Syllabus Content:

10.4 Introduction to Abstract Data Types (ADT)

Candidates should be able to:

- Show understanding that an ADT is a collection of data and a set of operations on those data.
- Show understanding that a stack, queue and linked list are examples of ADTs
- Describe the key features of a stack, queue and linked list and justify their use for a given situation
- Use a stack, queue and linked list to store data
- Candidates will not be required to write pseudocode for these structures, but they should be able to add, edit and delete data from these structures
- Describe how a queue, stack and linked list can be implemented using arrays.

ADTs (Abstract Data Type):

An **abstract data type** is a collection of data. When we want to use an abstract data type, we need a set of basic operations:

- create a new instance of the data structure
- find an element in the data structure
- insert a new element into the data structure
- delete an element from the data structure
- access all elements stored in the data structure in a systematic manner.

KEY TERMS

Abstract data type: a collection of data with associated operations

Abstract Data Types

Definition

An abstract data type is a type with associated operations, but whose representation is hidden.

The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.

It is called "abstract" because it gives an implementation independent view. The process of providing only the essentials and hiding the details is known as abstraction.

The user of data type need not know that data type is implemented, for example, we have been using **integer**, **float**, **char** data types only with the knowledge with values that can take and operations that can be performed on them without any idea of how these types are implemented. So a user only needs to know what a data type can do but not how it will do it.

Contact: 03004003666
Email: majidtahir61@gmail.com

We can think of ADT as a black box which hides the inner structure and design of the data type.

Now we'll define three **ADTs** namely **Stack** ADT, **Queue** ADT and **Linked List ADT**.

Stack ADT

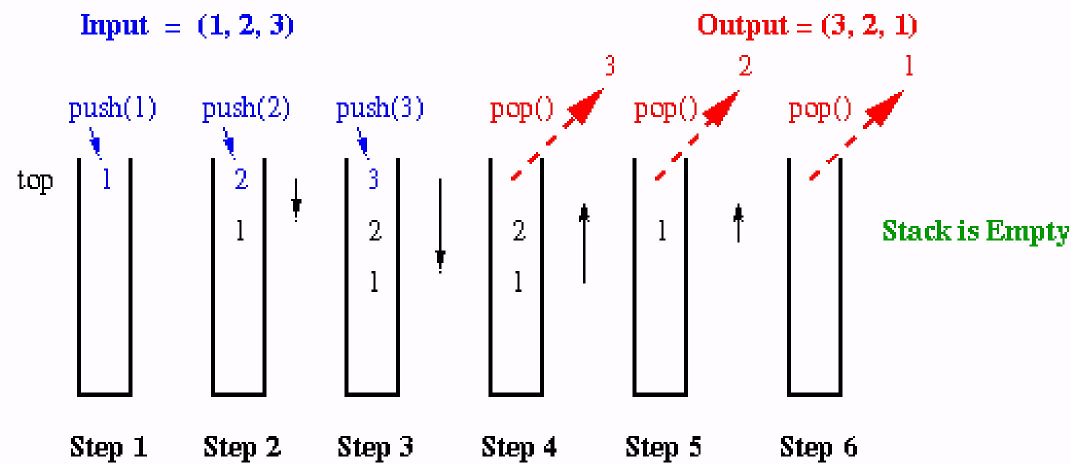
A Stack contains elements of same type arranged in sequential order. All operations takes place at a single end that is top of the stack and following operations can be performed:

To make a stack, we pile items on top of each other. The item that is accessible is the one on top of the stack. If we try to find an item in the stack and take it out, we are likely to cause the pile of items to collapse.

The **BaseofstackPointer** will always point to the first slot in the stack. The **TopOfStackPointer** will point to the last element pushed onto the stack.

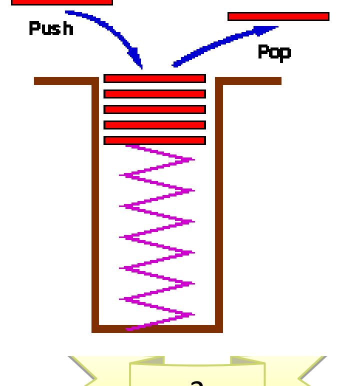
When an element is removed from the stack, the **TopOfStackPointer** will decrease to point to the element now at the top of the stack.

Figure below shows how we can represent a stack when we have added three items in this order: 1, 2, 3 push() adds the item in stack and pop() picks the item from stack.



The **'STACK'** is a **Last-In First-Out (LIFO)** List. Only the last item in the stack can be accessed directly.

push() – Insert an element at one end of the stack called top.
pop() – Remove and return the element at the top of the stack, if it is not empty.
peek() – Return the element at the top of the stack without removing it, if the stack is not empty.
size() – Return the number of elements in the stack.
isEmpty() – Return true if the stack is empty, otherwise return false.
isFull() – Return true if the stack is full, otherwise return false.



Contact: 03004003666
Email: majidtahir61@gmail.com

To Setup a Stack in **pseudocode** see the code below:

To Setup a Stack in **pseudocode** see the code below:

```
DECLARE stack Array[1-10] : INTEGER
DECLARE TopPointer : INTEGER
DECLARE BasePointer : INTEGER
DECLARE StackFull : INTEGER

         BasePointer = 1
         TopPointer = 0
         StackFull = 10
```

To add an item in STACK

To **Push** an item in a Stack, see the **pseudocode**

```
IF TopPointer < StackFull
   THEN
         Stack[TopPinter] = item
         TopPointer = TopPointer + 1
   ELSE
       OUTPUT("Stack is Full, can not push")
END IF
```

To remove an item from STACK

```
IF TopPointer = BasePointer - 1 //Remember BasePointer = 1 in start
   THEN
        OUTPUT("Stack is empty, can not pop")

   ELSE
        item = Stack[TopPinter]
        TopPointer = TopPointer - 1

END IF
```

Stacks in VB

```
Public Dim stack() As Integer = {Nothing, Nothing, Nothing, Nothing,
Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing}
Public Dim basePointer As Integer = 0
Public Dim topPointer As Integer = -1
Public Const stackFull As Integer = 10
Public Dim item As Integer
```

