Chapter 23 Computational Thinking and Problem-solving: Answers

Syllabus sections covered: 4.1 (Sections 4.1.1 – 4.1.3)

Task 23.01 Bubble sort (n elements)

```
n ← MaxIndex - 1
REPEAT
   NoMoreSwaps ← TRUE
   FOR j ← 1 TO n
      IF MyList[j] > MyList[j + 1]
         THEN
            Temp ← MyList[j]
            MyList[j] ← MyList[j + 1]
            MyList[j + 1] ← Temp
            NoMoreSwaps ← FALSE
      ENDIF
   ENDFOR
   n ← n - 1
UNTIL NoMoreSwaps = TRUE
```

Python

```
MyList = []
for Index in range(7):
   MyList.append( int(input("Enter a number: ")))
MaxIndex = 7
n = MaxIndex - 1
NoMoreSwaps = False
while NoMoreSwaps == False:
   NoMoreSwaps = True
   for j in range(n):
      if MyList[j] > MyList[j + 1]:
         Temp = MyList[j]
         MyList[j] = MyList[j + 1]
         MyList[j + 1] = Temp
         NoMoreSwaps = False
   n = n - 1

for Index in range(7):
   print(MyList[Index])
```

VB.NET

```
Module Module1
    Dim MyList(7) As Integer
    Dim Index, MaxIndex, n, j, Temp As Integer
    Dim NoMoreSwaps As Boolean

    Sub Main()
        For Index = 1 To 7
            Console.Write("Enter a number: ")
            MyList(Index) = Console.ReadLine()
        Next
```

```
        MaxIndex = 7
        n = MaxIndex - 1
        Do
            NoMoreSwaps = True
            For j = 1 To n
                If MyList(j) > MyList(j + 1) Then
                    Temp = MyList(j)
                    MyList(j) = MyList(j + 1)
                    MyList(j + 1) = Temp
                    NoMoreSwaps = False
                End If
            Next
            n = n - 1
        Loop Until NoMoreSwaps

        For Index = 1 To 7
            Console.Write(MyList(Index) & " ")
        Next
        Console.ReadLine()
    End Sub

End Module
```

Pascal

```
var MyList : Array[1..7] of integer;
    Index, MaxIndex, n, j, Temp : Integer;
    NoMoreSwaps : Boolean;

begin
  for Index := 1 to 7 do
     begin
        Write('Enter a number: ');
        ReadLn(MyList[Index]);
     end;
  MaxIndex := 7;
  n := MaxIndex - 1;
  repeat
     NoMoreSwaps := True;
     for j := 1 to n do
        if MyList[j] > MyList[j + 1]
           then
              begin
                 Temp := MyList[j];
                 MyList[j] := MyList[j + 1];
                 MyList[j + 1] := Temp;
                 NoMoreSwaps := False;
              end;
     n := n - 1;
  until NoMoreSwaps;
  for Index := 1 to 7 do
        write(MyList[Index], '  ');
  ReadLn;
end.
```

Task 23.02 part 1

NumberOfitems	Pointer	ItemToBeInserted	CurrentItem	List					
				[1	[2	[3	[4	[5	[6

				]	]	]	]	]	]
6				53	21	60	18	42	19
	2	21	1		53				
			0	21					
	3	60	2			60			
	4	18	3				60		
			2			53			
			1		21				
			0	18					
	5	42	4					60	
			3				53		
			2			42			
	6	19	5						60
			4					53	
			3				42		
			2			21			
			1		19				

Task 23.02 part 2

Python

```
NumberOfItems = 6
List = [0]
List.append(53)
List.append(21)
List.append(60)
List.append(18)
List.append(42)
List.append(19)
for Pointer in range(1, NumberOfItems + 1):
   print(List[Pointer], end=' ')
print();

for Pointer in range(2, NumberOfItems + 1):
   ItemToBeInserted = List[Pointer]
   CurrentItem = Pointer - 1
   while (List[CurrentItem] > ItemToBeInserted) and
(CurrentItem > 0):
      List[CurrentItem + 1] = List[CurrentItem]
      CurrentItem = CurrentItem - 1
   List[CurrentItem + 1] = ItemToBeInserted

for Pointer in range(1, NumberOfItems + 1):
   print(List[Pointer], end=' ')
```

VB.NET

```
Module Module1

    Sub Main()
        Dim Pointer, NumberOfItems, ItemToBeInserted, CurrentItem As
Integer
        Dim List(6) As Integer
        NumberOfItems = 6
        List(1) = 53
        List(2) = 21
        List(3) = 60
        List(4) = 18
        List(5) = 42
        List(6) = 19
        For Pointer = 1 To NumberOfItems
```

```
            Console.Write(List(Pointer) & " ")
        Next
        Console.WriteLine()

        For Pointer = 2 To NumberOfItems
            ItemToBeInserted = List(Pointer)
            CurrentItem = Pointer - 1
            While (List(CurrentItem) > ItemToBeInserted) And
(CurrentItem > 0)

                List(CurrentItem + 1) = List(CurrentItem)
                CurrentItem = CurrentItem - 1
            End While
            List(CurrentItem + 1) = ItemToBeInserted
        Next

        For Pointer = 1 To NumberOfItems
            Console.Write(List(Pointer) & " ")
        Next
        Console.ReadLine()
    End Sub

End Module
```

Pascal

```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;
var Pointer, NumberOfItems, ItemToBeInserted,
CurrentItem : integer;
    List : array[1..6] of integer;
begin
  NumberOfItems := 6;
  List[1] := 53;
  List[2] := 21;
  List[3] := 60;
  List[4] := 18;
  List[5] := 42;
  List[6] := 19;
   for Pointer := 1 to NumberOfItems do
write(List[Pointer], ' ');
   writeln;
  for Pointer := 2 to NumberOfitems do
     begin
        ItemToBeInserted := List[Pointer];
        CurrentItem := Pointer - 1;
        while (List[CurrentItem] > ItemToBeInserted)
and (CurrentItem > 0) do
           begin
              List[CurrentItem + 1] :=
List[CurrentItem];
              CurrentItem := CurrentItem - 1;
           end;
        List[CurrentItem + 1] := ItemToBeInserted;
     end;
  for Pointer := 1 to NumberOfItems do
```

```
write(List[Pointer], ' ');
     readln;
end.
```

Task 23.03

List																			
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]
7	12	19	23	27	33	37	41	45	56	59	60	62	71	75	80	84	88	92	99

MaxItems	SearchItem	Found	SearchFailed	First	Last	Middle	List[Middle]
20	60	F	F	1	20	10	56
				11		15	75
					14	12	60
		T					

While loop executed 3 times

MaxItems	SearchItem	Found	SearchFailed	First	Last	Middle	List[Middle]
20	34	F	F	1	20	10	56
					9	5	27
				6		7	37
					6	6	33
			T	7			

