3 A data structure is required to store 25 nodes. A linked list is maintained of all the nodes. A node contains a data value and two pointers: a left pointer and a right pointer.
Items in the list are initially linked using their `LeftChild` pointers.

Each node is implemented as an instance of the class `ConnectionNode`. The class `ConnectionNode` has the following properties:

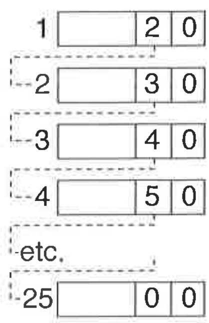
Class: `ConnectionNode`		
Attributes		
Identifier	Data Type	Description
`DataValue`	`STRING`	The node data
`LeftChild`	`INTEGER`	The left node pointer
`RightChild`	`INTEGER`	The right node pointer

The structure for the linked list is implemented as follows:

Identifier	Data Type	Description
`RobotData`	`ARRAY[1 : 25] OF ConnectionNode`	An array used to store the 25 nodes.
`Root`	`INTEGER`	Index for the root position of the `RobotData` array.
`NextFreeChild`	`INTEGER`	Index for the next available empty node.

The first available node is indicated by `NextFreeChild`. The initial value of `Root` is 1 and the initial value of `NextFreeChild` is 1.

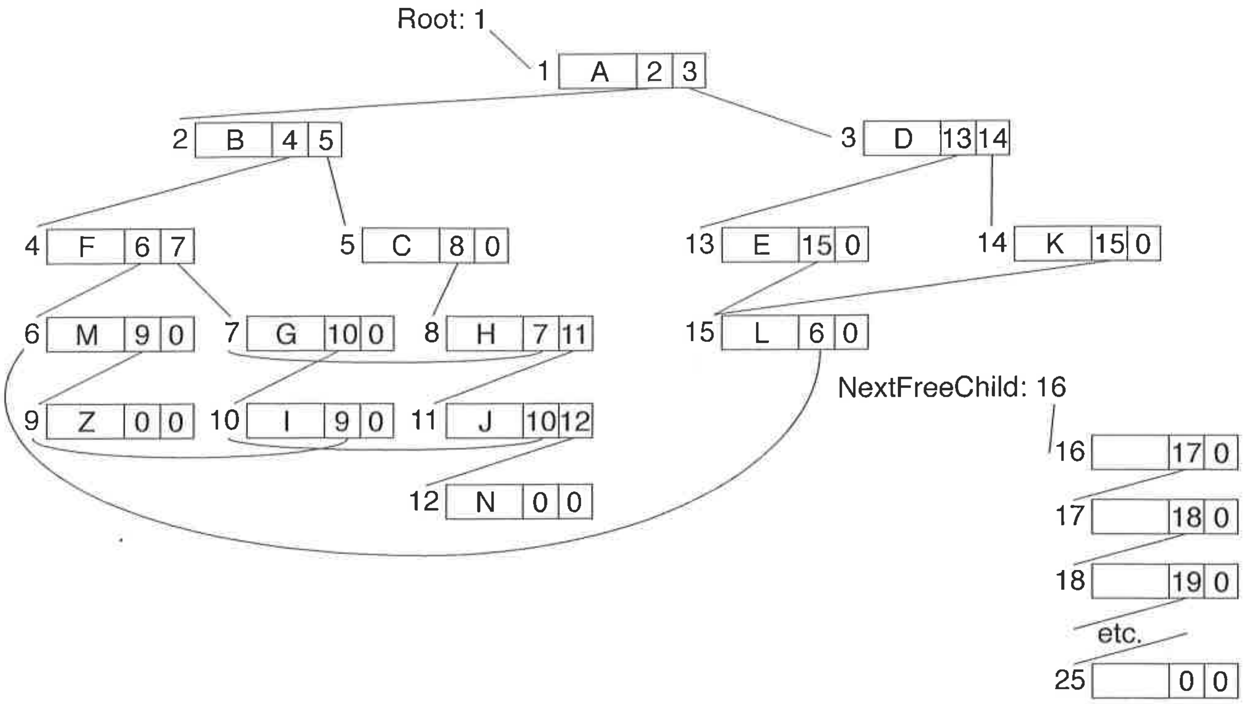
The diagram shows the empty data structure with the linked list to record the unused nodes.



Task 3.1
Write the program code to declare the **empty** data structure and linked list of 25 unused nodes. Add statement(s) to initialise the empty data structure.

