10(a)

One mark per mark point (**Max 3**)
MP1 Correct declaration of constant `Maximum`
MP2 Both correctly declared integers
MP3 Correct array declaration

Example answer

```
CONSTANT Maximum = 100
DECLARE Base : INTEGER
DECLARE Top : INTEGER
DECLARE StackArray : ARRAY[1:100] OF STRING
```

10(b)

One mark per mark point (**Max 3**)
MP1 Correct procedure definition structure
MP2 Base pointer with appropriate value (0 or 1)
MP3 Top pointer with appropriate value (–1, 0, 1)

Example answer

```
PROCEDURE InitialiseStack()
   Base ← 0
   Top ← 0
ENDPROCEDURE
```

13

One mark per mark point (**Max 7**)
MP1 `LDM #250` seen
MP2 Correct use of `LDD 563`
MP3 Correct use of `ADD` with labelled address `X`
MP4 Correct use of `SUB` with address `899`
MP5 At least **one** correct use of `STO`

Opcode	Operand
LDM	#250
STO	X
LDD	563
STO	Y
ADD	X
SUB	899
STO	Total

13

MP6 Correct setting up of at least **one** labelled address and its value
MP7 Correct setting up of remaining **two** labelled addresses and their values

Label	Contents
X:	250
Y:	125
Total:	312

4(a)

1 mark for each correct statement

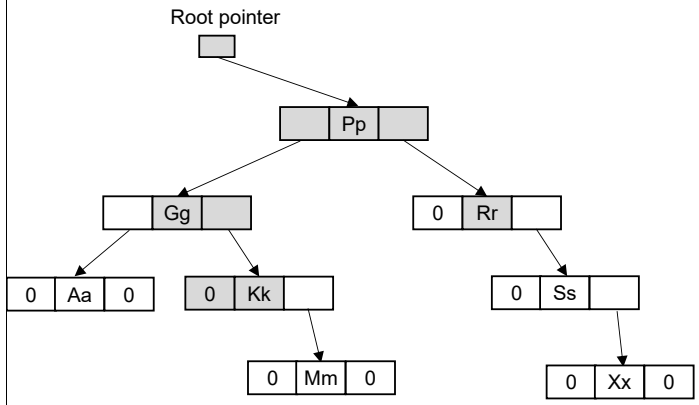
X = B NOR (A NOR C)

Y = (B AND C) XOR NOT D

4(b)

1 mark for 1st 4 rows
1 mark for 2nd 4 rows

A	B	C	Working space	X
0	0	0		**1**
0	0	1		**1**
0	1	0		**1**
0	1	1		**0**
1	0	0		**0**
1	0	1		**1**
1	1	0		**1**
1	1	1		**1**

11(a)	**One** mark per mark point MP1 Any two nodes added correctly with correct data (Aa, Mm, Ss, Xx) MP2 Remaining two nodes added with correct data (Aa, Mm, Ss, Xx) and all nodes with connecting arrows starting from correct pointer boxes MP3 Correct null pointers (0) added throughout MP4 … with no entries in other pointer boxes and all nodes correctly positioned and connected. 
11(b)	**One** mark for feature and **one** mark for example (**Max 2**) • Recursion is beneficial for algorithms when a problem naturally breaks down into smaller versions of itself. • … such as calculating a factorial / mathematical series / Fibonacci / compound interest / tree traversal / evaluation of RPN expressions.

8(a) Example solution:

```
PROCEDURE Assign(ThisRole : STRING, ThisPlayer : INTEGER)
   DECLARE Index : INTEGER
   DECLARE Done : BOOLEAN

   Done ← FALSE
   Index ← 1

   WHILE Index < 46 AND Done = FALSE
      IF Character[Index].Player = 0 AND __
         Character[Index].Role = ThisRole THEN
         Character[Index].Player ← ThisPlayer
         Done ← TRUE
      ELSE
         Index ← Index + 1
      ENDIF
   ENDWHILE

   IF Done = TRUE THEN
      OUTPUT Character[Index].Name, " the ",__
             Character[Index].Role, __
             " has been assigned to player ", ThisPlayer
   ELSE
      OUTPUT "No characters with this role are available"
   ENDIF

ENDPROCEDURE
```

