3	`FUNCTION Pop() RETURNS PopData`
	`  DECLARE ThisPop : PopData` MP 2
	`  IF SP < 1 // SP = 0 THEN` MP 3
	`  PopData.Exists ← FALSE // Stack is empty` MP 1
	`  ELSE`
	`  PopData.Data ← ThisStack[SP]` MP 4
	`  PopData.Exists ← TRUE` MP 1
	`  SP ← SP – 1` MP 5
	`  ENDIF`
	`  RETURN ThisPop`
	`ENDFUNCTION`
	
	Mark as follows:
	
	MP 1 One mark for **both** assignments to `PopData.Exists` (`FALSE` **and** `TRUE`)
	MP 2, 3, 4 ,5 One mark for each of remaining four bold parts

9(a)(i)	**One** mark for each correctly completed line (**Max 5**)
	
	`FUNCTION Pop() RETURNS STRING`
	`  DECLARE DataItem : STRING`
	`  DataItem ← ""`
	`  IF Top > –1 // Top >= Base THEN`
	`  DataItem ← StackArray[Top]`
	`  Top ← Top – 1`
	`  ELSE`
	`  DataItem ← "You cannot remove data; the stack is empty"`
	`  ENDIF`
	`  RETURN DataItem // StackArray[Top + 1]`
	`ENDFUNCTION`

9(a)(ii)	`OUTPUT "The data removed from the stack is ", Pop()`
9(b)	**One** mark per mark point (**Max 3**) MP1 A recursive algorithm must call itself / have a general case MP2 It must have a base case / have a stopping condition MP3 It must change its state and move towards the base case

2(a)	1 mark each
	• `Queue` declared as a (global) 1D array of 100 string elements all initialised to `""`
	• `QueueHead` and `QueueTail` initialised with `-1`, `NumberItems` initialised to `0`
	
	Example program code
	Java
	`public static String[] Queue = new String[100];`
	`public static Integer QueueHead;`
	`public static Integer QueueTail;`
	`public static Integer NumberItems;`
	`for(int x = 0; x < 100; x++){`
	`  Queue[x] = "";`
	`}`
	`QueueHead = -1;`
	`QueueTail = -1;`
	`NumberItems = 0;`
	
	VB.NET
	`  Dim Queue(99) As String`
	`  Dim QueueHead As Integer`
	`  Dim QueueTail As Integer`
	`  Dim NumberItems As Integer`
	`For x = 0 To 99`
	`  Queue(x) = ""`
	`Next`
	`QueueHead = -1`
	`QueueTail = -1`
	`NumberItems = 0`

2(a)	Python
	`global Queue, QueueHead, QueueTail, NumberItems`
	`Queue = []`
	`  for x in range(100):`
	`  Queue.append("")`
	`  QueueHead = -1`
	`  QueueTail = -1`
	`  NumberItems = 0`

2(b)

1 mark each

- Function header (and end where appropriate) taking one (string) parameter **and** checking full **and** returning `FALSE`
- (otherwise) Storing parameter in `QueueTail + 1`
- Incrementing `QueueTail` and `NumberItems`
- (Dealing with first element) If `QueueHead = -1` store `0` in `QueueHead` / increment `QueueHead`
- Return `TRUE` in **all** cases when inserted

Example program code.

Java

```
      public static Boolean Enqueue(String TheData){
            if(QueueHead == -1){
                  Queue[0] = TheData;
                  QueueHead = 0;
                  QueueTail = 0;
                  NumberItems++;
                  return true;
            }else if(QueueTail >= 99){
                  return false;
            }else{
                  Queue[QueueTail+1] = TheData;
                  QueueTail++;
                  NumberItems++;
                  return true;
            }

      }
```

2(b)

VB.NET

```
Function Enqueue(TheData)
    If QueueHead = -1 Then
        Queue(0) = TheData
        QueueHead = 0
        QueueTail = 0
        NumberItems += 1
        Return True
    ElseIf QueueTail >= 99 Then
        Return False
    Else
        Queue(QueueTail + 1) = TheData
        QueueTail += 1
        NumberItems += 1
        Return True
    End If
End Function
```

