PAPER 3 ADT

COMPUTER SCIENCE

Topic: Paper 3 ADT

INSTRUCTIONS

- Carry out every instruction in each task.
- Answer **all** questions.
- Use a black or dark blue pen.
- You may use an HB pencil for any diagram, graphs or rough working.
- **Calculator Not Allowed**.
- Show your workings if relevant.

INFORMATION

- The number of marks for each question or part question is shown in brackets [ ].

(d) The function `StackFull()` checks whether a stack is full.

The function uses the variable `TopOfStack` to represent the pointer to the most recent position used on the stack, and the variable `Max` to represent the maximum size of the stack. Assume `TopOfStack` and `Max` are global variables.

```
FUNCTION StackFull() RETURNS BOOLEAN
    IF TopOfStack = Max THEN
       RETURN TRUE
    ELSE
       RETURN FALSE
    ENDIF
ENDFUNCTION
```

An algorithm `AddInteger` is required to add a new integer data element to a stack.

The stack is implemented as an array `ArrayStack`.

The function `AddInteger()` calls `StackFull()` and returns an appropriate message.

Complete the pseudocode for the function `AddInteger()`.

```
FUNCTION AddInteger(NewInteger : INTEGER) RETURNS STRING
```

....

....

....

....

....

....

....

....

....

....

....

....

```
ENDFUNCTION
```

[5]

11 A simplified linked list is used to store the names of flowers in alphabetical order. It is implemented using two 1D arrays:

- `Flower` stores the names of the flowers.
- `NextPointer` stores the pointer to the next flower name in the list.

`HeadPointer` indicates the index of the first flower name in the linked list.

HeadPointer	6

When the end of the linked list is reached, the next pointer has the value of 0.

The following table shows the initial content of the arrays.

Index	**Flower**	**NextPointer**
1	Rose	7
2	Marigold	1
3	Foxglove	10
4	Iris	9
5	Daisy	3
6	Dahlia	5
7	Saxifrage	0
8	Lupin	2
9	Lily	8
10	Hydrangea	4

(a) Several flower names have been deleted from the linked list. These are crossed out in the following table.

Complete the table to show the new values of `HeadPointer` and `NextPointer` to keep the remaining flower names in alphabetical order.

HeadPointer	

Index	**Flower**	**NextPointer**
1	Rose	
2	~~Marigold~~	
3	Foxglove	
4	Iris	
5	Daisy	
6	~~Dahlia~~	
7	~~Saxifrage~~	
8	Lupin	
9	Lily	
10	~~Hydrangea~~	

[3]

(b) Complete the pseudocode algorithm so that it achieves the following when applied to the arrays:

- The flower name is input.
- The linked list is searched, in order, for the flower name.
- If the flower name is found, an appropriate message is output to indicate it has been found.
- If the flower name is not found, an appropriate message is output to indicate it has not been found.
- The algorithm terminates when the next pointer value is 0.

```
Pointer ← HeadPointer
Found ← 0
OUTPUT "Enter a flower name "

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

.....................................................................
   IF Flower[Pointer] = FlowerName THEN
      Found ← Pointer
      Pointer ← 0
   ELSE

      .....................................................................
   ENDIF
ENDWHILE

.....................................................................
   OUTPUT Flower[Found], " is found"
ELSE

   .....................................................................
ENDIF
```

[5]

(c) Explain how you could improve the simplified linked list structure.

....

....

....

.... [2]

10 The pseudocode algorithm shown copies an active accounts text file `ActiveFile.txt` to an archive accounts text file `ArchiveFile.txt`, one line at a time. Any blank lines found in the active accounts text file are replaced with the words "`Account not present`" in the archive accounts text file.

Complete this file-handling pseudocode.

```
DECLARE Account : STRING

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

OPENFILE "ArchiveFile.txt" FOR WRITE

WHILE NOT ..............................................................................................................
   READFILE "ActiveFile.txt", Account
   IF Account = "" THEN

      WRITEFILE "ArchiveFile.txt", " ........................................................................"
   ELSE

      WRITEFILE "ArchiveFile.txt", ..........................................................................
   ENDIF
ENDWHILE

.............................................................................................................................
CLOSEFILE "ArchiveFile.txt"
```

[5]

11 Pseudocode is to be written to implement a queue Abstract Data Type (ADT) with items of the string data type. This will be implemented using the information in the table.

