QUESTION 8.

6 A recursively defined procedure X is defined below:

```
PROCEDURE X(BYVALUE n : INTEGER)
   IF (n = 0) OR (n = 1)
      THEN
         OUTPUT n
      ELSE
         CALL X(n DIV 2)
         OUTPUT (n MOD 2)
   ENDIF
ENDPROCEDURE
```

(a) Explain what is meant by recursively defined.

.....

..... [1]

(b) Explain how a stack is used during the execution of a recursive procedure.

.....

.....

.....

..... [2]

(c) Dry run the procedure X by completing the trace table for the procedure call:

```
CALL X(40)
```

Call number	n	(n = 0) OR (n = 1)	n DIV 2	n MOD 2
1	40	FALSE	20	
2				
3				
4				
5				
6				

OUTPUT ..... [6]

(d) State the process that is carried out by procedure X.

.....

.....

(e) Write **program code** for procedure X.

Programming language .....

.....

.....

.....

.....

.....

.....

.....

..... [5]

QUESTION 9.

4 A binary tree Abstract Data Type (ADT) has these associated operations:

- create the tree (`CreateTree`)
- add an item to tree (`Add`)
- output items in ascending order (`TraverseTree`)

(a) Show the final state of the binary tree after the following operations are carried out.

```
CreateTree
Add("Dodi")
Add("Farai")
Add("Elli")
Add("George")
Add("Ben")
Add("Celine")
Add("Ali")
```

[4]

(b) The binary tree ADT is to be implemented as an array of nodes. Each node and two pointers.

Using pseudocode, a record type, `Node`, is declared as follows:

```
TYPE Node
   DECLARE Name : STRING
   DECLARE LeftPointer : INTEGER
   DECLARE RightPointer : INTEGER
ENDTYPE
```

The statement

```
DECLARE Tree : ARRAY[1:10] OF Node
```

reserves space for 10 nodes in array `Tree`.

The `CreateTree` operation links all nodes into a linked list of free nodes. It also initialises the `RootPointer` and `FreePointer`.

Show the contents of the `Tree` array and the values of the two pointers, `RootPointer` and `FreePointer`, after the operations given in **part (a)** have been carried out.

`RootPointer`

`FreePointer`

`Tree`

	Name	LeftPointer	RightPointer
[1]			
[2]			
[3]			
[4]			
[5]			
[6]			
[7]			
[8]			
[9]			
[10]			

[7]

(c) A programmer needs an algorithm for outputting items in ascending order. To
programmer writes a recursive procedure in pseudocode.

(i) Complete the pseudocode:

```
01 PROCEDURE TraverseTree(BYVALUE Root: INTEGER)
02     IF Tree[Root].LeftPointer ..........
03         THEN
04             TraverseTree(..........)
05     ENDIF
06     OUTPUT ..........Name
07     IF ..........<> 0
08         THEN
09             TraverseTree(..........)
10     ENDIF
11 ENDPROCEDURE
```

[5]

(ii) Explain what is meant by a recursive procedure. Give a line number from the code above that shows procedure `TraverseTree` is recursive.

.....

.....

.....

Line number .....[2]

(iii) Write the pseudocode call required to output all names stored in `Tree`.

.....

.....[1]

Question 5 begins on page 14.

QUESTION 10.

2 **(a)** **(i)** State what is meant by a recursively defined procedure.

.....

.....

(ii) Write the line number from the pseudocode shown in **part (b)** that shows the procedure `X` is recursive. ..... [1]

(b) The recursive procedure `X` is defined as follows:

```
01  PROCEDURE X(Index, Item)
02      IF MyList[Index] > 0
03          THEN
04              IF MyList(Index) >= Item
05                  THEN
06                      MyList[Index] ← MyList[Index + 1]
07              ENDIF
08              CALL X(Index + 1, Item)
09      ENDIF
10  ENDPROCEDURE
```

An array `MyList` is used to store a sorted data set of non-zero integers. Unused cells contain zero.

	1	2	3	4	5	6	7	8	9	10
MyList	3	5	8	9	13	16	27	0	0	0

(i) Complete the trace table for the dry-run of the pseudocode for `CALL X(1, 9)`.

		MyList									
Index	**Item**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	9	3	5	8	9	13	16	27	0	0	0

[4]

(ii) State the purpose of procedure `X` when used with the array `MyList`.

.....

.....[1]

QUESTION 11.