Python

```
def Enqueue(TheData):
    global Queue, QueueHead, QueueTail, NumberItems
    if(QueueHead == -1){
         Queue[0] = TheData
         QueueHead = 0
         QueueTail = 0
         NumberItems +=1
         return True
    elif QueueTail >= 99:
        return False
    else:
        Queue[QueueTail+1] = TheData
        QueueTail +=1
        NumberItems +=1
        return True
```


2(c)

1 mark each

- Function header (and end), checking if `Queue` is empty (`NumberItems = 0` or `QueueHead > QueueTail`) **and** returning "`False`"
- (otherwise) returning `Queue[QueueHead]`
- Incrementing `QueueHead`, decrementing `NumberItems`

Example program code

Java

```
public static String Dequeue(){
      String ReturnValue;
      if(NumberItems == 0){
            return "False";
      }else{
            ReturnValue = Queue[QueueHead];
            QueueHead++;
            NumberItems--;
            return ReturnValue;
      }
}
```

VB.NET

```
Function Dequeue()

    If NumberItems = 0 Then
        Return "False"
    Else
        Dequeue = Queue(QueueHead)
        QueueHead += 1
        NumberItems -= 1

    End If
End Function
```

2(c)

Python

```
def Dequeue():
    global Queue, QueueHead, QueueTail, NumberItems
    if NumberItems == 0:
        return "False"
    else:
        ReturnData = Queue[QueueHead]
        QueueHead += 1
        NumberItems -=1
        return ReturnData
```

2(d)

1 mark each to max 5

- Procedure header (and end where appropriate)
- Opening the file **and** closing the file (in appropriate place)
- Looping until EOF // looping through each line …
- … read in each value …
- … calling `Enqueue()` with read in value and store/use return value
- Exception handling with try except and appropriate output with all file access within try

Example program code

Java

```
public static void ReadData(){
          Boolean ReturnValue;
          Boolean FinishLoop = false;
          try{
               Scanner Scanner1 = new Scanner(new File("BinaryData.txt"));
               while(Scanner1.hasNextLine() && FinishLoop == false){
                    ReturnValue = Enqueue(Scanner1.nextLine());
                    if(ReturnValue.equals("False")){
                         FinishLoop = true;
                    }
               }
               Scanner1.close();
          } catch(FileNotFoundException ex){
               System.out.println("No file found");
          }
     }
```

2(d)

VB.NET

```
Sub ReadData()
     Dim ReturnValue As String
     Dim DataReader As New System.IO.StreamReader("BinaryData.txt")
     Dim FinishLoop As Boolean = True
     Do Until DataReader.EndOfStream Or FinishLoop = False
         ReturnValue = Enqueue(DataReader.ReadLine())
         If ReturnValue = False Then
             FinishLoop = False
         End If
     Loop
     DataReader.Close()
 End Sub
```

Python

```
def ReadData():

    TheFile = open("BinaryData.txt")
    for Line in TheFile:
        ReturnValue = Enqueue(Line.strip())
        if ReturnValue == False:
            break

    TheFile.close()
```


2(e)

1 mark each

- Procedure header (and end where appropriate), storing final string compressed data in global variable
- Call `Dequeue()` **and** storing return value …
- … repeatedly until the return value == "False"
- Comparing new value with previous …
- … maintaining counter of number of occurrences …
- … appending digit and number of occurrences to a string without overriding

Example program code

Java

```
public static void Compress(){
          String First = Dequeue();
          String NewLine = "";
          Integer Count;
          String NextChar;
          while(NumberItems > 0 && First != "False"){
               Count = 1;
               NextChar = Dequeue();
               while(NextChar.equals(First)){
                    Count++;
                    First = NextChar;
                    NextChar = Dequeue();
               }
               NewLine = First + Count;
               NewString = NewString + NewLine;
               First = NextChar;
          }}
```

2(e)

VB.NET

```
Sub Compress()
     Dim First As String = Dequeue()
     Dim NewLine As String = ""
     Dim Count As Integer
     Dim NextChar As String
     While NumberItems > 0 And First <> "False"
         Count = 1
         NextChar = Dequeue()

         While NextChar = First
             Count += 1
             First = NextChar
             NextChar = Dequeue()
         End While

         NewLine = First & Count
         NewString = NewString & NewLine
         First = NextChar
     End While
```