Mark as follows:
MP1 Loop until 'found' or all 45 elements considered
MP2 Test of `Player` field – i.e. not value **in a loop**

MP3 ... `AND Role` **–** i.e. match for **`ThisRole`** parameter **in a loop**

MP4 If available character found, assign `ThisPlayer` to the character **in a loop**
MP5 When character found set termination condition/flag
MP6 **Both** OUTPUT messages logically correctly placed

MP7 **Both** OUTPUT statements correctly formed

8(b)

Example solution:

```
PROCEDURE Save()
   DECLARE Index    : INTEGER
   DECLARE Line     : STRING
   CONSTANT SEP = '^'

   OPENFILE "SaveFile.txt" FOR WRITE
   FOR Index ← 1 TO 45
      Line ← NUM_TO_STR(Character[Index].Player) & SEP
      Line ← Line & Character[Index].Role & SEP
      Line ← Line & Character[Index].Name & SEP
      Line ← Line & NUM_TO_STR(Character[Index].Level)
      WRITEFILE "SaveFile.txt", Line
   NEXT Index

   CLOSEFILE "SaveFile.txt"
ENDPROCEDURE
```

Mark as follows:
MP1 Declaration of local integer for `Index` (**and** string type for `Line`)
MP2 Open `"SaveFile.txt"` in write mode **and** subsequently close
MP3 Loop through 45 elements

MP4 Attempt to form `Line` - four fields **in a loop**
MP5 Correct use of `NUM_TO_STR()x2 (Player and Level)` **in a loop**
MP6 Correct use of **three** &<separator>& `strings` **in a loop**

MP7 Line from MP4 written to file **in a loop**

8(c)

MP1 Encode `Status` as a character / string
MP2 Append the '^' separator and the character/string

8(d)

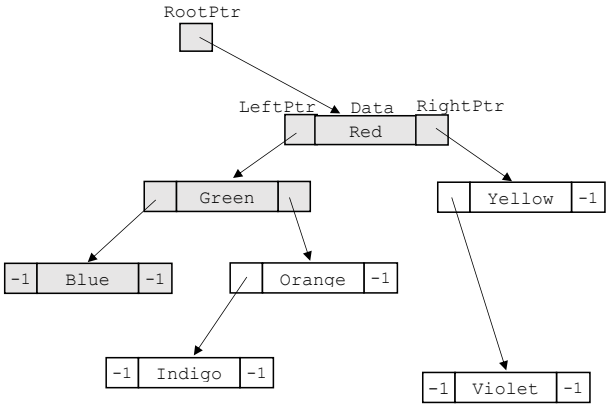
MP1 Method:
Create a filename **suffix** which is **incremented** for each file save

MP2 Example:
SaveFile01.txt, SaveFile02.txt

11(a)

One mark per mark point (**Max 4)**

MP1 Four additional nodes with correct data values
MP2 Correct null pointers in all added nodes (6) with no extra null pointers where the arrow points to the next node
MP3 Correct arrows to represent pointers joining parent nodes to child nodes
MP4 All nodes in correct order and no extra data added to pointers.



11(b)

One mark per mark point (**Max 4)**

MP1 Correct `Red` and `Green` rows
MP2 Correct `Yellow` and `Blue` rows
MP3 Correct `Orange`, `Indigo` and `Violet` rows
MP4 Correct `FreePtr` with blank row 7

RootPtr
0

FreePtr
7

Index	LeftPtr	Data	RightPtr
0	1	Red	2
1	3	Green	4
2	6	Yellow	-1
3	-1	Blue	-1
4	5	Orange	-1
5	-1	Indigo	-1
6	-1	Violet	-1
7			

11(c)

One mark for any correct row (**Max 4**)

```
FUNCTION SearchTree(Item : STRING) RETURNS INTEGER
   NowPtr ← RootPtr
   WHILE NowPtr <> -1
      IF BinTree[NowPtr].Data > Item THEN
         NowPtr ← BinTree[NowPtr].LeftPtr
      ELSE
         IF BinTree[NowPtr].Data < Item THEN
            NowPtr ← BinTree[NowPtr].RightPtr
         ELSE
            RETURN NowPtr
         ENDIF
      ENDIF
   ENDWHILE
   RETURN NowPtr
ENDFUNCTION
```


3(a)