3 `NameList` is a 1D array that stores a sorted list of names. A programmer decl… pseudocode as follows:

```
NameList : Array[0 : 100] OF STRING
```

The programmer wants to search the list using a binary search algorithm.

The programmer decides to write the search algorithm as a recursive function. The function, `Find`, takes three parameters:

- `Name`, the string to be searched for
- `Start`, the index of the first item in the list to be searched
- `Finish`, the index of the last item in the list to be searched

The function will return the position of the name in the list, or –1 if the name is not found.

Complete the **pseudocode** for the recursive function.

```
FUNCTION Find(BYVALUE Name : STRING, BYVALUE Start : INTEGER,
                                BYVALUE Finish : INTEGER) RETURNS INTEGER

   // base case

   IF ..........

      THEN

         RETURN -1

      ELSE

         Middle ← ..........

         IF ..........

            THEN

               RETURN ..........

            ELSE    // general case

               IF SearchItem > ..........

                  THEN

                     ..........

                  ELSE

                     ..........

               ENDIF

         ENDIF

   ENDIF

ENDFUNCTION
```

[7]

QUESTION 12.

2 An ordered binary tree Abstract Data Type (ADT) has these associated operation

- create tree
- add new item to tree
- traverse tree

The binary tree ADT is to be implemented as a linked list of nodes.

Each node consists of data, a left pointer and a right pointer.

(a) A null pointer is shown as Ø.

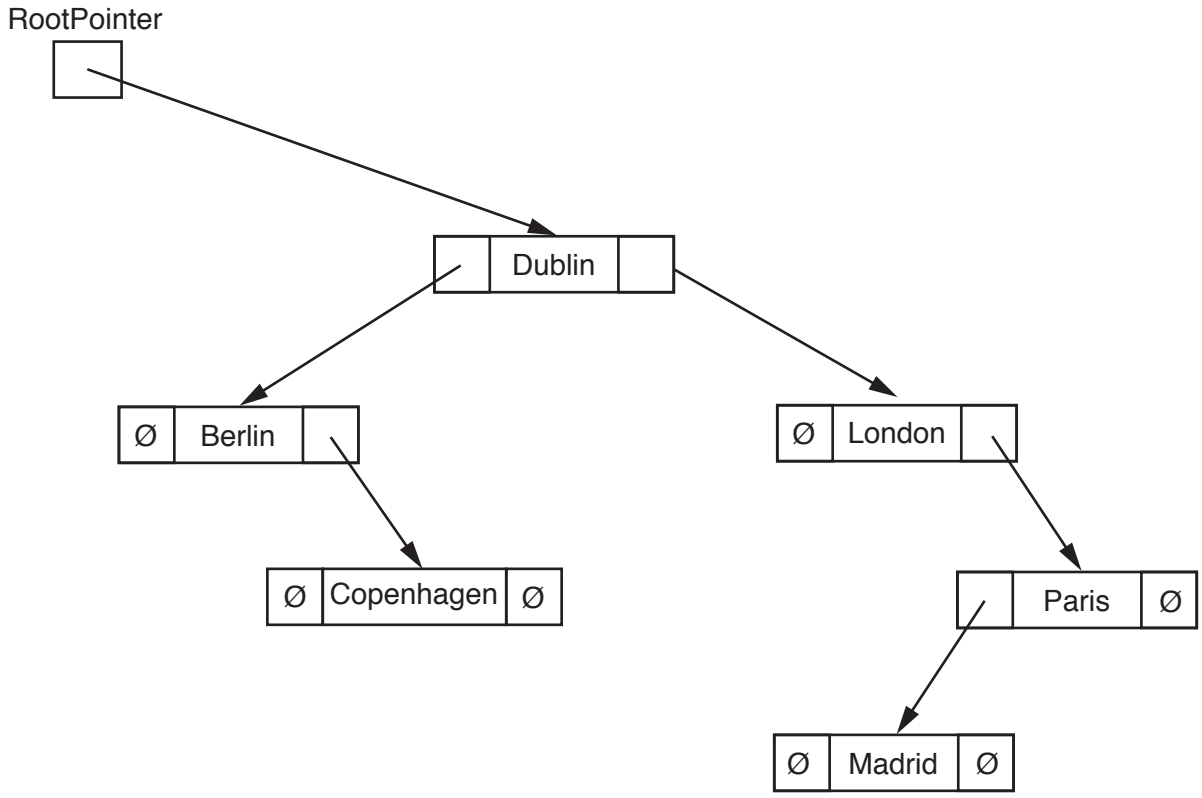
Explain the meaning of the term **null pointer**.

