Note: The Data Structures is a fairly complicated topic with lots of small things to take care of. As such it is quite difficult not to include these small things as they make quite a difference in operating these data structures. I have tried to keep things as simple as possible, leaving difficult things out where possible. However, you can choose for yourself which portions you want discard and which ones to use in your lectures. One example of this could be the "finding of required Node" in the Deletion section of Linked List. The entire "finding" section could be replaced with:

```
Find the required Node
If Node is found
......
Otherwise
......
Endif
```

Stacks

We need two variables. One for the maximum limit of the Stack (MAX) and the other for keeping track of the current element to be manipulated (Current). We also declare an array for the Stack.

```
MAX = 5
Current = 0
Array Stack[MAX]
```

This will declare an array with 5 elements.

Inserting a Value onto the Stack (Push)

Procedure Push (Value)

If Current is equal to MAX then
  Output an error that the Stack is full
```
This means no more space for data. Stack is full.
```
  Display an error message that there is no more space on the Stack.
Else
    Add 1 to Current
    Put data in Value into current position
Endif
End Procedure

Displaying a value from the Stack (Pop)
-----------------------------------------
Procedure Pop
if Current is equal to Zero then
    Output error that Stack is empty
Else
    Output value from Stack at Current position
    Subtract 1 from Current
Endif
End Procedure

===============================================================================

QUEUES
=======
Prerequisites:
End = 0
MAX = 5
To track the End element of the Queue.
Maximum number of elements (or whatever value you like as the number of elements in the Queue)
Array Queue[MAX]
Declares an array of 5 elements for the Queue.

Adding a Value
---------------
Procedure Insert(Value)
If End is equal to MAX then
    Output an error message that the Queue is full
Otherwise
    Add 1 to End
    Store data in Value into Queue at End position
Endif
End Procedure
Displaying value from Queue
----------------------------------
Procedure Display
If End is equal to Zero then
    'No values to display
    Output an error message that the Queue is empty
Otherwise
    Display value at position Queue[1]
    'Always display first value (FIFO)
    Start Loop from position 1 to End position
        Copy data from Next element to Current element
        'Moving data is step on its left to fill the gap in the 1 position
        Move to Next element
    End Loop
EndIf
End Procedure

===================================================================================================== 
Linked List
=============
Declaring a Linked List
------------------------
Integer : Data
Pointer:  Next
Declaring variables
----------------------
Pointer : Start
Start = NULL
'Nothing in the Linked List

Creating a New Node with above structure.
-----------------------------------------------
Procedure Create(Value)
If there is enough memory available then
    'Only create a Node if memory is available
    Create New Node in memory
    'Creating a New Node for the Linked List
    Point its Next pointer to NULL
    'Making a termination point for the Node
    If this is the first Node in the List then
        Point Start to the Node
        'There must be a starting point for the Linked List
        'Point Start to New Node only if it is the very first one
    End If
End If
Otherwise
  Go to memory location pointed by Start pointer
  Start Loop and run until Next points to NULL
    Go to the Node pointed by Next pointer
    End Loop
  Point Next of last Node to the newly created Node
  Put data in Value in new Node's Data
Endif
Otherwise
  Output error message that there is no more memory available
Endif
End Procedure

Deleting a Node from the Linked List (assuming values in Data are unique)

Procedure Delete(Value)
Pointer : Found
Found = NULL

If there are no Nodes in the List then
  Output error message that can't delete a Node
Otherwise
  Go to memory location pointed by Start pointer
  Start Loop and run until Next points to NULL
    If value in Data of current Node is equal to Value then
      Found = current Node
      Exit Loop
    Endif
  End Loop
  If the required Node is the first Node in the List then
    Point the Start pointer to NULL
  Otherwise
    Point Next of previous Node to the Node pointed by Next of Found
Endif
Endif
End Procedure
Displaying the entire Linked List
------------------------------------------
Procedure Display
  If Start points to NULL then
    'List is empty
    Output error message that the List is empty
  Otherwise
    Start Loop at Start position and run until Next points to NULL
    Output current Node's Data
    Go to Node pointed by Next pointer of current Node
  End Loop
Endif
End Procedure

====================================================================================================
Binary Trees
=============

Structure of Binary Tree Node

Integer : Data
 Pointer : Left
 Pointer : Right
 Pointer : Start
 Pointer : Current

Start = NULL
Current = NULL

Insertion in a Binary Tree
-----------------------------
Procedure Insertion(Value)
  Create a New Node
  Point its Left and Right pointers to NULL
  If Start is pointing to NULL then
    'Tree was empty so insert at Start
    Point Start to the New Node
  Otherwise
    'Search for appropriate position
    Current = Start
    Start Loop at Current and run until Left and Right pointers are NULL
  Endif
End Procedure
If data in Value is less than or equal to Data in Current Node then
   Go to the left of the Current Node
Otherwise
   Go to the right of the Current Node
Endif
If data in New Node is less than data in Current Node then
   Point the Left pointer of Current Node to the New Node
Otherwise
   Point the Right pointer of Current Node to the New Node
Endif
Insert data in Value in the New Node
End Loop
Endif
End Procedure

Traversals of Binary Tree
--------------------------

Procedure InOrder(Current)
If Left pointer of Current is not NULL then
   Call InOrder with Left pointer of Current
   This is a recursive call from InOrder to itself
Endif
Output Data in Current Node
If Right pointer of Current is not NULL then
   Call InOrder with Right pointer of Current
   This is a recursive call from InOrder to itself
Endif
End Procedure

Procedure PreOrder(Current)
Output Data in Current Node
If Left pointer of Current is not NULL then
   Call InOrder with Left pointer of Current
   This is a recursive call from PreOrder to itself
Endif
If Right pointer of Current is not NULL then
   Call InOrder with Right pointer of Current
   This is a recursive call from PreOrder to itself
Endif
End Procedure
Procedure PostOrder(Current)
If Left pointer of Current is not NULL then
    Call InOrder with Left pointer of Current
    'This is a recursive call from PostOrder to itself
Endif
If Right pointer of Current is not NULL then
    Call InOrder with Right pointer of Current
    'This is a recursive call from PostOrder to itself
Endif
Output Data in Current Node
End Procedure