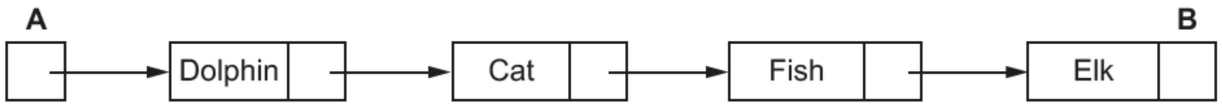
10.2. Computational thinking and problem-solving (Pastpapers 2015 – 2024)

- Abstraction
- 4.1.3 Abstract Data Types (ADT)

9618/22/M/J/21

1. The following diagram represents an Abstract Data Type (ADT).



(a) Identify this type of ADT.

.... [1]

(b) Give the technical term for the item labelled **A** in the diagram.

.... [1]

(c) Give the technical term for the item labelled **B** in the diagram.

Explain the meaning of the value given to this item.

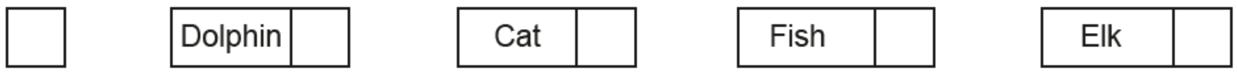
Term ....

Meaning ....

....

....[2]

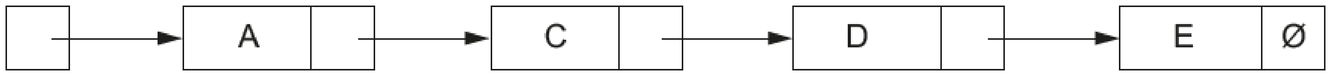
(d) Complete the diagram to show the ADT after the data has been sorted in alphabetical order.



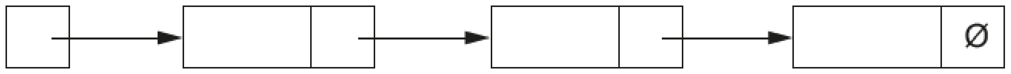
[2]

9618/22/M/J/21

2. The following diagram represents an Abstract Data Type (ADT) for a linked list.



The free list is as follows:



(a) Explain how a node containing data value B is added to the list in alphabetic sequence.

....

....

....

....

....

....

.... [4]

(b) Describe how the linked list in **part (a)** may be implemented using variables and arrays.

....

....

....

.... [2]

9618/22/M/J/23

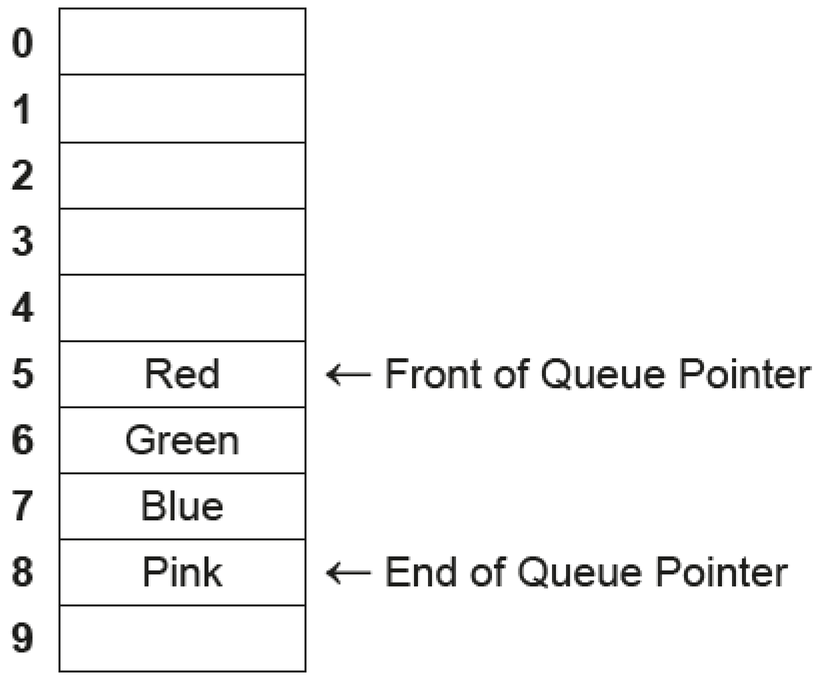
3 A program stores data in a text file. When data is read from the file, it is placed in a queue.

(a) The diagram below represents an Abstract Data Type (ADT) implementation of the queue.