While loop executed 4 times

Task 23.04

Python

```
# NullPointer should be set to -1 if using array element with index 0
NULLPOINTER = -1

# Declare record type to store data and pointer
class ListNode :
   def __init__(self) :
      self.Data = ""
      self.Pointer = NULLPOINTER

def InitialiseList() :
   List = [ListNode() for i in range(8)]
   StartPointer = NULLPOINTER          # set start pointer
   FreeListPtr = 0                     # set starting position of free list
   for Index in range(7) :             # link all nodes to make free list
      List[Index].Pointer = Index + 1
   List[7].Pointer = NULLPOINTER       #last node of free list
   return(List, StartPointer, FreeListPtr)

def InsertNode(List, StartPointer, FreeListPtr, NewItem) :
   if FreeListPtr != NULLPOINTER :
      # 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
      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

      if PreviousNodePtr == NULLPOINTER :
         # 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
   else :
      print("no space for more data")
   return(List, StartPointer, FreeListPtr)

def FindNode(List, StartPointer, DataItem) : # returns pointer to node
   CurrentNodePtr = StartPointer  # start at beginning of list
   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
   return(CurrentNodePtr)  # returns NullPointer if item not found

def DeleteNode(List, StartPointer, FreeListPtr, DataItem) :
   ThisNodePtr = StartPointer               # start at beginning of list
   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
   if ThisNodePtr != NULLPOINTER :  # node exists in list
      if ThisNodePtr == StartPointer : # first node to be deleted
         StartPointer = List[StartPointer].Pointer
      else :
         List[PreviousNodePtr].Pointer = List[ThisNodePtr].Pointer
      List[ThisNodePtr].Pointer = FreeListPtr
      FreeListPtr = ThisNodePtr
   else :
      print("data does not exist in list")
   return(List, StartPointer, FreeListPtr)

def OutputAllNodes(List, StartPointer) :
   CurrentNodePtr = StartPointer # start at beginning of list
   if StartPointer == NULLPOINTER :
      print("No data in list")
   while CurrentNodePtr != NULLPOINTER : # while not end of list
      print(CurrentNodePtr, "  ",List[CurrentNodePtr].Data)
      # follow the pointer to the next node
      CurrentNodePtr = List[CurrentNodePtr].Pointer

def GetOption() :
   print("1: insert a value")
   print("2: delete a value")
   print("3: find a value")
   print("4: output list")
   print("5: end program")
   option = input("Enter your choice: ")
   return(option)


List, StartPointer, FreeListPtr = InitialiseList()

Option = GetOption()
while Option != "5" :
   if Option == "1" :
      Data = input("Enter the value: ")
      List, StartPointer, FreeListPtr = InsertNode(List, StartPointer,
FreeListPtr, Data)
      OutputAllNodes(List, StartPointer)
   elif Option == "2" :
      Data = input("Enter the value: ")
      List, StartPointer, FreeListPtr = DeleteNode(List, StartPointer,
FreeListPtr, Data)
```

```
      OutputAllNodes(List, StartPointer)
   elif Option == "3" :
      Data = input("Enter the value: ")
      CurrentNodePtr = FindNode(List, StartPointer, Data)
      if CurrentNodePtr == NULLPOINTER :
         print("data not found")
      print(StartPointer, FreeListPtr)
      for i in range(8) :
         print(i, "  ", List[i].Data, "  ", List[i].Pointer)
   elif Option == "4" :
      OutputAllNodes(List, StartPointer)
   Option = GetOption()
```

VB.NET

```
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(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
                ' 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(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(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
                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  ' insert new node between previous node and this
node
                List(NewNodePtr).Pointer =
List(PreviousNodePtr).Pointer
                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()
                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
```

Pascal

```
program linkedList;

{$APPTYPE CONSOLE}

uses
  SysUtils;

// NullPointer should be set to -1 if using array element with index 0
  Const NULLPOINTER = -1;
    // Declare record type to store data and pointer
  type ListNode = record
    Data : String;
    Pointer : Integer;
  End;

  var List: array[0..7] of ListNode;
      StartPointer, FreeListPtr : Integer;

procedure InitialiseList;
var Index : integer;
  begin
    StartPointer := NULLPOINTER;         // set start pointer
    FreeListPtr := 0;                    // set starting position of free
list
    For Index := 0 To 7 do            // link all nodes to make free list
      List[Index].Pointer := Index + 1;
    List[7].Pointer := NULLPOINTER;       // last node of free list
  End;

Function FindNode(DataItem : String) : Integer; // returns pointer to node
var CurrentNodePtr : Integer;
  begin
    CurrentNodePtr := StartPointer;  // start at beginning of list
    While (CurrentNodePtr <> NULLPOINTER) And (List[CurrentNodePtr].Data <>
DataItem) do
    // not end of list,item not found
    // follow the pointer to the next node
      CurrentNodePtr := List[CurrentNodePtr].Pointer;
    WriteLn('data not found');
    FindNode := CurrentNodePtr;  // returns NullPointer if item not found
  End;

procedure DeleteNode(DataItem : string);
var ThisNodePtr, PreviousNodePtr : Integer;
  begin
    ThisNodePtr := StartPointer;
    // start at beginning of list
    While (ThisNodePtr <> NULLPOINTER) And (List[ThisNodePtr].Data <>
DataItem) do
      begin
        // 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;
      end;
    if ThisNodePtr = NULLPOINTER
      then WriteLn('data does not exist in list');
    If ThisNodePtr <> NULLPOINTER
      Then // node exists in list
        begin
          If ThisNodePtr = StartPointer Then // first node to be deleted
            StartPointer := List[StartPointer].Pointer
          Else List[PreviousNodePtr].Pointer := List[ThisNodePtr].Pointer;
          List[ThisNodePtr].Pointer := FreeListPtr;
          FreeListPtr := ThisNodePtr;
        End;
  End;

procedure InsertNode(NewItem : string);
var ThisNodePtr, NewNodePtr, PreviousNodePtr : Integer;
  begin
    If FreeListPtr <> NULLPOINTER
      Then
        begin
          // 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
          While (ThisNodePtr <> NULLPOINTER) And (List[ThisNodePtr].Data <
NewItem) do
            begin
              // while not end of list
              PreviousNodePtr := ThisNodePtr; // remember this node
              // follow the pointer to the next node
              ThisNodePtr := List[ThisNodePtr].Pointer;
            end;
          If PreviousNodePtr = NULLPOINTER
          Then // insert new node at start of list
            begin
              List[NewNodePtr].Pointer := StartPointer;
              StartPointer := NewNodePtr;
            end
          Else  // insert new node between previous node and this node
            begin
              List[NewNodePtr].Pointer := List[PreviousNodePtr].Pointer;
              List[PreviousNodePtr].Pointer := NewNodePtr;
            End
          end
        Else
          writeln('no space for more data');
    End;

procedure OutputAllNodes();
var CurrentNodePtr : Integer;
  begin
    CurrentNodePtr := StartPointer; // start at beginning of list
    If StartPointer = NULLPOINTER
      Then WriteLn('No data in list');
         While CurrentNodePtr <> NULLPOINTER do // while not end of list
           begin
             WriteLn(CurrentNodePtr , '  ' , List[CurrentNodePtr].Data);
             // follow the pointer to the next node
             CurrentNodePtr := List[CurrentNodePtr].Pointer;
           end;
  End;

Function GetOption() : char;
var Response : char;
  begin
    WriteLn('1: insert a value');
    WriteLn('2: delete a value');
    WriteLn('3: find a value');
    WriteLn('4: output list');
    WriteLn('5: end program');
    Write('Enter your choice: ');
    ReadLn(Response);
    GetOption := Response;
  End;

procedure Main();
var Choice : Char;
    Data : String;
    CurrentNodePtr, i : Integer;
  begin
    InitialiseList;
    Choice := GetOption();
    While Choice <> '5' do
      begin
        Case Choice of
        '1': begin
               Write('Enter the value: ');
               ReadLn(Data);
               InsertNode(Data);
               OutputAllNodes();
             end;
        '2': begin
               Write('Enter the value: ') ;
```

```
                ReadLn(Data);
                DeleteNode(Data);
                OutputAllNodes();
              end;
        '3' : begin
                Write('Enter the value: ');
                ReadLn(Data);
                CurrentNodePtr := FindNode(Data);
                WriteLn(CurrentNodePtr);
                OutputAllNodes();
              end;
        '4' : begin
                OutputAllNodes();
                WriteLn(StartPointer , ' ' , FreeListPtr);
                For i := 0 To 7  do
                  WriteLn(i , '  ' , List[i].Data , '  ' , List[i].Pointer);
              end;
        end;
        Choice := GetOption();
      end;
  End;

begin
   main;
end.
```