Identifier	**Data type**	**Description**
`FrontPointer`	`INTEGER`	points to the start of the queue
`RearPointer`	`INTEGER`	points to the end of the queue
`Length`	`INTEGER`	the current size of the queue
`Queue`	`STRING`	1D array to implement the queue

A constant, with identifier `MaxSize`, limits the size of the queue to 60 items.

(a) Write the pseudocode to declare `MaxSize`, `FrontPointer`, `RearPointer`, `Length` and `Queue`.

....

....

....

....

....

.... [3]

(b) Complete the following pseudocode for the function Dequeue to remove the front item from the queue.

```
FUNCTION Dequeue RETURNS STRING
   DECLARE Item : STRING

   .................................................................................... > 0 THEN

      Item ← ..........................................................................

      .................................................................................
      IF Length = 0 THEN
         CALL Initialise // reset the pointers
      ELSE
         IF FrontPointer > MaxSize THEN

         ............................................................................ ← 1
         ENDIF
      ENDIF
   ELSE
      OUTPUT "The print queue was empty - error!"
      Item ← ""
   ENDIF
   RETURN Item
ENDFUNCTION
```

[4]

(c) Explain how a new element can be added to the queue if it is implemented using two stacks.

....

....

....

....

....

.... [4]

12 (a) Describe what is meant by recursion.

....

....

....

.... [2]

11 (a) The pseudocode shown represents a queue Abstract Data Type (ADT) with procedures for initialisation and to add new items. It is incomplete.

```
CONSTANT MaxLength = 50
DECLARE FrontPointer : INTEGER
DECLARE RearPointer : INTEGER
DECLARE Length : INTEGER
DECLARE Queue : ARRAY[0 : MaxLength - 1] OF STRING

// initialisation of queue
PROCEDURE Initialise
   FrontPointer ← -1

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

   .......................................................... ← 0
ENDPROCEDURE

// adding a new item to the queue
PROCEDURE Enqueue(NewItem : STRING)

   IF .................................................................................... THEN

      RearPointer ← ..............................................................................
      IF RearPointer > MaxLength - 1 THEN
         RearPointer ← 0
      ENDIF

      ..........................................................................................................
      Length ← Length + 1
   ENDIF
ENDPROCEDURE
```

(i) Study the pseudocode and insert the identifiers to complete this table.

Identifier	Data type	Description
	STRING	An array to store the contents of the queue.
	INTEGER	Points to the last item of the queue.
	INTEGER	Indicates the number of items in the queue.
	INTEGER	Points to the first item of the queue.

[2]

(ii) Complete the given pseudocode. [5]

(b) Explain the reasons why a queue ADT works better than a stack ADT in organising print jobs.

....

....

....

....

....

.... [3]

10 The pseudocode algorithm shown copies an active accounts text file `ActiveFile.txt` to an archive accounts text file `ArchiveFile.txt`, one line at a time. Any blank lines found in the active accounts text file are replaced with the words "`Account not present`" in the archive accounts text file.

Complete this file-handling pseudocode.

```
DECLARE Account : STRING

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

OPENFILE "ArchiveFile.txt" FOR WRITE

WHILE NOT ..................................................................................
   READFILE "ActiveFile.txt", Account
   IF Account = "" THEN

      WRITEFILE "ArchiveFile.txt", "..............................................."
   ELSE

      WRITEFILE "ArchiveFile.txt", ................................................
   ENDIF
ENDWHILE

............................................................................................
CLOSEFILE "ArchiveFile.txt"
```

[5]

11 Pseudocode is to be written to implement a queue Abstract Data Type (ADT) with items of the string data type. This will be implemented using the information in the table.

Identifier	Data type	Description
`FrontPointer`	`INTEGER`	points to the start of the queue
`RearPointer`	`INTEGER`	points to the end of the queue
`Length`	`INTEGER`	the current size of the queue
`Queue`	`STRING`	1D array to implement the queue

A constant, with identifier `MaxSize`, limits the size of the queue to 60 items.