1 mark each

- `LinkedList` declared as **2D** array with (min) 20 × 2 elements (Integer) with all data initialised to `-1`, all nodes linked correctly
- (Global) `FirstNode` (Int) initialised as `-1` and (global) `FirstEmpty` (Int) initialised as `0`

VB.NET

```
Dim LinkedList(20, 2) As Integer
Dim FirstNode As Integer
Dim FirstEmpty As Integer

Sub Main(args As String())
        FirstNode = -1
        FirstEmpty = 0
        For x = 0 To 18
            LinkedList(x, 0) = -1
            LinkedList(x, 1) = x + 1

        Next
        LinkedList(19, 0) = -1
        LinkedList(19, 1) = -1
End Sub
```


3(a)

Python

```
LinkedList = [] #global
FirstNode = -1
FirstEmpty = 0
for x in range(0, 19):
    LinkedList.append([-1, x + 1])
LinkedList[19][0] = -1
LinkedList[19][1] = -1
```

Java

```
private static Integer[][] LinkedList = new Integer[20][2];
private static Integer FirstNode;
private static Integer FirstEmpty;
public static void main(String args[]){
      FirstNode = -1;
      FirstEmpty = 0;
      for(Integer X = 0; X < 19; X++){
            LinkedList[X][0] = -1;
            LinkedList[X][1] = X + 1;
      }
       LinkedList[19][0] = -1;
       LinkedList[19][1] = -1;
}
```

3(b)

1 mark each to max 6

- Procedure header (and end) taking (min) 5 data items as input from the user
- Checking if linked list is full (`FirstEmpty = -1`) …
- …ending procedure/loop/not doing anything further

- (otherwise) `LinkedList[FirstEmpty, 0]` = data input
- `LinkedList[FirstEmpty, 1] = FirstNode`
- `FirstNode = FirstEmpty`
- `FirstEmpty = LinkedList[FirstEmpty, 1]` **before** any update to `FirstEmpty` 's pointer

e.g.
Python

```
def InsertData():
    global LinkedList
    global FirstNode
    global FirstEmpty
    for _ in range(5):
        if FirstEmpty != -1:
            nextEmpty = LinkedList[FirstEmpty][1]
            LinkedList[FirstEmpty][0] = int(input("Value: "))
            LinkedList[FirstEmpty][1] = FirstNode
            FirstNode = FirstEmpty
            FirstEmpty = nextEmpty
```


3(b)

VB.NET

```
Sub InsertData()
    Dim NewItem As Integer
    Dim NextEmpty As Integer

    For x = 0 To 4
        Console.WriteLine("Enter the next number")
        NewItem = Console.ReadLine()

        If FirstEmpty = -1 Then
            x = 5
        Else
            NextEmpty = LinkedList(FirstEmpty, 1)
            LinkedList(FirstEmpty, 0) = NewItem
            LinkedList(FirstEmpty, 1) = FirstNode
            FirstNode = FirstEmpty
            FirstEmpty = NextEmpty
        End If

    Next x
End Sub
```

3(b)

Java

```
public static void InsertData(){
      Integer NewItem;
      Integer CurrentPointer = 0;
      Integer PreviousPointer = 0;
      Scanner scanner = new Scanner(System.in);
      Integer NextEmpty;

      for(Integer X = 0; X < 5; X++){
            System.out.println("Enter the next number");
            NewItem = Integer.parseInt(scanner.nextLine());
            if(FirstEmpty == -1){
                  X = 5;
            }else{
                  NextEmpty = LinkedList[FirstEmpty][1];
                  LinkedList[FirstEmpty][0] = NewItem;
                  LinkedList[FirstEmpty][1] = FirstNode;
                  FirstNode = FirstEmpty;
                  FirstEmpty = NextEmpty;
            }
      }
}
```


3(c)(i)

1 mark each

- Procedure header (and end) starting with node at index `FirstNode` and outputting data `LinkedList[FirstNode,0]`
- Following pointers until **end** reached and outputting data for each node

Python

```
def OutputLinkedList():
    global LinkedList
    global FirstNode
    global FirstEmpty
    CurrentPointer = FirstNode
    Flag = True
    while Flag:
        print(LinkedList[CurrentPointer][0])
        CurrentPointer = LinkedList[CurrentPointer][1]
        if CurrentPointer == -1:
            Flag = False
```