Python

```
def Compress():
    global NewString
    First = Dequeue()
    NewString = ""
    while NumberItems > 0 and First != "False":
        Count = 1
        NextChar = Dequeue()
        while NextChar == First:
            Count += 1
            First = NextChar
            NextChar = Dequeue()
        NewLine = First + str(Count)
        NewString = NewString + NewLine
        First = NextChar
```

2(f)(i)

1 mark each
- Calling `ReadData()` then `Compress()`
- Outputting compressed string

Example program code

Java

```
public static void main(String args[]){
           NewString = "";
           for(int x = 0; x < 100; x++){
                     Queue[x] = "";
                        }
                     QueueHead = -1;
                     QueueTail = -1;
                    NumberItems = 0;

           ReadData();
           Compress();
           System.out.println(NewString);
     }
```

VB.NET

```
Sub Main()
     NewString = ""
     For x = 0 To 99
       Queue(x) = ""
     Next
     QueueHead = -1
     QueueTail = -1
     NumberItems = 0

     ReadData()
     Compress()
     Console.WriteLine(NewString)
     Console.ReadLine()
 End Sub
```

2(f)(i)

Python

```
NewString = ""
Queue = []
for x in range(100):
        Queue.append("")
QueueHead = -1
QueueTail = -1
NumberItems = 0
ReadData()
Compress()
print(NewString)
```

2(f)(ii)

1 mark for screenshot showing output

Example

```
19051306
```


5

MP1 Pop an item off the stack
MP2 Update Stack pointer

MP3 Add it to the queue
MP4 Update Rear pointer

MP5 Repeat until the **stack is empty**

MP6 Remove an item from the queue
MP7 Update Front pointer

MP8 Push it onto the stack
MP9 Update the Stack pointer

MP10 Repeat until the **queue is empty**

Max 7

2(a)(i)

MP1 400
MP2 6

2(a)(ii)

Stack

Memory location	Value	
409		
408	'Y'	← `TopOfStack`
407	'Z'	
406	'C'	
405	'F'	
404	'K'	
403	'B'	
402	'S'	
401	'R'	
400	'D'	

MP1 `TopOfStack` pointing to 'Y' **and** value 'Y' in location 408
MP2 Values 'C' and 'Z' in 406 and 407
MP3 Values 'D' to 'F' unchanged in 400 to 405 **and** 409 is empty

2(b)(i)

Index	Data	Pointer		
0	"Neptune"		**2**	**MP3**
1	"Jupiter"		**4**	
2	"Saturn"		**5**	
3	"Earth"	**1**		**MP1**
4	"Mercury"		**0**	
5	"Uranus"	**-1**		**MP2**

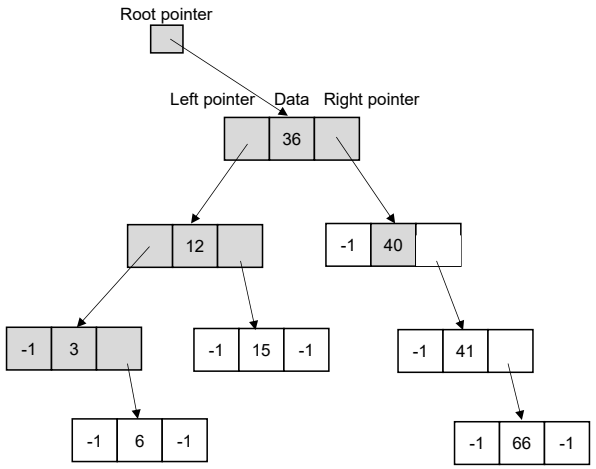
MP1 Earth pointer 1
MP2 Uranus pointer -1
MP3 Other four correct pointers

2(b)(ii) **3**

2(c)
MP1 First value added to queue is the first value removed // FIFO // LILO
MP2 Two **pointers** needed to indicate the **start** and **end** of the queue
MP3 May behave as a circular queue
MP4 Manipulated using `enqueue()` / `dequeue()` operations // by description

Max 2

4(a)
One mark per mark point
MP1 Any two nodes added correctly with correct data (6, 15, 41, 66) and arrows
MP2 Remaining two nodes added with correct data (6, 15, 41, 66) and all nodes with connecting arrows starting from pointer boxes
MP3 Correct null pointers (-1) added throughout
MP4 … with no entries in other pointer boxes and all nodes correctly positioned and connected