.....

.....[1]

(b) The following diagram shows an ordered binary tree after the following data have been added:

Dublin, London, Berlin, Paris, Madrid, Copenhagen



Another data item to be added is Athens.

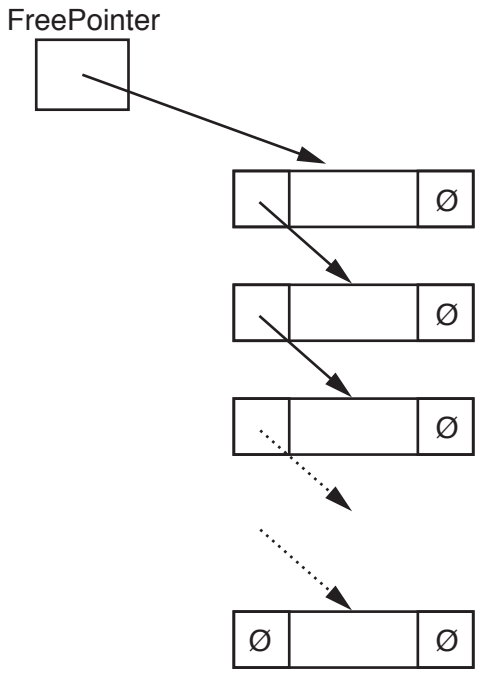
Make the required changes to the diagram when this data item is added. [2]

(c) A tree without any nodes is represented as:

RootPointer

Ø

Unused nodes are linked together as shown:



The following diagram shows an array of records that stores the tree shown in **part (b)**.

(i) Add the relevant pointer values to complete the diagram.

RootPointer

0

FreePointer

	LeftPointer	Tree data	RightPointer
[0]		Dublin	
[1]		London	
[2]		Berlin	
[3]		Paris	
[4]		Madrid	
[5]		Copenhagen	
[6]		Athens	
[7]			
[8]			
[9]			

[5]

(ii) Give an appropriate numerical value to represent the null pointer for th
your answer.

.....

.....

.....

.....[2]

(d) A program is to be written to implement the tree ADT. The variables and procedures to be used are listed below:

Identifier	Data type	Description
`Node`	`RECORD`	Data structure to store node data and associated pointers.
`LeftPointer`	`INTEGER`	Stores index of start of left subtree.
`RightPointer`	`INTEGER`	Stores index of start of right subtree.
`Data`	`STRING`	Data item stored in node.
`Tree`	`ARRAY`	Array to store nodes.
`NewDataItem`	`STRING`	Stores data to be added.
`FreePointer`	`INTEGER`	Stores index of start of free list.
`RootPointer`	`INTEGER`	Stores index of root node.
`NewNodePointer`	`INTEGER`	Stores index of node to be added.
`CreateTree()`		Procedure initialises the root pointer and free pointer and links all nodes together into the free list.
`AddToTree()`		Procedure to add a new data item in the correct position in the binary tree.
`FindInsertionPoint()`		Procedure that finds the node where a new node is to be added. Procedure takes the parameter `NewDataItem` and returns two parameters: • `Index`, whose value is the index of the node where the new node is to be added • `Direction`, whose value is the direction of the pointer (“`Left`” or “`Right`”).

(i) Complete the pseudocode to create an empty tree.

```
TYPE Node
    .....
    .....
    .....
ENDTYPE
DECLARE Tree : ARRAY[0 : 9] .....
DECLARE FreePointer : INTEGER
DECLARE RootPointer : INTEGER

PROCEDURE CreateTree()
   DECLARE Index : INTEGER
   .....
   .....
   FOR Index ← 0 TO 9   // link nodes
      .....
      .....
   ENDFOR
   .....
ENDPROCEDURE
```

[7]

(ii) Complete the pseudocode to add a data item to the tree.

```
PROCEDURE AddToTree(BYVALUE NewDataItem : STRING)
// if no free node report an error
   IF FreePointer .....
      THEN
         OUTPUT("No free space left")
      ELSE // add new data item to first node in the free list
         NewNodePointer ← FreePointer
         .....
         // adjust free pointer
         FreePointer ← .....
         // clear left pointer
         Tree[NewNodePointer].LeftPointer ← .....
         // is tree currently empty ?
         IF .....
            THEN // make new node the root node
               .....
            ELSE   // find position where new node is to be added
               Index ← RootPointer
               CALL FindInsertionPoint(NewDataItem, Index, Direction)
               IF Direction = "Left"
                  THEN   // add new node on left
                     .....
                  ELSE   // add new node on right
                     .....
               ENDIF
         ENDIF
   ENDIF
ENDPROCEDURE
```

[8]

(e) The traverse tree operation outputs the data items in alphabetical order. Th
as a recursive solution.

