QUESTION 7.

6 A queue Abstract Data Type (ADT) has these associated operations:

- create queue
- add item to queue
- remove item from queue

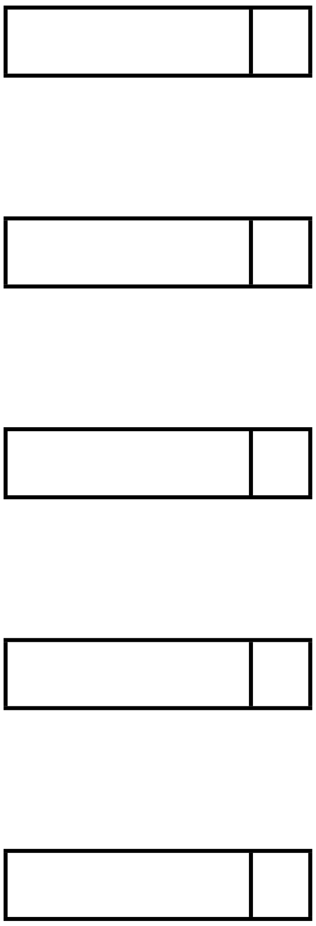
The queue ADT is to be implemented as a linked list of nodes.

Each node consists of data and a pointer to the next node.

(a) The following operations are carried out:

```
CreateQueue
AddName("Ali")
AddName("Jack")
AddName("Ben")
AddName("Ahmed")
RemoveName
AddName("Jatinder")
RemoveName
```

Add appropriate labels to the diagram to show the final state of the queue. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

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

```
TYPE Node
    DECLARE Name    : STRING
    DECLARE Pointer : INTEGER
ENDTYPE
```

The statement

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

reserves space for 10 nodes in array `Queue`.

(i) The `CreateQueue` operation links all nodes and initialises the three pointers that need to be used: `HeadPointer`, `TailPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers after `CreateQueue` has been executed.

HeadPointer

TailPointer

FreePointer

Queue

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

[4]

(ii) The algorithm for adding a name to the queue is written, using ps procedure with the header:

```
PROCEDURE AddName(NewName)
```

where `NewName` is the new name to be added to the queue.

The procedure uses the variables as shown in the identifier table.

Identifier	**Data type**	**Description**
`Queue`	`Array[1:10] OF Node`	Array to store node data
`NewName`	`STRING`	Name to be added
`FreePointer`	`INTEGER`	Pointer to next free node in array
`HeadPointer`	`INTEGER`	Pointer to first node in queue
`TailPointer`	`INTEGER`	Pointer to last node in queue
`CurrentPointer`	`INTEGER`	Pointer to current node

```
PROCEDURE AddName(BYVALUE NewName : STRING)
   // Report error if no free nodes remaining
   IF FreePointer = 0
      THEN
         Report Error
   ELSE
      // new name placed in node at head of free list
      CurrentPointer ← FreePointer
      Queue[CurrentPointer].Name ← NewName
      // adjust free pointer
      FreePointer ← Queue[CurrentPointer].Pointer
      // if first name in queue then adjust head pointer
      IF HeadPointer = 0
         THEN
            HeadPointer ← CurrentPointer
      ENDIF
      // current node is new end of queue
      Queue[CurrentPointer].Pointer ← 0
      TailPointer ← CurrentPointer
   ENDIF
ENDPROCEDURE
```

QUESTION 8.

5 A stack Abstract Data Type (ADT) has these associated operations:

- create stack
- add item to stack (push)
- remove item from stack (pop)

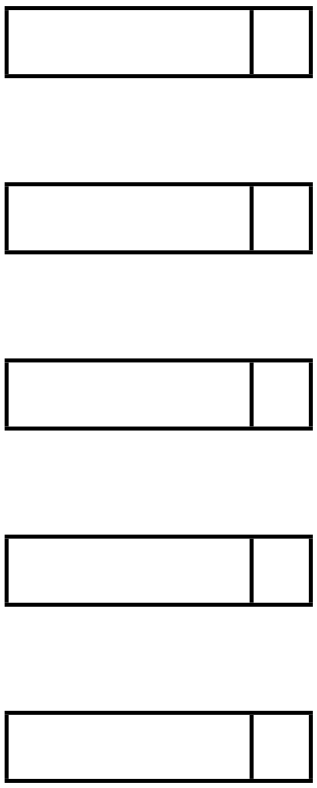
The stack ADT is to be implemented as a linked list of nodes.

