<table>
<tr><td>2(a)</td><td>

Two marks for working
- correct calculation/application of exponent seen
- correct method to find the final answer

Working:
Exponent = 8 + 2 + 1 = 11
// =0.111100101 x 2^{11}
// =11110010100.0 (moving bp 11 places to right)

Method to find the answer
// 1024 + 512 + 256 + 128 + 16 + 4

One mark for correct answer

Denary value: 1940

Example of solution using fractions:

</td></tr>
<tr><td>2(b)</td><td>

One mark per mark point (**Max 3**)
MP1 correct **method** to find the binary number
MP2 correct **use** of exponent
MP3 correct answer in the space provided

Working:
26.6875 converted to binary (0)11010.1011 // 16+8+2+0.5+0.125+0.0625 movement of binary point by 5 places

<table>
<tr><th colspan="10">Mantissa</th><th colspan="6">Exponent</th></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
</table>

</td></tr>
</table>

7(a)

Example solution Conditional Loop

```
FUNCTION FindCustomer(CustomerID : INTEGER) RETURNS
INTEGER

   DECLARE Index : INTEGER
   DECLARE Found : BOOLEAN
   CONSTANT Unused = 99999
   CONSTANT Upper = 1000
   Found ← FALSE

   Index ← 1

   IF CustomerID < 10001 OR CustomerID > 11000 THEN
      RETURN -1  // Out of range value for customer ID
   ENDIF

   WHILE Found = FALSE AND Loyalty[Index, 1] <> 99999
      IF Loyalty[Index,1] = CustomerID THEN
         Found ← TRUE
      ELSE
         Index ← Index + 1
      ENDIF
      IF Index > Upper THEN
         RETURN -1
      ENDIF
   ENDWHILE

   IF Found THEN
      RETURN Loyalty[Index, 2]
   ELSE
      RETURN -1
   ENDIF

ENDFUNCTION
```

Mark as follows:

1. Create function header and ending with correct parameter and return type
2. Check `CustomerID` is in range and if not return −1
3. (Conditional) loop iterating through each element in array
4. Check for current element in array contains required customer ID **in a loop**
5. … If found set value to terminate loop
6. Terminating loop when customer ID found
7. Terminating loop when current customer ID is 99999
8. Return either loyalty points for the customer found or `-1` if not found

7(a)

Alternative solution using FOR Loop

Example solution

```
FUNCTION FindCustomer(CustomerID : INTEGER) RETURNS
                                                    INTEGER

   DECLARE Index : INTEGER

   IF CustomerID < 10001 OR CustomerID > 11000 THEN
      RETURN -1  // Out of range value for customer ID
   ENDIF

   FOR Index ← 1 TO 1000
      IF Loyalty[Index, 1] = CustomerID THEN
         RETURN Loyalty[Index, 2]
      ENDIF
      IF Loyalty[Index, 1] = 99999 THEN
         RETURN -1
      ENDIF
   NEXT Index

ENDFUNCTION
```

Mark as follows:

1. Create function header and ending with correct parameter and return type
2. Check `CustomerID` is within range and if not return –1
3. FOR Loop
4. Loop for elements 1 to 1000 / 0 to 999
5. Check if current element in array contains required Customer ID **in a loop**
6. … If found correctly return loyalty points // store correct loyalty points and return after loop
7. Check if current element in array is `99999` and return `-1` / break loop **in a loop**
8. return –1 if Customer ID not found

Direct access solution
Alternative solution subtracting 10000 from `CustomerID`

```
FUNCTION FindCustomer(CustomerID : INTEGER) RETURNS
                                                    INTEGER

   DECLARE Index : INTEGER

   IF CustomerID < 10001 OR CustomerID > 11000 THEN
      RETURN -1  // Out of range value for customer ID
   ENDIF
```

7(a)

```
   Index = CustomerID - 10000

   IF Loyalty[Index, 1] = CustomerID THEN
      RETURN Loyalty[Index, 2]
   ENDIF

   RETURN -1


ENDFUNCTION
```