VB.NET

```
Sub OutputLinkedList()

        Dim CurrentPointer As Integer = FirstNode
        Dim Flag As Boolean = True
        While Flag
            Console.WriteLine(LinkedList(CurrentPointer, 0))
            CurrentPointer = LinkedList(CurrentPointer, 1)
            If CurrentPointer = -1 Then
                Flag = False
            End If
        End While
```

3(c)(i)

```
End Sub
```

Java

```
public static void OutputLinkedList(){
      Integer CurrentPointer = FirstNode;
      Boolean Flag = true;
            while(Flag){
            System.out.println(LinkedList[CurrentPointer][0]);
                  CurrentPointer = LinkedList[CurrentPointer][1];
            if(CurrentPointer == -1){Flag = false;}
      }
}
```

3(c)(ii)

1 mark for calling `InsertData()` then `OutputLinkedList()`

Python

```
InsertData()
OutputLinkedList()
```

VB.NET

```
InsertData()
OutputLinkedList()
```

Java

```
InsertData();
OutputLinkedList();
```

3(c)(iii)

1 mark for inputs of 5 1 2 3 8 and output of 8 3 2 1 5

3(d)(i)

1 mark each to max 5

- Procedure header (and end) with parameter
- Checking data in `FirstNode` against parameter …
- … (if found) updating `FirstNode` to `LinkedList[FirstNode, 1]`
- (Otherwise) following pointers in loop/recursive call …
- …comparing to data to remove each time
- … storing previous pointer through each loop…
- … when found, updating previous pointer to found node's pointer
- Adding deleted node to end of/start of empty list (and updating `FirstEmpty` if needed)

Python

```
def RemoveData(ItemToRemove):
    global LinkedList
    global FirstNode
    global FirstEmpty

    if LinkedList[FirstNode][0] == ItemToRemove:
        NewFirst = LinkedList[FirstNode][1]
        LinkedList[FirstNode][1] = FirstEmpty
        FirstEmpty = FirstNode
        FirstNode = NewFirst

    else:
        if FirstNode != -1:
            CurrentPointer = FirstNode
            PreviousNode = -1
            while(ItemToRemove != LinkedList[CurrentPointer][0] and CurrentPointer != -1):
                    PreviousNode = CurrentPointer
                    CurrentPointer = LinkedList[CurrentPointer][1]

            if ItemToRemove == LinkedList[CurrentPointer][0]:
                LinkedList[PreviousNode][1] = LinkedList[CurrentPointer][1]
                LinkedList[CurrentPointer][0] = -1
                LinkedList[CurrentPointer][1] = FirstEmpty
                FirstEmpty = CurrentPointer
```

3(d)(i)

VB.NET

```
Sub RemoveData(ItemToRemove)
    If LinkedList(FirstNode, 0) = ItemToRemove Then
        Dim NewFirst As Integer = LinkedList(FirstNode, 1)
        LinkedList(FirstNode, 1) = FirstEmpty
        FirstEmpty = FirstNode
        FirstNode = NewFirst
    Else
        If FirstNode <> -1 Then
            Dim CurrentPointer As Integer = FirstNode
            Dim PreviousNode As Integer = -1
            Dim Flag As Boolean = True
            Dim Found As Boolean = False
            While Flag And Not (Found)
                If (CurrentPointer <> -1) Then
                    If (ItemToRemove <> LinkedList(CurrentPointer, 0)) Then
                        PreviousNode = CurrentPointer
                        CurrentPointer = LinkedList(CurrentPointer, 1)
                    Else
                        Found = True
                    End If
                Else
                    Flag = False
                End If
            End While

            If Found Then
                LinkedList(PreviousNode, 1) = LinkedList(CurrentPointer, 1)
                LinkedList(CurrentPointer, 0) = -1
                LinkedList(CurrentPointer, 1) = FirstEmpty
                FirstEmpty = CurrentPointer
            End If
        End If
    End If
End Sub
```


3(d)(i)