Task 23.05

Python

```
# NullPointer should be set to -1 if using array element with index
0
NULLPOINTER = -1

# Declare record type to store data and pointer
class Node :
   def __init__(self) :
      self.Data = ""
      self.Pointer = NULLPOINTER

def InitialiseStack() :
   Stack = [Node() for i in range(8)]
   TopOfStack = NULLPOINTER          # set start pointer
   FreeListPtr = 0                     # set starting position of
free list
   for Index in range(7) :             # link all nodes to make
free list
      Stack[Index].Pointer = Index + 1
   Stack[7].Pointer = NULLPOINTER       #last node of free list
   return(Stack, TopOfStack, FreeListPtr)

def Push(Stack, TopOfStack, FreeListPtr, NewItem) :
   if FreeListPtr != NULLPOINTER :
      # there is space in the array
      # take node from free list and store data item
      NewNodePtr = FreeListPtr
      Stack[NewNodePtr].Data = NewItem
      FreeListPtr = Stack[FreeListPtr].Pointer
      # insert new node at top of stack
      Stack[NewNodePtr].Pointer = TopOfStack
      TopOfStack = NewNodePtr
   else :
      print("no space for more data")
   return(Stack, TopOfStack, FreeListPtr)

def Pop(Stack, TopOfStack, FreeListPtr) :
   if TopOfStack == NULLPOINTER :
      print("no data on stack")
      Value = ""
   else :
      Value = Stack[TopOfStack].Data
```

```
        ThisNodePtr = TopOfStack
        TopOfStack = Stack[TopOfStack].Pointer
        Stack[ThisNodePtr].Pointer = FreeListPtr
        FreeListPtr = ThisNodePtr
    return(Stack, TopOfStack, FreeListPtr, Value)

def OutputAllNodes(Stack, TopOfStack) :
    CurrentNodePtr = TopOfStack # start at beginning of list
    if TopOfStack == NULLPOINTER :
        print("No data on stack")
    while CurrentNodePtr != NULLPOINTER : # while not end of list
        print(CurrentNodePtr, "  ",Stack[CurrentNodePtr].Data)
        # follow the pointer to the next node
        CurrentNodePtr = Stack[CurrentNodePtr].Pointer

def GetOption() :
    print("1: push a value")
    print("2: pop a value")
    print("3: output stack")
    print("4: end program")
    option = input("Enter your choice: ")
    return(option)


Stack, TopOfStack, FreeListPtr = InitialiseStack()

Option = GetOption()
while Option != "4" :
    if Option == "1" :
        Data = input("Enter the value: ")
        Stack, TopOfStack, FreeListPtr = Push(Stack, TopOfStack,
FreeListPtr, Data)
        OutputAllNodes(Stack, TopOfStack)
    elif Option == "2" :
        Stack, TopOfStack, FreeListPtr, Value = Pop(Stack,
TopOfStack, FreeListPtr)
        print("Data popped: ", Value)
        OutputAllNodes(Stack, TopOfStack)
    elif Option == "3" :
        OutputAllNodes(Stack, TopOfStack)
        print(TopOfStack, FreeListPtr)
        for i in range(8) :
            print(i, "  ", Stack[i].Data, "  ", Stack[i].Pointer)

    Option = GetOption()
```

VB.NET

```
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 Node
        Dim Data As String
        Dim Pointer As Integer
    End Structure

    Dim Stack(7) As Node
    Dim TopOfStack As Integer
    Dim FreeListPtr As Integer

    Sub InitialiseStack()
        TopOfStack = NULLPOINTER         ' set start pointer
        FreeListPtr = 0                    ' set starting position of
free list
        For Index = 0 To 7              'link all nodes to make free
```

```
list
            Stack(Index).Pointer = Index + 1
        Next
        Stack(7).Pointer = NULLPOINTER       'last node of free list
    End Sub

    Function Pop()
        Dim ThisNodePtr As Integer
        Dim Value As String
        If TopOfStack = NULLPOINTER Then
            Console.WriteLine("no data on stack")
            Value = ""
        Else
            Value = Stack(TopOfStack).Data
            ThisNodePtr = TopOfStack
            TopOfStack = Stack(TopOfStack).Pointer
            Stack(ThisNodePtr).Pointer = FreeListPtr
            FreeListPtr = ThisNodePtr
        End If
        Return Value
    End Function

    Sub Push(NewItem)
        Dim NewNodePtr As Integer
        If FreeListPtr <> NULLPOINTER Then
            ' there is space in the array
            ' take node from free list and store data item
            NewNodePtr = FreeListPtr
            Stack(NewNodePtr).Data = NewItem
            FreeListPtr = Stack(FreeListPtr).Pointer
            ' insert new node at top of stack
            Stack(NewNodePtr).Pointer = TopOfStack
            TopOfStack = NewNodePtr
        Else
            Console.WriteLine("no space for more data")
        End If
    End Sub

    Sub OutputAllNodes()
        Dim CurrentNodePtr As Integer
        CurrentNodePtr = TopOfStack ' start at beginning of list
        If TopOfStack = NULLPOINTER Then
            Console.WriteLine("No data on stack")
        End If
        Do While CurrentNodePtr <> NULLPOINTER ' while not end of list
            Console.WriteLine(CurrentNodePtr & "  " &
Stack(CurrentNodePtr).Data)
            ' follow the pointer to the next node
            CurrentNodePtr = Stack(CurrentNodePtr).Pointer
        Loop
    End Sub

    Function GetOption()
        Dim Choice As Char
        Console.WriteLine("1: push a value")
        Console.WriteLine("2: pop a value")
        Console.WriteLine("3: output stack")
        Console.WriteLine("4: 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

        InitialiseStack()
        Choice = GetOption()
        Do While Choice <> "4"
            Select Case Choice
                Case "1"
                    Console.Write("Enter the value: ")
                    Data = Console.ReadLine()
                    Push(Data)
                    OutputAllNodes()
                Case "2"
                    Data = Pop()
                    Console.WriteLine("Data popped: " & Data)
                    OutputAllNodes()
                Case "3"
                    OutputAllNodes()
                    Console.WriteLine(TopOfStack & " " & FreeListPtr)
                    For i = 1 To 7
                        Console.WriteLine(i & "  " & Stack(i).Data & "
" & Stack(i).Pointer)
                    Next
            End Select
            Choice = GetOption()
        Loop
    End Sub

End Module
```