Complete the pseudocode for the recursive procedure `TraverseTree`.

```
PROCEDURE TraverseTree(BYVALUE Pointer : INTEGER)
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```

[5]

QUESTION 13.

4 The recursive algorithm for the `Calculate()` function is defined as follows:

```
01  FUNCTION Calculate(BYVALUE Number : INTEGER) RETURNS INTEGER
02      IF Number = 0
03          THEN
04              Calculate ← -10
05          ELSE
06              Calculate ← Number * Calculate(Number - 1)
07      ENDIF
08  ENDFUNCTION
```

(a) (i) State what is meant by a **recursive algorithm**.

.....

..... [1]

(ii) State the line number in `Calculate()` where the recursive call takes place.

..... [1]

Question 4(b) begins on the next page.

(b) The function is called with `Calculate(3)`.

Dry run the function **and** complete the trace table below. State the final value retu
your working.

```
01  FUNCTION Calculate(BYVALUE Number : INTEGER) RETURNS INTEGER
02      IF Number = 0
03          THEN
04              Calculate ← -10
05          ELSE
06              Calculate ← Number * Calculate(Number - 1)
07      ENDIF
08  ENDFUNCTION
```

Working .....

.....

.....

.....

Trace table:

Call number	Function call	Number = 0 ?	Return value

Final return value .....

[6]

(c) A recursive algorithm within a subroutine can be replaced with an iterative al

(i) Describe **one** problem that can occur when running a subroutine that has algorithm.

.....

.....

.....

..... [2]

(ii) Rewrite the `Calculate()` function in **pseudocode**, using an **iterative algorithm**.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

QUESTION 14.

6 A linked list abstract data type (ADT) is created. This is implemented as an array records are of type `ListElement`.

An example of a record of `ListElement` is shown in the following table.

Data Item	Value
`Country`	`"Scotland"`
`Pointer`	`1`

(a) (i) Use **pseudocode** to write a definition for the record type, `ListElement`.

.....

.....

.....

.....

..... [3]

(ii) Use **pseudocode** to write an array declaration to reserve space for only 15 nodes of type `ListElement` in an array, `CountryList`. The lower bound element is `1`.

..... [2]

(b) The program stores the position of the last node in the linked list in `LastNode`. The last node always has a `Pointer` value of `-1`. The position of the node at the head of the list is stored in `ListHead`.

After some processing, the array and variables are in the following state.

ListHead
1
LastNode
3

CountryList

	Country	**Pointer**
1	"Wales"	2
2	"Scotland"	4
3		-1
4	"England"	5
5	"Brazil"	6
6	"Canada"	7
7	"Mexico"	8
8	"Peru"	9
9	"China"	10
10		11
11		12
12		13
13		14
14		15
15		3

A **recursive** algorithm searches the list for a value, deletes that value, a
required pointers. When a node value is deleted, it is set to empty "" and the
to the end of the list.

A node value is deleted using the pseudocode statement

```
CALL DeleteNode("England", 1, 0)
```

Complete the following **pseudocode** to implement the `DeleteNode` procedure.

```
PROCEDURE DeleteNode(NodeValue: STRING, ThisPointer : INTEGER,
                                                PreviousPointer : INTEGER)

 IF CountryList[ThisPointer].Value = NodeValue

   THEN

     CountryList[ThisPointer].Value ← ""

     IF ListHead = ……

       THEN

         ListHead ← ……

       ELSE

         CountryList[PreviousPointer].Pointer ← CountryList[ThisPointer].Pointer

     ENDIF

     CountryList[LastNode].Pointer ← ……

     LastNode ← ThisPointer

     ……

   ELSE

     IF CountryList[ThisPointer].Pointer <> -1

       THEN

         CALL DeleteNode(NodeValue, ……,
                                                              ThisPointer)

       ELSE

         OUTPUT "DOES NOT EXIST"

     ENDIF

 ENDIF

ENDPROCEDURE
```

[5]

QUESTION 15.

3 Some algorithms can be written using recursion.

(a) State **two** features of recursion.

Feature 1 .....

Feature 2 .....

[2]

(b) Explain what a compiler has to do to implement recursion.

.....

.....

.....

.....

.....

..... [3]

BLANK PAGE