Stack Pop Operation

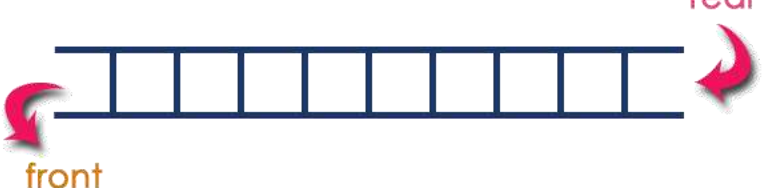
`topPointer` points to the top of stack

```
Sub pop()
        If topPointer = basePointer - 1 Then
                Console.WriteLine("Stack is empty, cannot pop")
        Else
                item = stack(topPointer)
                topPointer = topPointer - 1
        End If
End Sub
```

Stack Push Operation

```
Sub push(ByVal item)
        If topPointer < stackFull - 1 Then
        topPointer = topPointer + 1
                stack(topPointer) = item
        Else
                Console.WriteLine("Stack is full, cannot push")
        End if
End Sub
```

Queue ADT



Contact: 03004003666
Email: majidtahir61@gmail.com

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing of elements are performed at two different positions.

The insertion is performed at one end and deletion is performed at other end. In a queue data structure, the insertion operation is performed at a position which is known as '**rear**' and the deletion operation is performed at a position which is known as '**front**'.

In queue data structure, the insertion and deletion operations are performed based on **FIFO (First In First Out)** principle.
A Queue contains elements of same type arranged in sequential order. Operations takes place at both ends, insertion is done at end and deletion is done at front. Following operations can be performed:

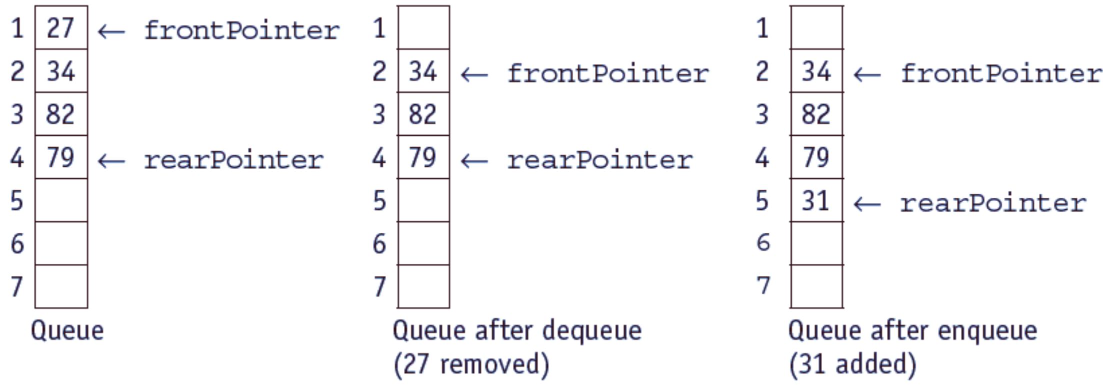
enqueue() – Insert an element at the end of the queue.
dequeue() – Remove and return the first element of queue, if the queue is not empty.
peek() – Return the element of the queue without removing it, if the queue is not empty.
size() – Return the number of elements in the queue.
isEmpty() – Return true if the queue is empty, otherwise return false.
isFull() – Return true if the queue is full, otherwise return false.

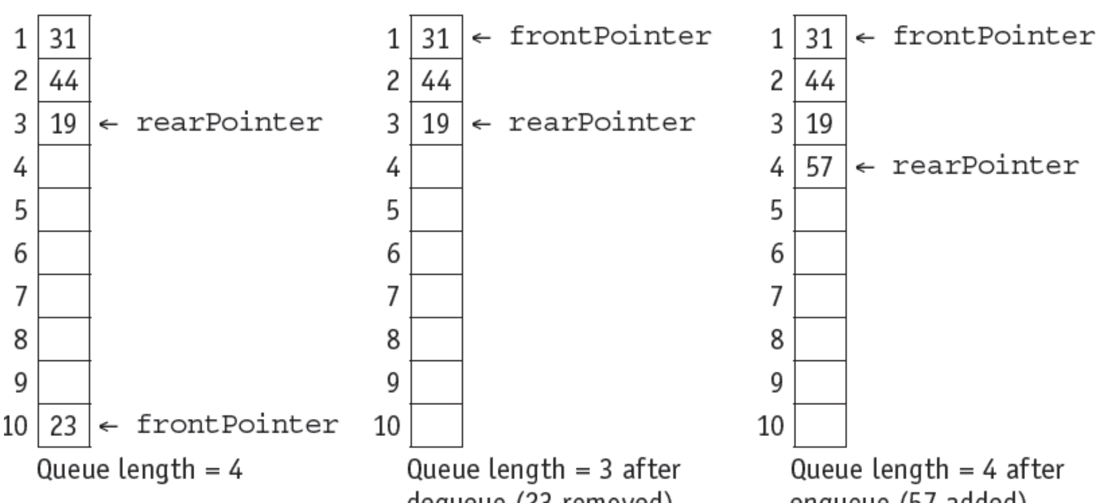
Queue after inserting 25, 30, 51, 60 and 85.

From these definitions, we can clearly see that the definitions do not specify how these ADTs will be represented and how the operations will be carried out. There can be different ways to implement an ADT, for example, the List ADT can be implemented using arrays, or singly linked list or doubly linked list. Similarly, stack ADT and Queue ADT can be implemented using arrays or linked lists.