Pascal

```
program linkedList;

{$APPTYPE CONSOLE}

uses
  SysUtils;

// NullPointer should be set to -1 if using array element with
index 0
  Const NULLPOINTER = -1;
    // Declare record type to store data and pointer
  type Node = record
    Data : String;
    Pointer : Integer;
  End;

  var Stack: array[0..7] of Node;
      TopOfStack, FreeListPtr : Integer;

procedure InitialiseStack;
var Index : integer;
  begin
    TopOfStack := NULLPOINTER;          // set start pointer
    FreeListPtr := 0;                     // set starting position
of free list
    For Index := 0 To 7 do             // link all nodes to make
free list
      Stack[Index].Pointer := Index + 1;
    Stack[7].Pointer := NULLPOINTER;        // last node of free
list
  End;
```

```
function Pop : string;
var ThisNodePtr : Integer;
  begin
    If TopOfStack = NULLPOINTER
      then
        begin
          WriteLn('no data on stack');
          Pop := '';
        end
      else
        begin
          Pop := Stack[TopOfStack].Data;
          ThisNodePtr := TopOfStack;
          TopOfStack := Stack[TopOfStack].Pointer;
          Stack[ThisNodePtr].Pointer := FreeListPtr;
          FreeListPtr := ThisNodePtr;
        end;
  End;

procedure Push(NewItem : string);
var NewNodePtr : Integer;
  begin
    If FreeListPtr <> NULLPOINTER
      Then
        begin
          // there is space in the array
          // take node from free list and store data item
          NewNodePtr := FreeListPtr;
          Stack[NewNodePtr].Data := NewItem;
          FreeListPtr := Stack[FreeListPtr].Pointer;
          // insert new node at top of stack
          Stack[NewNodePtr].Pointer := TopOfStack;
          TopOfStack := NewNodePtr;
        end
          Else  // insert new node between previous node and this
node
            writeln('no space for more data');
  End;

procedure OutputAllNodes();
var CurrentNodePtr : Integer;
  begin
    CurrentNodePtr := TopOfStack; // start at beginning of list
    If TopOfStack = NULLPOINTER
      Then WriteLn('No data on stack');
      While CurrentNodePtr <> NULLPOINTER do // while not end of
list
        begin
          WriteLn(CurrentNodePtr , '  ' ,
Stack[CurrentNodePtr].Data);
            // follow the pointer to the next node
            CurrentNodePtr := Stack[CurrentNodePtr].Pointer;
          end;
  End;

Function GetOption() : char;
var Response : char;
  begin
    WriteLn('1: push a value');
    WriteLn('2: pop a value');
    WriteLn('3: output Stack');
    WriteLn('4: end program');
    Write('Enter your choice: ');
    ReadLn(Response);
    GetOption := Response;
  End;

procedure Main();
```

```
var Choice : Char;
    Data : String;
    i : Integer;
  begin
    InitialiseStack;
    Choice := GetOption();
    While Choice <> '4' do
      begin
        Case Choice of
        '1': begin
               Write('Enter the value: ');
               ReadLn(Data);
               Push(Data);
               OutputAllNodes();
             end;
        '2': begin
               Data := Pop;
               writeln('Data popped: ', Data);
               OutputAllNodes();
             end;
       '3' : begin
               OutputAllNodes();
               WriteLn(TopOfStack , ' ' , FreeListPtr);
               For i := 0 To 7  do
                 WriteLn(i , '  ' , Stack[i].Data , '  ' ,
Stack[i].Pointer);
             end;
       end;
       Choice := GetOption();
     end;
  End;

begin
   main;
end.
```

Task 23.06

Python

```
# NullPointer should be set to -1 if using array element with index
0
NULLPOINTER = -1

# Declare record type to store data and pointer
class Node :
   def __init__(self) :
      self.Data = ""
      self.Pointer = NULLPOINTER

def InitialiseQueue() :
   Queue = [Node() for i in range(8)]
   HeadOfQueue = NULLPOINTER          # set Head of Queue pointer
   EndOfQueue = NULLPOINTER           # set End of Queue pointer
   FreeListPtr = 0                     # set starting position of
free list
   for Index in range(7) :             # link all nodes to make
free list
      Queue[Index].Pointer = Index + 1
   Queue[7].Pointer = NULLPOINTER       #last node of free list
   return(Queue, HeadOfQueue, EndOfQueue, FreeListPtr)

def JoinQueue(Queue, HeadOfQueue, EndOfQueue, FreeListPtr, NewItem)
:
   if FreeListPtr != NULLPOINTER :
      # there is space in the array
      # take node from free list and store data item
      NewNodePtr = FreeListPtr
```

```
      Queue[NewNodePtr].Data = NewItem
      FreeListPtr = Queue[FreeListPtr].Pointer
      Queue[NewNodePtr].Pointer = NULLPOINTER
      # find insertion point
      if EndOfQueue == NULLPOINTER :
         # insert new node at start of Queue
         HeadOfQueue = NewNodePtr
      else :
         Queue[EndOfQueue].Pointer = NewNodePtr
      EndOfQueue = NewNodePtr
   else :
      print("no space for more data")
   return(Queue, HeadOfQueue, EndOfQueue, FreeListPtr)

def LeaveQueue(Queue, HeadOfQueue, EndOfQueue, FreeListPtr) :
   if HeadOfQueue != NULLPOINTER :  # not an empty queue
      Value = Queue[HeadOfQueue].Data
      ThisNodePtr = Queue[HeadOfQueue].Pointer
      if ThisNodePtr == NULLPOINTER :
         EndOfQueue = NULLPOINTER # deleted last item in Queue
      Queue[HeadOfQueue].Pointer = FreeListPtr
      FreeListPtr = HeadOfQueue
      HeadOfQueue = ThisNodePtr
   else :
      print("queue empty")
      Value = ""
   return(Queue, HeadOfQueue, EndOfQueue, FreeListPtr, Value)

def OutputAllNodes(Queue, HeadOfQueue) :
   CurrentNodePtr = HeadOfQueue # start at beginning of queue
   if HeadOfQueue == NULLPOINTER :
      print("No data in list")
   while CurrentNodePtr != NULLPOINTER : # while not end of list
      print(CurrentNodePtr, "  ",Queue[CurrentNodePtr].Data)
      # follow the pointer to the next node
      CurrentNodePtr = Queue[CurrentNodePtr].Pointer

def GetOption() :
   print("1: join queue")
   print("2: leave queue")
   print("3: output queue")
   print("4: end program")
   option = input("Enter your choice: ")
   return(option)


Queue, HeadOfQueue, EndOfQueue, FreeListPtr = InitialiseQueue()

Option = GetOption()
while Option != "4" :
   if Option == "1" :
      Data = input("Enter the value: ")
      Queue, HeadOfQueue, EndOfQueue, FreeListPtr =
JoinQueue(Queue, HeadOfQueue, EndOfQueue, FreeListPtr, Data)
      OutputAllNodes(Queue, HeadOfQueue)
   elif Option == "2" :
      Queue, HeadOfQueue, EndOfQueue, FreeListPtr, Value =
LeaveQueue(Queue, HeadOfQueue, EndOfQueue, FreeListPtr)
      print("data leaving queue: ",Value)
      OutputAllNodes(Queue, HeadOfQueue)
   elif Option == "3" :
      OutputAllNodes(Queue, HeadOfQueue)
      print(HeadOfQueue, EndOfQueue, FreeListPtr)
      for i in range(8) :
         print(i, "  ", Queue[i].Data, "  ", Queue[i].Pointer)
   Option = GetOption()
```