(a) Write the pseudocode to declare `MaxSize`, `FrontPointer`, `RearPointer`, `Length` and `Queue`.

....

....

....

....

....

.... [3]

(b) Complete the following pseudocode for the function `Dequeue` to remove the front item from the queue.

```
FUNCTION Dequeue RETURNS STRING
   DECLARE Item : STRING

   ......................................................................................... > 0 THEN

      Item ← ..........................................................................................

      .................................................................................................
      IF Length = 0 THEN
         CALL Initialise // reset the pointers
      ELSE
         IF FrontPointer > MaxSize THEN

         ........................................................................................ ← 1
         ENDIF
      ENDIF
   ELSE
      OUTPUT "The print queue was empty - error!"
      Item ← ""
   ENDIF
   RETURN Item
ENDFUNCTION
```

[4]

(c) Explain how a new element can be added to the queue if it is implemented using two stacks.

....

....

....

....

....

.... [4]

12 (a) Describe what is meant by recursion.

....

....

....

.... [2]

10 A stack is to be set up using the information in the table.

Identifier	**Data type**	**Description**
`BasePointer`	`INTEGER`	points to the bottom of the stack
`TopPointer`	`INTEGER`	points to the top of the stack
`Stack`	`REAL`	1D array to implement the stack

A constant, with identifier `Capacity`, limits the size of the stack to 25 items.

(a) Write the **pseudocode** for the required declarations.

....

....

....

....

....

.... [3]

(b) Complete the pseudocode function `Pop()` to pop an item from `Stack`.

```
// popping an item from the stack

FUNCTION Pop() .......................................................................
   DECLARE Item : REAL
   Item ← 0

   ....................................................................... BasePointer THEN

      Item ← .......................................................................

      TopPointer ← .......................................................................
   ELSE
      OUTPUT "The stack is empty - error"
   ENDIF

   .......................................................................
ENDFUNCTION
```

[5]

(c) Compare and contrast the queue and stack Abstract Data Types (ADT).

....

....

....

....

....

.... [2]

9 **(a)** A stack Abstract Data Type (ADT) is to be implemented using pseudocode, with procedures to initialise it and to push new items onto the stack.

A 1D array `Stack` stores the contents of the stack.

(i) Study the pseudocode in **part (a)(ii)** and complete the table of identifiers by writing the missing data types and descriptions.

Identifier	Data type	Description
BasePointer		
TopPointer		
Stack	REAL	

[2]

(ii) Complete the pseudocode.

```
CONSTANT MaxSize = 40
DECLARE BasePointer : INTEGER
DECLARE TopPointer : INTEGER
DECLARE Stack : ARRAY[1:40] OF REAL

// initialisation of stack
PROCEDURE Initialise()

   ........................................................................ ← 1

   ........................................................................ ← 0
ENDPROCEDURE

// push an item onto the stack
PROCEDURE Push(NewItem : REAL)

   ........................................................................ MaxSize THEN

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

      Stack[TopPointer] ← .........................................................
   ENDIF
ENDPROCEDURE
```

[5]

(b) Justify the use of a linked list instead of an array to implement a stack.

....

....

....

.... [2]

(c) Explain how a compiler makes use of a stack when translating recursive programming code.

....

....

....

....

....

....

....

.... [4]

10 Describe the features of the SIMD and MISD computer architectures.

SIMD ....

....

....

....

MISD ....