Java

```
public static void RemoveData(Integer ItemToRemove){
      Integer CurrentPointer = 0;
      Integer PreviousNode = 0;
      Integer NewFirst = 0;

      if(LinkedList[FirstNode][0] == ItemToRemove){
            NewFirst = LinkedList[FirstNode][1];
            LinkedList[FirstNode][1] = FirstEmpty;
            FirstEmpty = FirstNode;
            FirstNode = NewFirst;

      }else{
            if (FirstNode != -1){
                  CurrentPointer = FirstNode;
                  PreviousNode = -1;
                     while(ItemToRemove != LinkedList[CurrentPointer][0]  && CurrentPointer
!= -1){
                              PreviousNode = CurrentPointer;
                           CurrentPointer = LinkedList[CurrentPointer][1];

                  }
                  if(ItemToRemove == LinkedList[CurrentPointer][0]){
                        LinkedList[PreviousNode][1] = LinkedList[CurrentPointer][1];
                        LinkedList[CurrentPointer][0] = -1;
                        LinkedList[CurrentPointer][1] = FirstEmpty;
                        FirstEmpty = CurrentPointer;

                  }
            }
      }
}
```

3(d)(ii)	1 mark for calling `RemoveData(5)`, outputting `"After"`, calling `OutputLinkedList()`
	
	Python
	`LinkedList = []`
	`FirstNode = -1`
	`FirstEmpty = 0`
	`for x in range(0, 19):`
	`  LinkedList.append([-1, x + 1])`
	`InsertData()`
	`OutputLinkedList()`
	`RemoveData(5)`
	`print("After")`
	`OutputLinkedList()`
	
	VB.NET
	`Sub Main(args As String())`
	`  FirstNode = -1`
	`  FirstEmpty = 0`
	`  For x = 0 To 19`
	`  LinkedList(x, 0) = -1`
	`  LinkedList(x, 1) = x + 1`
	`  Next`
	`  InsertData()`
	`  OutputLinkedList()`
	`  RemoveData(5)`
	`  Console.WriteLine("After")`
	`  OutputLinkedList()`
	`End Sub`

3(d)(ii)	Java
	`public static void main(String args[]){`
	`  FirstNode = -1;`
	`  FirstEmpty = 0;`
	`  for(Integer X = 0; X < 20; X++){`
	`  LinkedList[X][0] = -1;`
	`  LinkedList[X][1] = X + 1;`
	`  }`
	`  InsertData();`
	`  OutputLinkedList();`
	
	`  RemoveData(5);`
	`  System.out.println("After");`
	`  OutputLinkedList();`
	`}`

3(d)(iii)	1 mark for input and output. Test data 1: Input 5 6 8 9 5 'After' Output: 9 8 6 5 Test data 2: Input 10 7 8 5 6 "After" Output: 6 8 7 10

6(a)	Example solution:
	
	`Function Trim(Name : STRING) RETURNS STRING`
	`  CONSTANT Dots = "..."`
	`  CONSTANT Space = " "`
	
	`  IF LENGTH(Name) <= 16 THEN`
	`  RETURN Name`
	`  ENDIF`
	
	`  // Otherwise it has to be trimmed`
	
	`  WHILE LENGTH(Name) > 13`
	`  REPEAT`
	`  Name ← LEFT(Name, LENGTH(Name) - 1) // strip last char`
	`  UNTIL RIGHT(Name, 1) = Space // back to SPACE`
	`  ENDWHILE`
	
	`  Name ← LEFT(Name, LENGTH(Name) - 1) // remove the space`
	`  Name ← Name & Dots`
	`  RETURN Name`
	
	`ENDFUNCTION`
	
	Mark as follows:
	1. Function heading, ending, parameter and return type
	2. If length of original string <= 16 then return original string
	3. Any Conditional loop...
	4. until string is short enough
	5. Inner conditional loop / second condition to identify word(s) to remove from the end of string // Check to identify word(s) to remove from the end of string
	6. Attempt to strip characters back to space
	7. Correct removal of word/words from end of string **and** remove final space
	8. Concatenate `Dots` and return result
	
	Max 7 marks

6(b)(i)	A (very) large **file** is created // redundant zeroes are stored in the file
6(b)(ii)	One mark for: • Values are delimited by a special character / a separator character • First character indicates **sample** length **Max 1 mark**
6(b)(iii)	The **algorithm** to **store / extract / separate** the individual values is more complex / takes longer to execute / run / process NE Algorithm is more complicated