VB.NET

```
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 Node
        Dim Data As String
        Dim Pointer As Integer
    End Structure

    Dim Queue(7) As Node
    Dim HeadOfQueue As Integer
    Dim EndOfQueue As Integer
    Dim FreeListPtr As Integer

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

    Function LeaveQueue() As String
        Dim ThisNodePtr As Integer
        Dim Value As String
        If HeadOfQueue = NULLPOINTER Then
            Console.WriteLine("empty queue")
            Value = ""
        Else
            Value = Queue(HeadOfQueue).Data
            ThisNodePtr = Queue(HeadOfQueue).Pointer
            If ThisNodePtr = NULLPOINTER Then
                EndOfQueue = NULLPOINTER
            End If
            Queue(HeadOfQueue).Pointer = FreeListPtr
            FreeListPtr = HeadOfQueue
            HeadOfQueue = ThisNodePtr
        End If
        Return Value
    End Function

    Sub JoinQueue(NewItem)
        Dim NewNodePtr As Integer
        If FreeListPtr <> NULLPOINTER Then
            ' there is space in the array
            ' take node from free list and store data item
            NewNodePtr = FreeListPtr
            Queue(NewNodePtr).Data = NewItem
            FreeListPtr = Queue(FreeListPtr).Pointer
            Queue(NewNodePtr).Pointer = NULLPOINTER
            If EndOfQueue = NULLPOINTER Then
                HeadOfQueue = NewNodePtr
            Else
                Queue(EndOfQueue).Pointer = NewNodePtr
            End If
            EndOfQueue = NewNodePtr
        Else
```

```
            Console.WriteLine("no space for more data")
        End If
    End Sub

    Sub OutputAllNodes()
        Dim CurrentNodePtr As Integer
        CurrentNodePtr = HeadOfQueue ' start at beginning of queue
        If HeadOfQueue = NULLPOINTER Then
            Console.WriteLine("No data in queue")
        End If
        Do While CurrentNodePtr <> NULLPOINTER ' while not end of
list
            Console.WriteLine(CurrentNodePtr & "  " &
Queue(CurrentNodePtr).Data)
            ' follow the pointer to the next node
            CurrentNodePtr = Queue(CurrentNodePtr).Pointer
        Loop
    End Sub

    Function GetOption()
        Dim Choice As Char
        Console.WriteLine("1: join queue")
        Console.WriteLine("2: leave queue")
        Console.WriteLine("3: output queue")
        Console.WriteLine("4: 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

        InitialiseQueue()
        Choice = GetOption()
        Do While Choice <> "4"
            Select Case Choice
                Case "1"
                    Console.Write("Enter the value: ")
                    Data = Console.ReadLine()
                    JoinQueue(Data)
                    OutputAllNodes()
                Case "2"
                    Data = LeaveQueue()
                    Console.WriteLine("Data popped: " & Data)
                    OutputAllNodes()
                Case "3"
                    OutputAllNodes()
                    Console.Write(HeadOfQueue & "  " & "  " &
EndOfQueue & " ")
                    Console.WriteLine(FreeListPtr)
                    For i = 0 To 7
                        Console.Write(i & "  " & Queue(i).Data & "
")
                        Console.WriteLine(Queue(i).Pointer)
                    Next
            End Select
            Choice = GetOption()
        Loop
    End Sub
```

```
End Module
```

Pascal

```
program linkedList;

{$APPTYPE CONSOLE}

uses
  SysUtils;

// NullPointer should be set to -1 if using array element with
index 0
  Const NULLPOINTER = -1;
    // Declare record type to store data and pointer
  type Node = record
    Data : String;
    Pointer : Integer;
  End;

  var Queue: array[0..7] of Node;
      HeadOfQueue, EndOfQueue, FreeListPtr : Integer;

procedure InitialiseQueue;
var Index : integer;
  begin
    HeadOfQueue := NULLPOINTER;         // set start pointer
    EndOfQueue := NULLPOINTER;
    FreeListPtr := 0;                    // set starting position
of free list
    For Index := 0 To 7 do            // link all nodes to make
free list
      Queue[Index].Pointer := Index + 1;
    Queue[7].Pointer := NULLPOINTER;       // last node of free
list
  End;

function LeaveQueue : string;
var ThisNodePtr : Integer;
  begin
    If HeadOfQueue = NULLPOINTER
      then
        begin
          WriteLn('empty queue');
          LeaveQueue := '';
        end
      else
        begin
          LeaveQueue := Queue[HeadOfQueue].Data;
          ThisNodePtr := Queue[HeadOfQueue].Pointer;
          if ThisNodePtr = NULLPOINTER
            then EndOfQueue := NULLPOINTER;
          Queue[HeadOfQueue].Pointer := FreeListPtr;
          FreeListPtr := HeadOfQueue;
          HeadOfQueue := ThisNodePtr;
        end;
  End;

procedure JoinQueue(NewItem : string);
var NewNodePtr : Integer;
  begin
    If FreeListPtr <> NULLPOINTER
      Then
        begin
          // there is space in the array
          // take node from free list and store data item
```

```
          NewNodePtr := FreeListPtr;
          Queue[NewNodePtr].Data := NewItem;
          FreeListPtr := Queue[FreeListPtr].Pointer;
          // insert new node at top of stack
          Queue[NewNodePtr].Pointer := NULLPOINTER;
          if EndOfQueue = NULLPOINTER
            then
              HeadOfQueue := NewNodePtr
            Else
              Queue[EndOfQueue].Pointer := NewNodePtr;
          EndOfQueue := NewNodePtr;
        end
      else
        writeln('no space for more data');
  end;

procedure OutputAllNodes();
var CurrentNodePtr : Integer;
  begin
    CurrentNodePtr := HeadOfQueue; // start at beginning of list
    If HeadOfQueue = NULLPOINTER
      Then WriteLn('No data in queue');
      While CurrentNodePtr <> NULLPOINTER do // while not end of
list
        begin
          WriteLn(CurrentNodePtr , '  ' ,
Queue[CurrentNodePtr].Data);
             // follow the pointer to the next node
             CurrentNodePtr := Queue[CurrentNodePtr].Pointer;
        end;
  End;

Function GetOption() : char;
var Response : char;
  begin
    WriteLn('1: join queue');
    WriteLn('2: leave queue');
    WriteLn('3: output queue');
    WriteLn('4: end program');
    Write('Enter your choice: ');
    ReadLn(Response);
    GetOption := Response;
  End;

procedure Main();
var Choice : Char;
    Data : String;
    i : Integer;
  begin
    InitialiseQueue;
    Choice := GetOption();
    While Choice <> '4' do
      begin
        Case Choice of
        '1': begin
               Write('Enter the value: ');
               ReadLn(Data);
               JoinQueue(Data);
               OutputAllNodes();
             end;
        '2': begin
               Data := LeaveQueue;
               writeln('Data popped: ', Data);
               OutputAllNodes();
             end;
       '3' : begin
               OutputAllNodes();
               WriteLn(HeadOfQueue , ' ', EndOfQueue, '  ' ,
```

```
FreeListPtr);
               For i := 0 To 7  do
                 WriteLn(i , '  ' , Queue[i].Data , '  ' ,
Queue[i].Pointer);
             end;
       end;
       Choice := GetOption();
     end;
  End;

begin
   main;
end.
```