Each data item is stored in a separate location in the data structure. During initial design, the queue is limited to holding a maximum of 10 data items.

The operation of this queue may be summarised as follows:

- The Front of Queue Pointer points to the next data item to be removed.
- The End of Queue Pointer points to the last data item added.
- The queue is circular so that locations can be reused.



(i) Describe how the data items Orange and Yellow are added to the queue shown in the diagram.

....

....

....

....

....

....

....

.... [4]

(ii) The following diagram shows the state of the queue after several operations have been performed. All queue locations have been used at least once.

0	D4	
1	D3	← End of Queue Pointer
2	D27	
3	D8	
4	D33	
5	D17	← Front of Queue Pointer
6	D2	
7	D1	
8	D45	
9	D60	

State the number of data items in the queue.

.... [1]

(b) The design of the queue is completed and the number of locations is increased.

A function AddToQueue() has been written. It takes a string as a parameter and adds this to the queue. The function will return TRUE if the string was added successfully.

A procedure FileToQueue() will add each line from the file to the queue. This procedure will end when all lines have been added or when the queue is full.

Describe the algorithm for procedure FileToQueue().

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

....

....

....

....

....

....

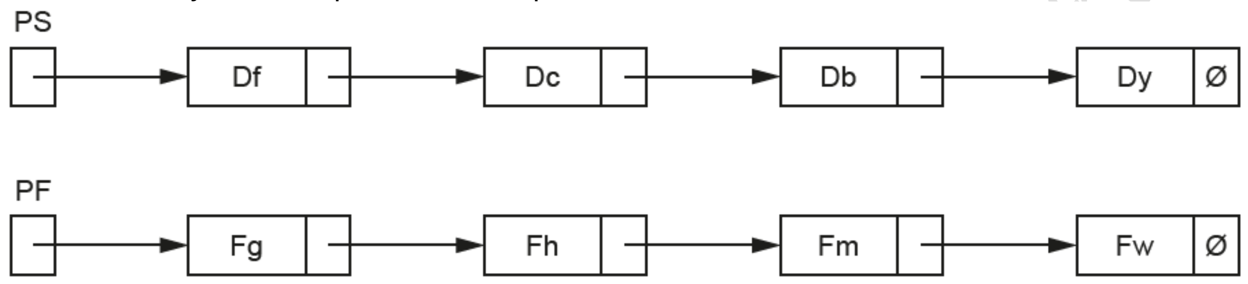
....

.... [5]

9618/21/M/J/24

4 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]**

9618/21/O/N/21
5 (a) The diagram below represents a queue Abstract Data Type (ADT) that can hold a maximum of eight items.
The operation of this queue may be summarised as follows:

- The front of queue pointer points to the next item to be removed.
- The end of queue pointer points to the last item added.
- The queue is circular so that empty storage elements can be reused.

0	Frog	← Front of queue pointer
1	Cat	
2	Fish	
3	Elk	← End of queue pointer
4		
5		
6		
7		

(i) Describe how "Octopus" is added to the given queue.

....

....

....

.... [2]

(ii) Describe how the next item in the given queue is removed and stored in the variable AnimalName.

....

....

.... [2]

(iii) Describe the state of the queue when the **front of queue** and the **end of queue** pointers have the same value.

....

.... [1]

(b) Some operations are carried out on the original queue given in **part (a)**.

(i) The current state of the queue is:

0	Frog
1	Cat
2	Fish
3	Elk
4	
5	
6	
7	

Complete the diagram to show the state of the queue after the following operations:
Add "**Wasp**", "**Bee**" and "**Mouse**", and then remove two data items. **[3]**

(ii) The state of the queue after other operations are carried out is shown:

0	Frog	
1	Cat	
2	Fish	
3	Elk	← Front of queue pointer
4	Wasp	
5	Bee	
6	Mouse	← End of queue pointer
7	Ant	

Complete the following diagram to show the state of the queue after the following operations:

Remove one item, and then add “Dolphin” and “Shark”.

0	
1	
2	
3	
4	
5	
6	
7	

[2]

(c) The queue is implemented using a 1D array.
Describe the algorithm that should be used to modify the **end of queue pointer** when adding an item to the queue.

Your algorithm should detect any potential error conditions.

....

....

....

....

....

