3 A program uses a stack to hold up to 30 integer values. The stack is implemented using a global integer variable and a global 1D array.

The array is declared in pseudocode:

```
DECLARE ThisStack : ARRAY[1:30] OF INTEGER
```

Stack design notes:

- The global variable `SP` acts as a stack pointer. `SP` contains the array index of the last value pushed onto the stack.
- If the stack is empty, then `SP` is assigned the value zero.
- The first item added to the stack will be stored in `ThisStack[1]`
- `SP` is incremented each time an item is added to the stack.

A function `Pop()` is written to remove an item from `ThisStack`. The function returns an item of type `PopData` which is defined in pseudocode:

```
TYPE PopData
   DECLARE Data   : INTEGER
   DECLARE Exists : BOOLEAN
ENDTYPE
```

The value removed from the stack is assigned to `Data` and `Exists` is set to `TRUE`. If it is **not** possible to remove a value from the stack, then `Exists` is set to `FALSE`

Complete the pseudocode for `Pop()`

```
FUNCTION Pop() RETURNS PopData

   DECLARE ThisPop : .....

   IF ..... THEN

      PopData.Exists ← ..... // Stack is empty

   ELSE

      PopData.Data ← .....

      PopData.Exists ← .....

      SP ← .....

   ENDIF

   RETURN ThisPop

ENDFUNCTION
```

[5]

9 **(a)** A stack has been implemented using pseudocode to store a maximum of 100 string items using the global variables in the following table:

Identifier	Data type	Description	Initialisation value
Base	INTEGER	pointer for the bottom of the stack	0
Top	INTEGER	pointer for the top of the stack	-1
StackArray	STRING	1D array to implement the stack	[0:99]
Max	INTEGER	maximum number of items in the stack	100

The value of `Top` is incremented each time a data item is added to the stack and decremented every time a data item is removed.

(i) Complete the **pseudocode** for the function to remove a data item from the stack.

```
FUNCTION Pop() .....
   DECLARE DataItem : STRING
   DataItem ← ""

   IF ..... THEN

      DataItem ← .....

      Top ← .....
   ELSE
      DataItem ← "You cannot remove data; the stack is empty"
   ENDIF

   .....
ENDFUNCTION
```

[5]

(ii) Write the **pseudocode** to output the data item removed from the stack with an appropriate message.

.....

..... [1]

(b) A stack is used to implement recursion.

State the **three** essential features of recursion.

1 .....

.....

2 .....

.....

3 .....

.....

[3]

10 Explain what is meant by **exception handling**.
Include an example of a possible cause of an exception in your answer.

Explanation .....

.....

.....

.....

Example .....

[3]

2 A program reads string data from a text file into a linear queue and then compresses the data.

The queue is created using the global 1D array `Queue`. The queue can store up to 100 elements. Each element is initialised to the empty string `""`

The queue has two pointers that are global to the program:

- `QueueHead` that points to the index of the first element in the queue, initialised to `-1`
- `QueueTail` that points to the index of the last element in the queue, initialised to `-1`

The global variable `NumberItems` stores the number of elements stored in the queue, initialised to `0`

(a) Write program code to declare and initialise `Queue`, `QueueHead`, `QueueTail` and `NumberItems`

Save your program as **Question2_N25**.

Copy and paste the program code into part **2(a)** in the evidence document.

[2]

(b) The function `Enqueue()` takes a string as a parameter. The function checks if the queue is full, and returns Boolean `FALSE` if the queue is full.

If the queue is not full, the parameter is inserted into the next position in `Queue`. The function updates the appropriate pointer(s), updates `NumberItems` and then returns Boolean `TRUE`

Write program code for `Enqueue()`

Save your program.

Copy and paste the program code into part **2(b)** in the evidence document.

[5]

(c) The function `Dequeue()` returns the string `"False"` if the queue is empty.

If the queue is not empty, the function returns the next element in the queue. The function updates the appropriate pointer(s) and updates `NumberItems`

Write program code for `Dequeue()`

Save your program.

Copy and paste the program code into part **2(c)** in the evidence document.

[3]

(d) The text file `BinaryData.txt` stores individual binary digits, `'1'` and `'0'`. Each digit is on a new line. For example, the first line in the text file stores `'1'`, the second line stores `'1'`

The procedure `ReadData()` reads in each line from the text file `BinaryData.txt` and inserts it into the queue using the appropriate method.

The procedure needs to work for a text file with any number of lines up to a maximum of 100.

Write program code for `ReadData()`

Save your program.

Copy and paste the program code into part **2(d)** in the evidence document.

