12(a)(i)

One mark for each correctly completed line (**Max 4**)

```
PROCEDURE Push(NewData : STRING)
   IF Top < Max - 1 THEN
      Top ← Top + 1
      StackArray[Top] ← NewData
   ELSE
      OUTPUT "Stack full; new data cannot be added"
   ENDIF
ENDPROCEDURE
```

12(a)(ii)

One mark per mark point (**Max 2**)
MP1 Input with variable, with or without prompt
MP2 Procedure call for Push with parameter used matching input variable

Example answer

```
INPUT MyData
CALL Push(MyData)
```

12(b)

One mark per mark point (**Max 3**)
MP1 Stacks store data in Last In First Out (LIFO) / First In Last Out (FILO) order
MP2 Each time a recursive algorithm calls itself data is pushed onto the stack
MP3 When the recursive algorithm reaches its base case / starts to unwind
MP4 … data is popped from the stack in the **reverse order** to which it was pushed onto it.

8

One mark per mark point - working (**Max 3**)
May be seen on diagram or in working section

MP1 Initialisation – setting Start to 0
MP2 … and the rest of the towns to ∞
MP3 Evidence to show values at nodes being updated
MP4 Evidence to show 'visited node(s)'
MP5 Evidence to **show a correct calculation** of at least one route
MP6 Evidence to show more than one route has been calculated for at least one town

Correct Answers (**Max 2**)
Two marks for all six correct values
One mark for four or five correct values.

T	V	W	X	Y	Z
6	10	13	18	21	28

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**