.... [3]

9608/41/M/J/15
Q. 6 /- A queue Abstract Data Type (ADT) has these associated operations:

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

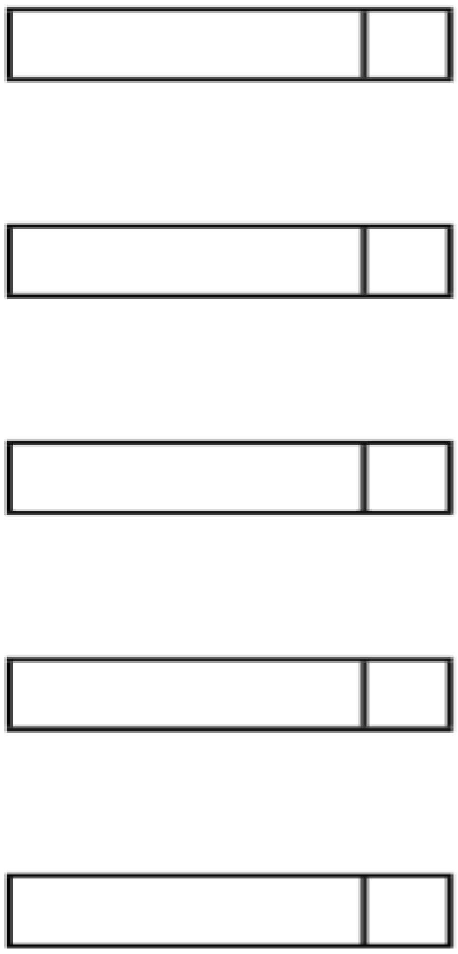
The queue ADT is to be implemented as a linked list of nodes.

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
END TYPE
```

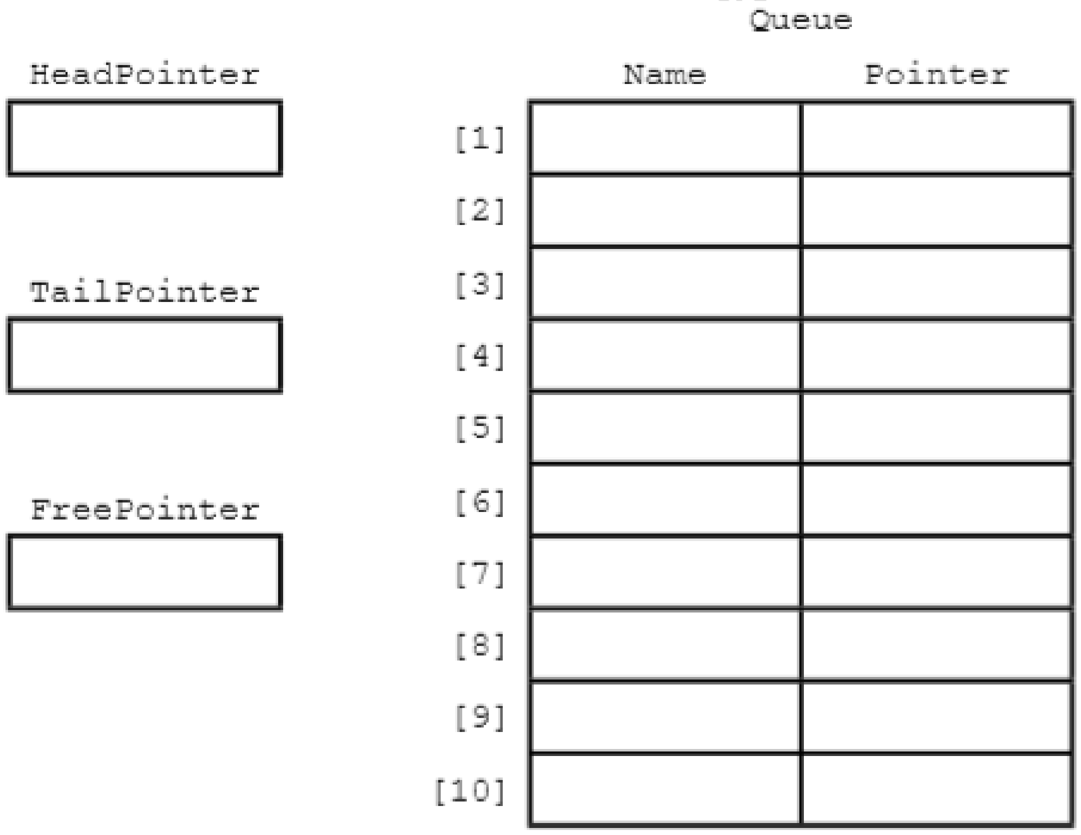
The statement

```
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.