[5]

(e) The string data in the text file is compressed.

The compression algorithm counts the number of times each binary digit appears consecutively, then stores the binary digit followed by the number of times it appears. The algorithm stores the compressed data in a single string.

For example, if the text file contains the data:

```
1
1
0
0
0
1
1
1
```

The compression algorithm will create the string `"120313"` because there are two `'1'` digits, followed by three `'0'` digits, followed by three `'1'` digits.

The procedure `Compress()` uses `Dequeue()` to remove each element from the queue in turn. The procedure then compresses the data following the compression algorithm described. The new compressed string is stored in the global variable `NewString`

You can assume that one binary digit will never appear more than nine times consecutively in the sequence.

You can assume that there will always be at least one item in the queue.

Write program code for `Compress()`

Save your program.

Copy and paste the program code into part **2(e)** in the evidence document.

[6]

(f) **(i)** Write program code for the main program to:

- call `ReadData()`
- call `Compress()`
- output the content of the compressed string.

Save your program.

Copy and paste the program code into part **2(f)(i)** in the evidence document.

[2]

(ii) Test your program.

Take a screenshot of the output(s).

Save your program.

Copy and paste the screenshot(s) into part **2(f)(ii)** in the evidence document.

[1]

5 Stacks and queues are both abstract data types.

A stack uses a top-of-stack pointer to indicate the location of the last item added to the stack.

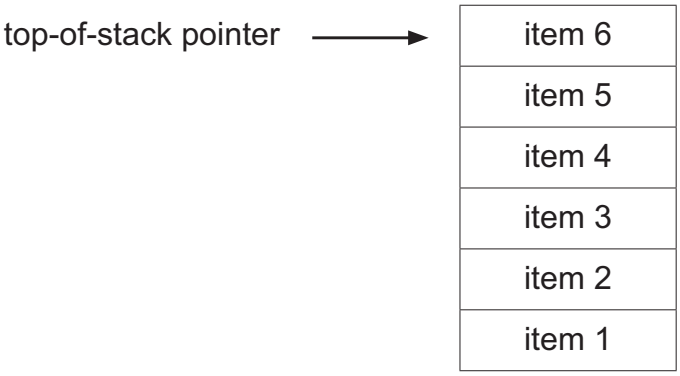
A queue uses two pointers:

- a front pointer to indicate the location of the next item to be removed from the queue
- a rear pointer to indicate the location of the next item to be added to the queue.

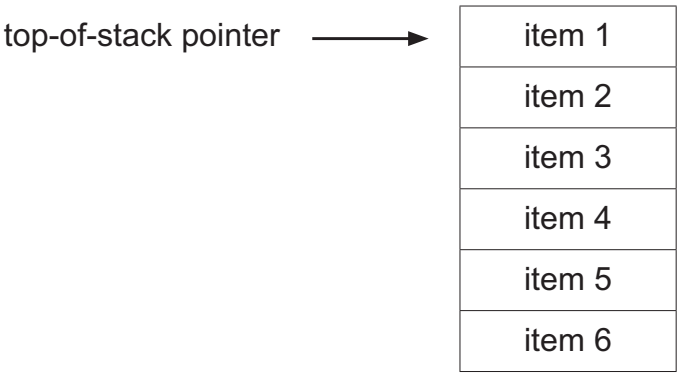
A queue can be used to reverse the items stored on a stack.

For example, if a stack contains six items:

Initial state of the stack:



Final state of the stack when the items have been reversed:



Describe how the queue could be used to reverse the items that are currently stored on the stack.

Your description must include how the pointers are used in both the stack and queue.

Assume:

- The stack initially contains an unknown number of items.
- The queue can store all the items currently stored on the stack.
- The queue is initially empty.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

2 A programmer uses a number of Abstract Data Types (ADT) in the programs that she is developing.

(a) A stack using memory locations 400 to 409 is used in one program.

The diagram represents the current state of the stack.

A variable `TopOfStack` pointer indicates the last value added to the stack.

Stack

Memory location	Value	
409		
408		
407		
406		
405		
404		
403	'P'	← `TopOfStack`
402	'X'	
401	'D'	
400	'B'	

(i) Complete the answer column in the following table:

	Answer
the memory location of the value that has been on the stack for the longest time	
the number of consecutive **push** operations that will result in the `TopOfStack` variable containing 409	

[2]

(ii) The diagram shows the current state of the stack.