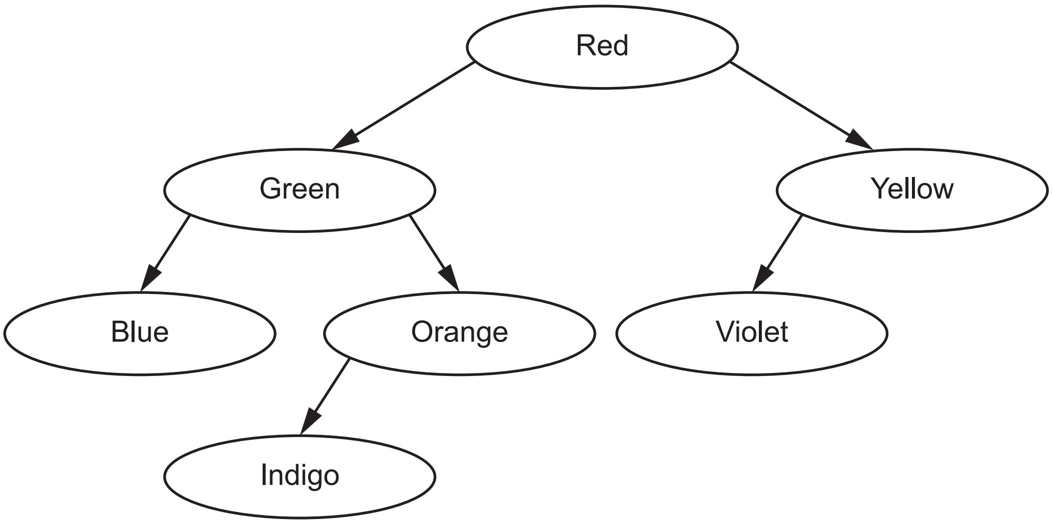
....

....

....

[4]

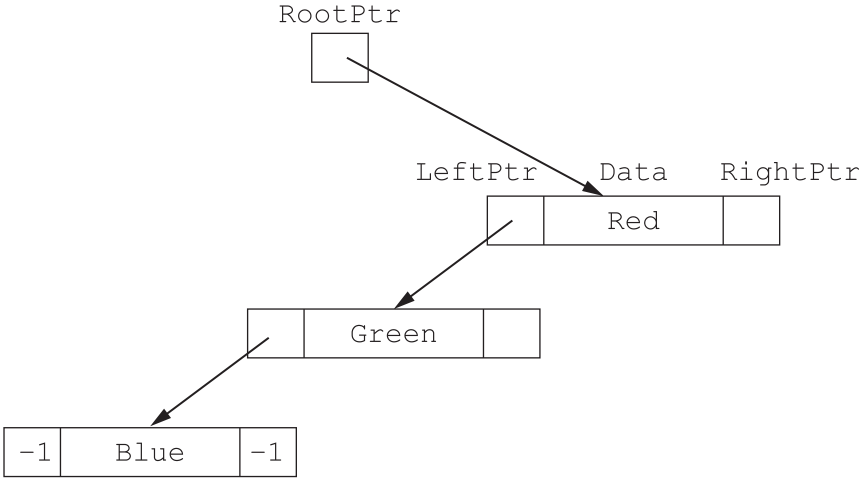
11 The following diagram shows an ordered binary tree.



(a) A linked list of nodes is used to store the data. Each node consists of a left pointer, the data and a right pointer.

–1 is used to represent a null pointer.

Complete this linked list to represent the given binary tree.



[4]

(b) A user-defined record structure is used to store the nodes of the linked list in part **(a)**.

Complete the diagram, using your answer for part **(a)**.

RootPtr

`0`

FreePtr

` `

Index	**LeftPtr**	**Data**	**RightPtr**
0		Red	
1		Green	
2		Yellow	
3		Blue	
4		Orange	
5		Indigo	
6		Violet	
7			

[4]

(c) The linked list in part **(a)** is implemented using a 1D array of records. Each record contains a left pointer, data and a right pointer.

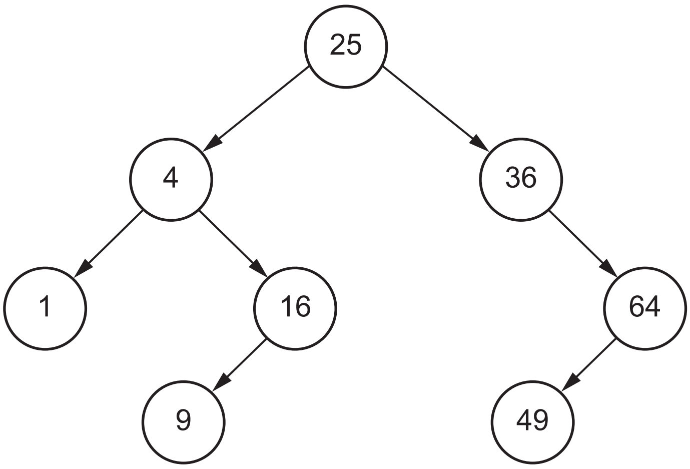
The following pseudocode represents a function that searches for an element in the array of records `BinTree`. It returns the index of the record if the element is found, or it returns a null pointer if the element is **not** found.