[4]

9608/43/M/J/15
Q7/- A stack Abstract Data Type (ADT) has these associated operations:

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

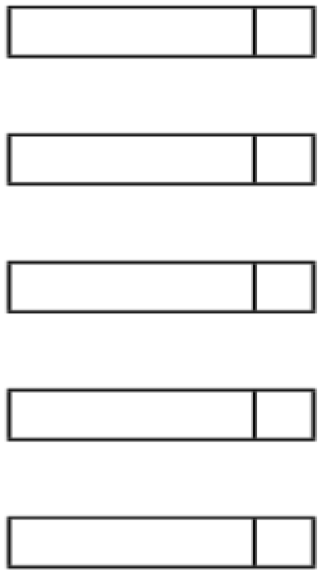
The stack ADT is to be implemented as a linked list of nodes.
Each node consists of data and a pointer to the next node.

(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]

(c) Using **pseudocode**, a record type, Node, is declared as follows:

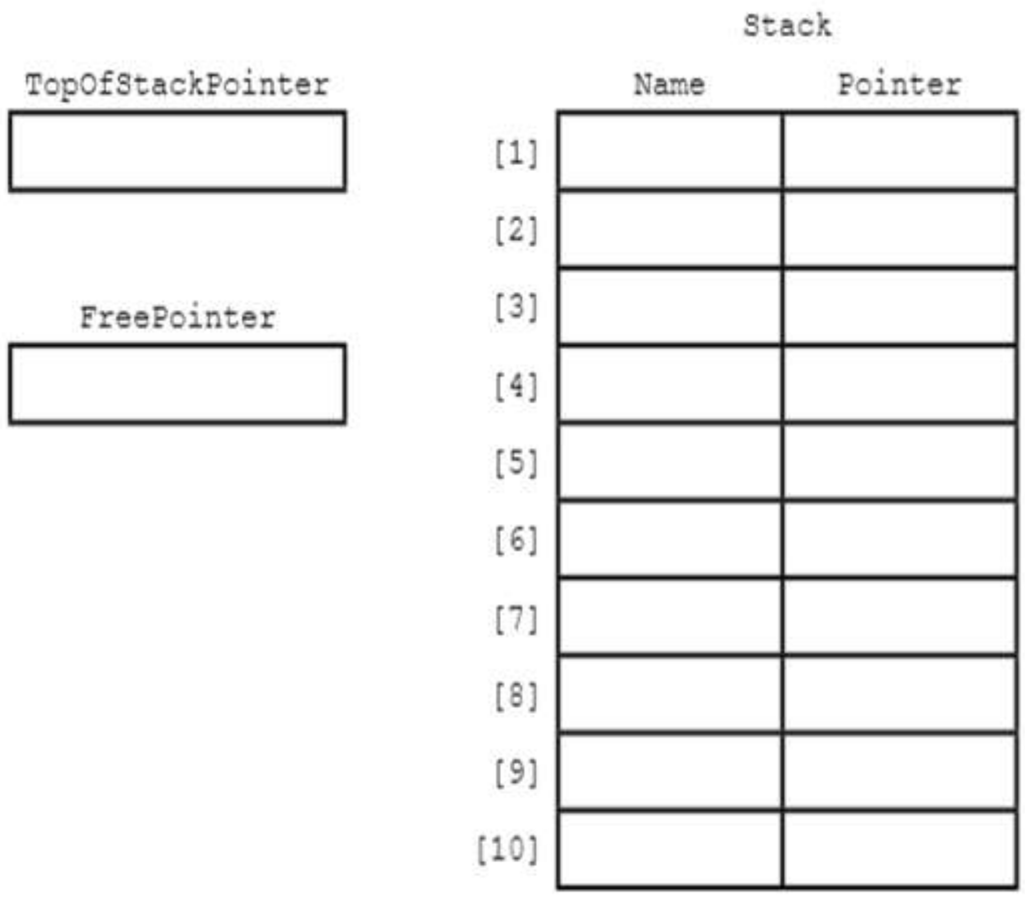
```
TYPE Node
   DECLARE Name: STRING
   DECLARE Pointer: INTEGER
END TYPE
```

The Statement

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

Reserves space for 10 nodes in array `Stack`

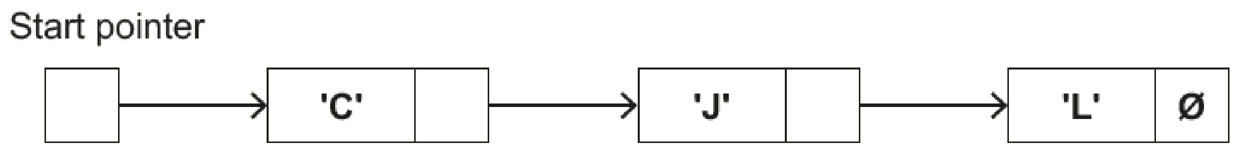
(i) The `CreateStack` operation links all the nodes and initializes `TopOfStackPointer` and `FreePointer`.
Complete the diagram to show the value of all Pointers after `CreateStack` has been executed.