Stack

Memory location	Value	
409		
408		
407	'A'	← TopOfStack
406	'C'	
405	'F'	
404	'K'	
403	'B'	
402	'S'	
401	'R'	
400	'D'	

The following sequence of operations are performed:

```
PUSH 'T'
POP
POP
PUSH 'Z'
PUSH 'X'
POP
PUSH 'Y'
```

Complete the following diagram to show the state of the stack after the operations have been performed.

Stack

Memory location	Value
409	
408	
407	
406	
405	
404	
403	
402	
401	
400	

10

[3]

(b) In a different program, the programmer decides to implement a linked list using an array of records.

(i) The array is represented in the diagram.

A pointer value of `-1` indicates the end of the linked list.

Complete the pointer field to produce a linked list that is in alphabetical order.

Index	Data	Pointer
0	"Neptune"	
1	"Jupiter"	
2	"Saturn"	
3	"Earth"	
4	"Mercury"	
5	"Uranus"	

[3]

(ii) The variable `StartPointer` contains the index of the first item in the linked list.

State the value of the variable `StartPointer`

..... [1]

(c) A third program is also being created to manage a queue.

Describe **two** features of a queue.

Feature one .....

.....

Feature two .....

.....

[2]

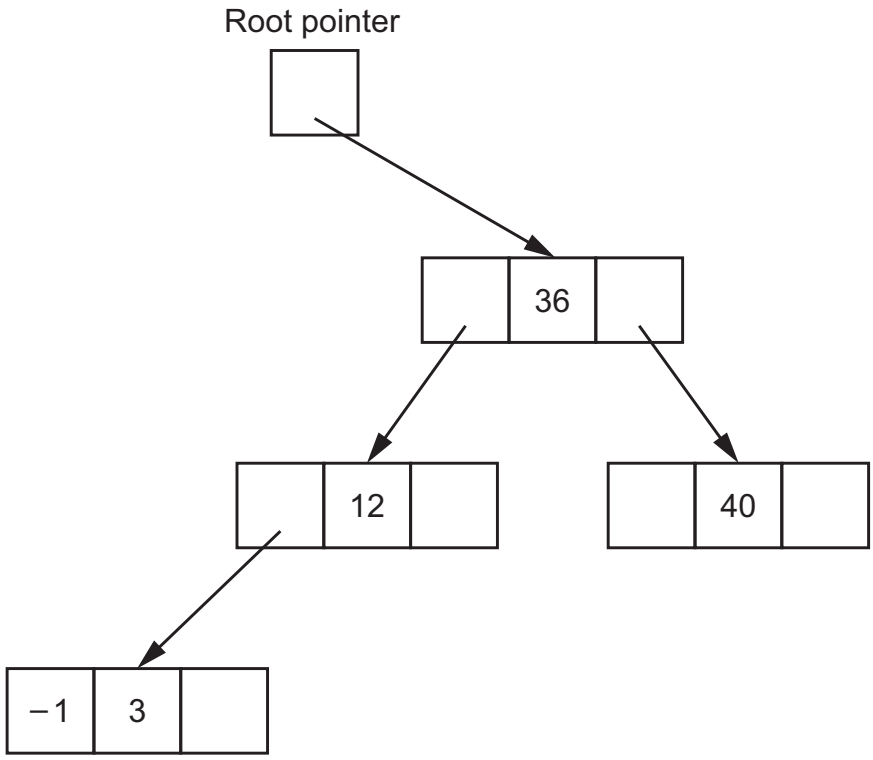
4 **(a)** A linked list of nodes is used to store an ordered list of integers. Each node consists of the data, a left pointer and a right pointer, for example:

The linked list will be organised as a binary tree.

−1 is used to represent a null pointer.

Complete the binary tree, including null pointers, to show how the data will be organised after the following integers have been added:

6, 15, 41, 66



[4]

(b) Describe what is meant by recursion.

.....

.....

.....

..... [2]

(c) A binary tree is a suitable Abstract Data Type (ADT) that a designer can implement using recursive algorithms.

Identify **one other** ADT that a designer can implement using recursive algorithms.

..... [1]

3 A program uses a stack to hold up to 60 numeric values.
The stack is implemented using two integer variables and a 1D array.

The array is declared in pseudocode as shown:

```
DECLARE ThisStack : ARRAY[1:60] OF REAL
```

The stack operates as follows:

- Global variable `SP` acts as a stack pointer that points to the next available stack location. The value of `SP` represents an array index.
- Global variable `OnStack` represents the number of values currently on the stack.
- The stack grows upwards from array element index 1.

(a) (i) Give the initial values that should be assigned to the **two** variables.

`SP` .....

`OnStack` .....
[1]

(ii) Explain why it is **not** necessary to initialise the array elements before the stack is used.