Evidence 6
Your program code for Task 3.1. [12]

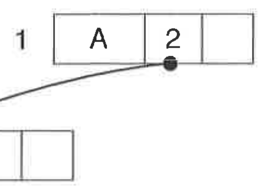
This data structure is used to record the possible routes for a robot to travel from a node A to a node Z. The following data structure illustrates many possible routes, for example, A→D→K→L→M→Z. It is only possible to move to one of two possible nodes; for example, from node A, the only move is to node B or node D.



This data structure has 15 nodes (A to N and Z) but for future development a maximum of 25 nodes is specified. All nodes are unique.

The pseudocode on the next page can be used to add a node to the data structure. The procedure `AddToRobotData` uses the parameters `NewDataItem`, `ParentItem` and `ThisMove`.

The parameter `ThisMove` holds the move made to create this new item ('L' for LeftChild, 'R' for RightChild, 'X' for initial state/root), and the `ParentItem` parameter holds the value of the parent item which points to this `NewDataItem`.



To add node B as shown, the procedure call would be `AddToRobotData('B', 'A', 'L')`. The parameters used would be:

`B`, the new node
`A`, the parent node
`L`, the location of the child (which has an index of 2) is recorded in `LeftChild` of A.

The following pseudocode (available in PSEUDOCODE_TASK_3_2.TXT) can be used to add a node to the data structure.

```
FUNCTION FindNode(NodeValue) RETURNS INTEGER
   Found ← FALSE
   CurrentPosition ← Root
   REPEAT
      IF RobotData[CurrentPosition].DataValue = NodeValue THEN
         Found ← TRUE
      ELSE
         CurrentPosition ← CurrentPosition + 1
      ENDIF
   UNTIL Found = TRUE OR CurrentPosition > 25
   IF CurrentPosition > 25 THEN
      RETURN 0
   ELSE
      RETURN CurrentPosition
   ENDIF
ENDFUNCTION

PROCEDURE AddToRobotData(NewDataItem, ParentItem, ThisMove)
      IF Root = 1 AND NextFreeChild = 1 THEN
         NextFreeChild ← RobotData[NextFreeChild].LeftChild
         RobotData[Root].LeftChild ← 0
         RobotData[Root].DataValue ← NewDataItem
      ELSE
         // does the parent exist?
         ParentPosition ← FindNode(ParentItem)
         IF ParentPosition > 0 THEN // parent exists
            // does the child exist?
            ExistingChild ← FindNode(NewDataItem)
            IF ExistingChild > 0 THEN // child exists
               ChildPointer ← ExistingChild
            ELSE
               ChildPointer ← NextFreeChild
               NextFreeChild ← RobotData[NextFreeChild].LeftChild
               RobotData[ChildPointer].LeftChild ← 0
               RobotData[ChildPointer].DataValue ← NewDataItem
            ENDIF
            IF ThisMove = 'L' THEN
               RobotData[ParentPosition].LeftChild ← ChildPointer
            ELSE
               RobotData[ParentPosition].RightChild ← ChildPointer
            ENDIF
         ENDIF
      ENDIF
ENDPROCEDURE
```

Task 3.2

Write code to implement AddToRobotData and FindNode from this pseudocode.
You may use the text file PSEUDOCODE_TASK_3_2.TXT as a basis for writing your code.

Evidence 7
Your program code for Task 3.2. [7]

Task 3.3

Write a procedure OutputData which displays the value of Root, the value of NextFreeChild and the contents of RobotData in index order.

Evidence 8
Your program code for Task 3.3. [6]

Task 3.4

The file SEARCHTREE.TXT contains the data for the search tree. Each row of the file contains three comma separated values, for example, the first row contains 'A', '0' and 'X'. The file is organised as:

```
NewDataItem,ParentItem,ThisMove
NewDataItem,ParentItem,ThisMove
...
<End of File>
```

There are a total of 20 lines in the SEARCHTREE.TXT file representing possible routes.

Write a main program to read the contents of this file and use AddToRobotData and FindNode to insert these routes into RobotData. Your program will then call the OutputData procedure.

Evidence 9
Your program code for Task 3.4. [6]

Evidence 10
Screenshot showing the output from running the program in Task 3.4. [2]

Task 3.5

Write a recursive pre-order tree traversal that will display all valid routes from A to Z by following the routes described in RobotData.

Evidence 11
Your program code for Task 3.5. [6]

Evidence 12
Screenshot showing the output from running the program in Task 3.5. [1]