Comp

The value of the frontPointer changes after dequeue but the value of the rearPointer changes after enqueue:





To set up a queue

```
DECLARE queue ARRAY[1:10] OF INTEGER
DECLARE rearPointer : INTEGER
DECLARE frontPointer : INTEGER
DECLARE queueful : INTEGER
DECLARE queueLength : INTEGER
frontPointer ← 1
endPointer ← 0
upperBound ← 10
queueful ← 10
queueLength ← 0
```

To add an item, stored in `item`, onto a queue

```
IF queueLength < queueful
  THEN
    IF rearPointer < upperBound
      THEN
        rearPointer ← rearPointer + 1
      ELSE
        rearPointer ← 1
    ENDIF
    queueLength ← queueLength + 1
    queue[rearPointer] ← item
  ELSE
    OUTPUT "Queue is full, cannot enqueue"
ENDIF
```

To remove an item from the queue and store in `item`

```
IF queueLength = 0
  THEN
    OUTPUT "Queue is empty, cannot dequeue"
  ELSE
    Item ← queue[frontPointer]
    IF frontPointer = upperBound
      THEN
        frontPointer ← 1
      ELSE
        frontPointer ← frontPointer + 1
    ENDIF
    queueLength ← queueLength - 1
ENDIF
```

Queue Operations in VB:

Contact: 03004003666
Email: majidtahir61@gmail.com

Empty Queue with no items and variables, set to public for subroutine access.

```
Public Dim queue() As Integer = {Nothing,  Nothing, Nothing, Nothing,
Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing}
Public Dim frontPointer As Integer = 0
Public Dim rearPointer As Integer = -1
Public Const queueFull As Integer = 10
Public Dim queueLength As Integer =  0
Public Dim item As Integer
```

Queue Enqueue (adding an item to queue)

```
Sub enQueue(ByVal item)
    If queueLength < queueFull Then
        If rearPointer < queue.length - 1 Then
            rearPointer = rearPointer + 1
        Else
            rearPointer = 0
        End If
        queueLength = queueLength + 1
        queue(rearPointer) = item
    Else
        Console.WriteLine("Queue is full, cannot enqueue")
    End If
End Sub
```

Queue Enqueue (adding an item to queue)

```
Sub deQueue()
    If queueLength = 0 Then
        Console.WriteLine("Queue is empty, cannot dequeue")
    Else
        item = queue(frontPointer)
        If frontPointer = queue.length - 1 Then
            frontPointer = 0
        Else
            frontPointer = frontPointer + 1
        End If
        queueLength = queueLength - 1
    End If
End Sub
```

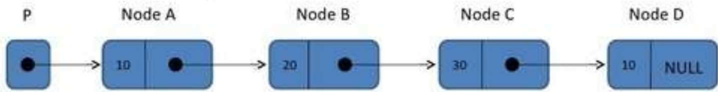
Linked lists

Earlier we used an **array** as a linear list. In an **Array** (Linear list), the list items are stored in consecutive locations. This is not always appropriate.

Another method is to store an individual list item in whatever location is available and link the individual item into an ordered sequence using pointers.

Linked List

- A list implemented by each item having a link to the next item.
- Head points to the first node.
- Last node points to NULL.



An element of a list is called a **node.** A node can consist of several data items and a **pointer,** which is a variable that stores the address of the node it points to.
A pointer that does not point at anything is called a **null pointer.** It is usually rep resented by ɸ. A variable that stores the address of the first element is called a **start pointer.**

KEY TERMS

Node: an element of a list

Pointer: a variable that stores the address of the node it points to

Null pointer: a pointer that does not point at anything

Start pointer: a variable that stores the address of the first element of a linked list

In Figure below, the data value in the node box represents the key field of that node. There are likely to be many data items associated with each node. The arrows represent the pointers.

It does not show at which address a node is stored, so the diagram does not give the value of the pointer, only where it conceptually links to.
Suppose **StartPointer points** to **B**, **B points** to **D** and **D points** to **L, L Points** to **NULL**

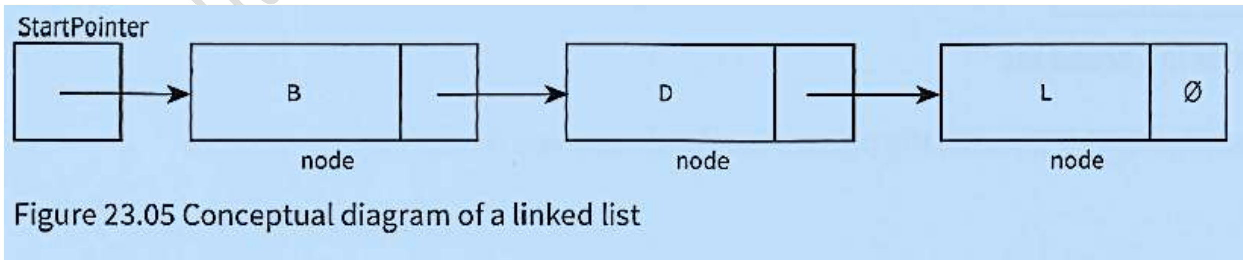


Figure 23.05 Conceptual diagram of a linked list

Add a node at the front: (A 4 steps process)

A new node, **A**, is inserted at the beginning of the list.
The content of **startPointer** is copied into the new node's pointer field and **startpointer** is set to point to the new node, **A**.

Contact: 03004003666
Email: majidtahir61@gmail.com

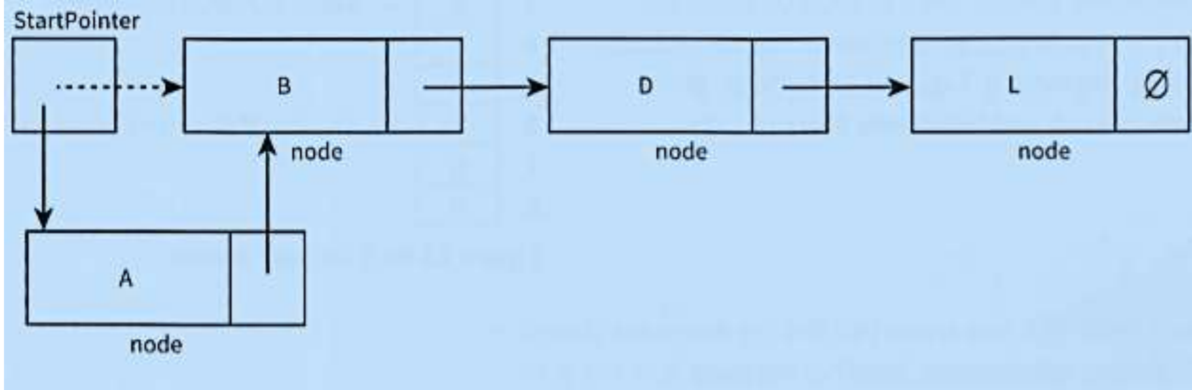


Figure 23.06 Conceptual diagram of adding a new node to the beginning of a linked list

Add a node after a given node:

We are given pointer to a node, and the new node is inserted after the given node.