Mark as follows:

1. Create function header and ending with correct parameter and return type
2. Declare Index as Integer
3. Check `CustomerID` is less than `10001` and return `-1` if it is
4. Check `CustomerID` is greater than `11000` and return `-1` if it is
5. Calculate Index by subtracting a `10000` form `CustomerID`
6. Check if array contains the `CusomerID`
7. Return Loyalty Points if `CustomerID` found
8. Return `-1` if not found

Binary Search Mark Scheme

Example Solution

```
FUNCTION FindCustomer(CustomerID : INTEGER) RETURNS
                                                    INTEGER
   DECLARE Start : INTEGER
   DECLARE End : INTEGER
   DECLARE Mid : INTEGER

   IF CustomerID < 10001 OR CustomerID > 11000 THEN
      RETURN -1  // Out of range value for customer ID
   ENDIF

   Start ← 1
   End ←1000

   WHILE Start <= End
      Mid ← (Start + End) DIV 2
      IF Loyalty[Mid, 1] = CustomerID THEN
         RETURN Loyalty[Mid, 2]
      ELSE IF Loyalty[Mid, 1] > CustomerID THEN
         End ← Mid - 1
      ELSE
         Start ← Mid + 1
      ENDIF
   ENDWHILE
   RETURN -1
ENDFUNCTION
```

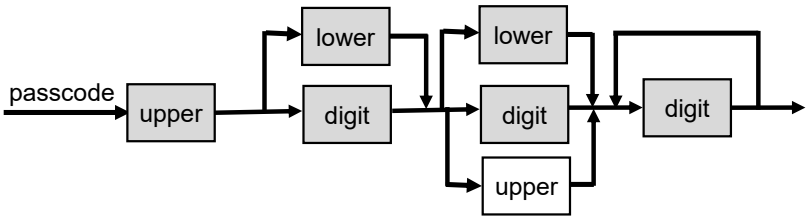
7(a)	Mark points 1. Create function header and ending with correct parameter and return type 2. Check `CustomerID` is within range and if not return –1 3. Conditional loop that halves array search space with each iteration 4. Check if `CustomerID` is stored in current array element **in loop** 5. Mechanism to end loop if `CustomerID` is found in array **in a loop** 6. Mechanism to end loop if `CustomerID` is not in array **in a loop** 7. If `CustomerID` found in array then return correct loyalty points 8. If `CustomerID` not in array then return –1

7(b)	Example solution
	
	`PROCEDURE PointsReport()`
	
	`DECLARE Count : INTEGER`
	`DECLARE Index : INTEGER`
	`DECLARE Sum : INTEGER`
	`DECLARE Average : REAL`
	
	`Index ← 1`
	`Count ← 0`
	`Sum ← 0`
	
	`WHILE Loyalty[Index, 1] <> 99999 AND Index <= 1000`
	`IF Loyalty[Index, 2] >= 11 THEN`
	`OUTPUT Loyalty[Index, 1]`
	`ENDIF`
	`Count ← Count + 1`
	`Sum ← Sum + Loyalty[Index, 2]`
	`Index ← Index + 1`
	`ENDWHILE`
	
	`Average ← Sum / Count`
	`OUTPUT "The average points of all the customers is ", Average`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1. Create procedure header and ending
	2. Declare and initialise `Index`, `Sum` and `Count`
	3. Loop through all elements in `Loyalty` array
	4. … or terminates when column1 of `Loyalty` array equals `99999` **in a loop**
	5. Check if loyalty points greater than or equal to `11` **in a loop**
	6. If true output customer ID
	7 Sum points **and** Increment `Count` **in a loop**
	8 Calculate **and** output average with an appropriate message
	
	Max 7

7(c)(i)	An array can only store data of **the same type** // An array cannot store data **of different types**
7(c)(ii)	1 mark for each point 1. a new (composite) data type / record is defined (that consists of both `INTEGER` and `STRING`) 2. an array based on this new type is **declared** Alternative 1 mark for each point 1. Converting loyalty points to string and concatenating / joining / append with Customer ID 2. Storing **concatenated** string in an **array of string**

9(a)	The second character must come from either lower or digit. It can't be upper.