Each node consists of data and a pointer to the next node.

(a) There is one pointer: the top of stack pointer, which points to the last item added to the stack. Draw a diagram to show the final state of the stack after the following operations are carried out.

```
CreateStack
Push("Ali")
Push("Jack")
Pop
Push("Ben")
Push("Ahmed")
Pop
Push("Jatinder")
```

Add appropriate labels to the diagram to show the final state of the stack. Use the space on the left as a workspace. Show your final answer in the node shapes on the right:



[3]

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

```
TYPE Node
  DECLARE Name : STRING
  DECLARE Pointer : INTEGER
ENDTYPE
```

The statement

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

reserves space for 10 nodes in array `Stack`.

(i) The `CreateStack` operation links all nodes and initialises the `TopOfStackPointer` and `FreePointer`.

Complete the diagram to show the value of all pointers after `CreateStack` has been executed.

TopOfStackPointer

FreePointer

Stack

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

[4]

(ii) The algorithm for adding a name to the stack is written, using ps procedure with the header

```
PROCEDURE Push (NewName)
```

Where `NewName` is the new name to be added to the stack. The procedure uses variables as shown in the identifier table.

Identifier	**Data type**	**Description**
`Stack`	`Array[1:10] OF Node`	
`NewName`	`STRING`	Name to be added
`FreePointer`	`INTEGER`	Pointer to next free node in array
`TopOfStackPointer`	`INTEGER`	Pointer to first node in stack
`TempPointer`	`INTEGER`	Temporary store for copy of `FreePointer`

```
PROCEDURE Push(BYVALUE NewName : STRING)
  // Report error if no free nodes remaining
  IF FreePointer = 0
     THEN
        Report Error
     ELSE
         // new name placed in node at head of free list
         Stack[FreePointer].Name ← NewName
         // take a temporary copy and
         // then adjust free pointer
         TempPointer ← FreePointer
         FreePointer ← Stack[FreePointer].Pointer
         // link current node to previous top of stack
         Stack[TempPointer].Pointer ← TopOfStackPointer
         // adjust TopOfStackPointer to current node
         TopOfStackPointer ← TempPointer
  ENDIF
ENDPROCEDURE
```

Complete the **pseudocode** for the procedure `Pop`. Use the variables listed in the iden

```
PROCEDURE Pop()

  // Report error if Stack is empty

.......................................................................................................

.......................................................................................................

.......................................................................................................

.......................................................................................................

    OUTPUT Stack[.......................................................].Name

    // take a copy of the current top of stack pointer

.......................................................................................................

    // update the top of stack pointer

.......................................................................................................

    // link released node to free list

.......................................................................................................

.......................................................................................................

.......................................................................................................

ENDPROCEDURE
```

[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. T
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.

1 A linked list abstract data type (ADT) is to be used to store and organise surname

This will be implemented with a 1D array and a start pointer. Elements of the array c user-defined type. The user-defined type consists of a data value and a link pointer.

Identifier	Data type	Description
`LinkedList`	`RECORD`	User-defined type
`Surname`	`STRING`	Surname string
`Ptr`	`INTEGER`	Link pointers for the linked list

(a) **(i)** Write **pseudocode** to declare the type `LinkedList`.

....

....

....

....[3]

(ii) The 1D array is implemented with an array `SurnameList` of type `LinkedList`.

Write the **pseudocode** declaration statement for `SurnameList`. The lower and upper bounds of the array are 1 and 5000 respectively.

....[2]

(b) The following surnames are organised as a linked list with a start pointer `StartPtr`.

`StartPtr: 3`

	1	2	3	4	5	6		5000
Surname	Liu	Yang	Chan	Wu	Zhao	Huang	...	
Ptr	4	5	6	2	0	1	...	

State the value of the following:

(i) `SurnameList[4].Surname` ....[1]

(ii) `SurnameList[StartPtr].Ptr` ....[1]

(c) Pseudocode is to be written to search the linked list for a surname input by t

Identifier	**Data type**	**Description**
`ThisSurname`	`STRING`	The surname to search for
`Current`	`INTEGER`	Index to array `SurnameList`
`StartPtr`	`INTEGER`	Index to array `SurnameList`. Points to the element at the start of the linked list

(i) Study the pseudocode in **part (c)(ii)**.