Task 23.07

Python

```
# NullPointer should be set to -1 if using array element with index
0
NULLPOINTER = -1

# Declare record type to store data and pointer
class TreeNode :
   def __init__(self) :
      self.Data = ""
      self.LeftPointer = NULLPOINTER
      self.RightPointer = NULLPOINTER

def InitialiseTree() :
   Tree = [TreeNode() for i in range(8)]
   RootPointer = NULLPOINTER          # set Root pointer
   FreePtr = 0                      # set starting position of free
list
   for Index in range(7) :             # link all nodes to make
free list
      Tree[Index].LeftPointer = Index + 1
   return(Tree, RootPointer, FreePtr)

def InsertNode(Tree, RootPointer, FreePtr, NewItem) :
   if FreePtr != NULLPOINTER :
      # there is space in the array
      # take node from free list and store data item
      NewNodePtr = FreePtr
      Tree[NewNodePtr].Data = NewItem
      FreePtr = Tree[FreePtr].LeftPointer
      Tree[NewNodePtr].LeftPointer = NULLPOINTER
      # check if empty tree
      if RootPointer == NULLPOINTER :
         # insert new node at root
         RootPointer = NewNodePtr
      else : # find insertion point
         ThisNodePtr = RootPointer
         while ThisNodePtr != NULLPOINTER : # while not a leaf node
            PreviousNodePtr = ThisNodePtr # remember this node
            if Tree[ThisNodePtr].Data > NewItem :
               TurnedLeft = True # # follow left pointer
               ThisNodePtr = Tree[ThisNodePtr].LeftPointer
            else :
               TurnedLeft = False
               ThisNodePtr = Tree[ThisNodePtr].RightPointer
         if TurnedLeft :
            Tree[PreviousNodePtr].LeftPointer = NewNodePtr
         else :
            Tree[PreviousNodePtr].RightPointer = NewNodePtr
   else :
      print("no space for more data")
   return(Tree, RootPointer, FreePtr)
```

```
def FindNode(Tree, RootPointer, SearchItem) :
   ThisNodePtr = RootPointer # start at the root of the tree
   while ThisNodePtr != NULLPOINTER and Tree[ThisNodePtr].Data !=
SearchItem :
      # while there is a pointer to follow and search item not
found
      if Tree[ThisNodePtr].Data > SearchItem :
         ThisNodePtr = Tree[ThisNodePtr].LeftPointer # follow left
pointer
      else :
         ThisNodePtr = Tree[ThisNodePtr].RightPointer # follow
right pointer
   return(ThisNodePtr)

def TraverseTree(Tree, RootPointer) :
   if RootPointer != NULLPOINTER :
      TraverseTree(Tree, Tree[RootPointer].LeftPointer)
      print(Tree[RootPointer].Data)
      TraverseTree(Tree, Tree[RootPointer].RightPointer)



def GetOption() :
   print("1: add data")
   print("2: find data")
   print("3: traverse tree")
   print("4: end program")
   option = input("Enter your choice: ")
   return(option)


Tree, RootPointer, FreePtr = InitialiseTree()

Option = GetOption()
while Option != "4" :
   if Option == "1" :
      Data = input("Enter the value: ")
      Tree, RootPointer, FreePtr = InsertNode(Tree, RootPointer,
FreePtr, Data)
      TraverseTree(Tree, RootPointer)
   elif Option == "2" :
      Data = input("Enter search value: ")
      ThisNodePtr = FindNode(Tree, RootPointer, Data)
      if ThisNodePtr == NULLPOINTER :
         print("value not found")
      else :
         print("value found at ", ThisNodePtr)
      print(RootPointer, FreePtr)
      for i in range(8) :
         print(i, "  ", Tree[i].LeftPointer, "  ", Tree[i].Data, "
", Tree[i].RightPointer)
   elif Option == "3" :
      TraverseTree(Tree, RootPointer)
   Option = GetOption()
```