Complete the pseudocode for the function.

```
FUNCTION SearchTree(Item : STRING) ..........................................................

   NowPtr ← ..........................................................................................
   WHILE NowPtr <> -1

      IF ......................................................................................... THEN
         NowPtr ← BinTree[NowPtr].LeftPtr
      ELSE
         IF BinTree[NowPtr].Data < Item THEN

            ......................................................................................
         ELSE
            RETURN NowPtr
         ENDIF
      ENDIF
   ENDWHILE
   RETURN NowPtr
ENDFUNCTION
```

[4]

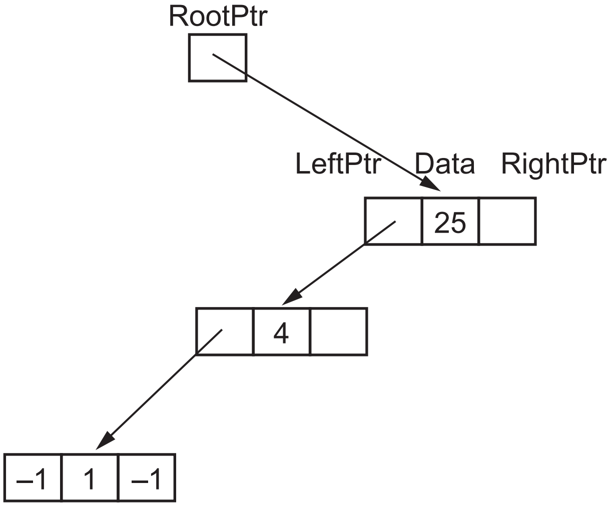
11 This binary tree shows an ordered list of integers.



(a) A linked list of nodes is used to store the data. Each node consists of a left pointer, the data and a right pointer.

−1 is used to represent a null pointer.

Complete this linked list to represent the given binary tree organisation.



[4]

(b) A 2D array is used to store the nodes of the linked list in part **(a)**.

Complete the diagram using your answer for part **(a)**.

RootPtr
0

FreePtr

Index	LeftPtr	Data	RightPtr
0		25	
1		4	
2		36	
3		1	
4		16	
5		64	
6		9	
7		49	
8			

[4]

(c) The linked list in part **(a)** is implemented using a 1D array of records. Each record contains a left pointer, data and a right pointer.

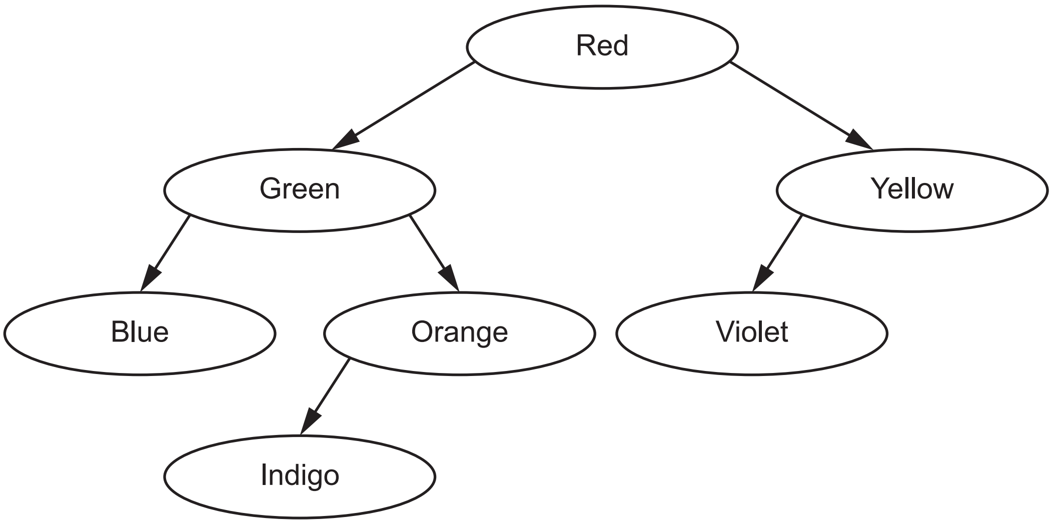
The following pseudocode represents a function that searches for an element in the array of records `LinkList`. It returns the index of the record if the element is found, or it returns a null pointer if the element is not found.