Complete the table above by adding the missing identifier details. [2]

(ii) Complete the pseudocode.

```
Current ← ..........................................................................................
IF Current = 0
   THEN
      OUTPUT ....................................................................................
   ELSE
      IsFound ← .................................................................................
      INPUT ThisSurname
      REPEAT
         IF ....................................................................... = ThisSurname
            THEN
               IsFound ← TRUE
               OUTPUT "Surname found at position ", Current
            ELSE
               // move to the next list item
               ..........................................................................
         ENDIF
      UNTIL IsFound = TRUE OR ..................................................................
      IF IsFound = FALSE
         THEN
            OUTPUT "Not Found"
      ENDIF
ENDIF
```

[6]

QUESTION 11.

4 An ordered linked list Abstract Data Type (ADT) has these associated operations.

- create list
- add item to list
- output list to console

The ADT is to be implemented using object-oriented programming as a linked list of nodes.

Each node consists of data and a pointer.

(a) There are two classes, `LinkedList` and `Node`.

(i) State the term used to describe the relationship between these classes.

....[1]

(ii) Draw the appropriate diagram to represent this relationship. Do not list the attributes and methods of the classes.

[2]

(b) The design for the `Node` class consists of:

- attributes
  - `Data : STRING`
  - `Pointer : INTEGER`
- methods
  - `CreateNode(Data, Pointer)`
  - `SetData(Data)`
  - `SetPointer(Pointer)`
  - `GetData() RETURNS STRING`
  - `GetPointer() RETURNS INTEGER`

The constructor method sets the attributes to the initial values that are passed as parameters.

Write **program code** for:

- the `Node` class declaration
- the constructor.

Programming language used ....

Program code

....

....

....

....

....

....

....

....

....

....

....

....

....

....

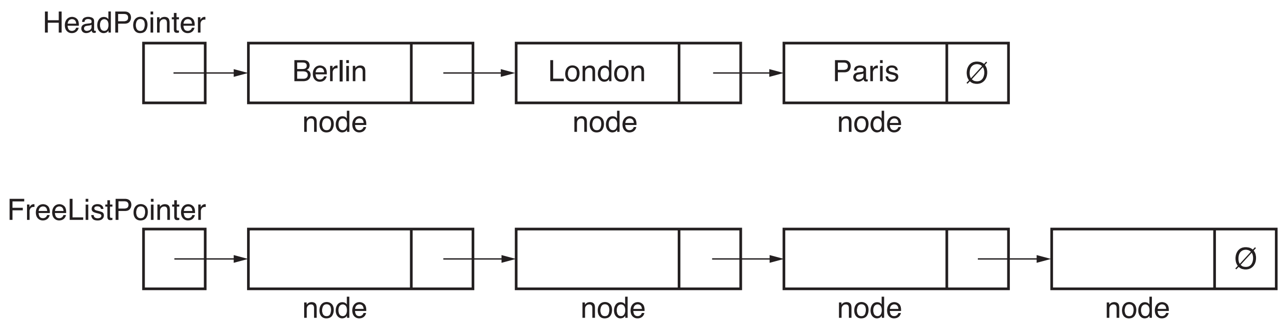
....

....[5]

(c) The identifier table for the `LinkedList` class is:

Identifier	**Data type**	**Description**
`HeadPointer`	`INTEGER`	Pointer to the first node in the ordere
`FreeListPointer`	`INTEGER`	Pointer to the first node in the free list.
`NodeArray`	`ARRAY[0 : 7] OF Node`	1D array stores the nodes that make the ordered linked list. The unused nodes are linked together into a free list.
`Constructor()`		Constructor instantiates an object of `LinkedList` class, initialises `HeadPointer` to be a null pointer and links all nodes to form the free list.
`FindInsertionPoint()`		Procedure that takes the new data item as the parameter `NewData` and returns two parameters: • `PreviousPointer`, whose value is: ◦ either pointer to node before the insertion point ◦ or the null pointer if the new node is to be inserted at the beginning of the list. • `NextPointer`, whose value is a pointer to node after the insertion point.
`AddToList(NewString)`		Procedure that takes as a parameter a unique string and links it into the correct position in the ordered list.
`OutputListToConsole()`		Procedure to output all the data from the list pointed to by `HeadPointer`.