4(b)
One mark per mark point
MP1 A technique used to solve problems using a function/procedure/subroutine that calls itself (general case)
MP2 … until the terminating condition / base case is achieved, when no further recursive calls are made

4(c)
One mark
- Stack

3(a)(i)
`SP`: 1
`OnStack`: 0

One mark for both correct values

3(a)(ii)
MP1 Unused values cannot be popped / taken off the stack // initialised values would never be used / unused elements cannot be accessed

MP2 ... until a value has first been pushed / written // overwrites previous value

3(b)
Example solution:

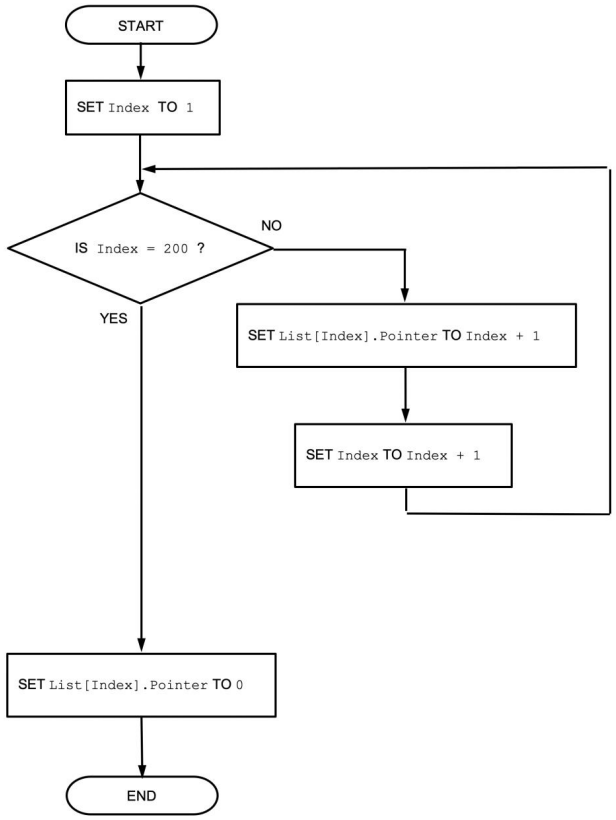
```
FUNCTION Push(ThisValue : REAL) RETURNS BOOLEAN
   DECLARE ReturnValue : BOOLEAN
   IF OnStack = 60 / >59 // SP = 61 / SP > 60 //
                                    SP outside the range 1 to 60 THEN
      RETURN FALSE // Stack is already full
   ENDIF
   ThisStack[SP] ← ThisValue
   SP ← SP + 1
   OnStack ← OnStack + 1
   RETURN TRUE
ENDFUNCTION1
```

Mark as follows:
One mark per gap

3(a)(i) 0 is not a valid array / `List` index value // No 0^{th} element in the array

3(a)(ii) `0` to `200`

3(b)



Mark as follows:
MP1 Use of variable as array index **and** initialisation to 1 / to first element of array
MP2 Loop for 199 / 200 iterations
MP3 Assign Index+1 to each pointer field in a loop
MP4 Assign 0 to 200th pointer field

3(c)

One mark per step:

MP1 Assign `HeadPointer` to `ThisPointer` / Current Pointer // Identify first node using headpointer

MP2 Loop the following until `ThisPointer` value is 0 (zero) / null pointer reached

MP3 … Output data (field) from array element at index `ThisPointer` // Output data (field) of the current node / array element

MP4 … Assign pointer field from current array element / index / to `ThisPointer` / Current Pointer // Assign pointer field from current node to `ThisPointer` / Current Pointer

3(a)

One mark per point:

1 The PS contains a null pointer
2 The PF points to the first element on the free list
3 All the nodes are on the free list

3(b)

Max 2 marks for 'Variables':

The two variables will be of type **Integer**
The two variables will be used as **pointers / indexes** to the arrays.
The values stored in the two variables will indicate **the first element in each list**
The first 1D array will be of type **String**
The first 1D array will be used to **store the values // data items // User IDs**
The second 1D array will be of type **Integer**
The second 1D array will be used to **store the pointers // point to next item**

Mark as follows:

One mark for **each** of the first three rows
One mark for **both** Array 1 rows
One mark for **both** Array 2 rows