9(b)	**Max 3**
	
	One mark
	MP1 `<upper> ::= J | K | L | V | X | Z`
	
	Either: fully written out answer
	MP2 Any two correct options for `<passcode>`
	MP3 Remaining two options correct for `<passcode>`
	
	Example answer
	`<passcode> ::=`
	`<upper><lower><lower><digit>|`
	`<upper><lower><digit><digit>|`
	`<upper><digit><digit><digit>|`
	`<upper><digit><lower><digit>`
	
	Or: answer with interim expression
	MP4 `<passcode> ::= <upper><middle><middle><digit>`
	MP5 `<middle> ::= <lower>|<digit>`

9(c)	**One** mark per mark point MP1 Box for upper in correct place with correct connections MP2 Correct repetition arrow for final digit 

2(a)

1 mark each to max
- Record structure or class with constructor `Queue` (and end where appropriate) …
- … containing a 1D array of (100) integers `QueueArray`
- … containing `HeadPointer` and `TailPointer` as integers

e.g.
Python
```
class Queue:
    def __init__(self):
        self.QueueArray = []
        HeadPointer = 0 #integer
        TailPointer = 0 #integer
        for x in range(0, 100):
            self.QueueArray.append(-1)
```

VB.NET
```
Structure Queue
    Dim QueueArray() As Integer
    Dim HeadPointer As Integer
    Dim TailPointer As Integer
End Structure
```

Java
```
class queue{
     private static Integer[] QueueArray = new Integer[100];
     private static Integer HeadPointer;
     private static Integer TailPointer;

     public queue(){
     }
}
```

2(b)

1 mark each
- New `Queue` record/object created/instance of class
- Queue **field/attribute** head pointer initialised to `-1`, tail pointer to `0`
- All 100 array **field/attribute** elements initialised with `-1`

e.g.
Python
```
class Queue:
    def __init__(self):
        self.QueueArray = []
        for x in range(0, 100):
            self.QueueArray.append(-1)

        self.HeadPointer = -1
        self.TailPointer = 0
TheQueue= Queue()
```

VB.NET
```
 Dim TheQueue As New Queue
        TheQueue.HeadPointer = -1
        TheQueue.TailPointer = 0
        ReDim TheQueue.QueueArray(100)
        For x = 0 To 99
            TheQueue.QueueArray(x) = -1
        Next
```

Java
```
class queue{
     private static Integer[] QueueArray = new Integer[100];
     private static Integer HeadPointer;
     private static Integer TailPointer;
```

2(b)

```
      public queue(){
            HeadPointer = -1;
            TailPointer = 0;
            for(Integer x = 0; x < 100; x++){
                  QueueArray[x] = -1;
            }
      }
}
public static void main(String args[]){
      queue TheQueue = new queue();
}
```

2(c)

1 mark for each completed statement to max 3
1 mark for correct values returned in correct places
1 mark for function header taking (at least) one parameter and the rest of function correct and using the record/class data structure accurately.

Pseudocode

```
FUNCTION Enqueue(BYREF AQueue : Queue, BYVAL TheData : INTEGER)
                  RETURNS INTEGER
   IF AQueue.HeadPointer = -1 THEN
      AQueue.QueueArray[AQueue.TailPointer] ← TheData
      AQueue.HeadPointer ← 0
      AQueue.TailPointer ← AQueue.TailPointer + 1
      RETURN 1
   ELSE
      IF AQueue.TailPointer > 99 THEN
        RETURN -1
      ELSE
        AQueue.QueueArray[AQueue.TailPointer] ← TheData
        AQueue.TailPointer ← AQueue.TailPointer + 1
        RETURN 1
      ENDIF
   ENDIF
ENDFUNCTION
```

2(c)

e.g.