VB.NET

```
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 TreeNode
        Dim Data As String
        Dim LeftPointer, RightPointer As Integer
    End Structure
```

```
    Dim Tree(7) As TreeNode
    Dim RootPointer As Integer
    Dim FreePtr As Integer

    Sub InitialiseTree()
        RootPointer = NULLPOINTER          ' set start pointer
        FreePtr = 0                      ' set starting position of
free list
        For Index = 0 To 7               'link all nodes to make free
list
            Tree(Index).LeftPointer = Index + 1
            Tree(Index).RightPointer = NULLPOINTER
            Tree(Index).Data = ""
        Next
        Tree(7).LeftPointer = NULLPOINTER       'last node of free
list
    End Sub

    Function FindNode(SearchItem) As Integer
        Dim ThisNodePtr As Integer
        ThisNodePtr = RootPointer
        Try
            Do While ThisNodePtr <> NULLPOINTER And
Tree(ThisNodePtr).Data <> SearchItem
                If Tree(ThisNodePtr).Data > SearchItem Then
                    ThisNodePtr = Tree(ThisNodePtr).LeftPointer
                Else
                    ThisNodePtr = Tree(ThisNodePtr).RightPointer
                End If
            Loop
        Catch ex As Exception
        End Try
        Return ThisNodePtr
    End Function

    Sub InsertNode(NewItem)
        Dim NewNodePtr, ThisNodePtr, PreviousNodePtr As Integer
        Dim TurnedLeft As Boolean
        If FreePtr <> NULLPOINTER Then
            ' there is space in the array
            ' take node from free list and store data item
            NewNodePtr = FreePtr
            Tree(NewNodePtr).Data = NewItem
            FreePtr = Tree(FreePtr).LeftPointer
            Tree(NewNodePtr).LeftPointer = NULLPOINTER
            ' check if empty tree
            If RootPointer = NULLPOINTER Then
                RootPointer = NewNodePtr
            Else ' find insertion point
                ThisNodePtr = RootPointer
                Do While ThisNodePtr <> NULLPOINTER
                    PreviousNodePtr = ThisNodePtr
                    If Tree(ThisNodePtr).Data > NewItem Then
                        TurnedLeft = True
                        ThisNodePtr = Tree(ThisNodePtr).LeftPointer
                    Else
                        TurnedLeft = False
                        ThisNodePtr = Tree(ThisNodePtr).RightPointer
                    End If
                Loop
                If TurnedLeft Then
                    Tree(PreviousNodePtr).LeftPointer = NewNodePtr
```

```
                Else
                    Tree(PreviousNodePtr).RightPointer = NewNodePtr
                End If
            End If
        Else
            Console.WriteLine("no spce for more data")
        End If
    End Sub

    Sub TraverseTree(RootPointer)
        If RootPointer <> NULLPOINTER Then
            TraverseTree(Tree(RootPointer).LeftPointer)
            Console.WriteLine(Tree(RootPointer).Data)
            TraverseTree(Tree(RootPointer).RightPointer)
        End If
    End Sub

    Function GetOption()
        Dim Choice As Char
        Console.WriteLine("1: add data")
        Console.WriteLine("2: find data")
        Console.WriteLine("3: traverse tree")
        Console.WriteLine("4: 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 ThisNodePtr As Integer

        InitialiseTree()
        Choice = GetOption()
        Do While Choice <> "4"
            Select Case Choice
                Case "1"
                    Console.Write("Enter the value: ")
                    Data = Console.ReadLine()
                    InsertNode(Data)
                    TraverseTree(RootPointer)
                Case "2"
                    Console.Write("Enter search value: ")
                    Data = Console.ReadLine()
                    ThisNodePtr = FindNode(Data)
                    If ThisNodePtr = NULLPOINTER Then
                        Console.WriteLine("Value not found")
                    Else
                        Console.WriteLine("value found at: " &
ThisNodePtr)
                    End If
                    Console.WriteLine(RootPointer & "  " & FreePtr)
                    For i = 0 To 7
                        Console.WriteLine(i & "  " &
Tree(i).LeftPointer & "  " & Tree(i).Data & "  " &
Tree(i).RightPointer)
                    Next
                Case "3"
                    TraverseTree(RootPointer)
            End Select
            Choice = GetOption()
```

```
        Loop
    End Sub

End Module
```

Pascal

```
program linkedList;

{$APPTYPE CONSOLE}

uses
  SysUtils;

// NullPointer should be set to -1 if using array element with
index 0
  Const NULLPOINTER = -1;
    // Declare record type to store data and pointer
  type TreeNode = record
    Data : String;
    LeftPointer, RightPointer : Integer;
  End;

  var Tree: array[0..7] of TreeNode;
      RootPointer, FreePtr : Integer;

procedure InitialiseTree;
var Index : integer;
begin
   RootPointer := NULLPOINTER;          // set Root pointer
   FreePtr := 0;                      // set starting position of
free list
   for Index := 0 to 7 do
     begin            // link all nodes to make free list
      Tree[Index].LeftPointer := Index + 1;
      Tree[Index].RightPointer := NULLPOINTER;
      Tree[Index].Data := '';
     end;
   Tree[7].LeftPointer := NULLPOINTER;
end;

procedure InsertNode(NewItem : string);
var NewNodePtr, ThisNodePtr, PreviousNodePtr : integer;
TurnedLeft : Boolean;
begin
   if FreePtr <> NULLPOINTER
     then
       begin
         // there is space in the array
         // take node from free list and store data item
         NewNodePtr := FreePtr;
         Tree[NewNodePtr].Data := NewItem;
         FreePtr := Tree[FreePtr].LeftPointer;
         Tree[NewNodePtr].LeftPointer := NULLPOINTER;
         // check if empty tree
         if RootPointer = NULLPOINTER
           then
             // insert new node at root
             RootPointer := NewNodePtr
           else // find insertion point
             begin
               ThisNodePtr := RootPointer;
               while ThisNodePtr <> NULLPOINTER do // while not a
leaf node
                 begin
                   PreviousNodePtr := ThisNodePtr; // remember this
node
```

```
                    if Tree[ThisNodePtr].Data > NewItem
                      then
                        begin
                          TurnedLeft := True; // follow left pointer
                          ThisNodePtr :=
Tree[ThisNodePtr].LeftPointer;
                        end
                      else
                        begin
                          TurnedLeft := False;
                          ThisNodePtr :=
Tree[ThisNodePtr].RightPointer;
                        end;
                  end;
                if TurnedLeft
                  then
                    Tree[PreviousNodePtr].LeftPointer := NewNodePtr
                  else
                    Tree[PreviousNodePtr].RightPointer :=
NewNodePtr;
              end
       end
     else
       writeln('no space for more data');
end;

function FindNode(SearchItem : string) : integer;
var ThisNodePtr : integer;
begin
   ThisNodePtr := RootPointer; // start at the root of the tree
   while (ThisNodePtr <> NULLPOINTER) and (Tree[ThisNodePtr].Data
<> SearchItem) do
      // while there is a pointer to follow and search item not
found
      if Tree[ThisNodePtr].Data > SearchItem
        then
          ThisNodePtr := Tree[ThisNodePtr].LeftPointer // follow
left pointer
        else
         ThisNodePtr := Tree[ThisNodePtr].RightPointer; // follow
right pointer
   FindNode := ThisNodePtr;
end;

procedure TraverseTree(RootPointer : integer);
begin
   if RootPointer <> NULLPOINTER
     then
       begin
         TraverseTree(Tree[RootPointer].LeftPointer);
         writeln(Tree[RootPointer].Data);
         TraverseTree(Tree[RootPointer].RightPointer);
       end;
end;


function GetOption :  char;
var Response : char;
begin
   writeln('1: add data');
   writeln('2: find data');
   writeln('3: traverse tree');
   writeln('4: end program');
   Write('Enter your choice: ');
   ReadLn(Response);
   GetOption := Response;
end;
```

```
procedure Main();
var Choice : Char;
    Data : String;
    ThisNodePtr, i : Integer;
  begin
    InitialiseTree;
    Choice := GetOption();
    While Choice <> '4' do
      begin
        Case Choice of
        '1': begin
               Write('Enter the value: ');
               ReadLn(Data);
               InsertNode(Data);
               TraverseTree(RootPointer);
             end;
        '2': begin
               Write('Enter search value: ');
               ReadLn(Data);
               ThisNodePtr := FindNode(Data);
               if ThisNodePtr = NULLPOINTER
                 then
                   writeln('Value not found')
                 else
                   writeln('value found at: ', ThisNodePtr);
               WriteLn(RootPointer , ' ',FreePtr);
               For i := 0 To 7  do
                 WriteLn(i ,Tree[i].LeftPointer ,'  ' ,
Tree[i].Data , '  ' , Tree[i].RightPointer);
             end;
       '3' : TraverseTree(RootPointer);

       end;
       Choice := GetOption();
     end;
  End;

begin
   main;
end.
```