The following diagram shows an example of a linked list object. This example list consists of three nodes, linked in alphabetical order of the data strings. The unused nodes are linked to form a free list.



The symbol Ø represents a null pointer.

(i) Explain the meaning of the term **null pointer**.

....

[1]

(ii) Give an appropriate value to represent the null pointer for this des answer.

....

....

....

.... [2]

(iii) Write **program code** for the `LinkedList` class declaration **and** the constructor.

Programming language used ....

Program code

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

.... [7]

(iv) Write **program code** to instantiate a linked list object with the `contact`

Programming language used ....

Program code

....

....[1]

(v) The `OutputListToConsole` method is to output all the data stored in the linked list. `HeadPointer` points to the first list node.

Write **program code** for this method.

Programming language used ....

Program code

....

....

....

....

....

....

....

....

....

....

....[5]

Question 4 continues on page 14.

(vi) The structured English for the `AddToList(NewString)` method is as

```
Make a copy of the value of free list pointer, name it NewNod

Store new data item in free node pointed to by NewNodePointe

Adjust free list pointer to point to next free node

IF linked list is currently empty

   THEN

      Make this node the first node

      Set pointer of this node to null pointer

   ELSE

      Find insertion point using the FindInsertionPoint method

      // FindInsertionPoint provides

      // pointer to previous node and pointer to next node

      IF previous pointer is null pointer

         THEN

            Link this node to front of list

         ELSE

            Link this node between previous node and next node
```

The `FindInsertionPoint` method receives the new data item as the parameter `NewString`. It returns two parameters:

- `PreviousPointer`, whose value is:
  - either the pointer to the node before the insertion point
  - or the null pointer, if the new node is to be inserted at the beginning of the list.
- `NextPointer`, whose value is the pointer to the node after the insertion point.

Write **program code** for the `AddToList` method.

Programming language used ....

Program code

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....[6]

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.

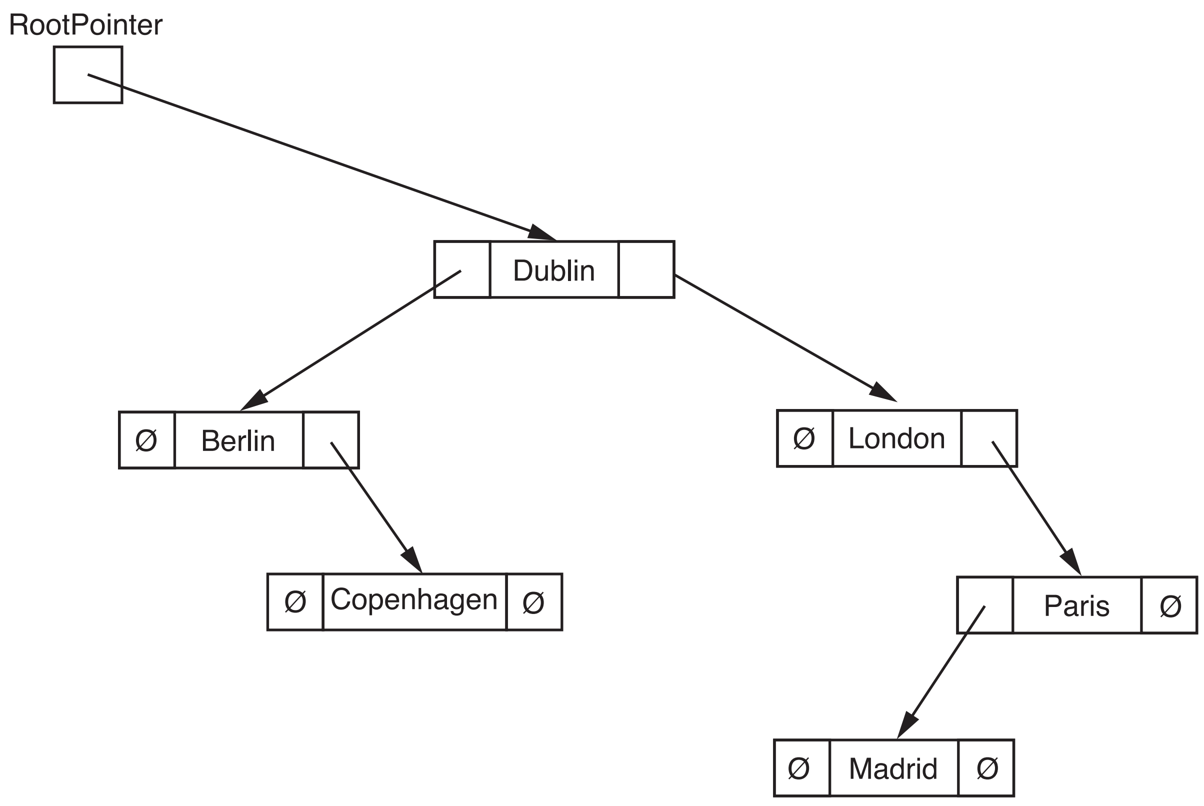
Explain the meaning of the term **null pointer**.