To insert a new node, **C**, between existing nodes, Band D (Figure 23.10), we copy the pointer field of node **B** into the pointer field of the new node, **C**. We change the pointer field of node B to point to the new node, **C**.

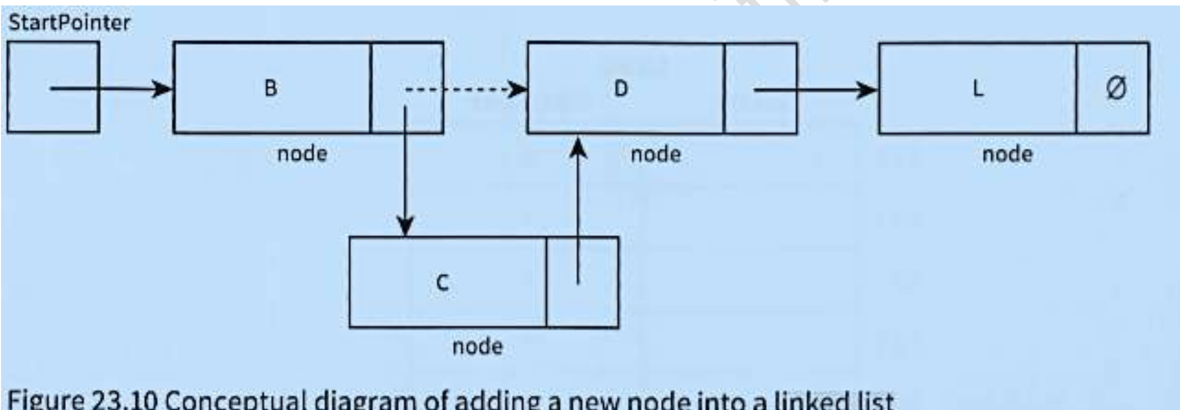


Figure 23.10 Conceptual diagram of adding a new node into a linked list

Add a node at the end:

In Figure 23.07, a new node, **P**, is inserted at the end of the list. The pointer field of node L points to the new node, **P**. The pointer field of the new node, P, contains the null pointer.

Computer Science with Majid

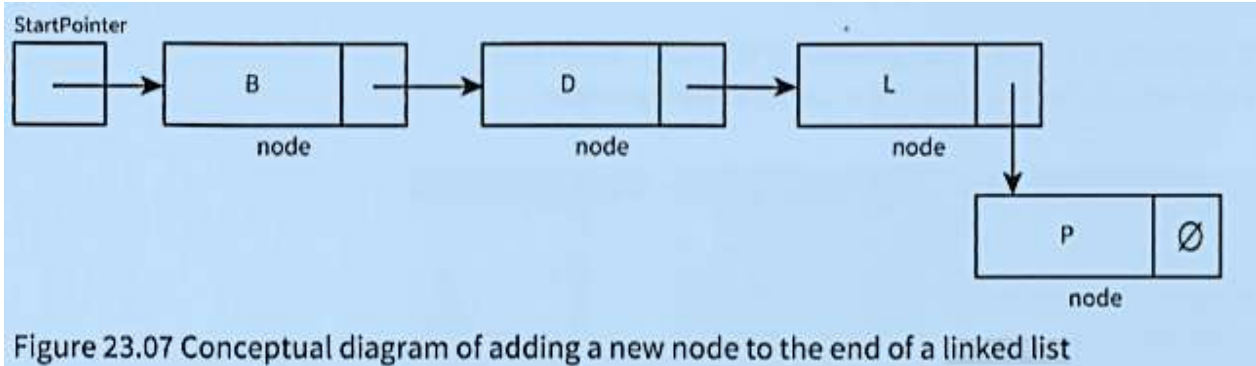


Figure 23.07 Conceptual diagram of adding a new node to the end of a linked list

Deleting the First node in the list:

To delete the first node in the list (Figure 23.08), we copy the pointer field of the node to be deleted into **StartPointer**

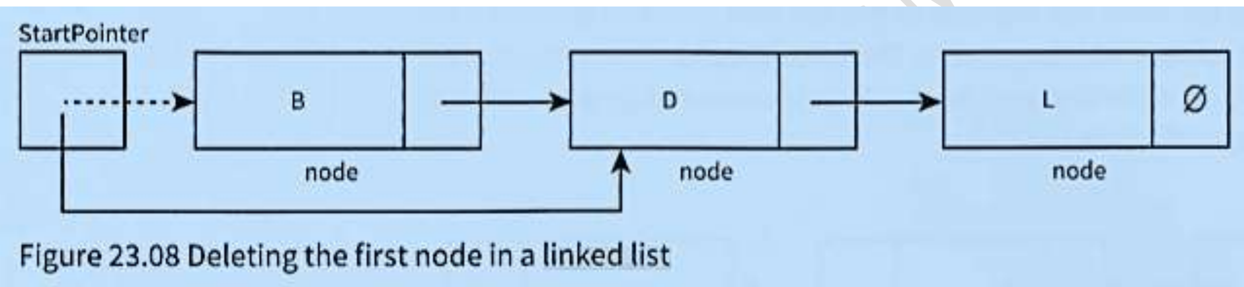


Figure 23.08 Deleting the first node in a linked list

Deleting the Last node in the list:

To delete the last node in the list (Figure 23.09), we set the pointer field for the previous node to the null pointer.

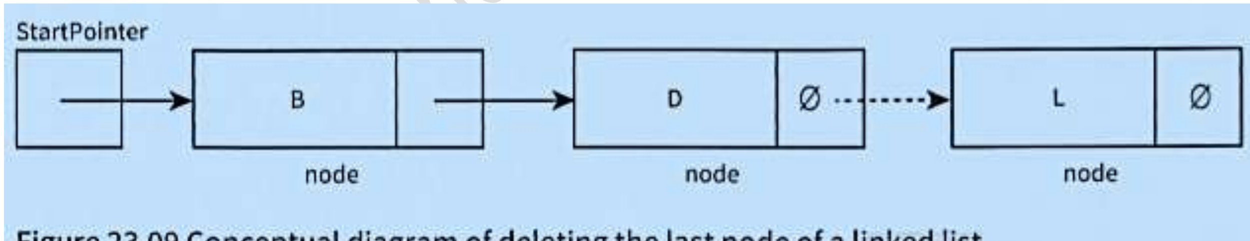


Figure 23.09 Conceptual diagram of deleting the last node of a linked list

Deleting a node within the list:

To delete a node, D, within the list (Figure 23.11), we copy the pointer field of the node to be deleted, D, into the pointer field of node B.