Python

```
def Enqueue(AQueue, TheData):

    if AQueue.HeadPointer == -1:
        AQueue.HeadPointer = 0
        AQueue.QueueArray[AQueue.HeadPointer] = TheData
        AQueue.TailPointer +=1
        return AQueue, 1
    elif AQueue.TailPointer > 99:
        return AQueue, -1
    else:
        AQueue.QueueArray[AQueue.TailPointer] = TheData
        AQueue.TailPointer = AQueue.TailPointer + 1
        return AQueue, 1
```

VB.NET

```
Function Enqueue(ByRef AQueue As Queue, ByVal TheData As Integer)
     If AQueue.HeadPointer = -1 Then
         AQueue.QueueArray(AQueue.TailPointer) = TheData
         AQueue.HeadPointer = 0
          AQueue.TailPointer += 1
         Return 1
     ElseIf AQueue.TailPointer > 99 Then
         Return -1
     Else
         AQueue.QueueArray(AQueue.TailPointer) = TheData
         AQueue.TailPointer += 1
         Return 1
     End If

End Function
```

2(c)

Java

```
public static Integer Enqueue(Integer TheData){
      if(GetHeadPointer() == -1){
            SetData(TheData);
            SetHeadPointer(0);
            SetTailPointer(GetTailPointer() + 1);
            return 1;
      }else if(GetTailPointer() > 99){
            return -1;
      }else{
            SetData(TheData);
            SetTailPointer(GetTailPointer() + 1);
            return 1;
      }
}
```

2(d)

1 mark each to max 3

- Function header (and end) iterating through each element in the queue
- Starting at `HeadPointer` and incrementing until `TailPointer - 1` …
- … concatenating and returning all integer values with a space between

e.g.

Python

```
def ReturnAllData(TheQueue):
    Temp = ""

    for X in range(TheQueue.HeadPointer, TheQueue.TailPointer):
        Temp = Temp + str(TheQueue.QueueArray[X]) + " "
    return Temp
```

2(d)	VB.NET
	`Function ReturnAllData(AQueue As Queue)`
	`  Dim Temp As String = ""`
	`  For X = AQueue.HeadPointer To AQueue.TailPointer - 1`
	`  Temp = Temp & AQueue.QueueArray(X).ToString() & " "`
	`  Next X`
	`  Return Temp`
	`End Function`
	
	Java
	`public static String ReturnAllData(){`
	`  String Temp = "";`
	`  Integer Counter = 0;`
	`  for(int X = HeadPointer; X < TailPointer; X++){`
	`  Temp = Temp + Integer.toString(QueueArray[X]) + " ";`
	`  }`
	`  return Temp;`
	`}`

2(e)(i)	1 mark each • Taking only **10** inputs in loop/one at a time • Calling `Enqueue()` with each input (min) **and** storing/using return value … • … only calling `Enqueue()` **once** when each input is an integer > = 0. Do **not** award if this validation stops 10 valid inputs being enqueued. • Outputting message if each item is inserted **and** outputting a message if queue is full • Calling `ReturnAllData()` and outputting return value at the end

2(e)(i)

e.g.

Python

```
for x in range(0, 10):
    Continue = True
    while(Continue == True):
        DataInput = int(input("Enter an integer that is 0 or more"))
        if DataInput > -1:
            Continue = False

    TheQueue, ReturnValue = Enqueue(TheQueue, DataInput)

    if(ReturnValue == -1):
        print("Queue full")
    else:
        print("Item inserted")

print(ReturnAllData(TheQueue))
```

VB.NET

```
Dim ContinueLoop As Boolean
Dim DataInput As Integer
Dim ReturnValue As Integer
For x = 0 To 9
    ContinueLoop = True
    While ContinueLoop = True
        Console.WriteLine("Enter an integer that is 0 or more")
        DataInput = Console.ReadLine
        If DataInput > -1 Then
            ContinueLoop = False
```