[4]

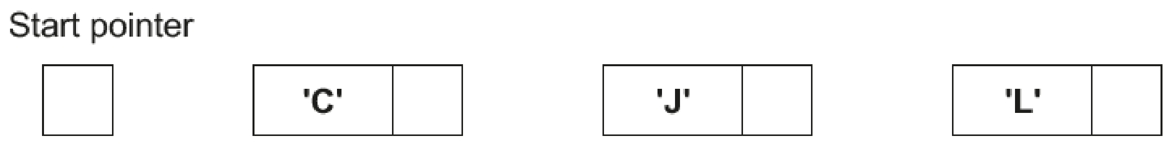
9618/21/O/N/22

8 (a) The following diagram shows an Abstract Data Type (ADT) representation of an ordered linked list. The data item stored in each node is a single character. The data will be accessed in alphabetical order.
The symbol Ø represents a null pointer.



(i) Nodes with data 'A' and 'K' are added to the linked list. Nodes with data 'J' and 'L' are deleted.
After the changes, the data items still need to be accessed in alphabetical order.

Complete the diagram to show the new state of the linked list.



[4]

(ii) The original data could have been stored in a 1D array in which each element stores a character.
For example:

'C'	'J'	'L'		

Explain the advantages of making the changes described in **part (a)(i)** when the data is stored in the linked list instead of an array.

....

....

....

.... [2]

(iii) Explain the disadvantages of making the changes described in **part (a)(i)** when the data is stored in the linked list instead of an array.

....

....

....

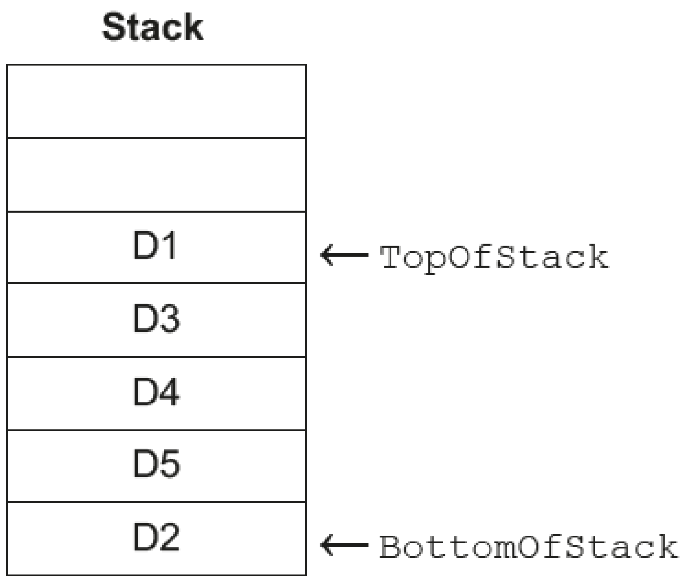
....

.... [2]

9618/21/O/N/23

9 The diagram represents an Abstract Data Type (ADT).
The operation of this stack may be summarised as follows:

- The `TopOfStack` pointer points to the last item added to the stack.
- The `BottomOfStack` pointer points to the first item on the stack.



(a) The stack is implemented using two variables and a 1D array of 8 elements as shown.

The variables are used to reference individual elements of the array, in such a way that:

- the array is filled from the lowest indexed element towards the highest
- all the elements of the array are available for the stack.

Complete the diagram to represent the state of the stack as shown above.

Array element	Data
8	
7	
6	
5	
4	
3	
2	
1	

Variable	
`TopOfStack`	
`BottomOfStack`	

[3]

9(b) A function **`Push()`** will add a value onto the stack by manipulating the array and variables in **part (a)**.
Before adding a value onto the stack, the algorithm will check that space is available.

If the value is added to the stack, the function will return TRUE, otherwise it will return FALSE.

The algorithm is expressed in five steps.

Complete the steps.