Contact: 03004003666
Email: majidtahir61@gmail.com

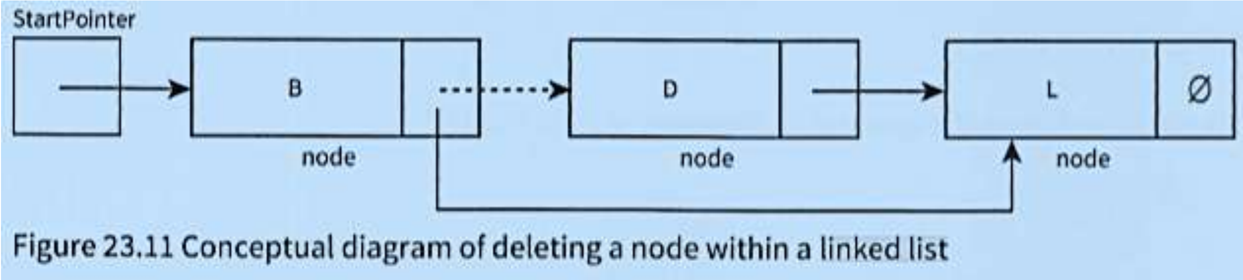


Figure 23.11 Conceptual diagram of deleting a node within a linked list

- Remember that, in real applications, the data would consist of much more than a **key field** and one **data item**.
- When list elements need reordering, only pointers need changing in a linked list. In an **Array** (**linear list**), all data items would need to be moved.
- This is why linked lists are preferable to **Arrays** (linear lists).
- **Linked lists** saves time, however we need more storage space for the **pointer fields**.

Using Linked Lists:

- We can store the linked list in an array of records. One **record** represents a **node** and consists of the **data and a pointer**.
- When a node is **inserted** or **deleted**, only the **pointers need to change**. A pointer value is the **array index** of the node pointed to.
- Unused nodes need to be easy to find.
- A suitable technique is to **link the unused nodes** to form another linked list: the **free list**. Figure 23.12 shows our **linked list** and its **free list**.

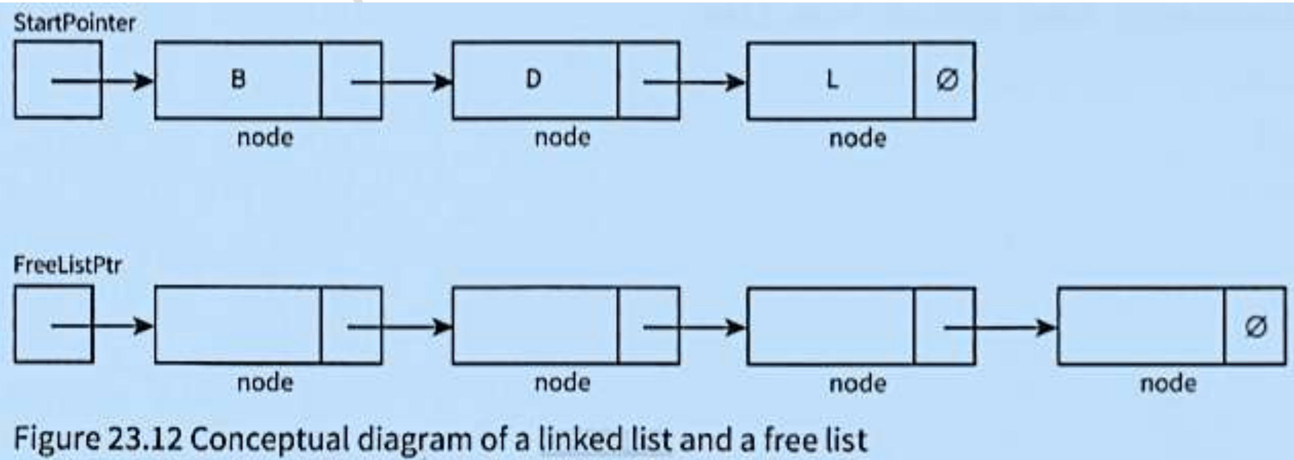


Figure 23.12 Conceptual diagram of a linked list and a free list

Science (9608) with Sir Majid Tahir

- When an array of nodes is first **initialised** to work as a linked list, the **linked list will be empty.**
- So the **start pointer** will be the **null pointer**.
- All nodes need to be **linked to form the free list**.
- Figure 23.13 shows an example of an implementation of a linked list before any data is inserted into it.

StartPointer: Ø

FreeListPtr: 1

List

	Data	Pointer
[1]		2
[2]		3
[3]		4
[4]		5
[5]		6
[6]		7
[7]		Ø

Figure 23.13 A linked list before any nodes are used

We now code the basic operations discussed using the conceptual diagrams in Figures 23.05 to 23.12.