2(e)(i)

```
        End If
    End While
    ReturnValue = Enqueue(TheQueue, DataInput)
    If ReturnValue = 2 Then
        Console.WriteLine("Queue full")
    Else
        Console.WriteLine("Item inserted")
    End If
Next

Console.WriteLine(ReturnAllData(TheQueue))
```

Java

```
Boolean Continue = true;
Integer DataInput = -1;
Scanner scanner = new Scanner(System.in);
Integer ReturnValue;

for(Integer x = 0; x < 10; x++){
      Continue = true;
      while(Continue == true){
            System.out.println("Enter an integer that is 0 or more");
            DataInput = Integer.parseInt(scanner.nextLine());
            if(DataInput > -1){
                  Continue = false;
            }
      }
      ReturnValue = Enqueue(DataInput);
      if(ReturnValue == -1){
            System.out.println("Queue full");
      }else{
            System.out.println("Item inserted");
      }
}
System.out.println(ReturnAllData());
```

2(e)(ii)	1 mark for each • All values input and 10 messages 'Inserted' (i.e. –1 is not inserted) • Screenshot show 10 9 8 7 6 5 4 3 2 1 on one line with a space between each number e.g. Enter an integer that is 0 or more10 Item inserted Enter an integer that is 0 or more9 Item inserted Enter an integer that is 0 or more-1 Enter an integer that is 0 or more8 Item inserted Enter an integer that is 0 or more7 Item inserted Enter an integer that is 0 or more6 Item inserted Enter an integer that is 0 or more5 Item inserted Enter an integer that is 0 or more4 Item inserted Enter an integer that is 0 or more3 Item inserted Enter an integer that is 0 or more2 Item inserted Enter an integer that is 0 or more1 Item inserted 10 9 8 7 6 5 4 3 2 1

2(f)	1 mark each
	• Function `Dequeue()` (head and close), returning a value in all cases
	• Checking if empty **and** returning –1
	• Returning item at `HeadPointer` **without deleting**/changing it
	• Incrementing `HeadPointer`
	
	Example program code:
	Python
	`def Dequeue(AQueue):`
	`    if AQueue.HeadPointer = 100 or AQueue.HeadPointer == -1 or AQueue.HeadPointer == AQueue.TailPointer:`
	`        return AQueue, -1`
	`    else:`
	`        Temp = AQueue.QueueArray[AQueue.HeadPointer]`
	`        AQueue.HeadPointer = AQueue.HeadPointer + 1`
	`        return AQueue, Temp`

2(f)

VB.NET

```
  Function Dequeue(ByRef AQueue As Queue)
        If AQueue.HeadPointer = 100 or AQueue.HeadPointer = -1 or AQueue.HeadPointer =
AQueue.TailPointer Then
            Return -1
        Else
            Dim Temp As Integer = AQueue.QueueArray(AQueue.HeadPointer)
            AQueue.HeadPointer += 1
            Return Temp
        End If
  End Function
```

Java

```
public static Integer Dequeue(){
     if(GetHeadPointer() = 100 || GetTailpointer() == -1 || GetHeadPointer() ==
GetTailpointer()){
           return -1;
     }else{
           Integer Temp = GetData(GetHeadPointer());
           SetHeadPointer(GetHeadPointer() + 1);
           return Temp;
     }
}
```

2(g)(i)	1 mark each
	• Calls `Dequeue()` **twice and** stores/uses return value
	• Outputs "Queue empty" when each return values is –1 and outputs return value otherwise
	• Calls `ReturnAllData()` at the end
	
	e.g.
	Python
	`TheQueue, ReturnValue = Dequeue(TheQueue)`
	`if ReturnValue == -1:`
	`  print("Queue empty")`
	`else:`
	`  print(ReturnValue, " is returned")`
	`TheQueue, ReturnValue = Dequeue(TheQueue)`
	`if ReturnValue == -1:`
	`  print("Queue empty")`
	`else:`
	`  print(ReturnValue, " is returned")`
	`print(ReturnAllData(TheQueue))`
	