1. If .... then return FALSE

2. Otherwise .... `TopOfStack`

3. Use `TopOfStack` as an .... to the array.

4. Set the element at this .... to the .... being added.

5. Return ....

[5]

9618/22/M/J/21

ANSWERS

Q1

Question Answer Marks

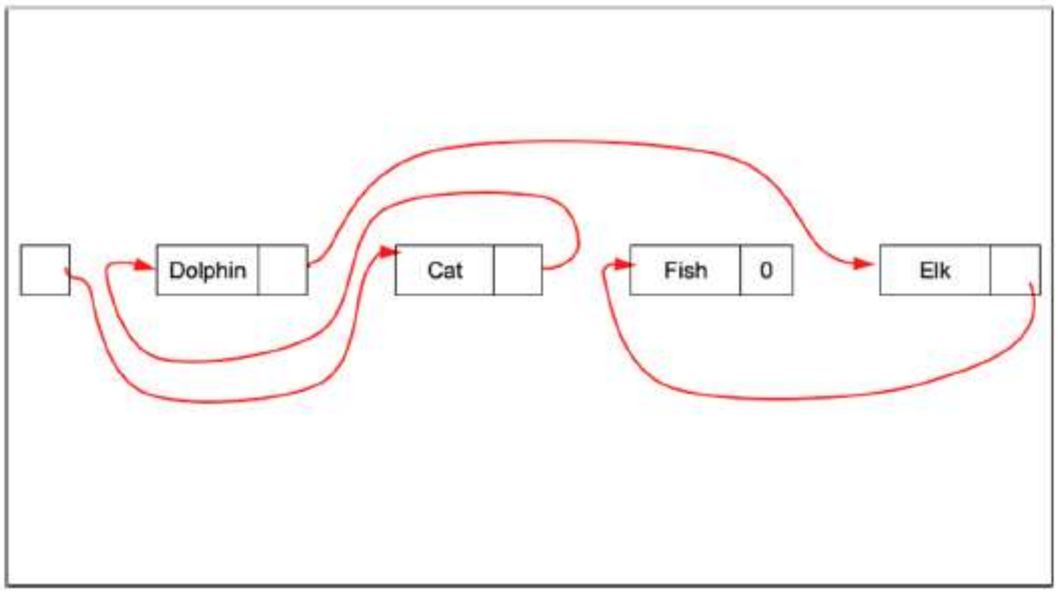
(a)	Linked list	**[1]**
(b)	Start pointer	**[1]**

(c) One mark for each:

Name: Null pointer

Meaning: There are no further nodes in the list **[2]**

(d)



[2]

One mark for:

- Start Pointer pointing to 'Cat' node
- Remaining arrows: Cat ← Dolphin ← Elk ← Fish

9618/21/M/J/21

2(a) One mark per point:

1 Check for a free node
2 Search for correct insertion point
3 Assign data value B to first node in free list / node pointed to by startpointer of free list
4 Pointer from A will be changed to point to node containing B (instead of C)
5 Pointer from B will be changed to point to node containing C
6 Start pointer in free list moved to point to next free node
Note: max 4 marks

[4]

2(b) One mark per point:
 An array (1D) to store the data **and** a second array (1D) to store the pointers
 An (integer) variable to hold the start pointer **and** an (integer) variable to store the next free pointer

ALTERNATIVE:
 Define a record type comprising a data element and a pointer **and** declare an array (1D) of this type
 An integer variable to hold the start pointer **and** an integer variable to store the next free pointer

[2]

9618/22/M/J/23
3(a)(i) One mark per point:
1 Check that the queue is not full
2 EoQ pointer will move to point to location 9
3 Data item Orange will be stored in location referenced by EoQ pointer
4 EoQ pointer will move to point to location 0
5 Data item Yellow will be stored in location referenced by EoQ pointer

[4]

Note: max 4 marks

3(a)(ii) **7** [1]

3(b) One mark per bullet:
1 Open file in READ mode
2 Loop to EOF()// read / process all the lines in file
3 Loop will end when return value from AddToQueue() is FALSE / queue is full
4 Read a line from the file in a loop
5 Pass string to AddToQueue()// AddToQueue()is executed with line as parameter

[5]

9618/21/M/J/24
4(a) One mark per point:
1 The PS contains a null pointer
2 The PF points to the first element on the free list
3 All the nodes are on the free list

[3]