Create a new linked list

```
CONSTANT NullPointer=0 //NullPointer should be set to -1 if using array element with index 0
1
TYPE ListNode       // Declare record type to store data and pointer
DECLARE Data STRING
DECLARE Pointer INTEGER
ENDTYPE

DECLARE StartPointer : INTEGER // Declare start pointer to point to first item in list
DECLARE FreeListPtr : INTEGER // Declare free pointer to add data in free memory slot.
DECLARE List[1:7] OF ListNode

PROCEDURE InitialiseList
      StartPointer ← NullPointer       // set start pointer, start of list
      FreeListPtr ← 1                  // set starting position of free list
      FOR Index ← 1 TO 6               // link all nodes to make free list
            List[Index].Pointer ← Index + 1
      NEXT
      List[7].Pointer ← Null Pointer      //last node of free list
END PROCEDURE
```

Create a new linked list in Visual Studio

```
Module Module1
  ' NullPointer should be set to -1 if using array element with index 0
  Const NULLPOINTER = -1 ' Declare record type to store data and pointer

    Structure ListNode
        Dim Data As String
        Dim Pointer As Integer
    End Structure

    Dim List(7) As ListNode
    Dim StartPointer As Integer
    Dim FreeListPtr As Integer

    Sub InitialiseList()
        StartPointer = NULLPOINTER    ' set start pointer
        FreeListPtr = 0                ' set starting position of free list
        For Index = 0 To 7             'link all nodes to make free list
            List(Index).Pointer = Index + 1
        Next
        List(7).Pointer = NULLPOINTER       'last node of free list
    End Sub
```

Insert a new node into an ordered linked list

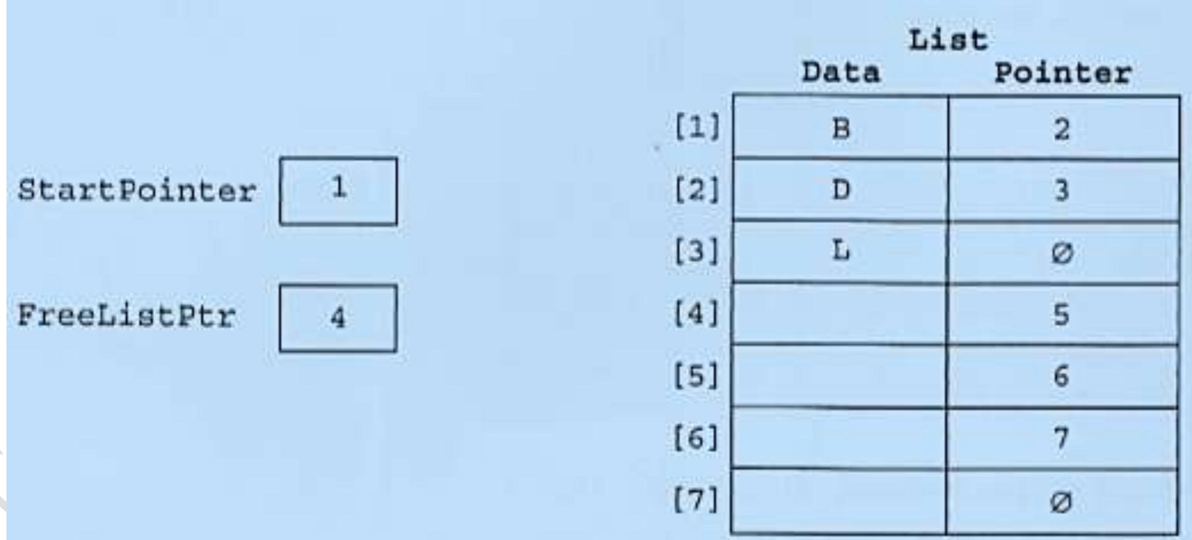


Figure 23.14 Linked list of three nodes and free list of four nodes

(9608) with Sir Majid Tahir

Here is the identifier table.

Identifier	Description
startPointer	Start of the linked list
heapStartPointer	Start of the heap
nullPointer	Null pointer set to -1
itemAdd	Item to add to the list
tempPointer	Temporary pointer

The algorithm to insert an item in the linked list myLinkedList could be written as a procedure in pseudocode as shown below.

```
DECLARE itemAdd : INTEGER
DECLARE startPointer : INTEGER
DECLARE heapstartPointer : INTEGER
DECLARE tempPointer : INTEGER
CONSTANT nullPointer = -1
PROCEDURE linkedListAdd(itemAdd)
  // check for list full
  IF heapStartPointer = nullPointer
    THEN
      OUTPUT "Linked list full"
    ELSE
      // get next place in list from the heap
      tempPointer ← startPointer // keep old start pointer
      startPointer ← heapStartPointer  // set start pointer to next position in heap
      heapStartPointer ← myLinkedListPointers[heapStartPointer] // reset heap start pointer
      myLinkedList[startPointer] ← itemAdd // put item in list
      myLinkedListPointers[startPointer] ← tempPointer  // update linked list pointer
  ENDIF
ENDPROCEDURE
```

Insert a new node into an ordered linked list

```
DECLARE startpointer : INTEGER
DECLARE heapStartPointer : INTEGER
DECLARE itemAdd : INTEGER
DECLARE tempPointer : INTEGER

CONSTANT nullPointer = -1
PROCEDURE
PROCEDURE InsertNode(Newitem)
      IF FreeListPtr <> NullPointer
      THEN // there is space in the array
            NewNodePtr ← FreeListPtr //take node from free list and store data item
            List[NewNodePtr].Data ← Newitem
```

```
                FreeListPtr ← List[FreeListPtr].Pointer
                // find insertion point
                ThisNodePtr ← StartPointer // start at beginning of list

                WHILE ThisNodePtr <> NullPointer // while not end of list
                AND List[ThisNodePtr].Data < NewItem
                PreviousNodePtr ← ThisNodePtr //remember this node
                                //follow the pointer to the next node
                ThisNodePtr ← List[ThisNodePtr].Pointer
                ENDWHILE

                IF PreviousNodePtr = StartPointer
                    THEN //insert new node at start of list
                    List[NewNodePtr].Pointer ← StartPointer
                    StartPointer ← NewNodePtr
                    ELSE //insert new node between previous node and this node
                    List[NewNodePtr].Pointer ← List[PreviousNodePtr].Pointer
                    List[PreviousNodePtr].Pointer ← NewNodePtr
                ENDIF
        ENDIF
END PROCEDURE
```

After three data items have been added to the linked list, the array contents are as shown in Figure 23.14.