	VB.NET
	`ReturnValue = Dequeue(TheQueue)`
	`If ReturnValue = -1 Then`
	`  Console.WriteLine("Queue empty")`
	`Else`
	`  Console.WriteLine(ReturnValue, " is returned")`
	`End If`

2(g)(i)

```
ReturnValue = Dequeue(TheQueue)
If ReturnValue = -1 Then
    Console.WriteLine("Queue empty")
Else
    Console.WriteLine(ReturnValue, " is returned")
End If
Console.WriteLine(ReturnAllData(TheQueue))
```

Java

```
ReturnValue = Dequeue();
if(ReturnValue == -1){
      System.out.println("Queue empty");
}else{
      System.out.println(ReturnValue + " is returned");
}

ReturnValue = Dequeue();
if(ReturnValue == -1){
      System.out.println("Queue empty");
}else{
      System.out.println(ReturnValue + " is returned");
}
ReturnAllData(TheQueue);
```

2(g)(ii)

1 mark each

- Screenshot shows the input of 10 9 8 7 6 5 4 3 2 1

Output for 10 returned
Output for 9 is returned

e.g.

```
Enter an integer that is 0 or more10
Item inserted
Enter an integer that is 0 or more9
Item inserted
Enter an integer that is 0 or more-1
Enter an integer that is 0 or more8
Item inserted
Enter an integer that is 0 or more7
Item inserted
Enter an integer that is 0 or more6
Item inserted
Enter an integer that is 0 or more5
Item inserted
Enter an integer that is 0 or more4
Item inserted
Enter an integer that is 0 or more3
Item inserted
Enter an integer that is 0 or more2
Item inserted
Enter an integer that is 0 or more1
Item inserted
10 9 8 7 6 5 4 3 2 1
10  is returned
9  is returned
8 7 6 5 4 3 2 1
```

6(a)

One mark for working, all five columns P, Q, R, S and T
One mark for first four rows of column Z
One mark for second four rows of column Z

			Working space					
A	**B**	**C**	**P**	**Q**	**R**	**S**	**T**	**Z**
0	0	0	1	1	1	1	1	0
0	0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0	1
0	1	1	1	0	0	1	0	1
1	0	0	0	1	1	0	0	0
1	0	1	0	1	0	1	0	1
1	1	0	0	0	1	1	0	1
1	1	1	0	0	0	1	0	1

6(b)

Two marks for all six correct terms only
One mark for any three correct terms

(Z =) $\overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$

6(c)(i)

Two marks if all correct
One mark if only one error present

A \ BC	**00**	**01**	**11**	**10**
0	1	0	1	1
1	1	0	1	1

6(c)(ii)

One mark for each correct loop **(Max 2)**

A \ BC	**00**	**01**	**11**	**10**
0	1	0	1	1
1	1	0	1	1

6(c)(iii)

B + $\underline{C}$

<table>
<tr><td>2(a)</td><td>Two marks for all protocols in correct position
One mark for at least two protocols in correct position

<table><tr><td>Application</td></tr><tr><td>Transport</td></tr><tr><td>Internet</td></tr><tr><td>Link</td></tr></table></td></tr>
<tr><td>2(b)</td><td>One mark per mark point (Max 2)
MP1 The transport layer is responsible for delivery of data from the source host to the destination host
MP2 It is where data is broken up into packets and sent to the internet layer
MP3 Adds the sequence number to the packet header
MP4 It establishes end to end contact
MP5 It ensures data arrives error free // It retransmits packets if lost.</td></tr>
<tr><td>2(c)</td><td>One mark for name of protocol and one mark for expansion (Max 2)

HTTP(S) – responsible for correct transfer of files / hypertext documents that make up web pages on the world wide web

FTP – used when transferring files from a server to a client on a network

POP3 – handles the receiving of emails

IMAP – handles the receiving of emails

SMTP – handles the sending of emails

BitTorrent – provides peer-to-peer file sharing</td></tr>
</table>