==========

DATA STRUCTURE
PSEUDO-CODE EXAMPLES

(c) Mubashir N. Mir - www.mubashirnabi.com

==========

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     'This means no more space for data. Stack is full.
  Output an error that the Stack is full   'Display an error message that there is no more space on the Stack.
```

```
Else
  Add 1 to Current                                  'Go to the next available element in the Stack.
  Put data in Value into current position           'Put (Push) the data sent in Value parameter in the new element.
Endif
End Procedure
```

Displaying a value from the Stack (Pop)

```
----------------------------------------
Procedure Pop
if Current is equal to Zero then                    'If there are no values in the Stack.
  Output error that Stack is empty                  'Nothing to output.
Else
  Output value from Stack at Current position       'Output (POP) a value in the Current element (LIFO).
  Subtract 1 from Current
Endif
End Procedure
```

==========

QUEUES

=======

Prerequsites:

```
End = 0                                             To track the End element of the Queue.
MAX = 5                                             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                         'No more space in Queue
  Output an error message that the Queue is full
Otherwise
  Add 1 to End                                      'Go to next available element in Queue
  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 'Looping through all elements with values
    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 'There must be a starting point for the Linked List
    Point Start to the Node 'Point Start to New Node only if it is the very first one

```
  Otherwise                                                    'If other Nodes exist then find the last Node in the List
    Go to memory location pointed by Start pointer             'Begin at the Start (first Node)
    Start Loop and run until Next points to NULL               'Check each Node until Next pointer points to NULL
      Go to the Node pointed by Next pointer                   'Move from Node to Node
    End Loop
    Point Next of last Node to the newly created Node          'When last Node found, point its Next pointer to New Node
    Put data in Value in new Node's Data                       'Put the required data (Value) in New Node's Data variable
  Endif
Otherwise                                                      'Out of memory. Can't create a New Node
  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                                                'Pointer for finding required Node to Delete
Found = NULL                                                   'Assuming required Node not found

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             'Begin at the Start (first Node)
    Start Loop and run until Next points to NULL               'Check each Node until Next pointer points to NULL
      If value in Data of current Node is equal to Value then  'If we find our required Node
        Found = current Node                                   'Mark the current Node with Found pointer
        Exit Loop                                              'No need to finish the loop
      Endif
      Go to Node pointed by Next pointer of current Node
    End Loop
    If the required Node is the first Node in the List then
      Point the Start pointer to NULL                          'Node deleted
    Otherwise
      Point Next of previous Node to the Node pointed by Next of Found 'Node deleted (Previous.Next = Found.Next)
    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
  Point Start to the New Node                                       'Tree was empty so insert at Start
Otherwise
  Current = Start                                                   'Search for appropriate position
  Start Loop at Current and run until Left and Right pointers are NULL
```

```
    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
```

==========