StartPointer: 1

FreeListPtr: 4

List	Data	Pointer
[1]	B	2
[2]	D	3
[3]	L	Ø
[4]		5
[5]		6
[6]		7
[7]		Ø

Figure 23.14 Linked list of three nodes and free list of four nodes

Insert a new node into an ordered linked list in Visual Studio:

```
Sub InsertNode(ByVal NewItem)
```

Contact: 03004003666
Email: majidtahir61@gmail.com

```
Dim TempPtr, NewNodePtr, PreviousNodePtr As Integer ' TemportatryPointer, NextNode
Pointer and PreviousPointer to Swap values of pointers
   If FreeListPtr <> NULLPOINTER Then   ' there is space in the array, take node from
free list and store data item
            NewNodePtr = FreeListPtr
            List(NewNodePtr).Data = NewItem
            FreeListPtr = List(FreeListPtr).Pointer ' find insertion point
            PreviousNodePtr = NULLPOINTER
            TempPtr = StartPointer ' start at beginning of list
            Try
                Do While (TempPtr <> NULLPOINTER) And (List(TempPtr).Data < NewItem) '
while not end of list
                    PreviousNodePtr = TempPtr ' remember this node follow the pointer to
the next node
                    TempPtr = List(TempPtr).Pointer
                Loop
            Catch ex As Exception
            End Try
            If PreviousNodePtr = NULLPOINTER Then ' insert new node at start of list

                List(NewNodePtr).Pointer = StartPointer
                StartPointer = NewNodePtr

            Else : List(NewNodePtr).Pointer = List(PreviousNodePtr).Pointer  ' insert new
node between previous node and this node
                List(PreviousNodePtr).Pointer = NewNodePtr
            End If
        Else : Console.WriteLine("no space for more data")
        End If
    End Sub
```

Find an element in an ordered linked list

```
FUNCTION FindNode(Dataitem) RETURNS INTEGER // returns pointer to node
       CurrentNodePtr ⟵ StartPointer //start at beginning of list
       WHILE CurrentNodePtr <> NullPointer //not end of list
       AND List[CurrentNodePtr].Data <> Dataitem // item not found
       //follow the pointer to the next node
       CurrentNodePtr ⟵ List [CurrentNodePtr].Pointer
       ENDWHILE
RETURN CurrentNodePtr // returns NullPointer if item not found
END FUNCTION
```

Finding an element Visual Studio Code:

```
Function FindNode(ByVal DataItem) As Integer ' returns pointer to node
Dim CurrentNodePtr As Integer
CurrentNodePtr = StartPointer  ' start at beginning of list

Try
Do While CurrentNodePtr <> NULLPOINTER And List(CurrentNodePtr).Data <>
DataItem ' not end of list,item(Not found)
```

Computer Science(9608) with Sir Majid Tahir

Contact: 03004003666
Email: majidtahir61@gmail.com

```
                ' follow the pointer to the next node
CurrentNodePtr = List(CurrentNodePtr).Pointer
Loop
Catch ex As Exception
Console.WriteLine("data not found")
End Try
Return (CurrentNodePtr)  ' returns NullPointer if item not found
End Function
```

Delete a node from an ordered linked list

```
PROCEDURE DeleteNode(Dataitem)
            ThisNodePtr ←  StartPointer //start at beginning of list
      WHILE ThisNodePtr <> NullPointer  //while not end of list
      AND List[ThisNodePtr].Data <> Dataitem //and item not found
      PreviousNodePtr ← ThisNodePtr //remember this node
                                    // follow the pointer to the next node
            ThisNodePtr ← List[ThisNodePtr].Pointer
      ENDWHILE
      IF ThisNodePtr <> NullPointer    //node exists in list
      THEN
            IF ThisNodePtr = StartPointer //first node to be deleted
            THEN
                  StartPointer ← List[StartPointer].Pointer
            ELSE
                  List[PreviousNodePtr] ← List[ThisNodePtr].Pointer
            ENDIF
      ENDIF
   List[ThisNodePtr].Pointer ← FreeListPtr
   FreeListPtr ← ThisNodePtr
END PROCEDURE
```

VB Code

```
Sub DeleteNode(ByVal DataItem)
        Dim ThisNodePtr, PreviousNodePtr As Integer
        ThisNodePtr = StartPointer
        Try           ' start at beginning of list

            Do While ThisNodePtr <> NULLPOINTER And List(ThisNodePtr).Data <>
DataItem                ' while not end of list and item not found

                PreviousNodePtr = ThisNodePtr    ' remember this node

                ' follow the pointer to the next node
                ThisNodePtr = List(ThisNodePtr).Pointer
            Loop
        Catch ex As Exception
            Console.WriteLine("data does not exist in list")
        End Try
        If ThisNodePtr <> NULLPOINTER Then ' node exists in list

            If ThisNodePtr = StartPointer Then ' first node to be deleted

                StartPointer = List(StartPointer).Pointer
```

Computer Science (9608) with Sir Majid Tahir

```
            Else : List(PreviousNodePtr).Pointer = List(ThisNodePtr).Pointer
            End If
            List(ThisNodePtr).Pointer = FreeListPtr
            FreeListPtr = ThisNodePtr
        End If
    End Sub
```

Access all nodes stored in the linked list

```
PROCEDURE OutputAllNodes
         CurrentNodePtr ← StartPointer   //start at beginning of list
     WHILE CurrentNodePtr <> NullPointer     //while not end of list
         OUTPUT List[CurrentNodePtr].Data   //follow the pointer to the next node
         CurrentNodePtr ← List[CurrentNodePtr].Pointer
     ENDWHILE
ENDPROCEDURE
```

VB Code

```
Sub OutputAllNodes()
        Dim CurrentNodePtr As Integer
        CurrentNodePtr = StartPointer ' start at beginning of list
        If StartPointer = NULLPOINTER Then
            Console.WriteLine("No data in list")
        End If
        Do While CurrentNodePtr <> NULLPOINTER ' while not end of list

            Console.WriteLine(CurrentNodePtr & "   " & List(CurrentNodePtr).Data)
' follow the pointer to the next node
            CurrentNodePtr = List(CurrentNodePtr).Pointer
        Loop
    End Sub
```

VB Program for Linked Lists

```
Module Module1
            ' NullPointer should be set to -1 if using array element with index 0
        Const NULLPOINTER = -1   ' Declare record type to store data and pointer
    Structure ListNode
        Dim Data As String
        Dim Pointer As Integer
    End Structure

    Dim List(7) As ListNode
    Dim StartPointer As Integer
    Dim FreeListPtr As Integer

    Sub InitialiseList()
        StartPointer = NULLPOINTER        ' set start pointer
        FreeListPtr = 0                    ' set starting position of free list
        For Index = 0 To 7              'link all nodes to make free list
            List(Index).Pointer = Index + 1
        Next
        List(7).Pointer = NULLPOINTER       'last node of free list
    End Sub

    Function FindNode(ByVal DataItem) As Integer ' returns pointer to node
        Dim CurrentNodePtr As Integer
        CurrentNodePtr = StartPointer   ' start at beginning of list
        Try
```