4(b) Max 2 marks for 'Variables':
The two variables will be of type **Integer**
The two variables will be used as **pointers / indexes** to the arrays.
The values stored in the two variables will indicate **the first element in each list**
The first 1D array will be of type **String**
The first 1D array will be used to **store the values // data items // User IDs**
The second 1D array will be of type **Integer**
The second 1D array will be used to **store the pointers // point to next item**
Mark as follows:
One mark for **each** of the first three rows
One mark for **both** Array 1 rows
One mark for **both** Array 2 rows **[5]**

9618/21/O/N/21

5(a)(i) One mark per point:

- `EoQ` pointer will move to point to location 4 // incremented `EoQ` (by 1)
- Data value "Octopus" will be stored in location pointed to be `EoQ` / location 4

[2]

5(a)(ii) One mark for each bullet

- Value "Frog" // value pointed to by `FoQ` / location 0 is assigned to variable `AnimalName`
- `FoQ` pointer will move to point to location 1 / point to "Cat" // incremented `FoQ` (by 1)

0	Frog	← **Front of queue pointer**
1	Cat	
2	Fish	
3	Elk	← **End of queue pointer**

5(a)(iii) There is only one data item in the queue **[1]**

5(b)(i) One mark for data values plus one mark for pointers

0	Frog	
1	Cat	
2	Fish	← **Front of queue pointer**
3	Elk	
4	**Wasp**	
5	**Bee**	
6	**Mouse**	← **End of queue pointer**
7		

One mark for each pointer
One mark for three new data values **[3]**

5(b)(ii)

0	**Shark**	← **End of queue pointer**
1	(Cat)	
2	(Fish)	
3	(Elk)	
4	Wasp	← **Front of queue pointer**
5	Bee	
6	Mouse	
7	**Dolphin**	

[2]

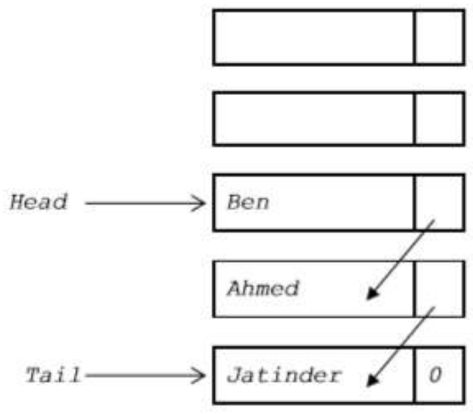
One mark for BOTH pointers
One mark for all data values as shown **[2]**

5(c) One mark per point:
1 If incremented `EoQ` = `FoQ` then error condition: queue is full
2 Increment the `EoQ`
3 Manage wrap-around

[3]

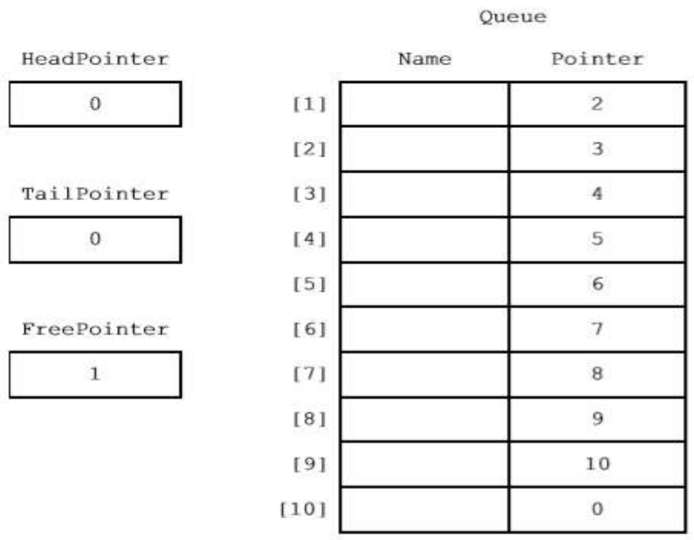
9608/41/M/J/1
Answers:
Q6/-

(a)



1 mark for Head and Tail pointers
1 mark for 3 correct items – linked as shown
1 mark for correct order with null pointer in last nod **[3]**

(b) (i)



