Data Structures

Dynamic Data Structures
- Arrays - fixed size
- Dynamic structures
  - linked lists, stacks, trees, networks, collections, dictionaries
  - size changes during execution

Linked Lists
- A linear collection of self-referential class objects, called nodes, connected by reference links

Self-Referential Class - Linked Lists
- CListNode Class definition
  - Private mNodeData As Variant ' serve as a node to store data
  - Private mNextNode As CListNode ' serve as a link
  - Public Property Get Data () As Variant
    Data = mNodeData
  - End Property
  - Public Property Let Data (ByVal vNewValue As Variant)
    mNodeData = vNewValue
  - End Property
  - Public Property Get NextNode () As CListNode
    Set NextNode = mNextNode
  - End Property
  - Public Property Let NextNode (ByVal vNewNode As CListNode)
    Set mNextNode = vNewNode
  - End Property

Advantages and Disadvantages of Linked Lists
- Useful when number of data items is unpredictable
- inserting data into a list is fast
- references incur overhead
- access of data is not as direct as using index of an array

Operations on Linked Lists
- Check if a list is empty
- Insert a data item
  - at beginning of list
  - at end of list
- delete
  - at beginning of list
  - at end of list
Defining a Linked List - Class CList - and Related Operations
Defining Related Operations
Defining Related Operations (cont.)
Test Program
Delete Nth element (form procedure)
Delete nth element (CLlist function)

Public Function removeNth(n As Integer)
Dim nextNode, nthNode, preNode As CListNode
Dim i As Integer
Set preNode = mFirstNode
i = 0
Do Until i = n - 1 Or preNode is Nothing 'find the node before the nth node
Set preNode = preNode.nextNode
i = i + 1
Loop
If preNode Is Nothing Then
Call MsgBox("The list does not have "+Str(n)+" item(s).")
Else
Set nthNode = preNode.nextNode 'find the nth node
If nthNode Is Nothing Then
Call MsgBox("The list does not have "+n+" item(s).")
Else
Set nextNode = nthNode.nextNode 'find the node following nth node
preNode.nextNode = nextNode 'connect preNode with NextNode
Set nthNode = Nothing 'remove nth node
End If
End If
End Function

Stack

• First In Last Out (FILO)
• Linked list
• Insert at front (Push)
• Read data from front
• Delete at front (Pop)

Clist definition (as before)

Fig. 21.3
Class CList
Option Explicit
Private mFirstNode As CListNode 'refers to first node in list
Private mLastNode As CListNode 'refers to last node in list
Public Function IsEmpty() As Boolean
Public Sub InsertAtFront(insertItem As Variant)
Public Sub InsertAtBack(insertItem As Variant)
Public Function RemoveFromFront()
Public Function RemoveFromBack()
Public Property Get First() As CListNode
Set First = mFirstNode
End Property

Stack Class Definition

Fig. 21.9
Class CStack
Option Explicit
Private list As New CList
Public Sub Push(value As Variant)
list.InsertAtFront(value)
End Sub
Public Function Pop() As Variant
Pop = list.RemoveFromFront()
End Function
Public Function IsStackEmpty() As Boolean
IsStackEmpty = list.IsEmpty()
End Function
Public Property Get First() As CListNode
Set First = list.First
End Property

Queue

• First In First Out (FIFO)
• Linked list
• Insert at end (Enqueue)
• Read data from front
• Delete at front (Dequeue)
Tree definition

```
' Fig. 21.15
' Class CTreeNode
Option Explicit
Private mRoot As CTreeNode
Private mLeft As CTreeNode
Private mRight As CTreeNode
Public Property Get mRoot() As CTreeNode
    Set mRoot = mRoot
End Property
Public Property Let mRoot(ByVal vNewValue As CTreeNode)
    Set mRoot = vNewValue
End Property
Public Property Get mLeft() As CTreeNode
    Set mLeft = mLeft
End Property
Public Property Let mLeft(ByVal vNewValue As CTreeNode)
    Set mLeft = vNewValue
End Property
Public Property Get mRight() As CTreeNode
    Set mRight = mRight
End Property
Public Property Let mRight(ByVal vNewValue As CTreeNode)
    Set mRight = vNewValue
End Property
```

Tree traversal

- Traversal of a tree - visit all elements in a tree in a predefined order

- Traversal orders
  - Preorder - visit root, traverse left subtree, traverse right subtree
  - In order - traverse left subtree, visit root, traverse right subtree
  - Post order - traverse left subtree, traverse right subtree, visit root

Preorder traversal - 47, 25, 11, 7, 17, 31, 44, 77, 65, 68, 93

In order traversal - 7, 11, 17, 25, 31, 44, 47, 65, 68, 77, 93

Post order traversal - 7, 17, 11, 31, 44, 43, 25, 68, 65, 93, 77, 47
1. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

2. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

3. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

4. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

5. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

6. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

7. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

8. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

9. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

10. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

11. \(-17 \times (31 + 44) + \sqrt{68} / 93\)

\[\text{Result} = -13.621\]

**Implementation ideas**

- Use a binary tree
- Number, operator sequence - operator as root, number as left branch
- operator, number sequence - operator as root, number as right branch
- when a tree is complete (left branch can be empty if operator is an uni-operator) and there is no parenthesis before the last number, perform the operation, and read the next input.

---

**Collection Object**

- Predefined in VB
- elements of a collection can be of any type
- Manipulations
  - Add - for adding a member to a collection
  - Remove - for deleting an item
  - Item - for retrieval of an item (using an index)
  - Count - for determining size of a collection
- members can be reference by index
- can be processed as an array

---

**Dictionary**

- For storing keyed data in a table and retrieve them by their keys
- Data stored in key-value pairs
- Data retrieved using keys
- Dictionary methods
  - aDictionary.Exists(key) - returns True or False
  - aDictionary.Add(key, value)
  - aDictionary.Remove(key)
  - aDictionary.RemoveAll
  - aDictionary.Item(key) - returns Value if key exists