer].Data <> Item
         THEN
            CurrentPointer ← List[CurrentPointer].Pointer
         ELSE
            RETURN CurrentPointer
      ENDIF
   ENDWHILE
   CurrentPointer ← NullPointer
   RETURN CurrentPointer
ENDFUNCTION
```

Question No. 5

4(c)(i)	To point to the start/first of the empty node/nodes	1
4(c)(ii)	–1 or below // 101 or above	1

4(c)(iii) — 2

1 mark for 23 at top, with 5 below left, 100 below right
1 mark for remaining in correct places below 5 and 100



4(c)(iv) — 5

1 mark for each completed statement

```
PROCEDURE AddData(NewNode)
  BinaryTree[FreePointer] ← NewNode
  BinaryTree[FreePointer].LeftPointer ← -1
  BinaryTree[FreePointer].RightPointer ← -1

  DECLARE PositionFound : BOOLEAN
  DECLARE PointerCounter : INTEGER
  PositionFound ← FALSE
  PointerCounter ← RootNode

  WHILE NOT PositionFound
    IF NewNode.Data < BinaryTree[PointerCounter].Data
      THEN
        IF BinaryTree[PointerCounter].LeftPointer = -1
          THEN
            BinaryTree[PointerCounter].LeftPointer ← FreePointer
            PositionFound ← TRUE
          ELSE
            PointerCounter ← BinaryTree[PointerCounter].LeftPointer
        ENDIF
      ELSE
        IF BinaryTree[PointerCounter].RightPointer = -1
          THEN
            BinaryTree[PointerCounter].RightPointer ← FreePointer
            PostionFound ← True
          ELSE
            PointerCounter ← BinaryTree[PointerCounter].RightPointer
        ENDIF
    ENDIF
  ENDWHILE
  FreePointer ← FreePointer + 1
ENDPROCEDURE
```

Question No. 6

2(a)	• A pointer that doesn't point to another node/other data/address // indicates the end of the branch	1
2(b)	one mark per bullet • node with 'Athens' linked to left pointer of Berlin (ignore null pointer) • null pointers in left and right pointers of Athens	2
2(c)(i)	RootPointer: 0 FreePointer: 7 1 mark	5
2(c)(ii)	• –1 • It is not the number for any node.	2

	LeftPointer	Tree Data	RightPointer
[0]	2	Dublin	1
[1]	-1/∅	London	3
[2]	6	Berlin	5
[3]	4	Paris	-1/∅
[4]	-1/∅	Madrid	-1/∅
[5]	-1/∅	Copenhagen	-1/∅
[6]	-1/∅	Athens	-1/∅
[7]	8		-1/∅
[8]	9		-1/∅
[9]	-1/∅		-1/∅

2(d)(i) — 7

```
TYPE Node
   LeftPointer : INTEGER
   RightPointer : INTEGER                                   1
   Data : STRING
ENDTYPE
DECLARE Tree : ARRAY[0 : 9] OF Node                         1
DECLARE FreePointer : INTEGER
DECLARE RootPointer : INTEGER

PROCEDURE CreateTree()
   DECLARE Index : INTEGER
   RootPointer ← -1                                         1
   FreePointer ← 0                                          1
   FOR Index ← 0 TO 9   // link nodes
      Tree[Index].LeftPointer ← Index + 1                   1
      Tree[Index].RightPointer ← -1                         1
   ENDFOR
   Tree[9].LeftPointer ← -1                                 1
ENDPROCEDURE
```

2(d)(ii) — 8

```
PROCEDURE AddToTree(ByVal NewDataItem : STRING)
// if no free node report an error
   IF FreePointer = -1                                      1
      THEN
         ERROR("No free space left")
      ELSE   // add new data item to first node in the free list
         NewNodePointer ← FreePointer
         Tree[NewNodePointer].Data ← NewDataItem            1
         // adjust free pointer
         FreePointer ← Tree[FreePointer].LeftPointer        1
         // clear left pointer
         Tree[NewNodePointer].LeftPointer ← -1              1
         // is tree currently empty ?
         IF RootPointer = -1                                1
            THEN   // make new node the root node
               RootPointer ← NewNodePointer                 1
            ELSE   // find position where new node is to be added
               Index ← RootPointer
               CALL FindInsertionPoint(NewDataItem, Index, Direction)
```

```
               IF Direction = "Left"
                  THEN   // add new node on left
            Tree[Index].LeftPointer ← NewNodePointer        1
                  ELSE   // add new node on right
            Tree[Index].RightPointer ← NewNodePointer       1
               ENDIF
            ENDIF
      ENDIF
ENDPROCEDURE
```

2(e)	**1 mark per bullet**
	• test for base case (null/-1)
	• recursive call for left pointer
	• output data
	• recursive call for right pointer
	• order, visit left, output, visit right
	`IF Pointer <> NULL` 1
	`THEN`
	`TraverseTree(Tree[Pointer].LeftPointer)` 1
	`OUTPUT Tree[Pointer].Data` 1 + 1
	`TraverseTree(Tree[Pointer].RightPointer)` 1
	`ENDIF`
	`ENDPROCEDURE`
		5

Question No. 7

4(a)	1 mark per bullet point
	• Record declaration with identifier Node …
	• … all three fields declared with type integer
	
	Example:
	`TYPE Node`
	`  DECLARE LeftPointer : INTEGER`
	`  DECLARE Data : INTEGER`
	`  DECLARE RightPointer : INTEGER`
	`ENDTYPE`
		2

4(b)	1 mark per bullet point:
	• Declaration with correct identifier (Node100) of type Node
	• Assigning `LeftPointer` to 1 and `RightPointer` to 4
	• Assigning 100 to the Data
	
	Example pseudocode
	
	`DECLARE Node100 : Node`
	`Node100.LeftPointer ← 1`
	`Node100.Data ← 100`
	`Node100.RightPointer ← 4`
		3