HeadPointer: 0

TailPointer: 0

FreePointer: 1

Queue

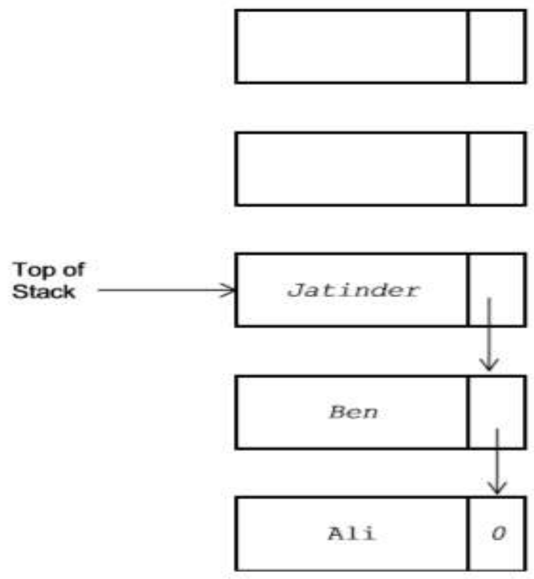
	Name	Pointer
[1]		2
[2]		3
[3]		4
[4]		5
[5]		6
[6]		7
[7]		8
[8]		9
[9]		10
[10]		0

Mark as follows:

`HeadPointer` *=0* **&** `TailPointer` *= 0*
`FreePointer` ***assigned a value***
`Pointers[1]` **to** `[9]` ***links the nodes together***
`Pointer[10]` *=* `'Null'` **[4]**

9608/43/M/J/15
Q7/-

(a)



1 mark for Top of Stack pointer
1 mark for 3 correct items
1 mark for correct order with null pointer in last node **[3]**

(b) (i)

Stack

TopOfStackPointer

0

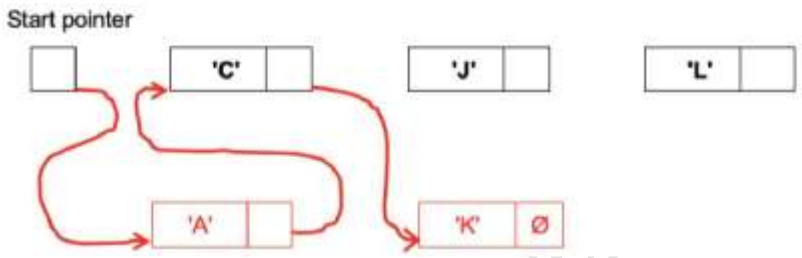
FreePointer

1

	Name	Pointer
[1]		2
[2]		3
[3]		4
[4]		5
[5]		6
[6]		7
[7]		8
[8]		9
[9]		10
[10]		0

Mark as follows:
TopOfStackPointer
FreePointer
Pointers[1] to [9]
Pointer[10] **[4]**

Q8 (a)
One mark for each:
1 Data A and K stored in new / existing nodes
2 Start pointer points to Node A
3 Node A points to Node C **and** Node C points to Node K
4 Node K contains Null Pointer



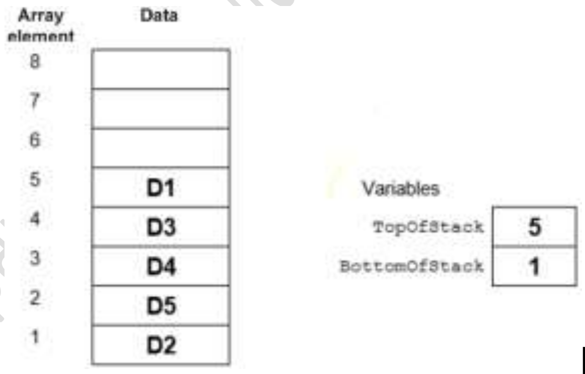
[4]

8(a)(ii)
One mark per point:
1 Pointers determine the ordering of data // only the pointers need to be changed when data changed
2 Easier to add / delete data (to maintain correct sequence) in a linked list // description of moving data to maintain correct sequence when array used **[2]**

8(a)(iii)
One mark per point:
1 Need to store pointers as well as data
2 More complex (to setup / implement) **[2]**

9618/21/O/N/23

Q9 (ANSWER)



Array element	Data
8	
7	
6	
5	D1
4	D3
3	D4
2	D5
1	D2

Variables

TopOfStack	5
BottomOfStack	1

[3]

MP1 all values in the order and location shown
MP2 `TopOfStack` value is index of element containing D1
MP3 `BottomOfStack` value is index of element containing D2

9(b)
MP1 If **`TopOfStack` = 8 // (stack) full** then return `FALSE`
MP2 Otherwise, **increment** `TopOfStack`
MP3 Use `TopOfStack` as an **index** to the Array
MP4 Set the element at this **index / location / position** to the **value / data / item** being added
MP5 Return **`TRUE`** [5]