....

....[1]

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

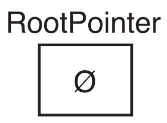
Dublin, London, Berlin, Paris, Madrid, Copenhagen



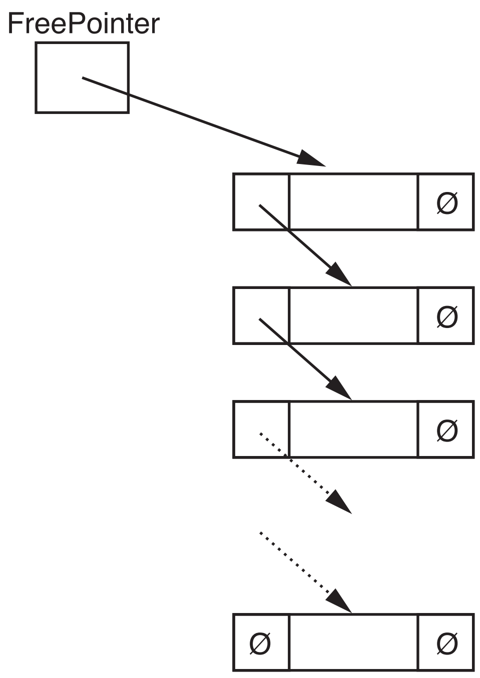
Another data item to be added is Athens.

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

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



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.

2 Kendra collects books. She is writing a program to store and analyse information

Her program stores information about each book as a record. The following table information that will be stored about each book.

Field name	**Description**
`Title`	The title of the book
`Author`	The first listed author of the book
`ISBN`	A 13-digit code that uniquely identifies the book, for example: `"0081107546738"`
`Fiction`	If the book is fiction (`TRUE`) or non-fiction (`FALSE`)
`LastRead`	The date when Kendra last read the book

(a) Write **pseudocode** to declare an Abstract Data Type (ADT) named `Book`, to store the information in the table.

....

....

....

....

....

....

....

....

.... [4]

(b) The records are stored in a random access file.

The function, `Hash()`, takes as a parameter the ISBN and returns the hash value.

The disk address of the record in the hash table is calculated as: ISBN modulus 2000 p

Write **program code** for the function `Hash()`.

Programming language ....

Program code ....

....

....

....

....

....

....

.... [4]

(c) The random access file, `MyBooks.dat`, stores the data about the books in t

```
<Title>
<Author>
<ISBN>
<Fiction>
<LastRead>
```

A procedure, `FindBook()`:

- prompts the user to input the ISBN of a book until the ISBN contains 13 numeric digits
- uses the function `Hash()` to calculate the disk address of the record
- reads the record for that book from `MyBooks.dat` into a variable of type `Book`
- outputs all the data about the book.

Use **pseudocode** to write the procedure `FindBook()`.

You can assume that the record exists at the disk address generated.

[8]

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)
```

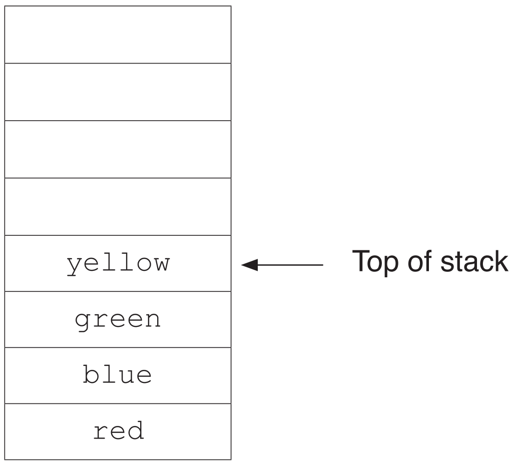
Complete the following **pseudocode** to implement the `DeleteNode` procedure.