Task 23.08

P332 Dictionary pseudocode

<code>

```
PROCEDURE AddEntry(English, French)
   DECLARE Entry : DictionaryEntry
   Entry.Key ← English
   Entry.Value ← French
   Index ← Hash(Entry.Key)
   WHILE EnglishFrench[Index] NOT empty
      Index ← Index + 1 // go to next slot
      IF Index > 999 // beyond table boundary?
         THEN // wrap around to beginning of table
            Index ← 0
```

```
      ENDIF
   ENDWHILE
   EnglishFrench[Index] ← Entry
ENDPROCEDURE
```

</code>
<code>

```
FUNCTION FrenchFor(EnglishWord) RETURNS STRING
   Index ← Hash(EnglishWord)
   WHILE (EnglishFrench[Index].Key <> EnglishWord)
      AND (EnglishFrench[Index] NOT empty)
      Index ← Index + 1 // go to next slot
      IF Index > 999 // beyond table boundary?
         THEN // wrap around to beginning of hash table
            Index ← 0
      ENDIF
   ENDWHILE
   IF EnglishFrench[Index] NOT empty // if English word found
      THEN
         RETURN EnglishFrench[Index].Value // return the
French word
      ELSE
         RETURN "Word does not exist in dictionary"
   ENDIF
ENDFUNCTION
```

</code>

Exam style questions

1 a

```
FUNCTION FindName(s : STRING) RETURNS INTEGER
   Index = -1
   First ← 0
   Last ← 50
   WHILE (Last >= First) AND Index = -1
      Middle ← (First + Last) DIV 2
      IF Names[Middle] = s
         THEN
            Index ← Middle
```

```
            ELSE
               IF Names[Middle] < s
                  THEN
                     Last ← Middle + 1
                  ELSE
                     First ← Middle - 1
               ENDIF
         ENDIF
      ENDWHILE
ENDFUNCTION
```

b the binary search does not work if the data in the array being searched is not sorted in ascending order.

c i the function returns the index of the search item

ii the function returns -1

2 a i and ii

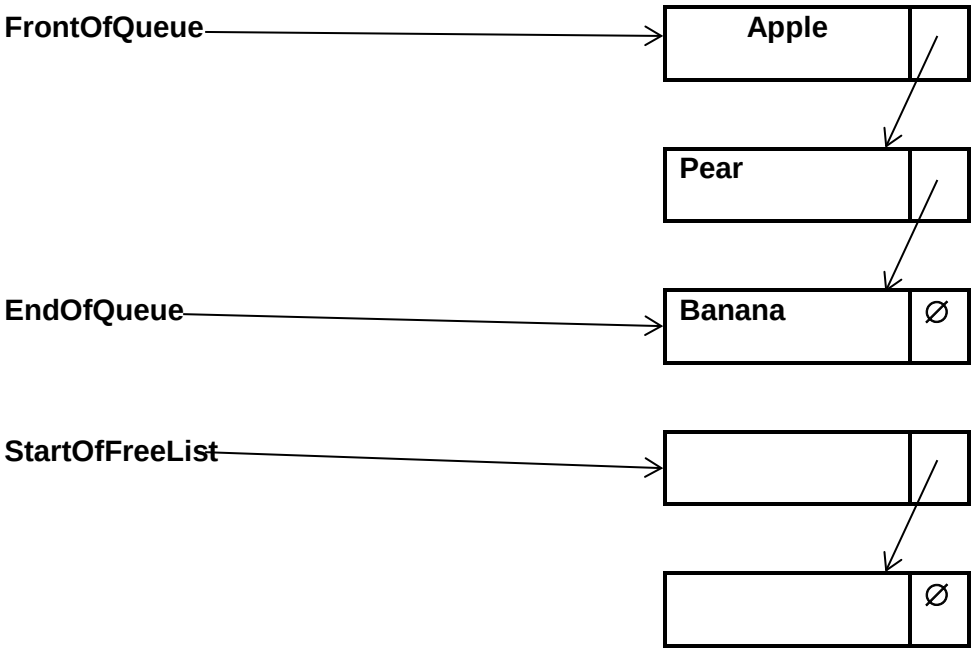

Figure 23.04

b i

Python	`class Node :`
	`  def __init__(self) :`
	`  self.Data = ""`
	`  self.Pointer = 0`

VB.NET	`Structure Node`
	`  Data As String`
	`  Pointer As Integer`
	`End Structure`

Pascal	`type Node = record` `  Data : String;`

```
   Pointer : Integer;
end;
```

ii

Python	`Queue = [Node() for i in range(51)]`
	
	`FrontOfQueue = 0`
	`EndOfQueue = 0`
	`StartOfFreeList = 0`
	`for i in range(1, 50) :`
	`  Queue[i].Pointer = i + 1`

VB.NET	`Dim Queue(50) As Node`
	
	`FrontOfQueue = 0`
	`EndOfQueue = 0`
	`StartOfFreeList = 1`
	`For i = 1 to 49`
	`  Queue(i) = i + 1`
	`Next`
	`Queue(50) = 0`

Pascal	`var Queue : array[1..50] of Node;`
	
	`FrontOfQueue := 0;`
	`EndOfQueue := 0;`
	`StartOfFreeList := 1;`
	`for i := 1 to 49 do`
	`  Queue[i] := i + 1;`
	`Queue[50] := 0;`

c i

Identifier	Data Type	Description
`NullPointer`	`INTEGER`	constant set to -1
`Queue`	`Node`	Array to store queue data
`NewItem`	`STRING`	Value to be added
`StartOfFreeList`	`INTEGER`	Pointer to next free node in array
`FrontOfQueue`	`INTEGER`	Pointer to first node in queue
`EndOfQueue`	`INTEGER`	Pointer to last node in queue
`NewNodePointer`	`INTEGER`	Pointer to node to be added

ii

```
PROCEDURE JoinQueue(NewItem : STRING)
```

```
   // Report error if no free nodes remaining
   IF StartOfFreeList = NullPointer
      THEN
         Report Error
      ELSE
         // new data item placed in node at start of free list
         NewNodePointer ← StartOfFreeList
         Queue[NewNodePointer].Data ← NewItem
         // adjust free list pointer
         StartOfFreeList ← Queue[NewNodePointer].Pointer
         Queue[NewNodePointer].Pointer ← NullPointer
         // if first item in queue then adjust front of queue
pointer
         IF FrontOfQueue = NullPointer
            THEN
               FrontOfQueue ← NewNodePointer
         ENDIF
         // new node is new end of queue
         Queue[EndOfQueue].Pointer ← NewNodePointer
         EndOfQueue ← NewNodePointer
   ENDIF
ENDPROCEDURE
```