.....

.....

.....

..... [2]

(b) A function to add a value to `ThisStack` is expressed in pseudocode as shown.
The function will return a value to indicate whether the operation was successful or not.

Complete the pseudocode by filling in the gaps.

```
FUNCTION Push(ThisValue : REAL) RETURNS BOOLEAN
   DECLARE ReturnValue : BOOLEAN
   IF ..... THEN
      RETURN ..... // stack is already full
   ENDIF
   ..... ← ThisValue
   SP ← .....
   OnStack ← OnStack + 1
   RETURN TRUE
ENDFUNCTION
```

[4]

3 The implementation of a linked list uses an integer variable and a 1D array `List` of type `Node`.

Record type `Node` is declared in pseudocode as follows:

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

The array `List` is declared in pseudocode as follows:

```
DECLARE List : ARRAY[1:200] OF Node
```

The linked list will operate as follows:

- Integer variable `HeadPointer` will store the array index for the first node in the linked list.
- The `Pointer` field of a node contains the index value of the next array element (the next node) in the linked list.
- The value 0 is used as a null pointer.

(a) (i) State why the value 0 has been selected as the null pointer.

.....

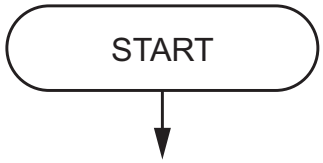
..... [1]

(ii) Give the range of valid values that could be assigned to variable `HeadPointer`.

..... [1]

(b) The array `List` will be initialised so that each node points to the following node. The last node will contain a null pointer.

Complete the program flowchart to represent the algorithm for this operation.



[4]

(c) An algorithm outputs the `Data` field from all nodes in the array `List`. The order the `Data` is output should be the same order it is stored in the linked list.

Describe the algorithm in **four** steps.

Do **not** use pseudocode statements in your answer.

Step 1 .....

.....

.....

Step 2 .....

.....

.....

Step 3 .....

.....

.....

Step 4 .....

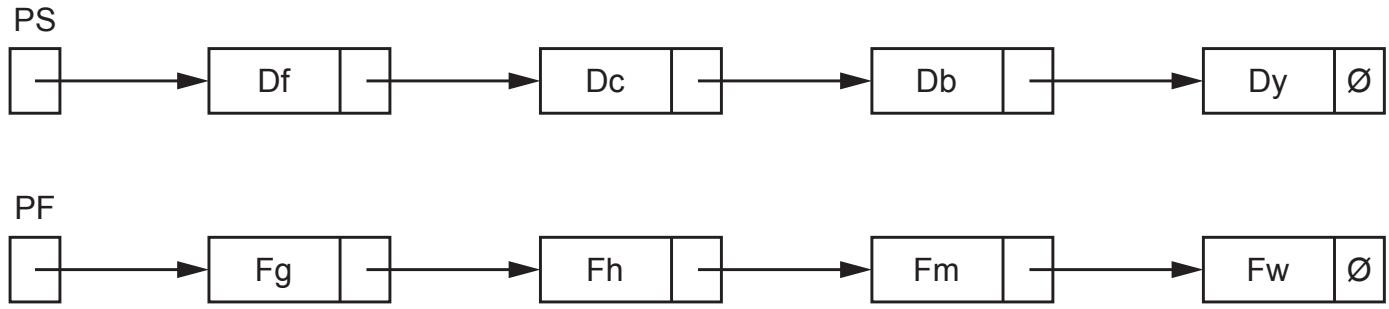
.....

.....

[4]

3 The diagram shows an Abstract Data Type (ADT) representation of a linked list after data items have been added.

- PS is the start pointer.
- PF is the free list pointer.
- Labels Df, Dc, Db and Dy represent the data items of nodes in the list.
- Labels Fg, Fh, Fm and Fw represent the data items of nodes in the free list.
- The symbol Ø represents a null pointer.



(a) Describe the linked list immediately after initialisation, before **any** data items are added.

.....

.....

.....

.....

.....

..... [3]

(b) A program will be written to include a linked list to store alphanumeric user IDs.

The design uses two variables and two 1D arrays to implement the linked list.
Each array element contains data of a single data type and **not** a record.

The statements below describe the design.

Complete the statements.

The two variables will be of type ..... .

The two variables will be used as ..... to the arrays.

The values stored in the two variables will indicate .....

..... .

The first 1D array will be of type ..... .

The first 1D array will be used to ..... .

The second 1D array will be of type ..... .

The second 1D array will be used to ..... .

[5]