Computer Science(9608) with Sir Majid Tahir

```
            Do While CurrentNodePtr <> NULLPOINTER And List(CurrentNodePtr).Data <>
DataItem ' not end of list,item(Not found)
                ' follow the pointer to the next node
                CurrentNodePtr = List(CurrentNodePtr).Pointer
            Loop
        Catch ex As Exception
            Console.WriteLine("data not found")
        End Try
        Return (CurrentNodePtr)  ' returns NullPointer if item not found
    End Function

    Sub DeleteNode(ByVal DataItem)
        Dim ThisNodePtr, PreviousNodePtr As Integer
        ThisNodePtr = StartPointer
        Try             ' start at beginning of list

            Do While ThisNodePtr <> NULLPOINTER And List(ThisNodePtr).Data <> DataItem
' while not end of list and item not found

                PreviousNodePtr = ThisNodePtr    ' remember this node

                ' follow the pointer to the next node
                ThisNodePtr = List(ThisNodePtr).Pointer
            Loop
        Catch ex As Exception
            Console.WriteLine("data does not exist in list")
        End Try
        If ThisNodePtr <> NULLPOINTER Then ' node exists in list

            If ThisNodePtr = StartPointer Then ' first node to be deleted

                StartPointer = List(StartPointer).Pointer
            Else : List(PreviousNodePtr).Pointer = List(ThisNodePtr).Pointer
            End If
            List(ThisNodePtr).Pointer = FreeListPtr
            FreeListPtr = ThisNodePtr
        End If
    End Sub

    Sub InsertNode(ByVal NewItem)

        Dim ThisNodePtr, NewNodePtr, PreviousNodePtr As Integer
        If FreeListPtr <> NULLPOINTER Then   ' there is space in the array
            '                                 take node from free list and store data
item
            NewNodePtr = FreeListPtr
            List(NewNodePtr).Data = NewItem
            FreeListPtr = List(FreeListPtr).Pointer             ' find insertion point
            PreviousNodePtr = NULLPOINTER
            ThisNodePtr = StartPointer            ' start at beginning of list
            Try
                Do While (ThisNodePtr <> NULLPOINTER) And (List(ThisNodePtr).Data <
NewItem)                    ' while not end of list
                    PreviousNodePtr = ThisNodePtr ' remember this node
                    '                             follow the pointer to the next node
                    ThisNodePtr = List(ThisNodePtr).Pointer
```

Computer Science(9608) with Sir Majid Tahir

```
                Loop
            Catch ex As Exception
            End Try
            If PreviousNodePtr = NULLPOINTER Then ' insert new node at start of list

                List(NewNodePtr).Pointer = StartPointer
                StartPointer = NewNodePtr

            Else : List(NewNodePtr).Pointer = List(PreviousNodePtr).Pointer
                ' insert new node between previous node and this node
                List(PreviousNodePtr).Pointer = NewNodePtr
            End If
        Else : Console.WriteLine("no space for more data")
        End If
    End Sub

    Sub OutputAllNodes()
        Dim CurrentNodePtr As Integer
        CurrentNodePtr = StartPointer ' start at beginning of list
        If StartPointer = NULLPOINTER Then
            Console.WriteLine("No data in list")
        End If
        Do While CurrentNodePtr <> NULLPOINTER ' while not end of list

            Console.WriteLine(CurrentNodePtr & "   " & List(CurrentNodePtr).Data)
' follow the pointer to the next node
            CurrentNodePtr = List(CurrentNodePtr).Pointer
        Loop
    End Sub

    Function GetOption()
        Dim Choice As Char
        Console.WriteLine("1: insert a value")
        Console.WriteLine("2: delete a value")
        Console.WriteLine("3: find a value")
        Console.WriteLine("4: output list")
        Console.WriteLine("5: end program")
        Console.Write("Enter your choice: ")
        Choice = Console.ReadLine()
        Return (Choice)
    End Function

    Sub Main()
        Dim Choice As Char
        Dim Data As String
        Dim CurrentNodePtr As Integer

        InitialiseList()
        Choice = GetOption()
        Do While Choice <> "5"
            Select Case Choice
                Case "1"
                    Console.Write("Enter the value: ")
                    Data = Console.ReadLine()
                    InsertNode(Data)
                    OutputAllNodes()
```

Computer Science (9608) with Sir Majid Tahir

Contact: 03004003666
Email: majidtahir61@gmail.com

```
                Case "2"
                    Console.Write("Enter the value: ")
                    Data = Console.ReadLine()
                    DeleteNode(Data)
                    OutputAllNodes()
                Case "3"
                    Console.Write("Enter the value: ")
                    Data = Console.ReadLine()
                    CurrentNodePtr = FindNode(Data)
                Case "4"
                    OutputAllNodes()
                    Console.WriteLine(StartPointer & " " & FreeListPtr)
                    For i = 0 To 7
                        Console.WriteLine(i & "   " & List(i).Data & "   " &
List(i).Pointer)
                    Next
            End Select
            Choice = GetOption()
        Loop
    End Sub
End Module
```

References:
Computer Science AS & A Level Coursebook by Sylvia Langfield & Dave Duddell
https://www.geeksforgeeks.org/abstract-data-types/
https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/
http://btechsmartclass.com/DS/U2_T7.html
http://www.teach-ict.com/as_as_computing/ocr/H447/F453/3_3_5/data_structures/miniweb/pg15.htm
https://www.geeksforgeeks.org/binary-tree-set-1-introduction/
https://www.thecrazyprogrammer.com/2017/08/difference-between-tree-and-graph.html
https://www.codeproject.com/Articles/4647/A-simple-binary-tree-implementation-with-VB-NET

Sir Majid Tahir
Computer Science

Contact: 03004003666
Email: majidtahir61@gmail.com