```
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.

1 **(a)** A stack contains the values `'red'`, `'blue'`, `'green'` and `'yellow'`.

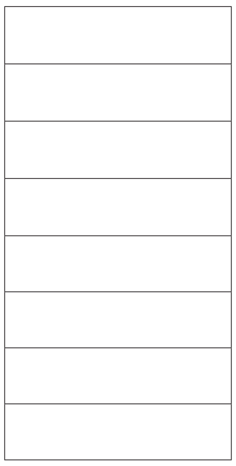


(i) Show the contents of the stack in **part(a)** after the following operations.

```
POP()

PUSH('purple')

PUSH('orange')
```



[1]

(ii) Show the contents of the stack from **part(a)(i)** after these further operat

```
POP()

POP()

PUSH('brown')

POP()

PUSH('black')
```


[1]

(b) A queue is an alternative Abstract Data Type (ADT).

Describe a **queue**.

....

....

....

....

....

.... [3]

QUESTION 16.

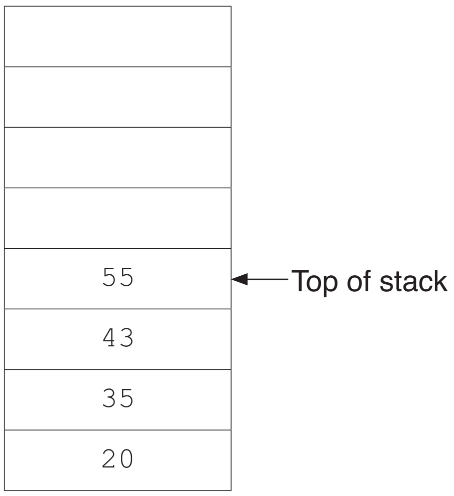
2 A stack is an Abstract Data Type (ADT).

(a) Tick (✓) **one** box to show the statement that describes a stack data structure.

Statement	**Tick (✓)**
Last in first out	
First in first out	
Last in last out	

[1]

(b) A stack contains the values `20`, `35`, `43`, `55`.



(i) Show the contents of the stack in **part (b)** after the following operations.

```
POP()

POP()

PUSH(10)
```

[1]

(ii) Show the contents of the stack from **part (b)(i)** after these further opera

```
POP()

PUSH(50)

PUSH(55)

POP()

PUSH(65)
```


[1]

(iii) The stack is implemented as a 1D array, with eight elements, and giv
`ArrayStack`.

The global variable `Top` contains the index of the last element in the stack, or
stack is empty.

The function `Push()`:

- takes as a parameter an `INTEGER` value to place on the stack
- adds the value to the top of the stack and returns `TRUE` to show that the operation was successful
- returns `FALSE` if the stack is full.

Write an algorithm in **pseudocode** for the function `Push()`.

[7]

QUESTION 17.

2 The number of cars that cross a bridge is recorded each hour. This number is placed in a circular queue before being processed.

(a) The queue is stored as an array, `NumberQueue`, with eight elements. The function `AddToQueue` adds a number to the queue. `EndPointer` and `StartPointer` are global variables.

Complete the following **pseudocode** algorithm for the function `AddToQueue`.

```
FUNCTION AddToQueue(Number : INTEGER) RETURNS BOOLEAN

   DECLARE TempPointer : INTEGER

   CONSTANT FirstIndex = 0

   CONSTANT LastIndex = .....................................................

   TempPointer ← EndPointer + 1

   IF ..................................................... > LastIndex

       THEN

          TempPointer ← .....................................................

   ENDIF

   IF TempPointer = StartPointer

       THEN

          RETURN .....................................................

       ELSE

          EndPointer ← TempPointer

          NumberQueue[EndPointer] ← .....................................................

          RETURN TRUE

   ENDIF

ENDFUNCTION
```

[5]

(b) Describe how a number is removed from the circular queue to be processed.

....

....

....

....

....

....

....

.... [4]

(c) A queue is one example of an Abstract Data Type (ADT).

Identify **three other** Abstract Data Types.

1 ....

2 ....

3 ....

[3]