Complete the pseudocode for the function.

```
FUNCTION SearchList(Item : INTEGER) ..................................................................
   NullPtr ← -1

   ............................................................... ← RootPtr
   WHILE NowPtr <> NullPtr
      IF LinkList[NowPtr].Data < Item THEN
         NowPtr ← LinkList[NowPtr].RightPtr
      ELSE

         IF .......................................................................................... THEN

            NowPtr ← ............................................................................................
         ELSE
            RETURN NowPtr
         ENDIF
      ENDIF
   ENDWHILE
   RETURN NullPtr
ENDFUNCTION
```

[4]

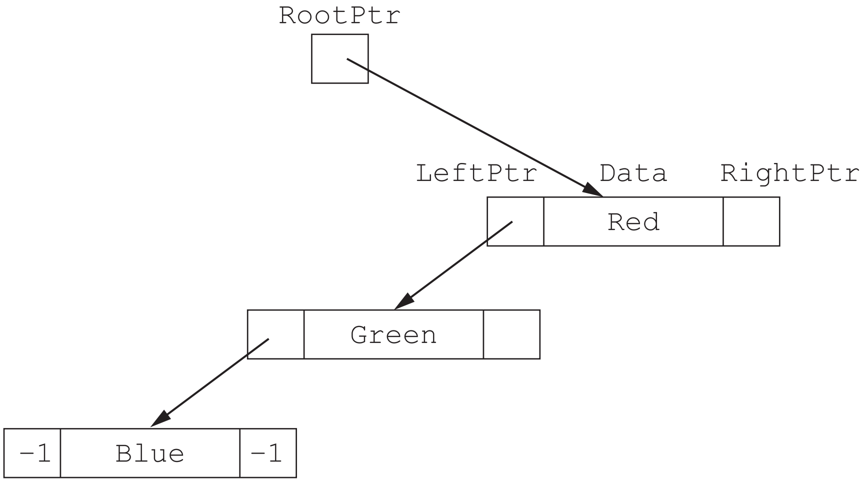
11 The following diagram shows an ordered binary tree.



(a) A linked list of nodes is used to store the data. Each node consists of a left pointer, the data and a right pointer.

–1 is used to represent a null pointer.

Complete this linked list to represent the given binary tree.



[4]

(b) A user-defined record structure is used to store the nodes of the linked list in part **(a)**.

Complete the diagram, using your answer for part **(a)**.

RootPtr

0

FreePtr

Index	**LeftPtr**	**Data**	**RightPtr**
0		Red	
1		Green	
2		Yellow	
3		Blue	
4		Orange	
5		Indigo	
6		Violet	
7			

[4]

(c) The linked list in part **(a)** is implemented using a 1D array of records. Each record contains a left pointer, data and a right pointer.

The following pseudocode represents a function that searches for an element in the array of records `BinTree`. It returns the index of the record if the element is found, or it returns a null pointer if the element is **not** found.

Complete the pseudocode for the function.

```
FUNCTION SearchTree(Item : STRING) ..............................................................

   NowPtr ← .........................................................................................
   WHILE NowPtr <> -1

      IF ........................................................................................... THEN
         NowPtr ← BinTree[NowPtr].LeftPtr
      ELSE
         IF BinTree[NowPtr].Data < Item THEN

            ...........................................................................................
         ELSE
            RETURN NowPtr
         ENDIF
      ENDIF
   ENDWHILE
   RETURN NowPtr
ENDFUNCTION
```

[4]