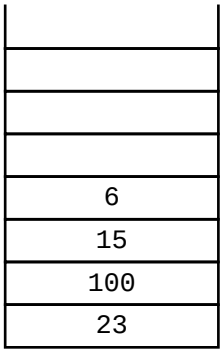
1(a). A computer program stores data input on a stack named `dataItems`. The stack has two subprograms to add and remove data items from the stack. The stack is implemented as a 1D array, `dataArray`.

Sub-program	Description
`push()`	The parameter is added to the top of the stack
`pop()`	The parameter is added to the top of the stack

The current contents of `dataItems` are shown:



The main program asks a user to push or pop an item from the stack. If the user chooses ‘push’, the data item is added to the stack. If the user chooses “pop”, the next item is removed from the stack, multiplied by 3 and output.

The main program is shown:

```
01  userAnswer = input("Would you like to push or pop an item?")
02  if userAnswer == "push" then
03    push(input("Enter data item"))
04  else
05    print(pop() * 3)
06  endif
```

(i) Before the sub-programs, `push()` and `pop()`, can add or remove items from the stack, a selection statement is used to decide if each action is possible.

Describe the decision that needs to be made in each sub-program and how this impacts the next process.

`push()` ----------

`pop()` ---

---

[4]

(ii) The algorithm does not work when the user enters "`PUSH`" or "`Push`". The algorithm needs to be changed in order to accept these inputs.

Identify the line number to be changed and state the change that should be made.

Line number ---

Change ---

---

[2]

(b). The stack is implemented as a 1D array, `dataArray`.

Describe how a 1D array can be set up and used to push and pop items as a stack.

[3]

2. A user enters whole numbers into a computer program. Each number entered is placed onto a stack. The stack is created using an array with a maximum of 20 elements.

Part of the array, `numStack`, is shown when one number has been input.

top	1

index	stackItem
9	
8	
7	
6	
5	
4	
3	
2	
1	
0	20

The pointer, `top`, points to the next free space in the stack.

A function, `addItem`, takes a number as a parameter and adds the number to the stack. The function returns `true` if this was successful, and `false` if the stack is already full.

(i) Give **one** reason why a function is used instead of a procedure in this scenario.

---

[1]

(ii) The parameter can be passed by value or by reference.

Describe what is meant by passing a parameter by value and by reference.

By value ---

By reference ----------

---

[4]

(iii) The function `addItem` is written but is incomplete.

Complete the function, `addItem`.

```
function addItem (number)
      if top == ..................................... then
           return false
      else
           numStack [.....................] = .....................
           top = ............................. + 1
           ..................................................
      endif
endfunction
```

[5]

(iv) The procedure, `calculate`, takes each item in turn from the stack. It alternately adds then subtracts the numbers until there are none left.

For example, if `numStack` contains:

2
6
5
12

It would perform 2 + 6 – 5 + 12 and output 15.

```
01   procedure calculate()
02        total = 0
03        add = true
04        if top == 0 then
05             print("Stack empty")
06        else
07             total = numStack[top - 1]
08             top = top ? 1
09             while top != 0
10                  if add == true then
11                       total = total + numStack[top - 1]
12                       add = false
13                  else
14                       total = total ? numStack[top - 1]
15                       add = true
16                  endif
17                  top = top - 1
18             endwhile
19             print(total)
20        endif
21   endprocedure
```

Complete the trace table for the procedure `calculate`. The current array and pointer values when the procedure is called are on the first line of the trace table.

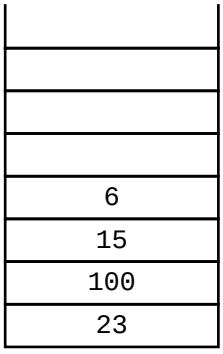
top	numstak						total	add	output
	0	1	2	3	4	5			
5	20	2	6	12	8				

[6]

3. A computer program stores data input on a stack named `dataItems`. The stack has two subprograms to add and remove data items from the stack. The stack is implemented as a 1D array, `dataArray`.

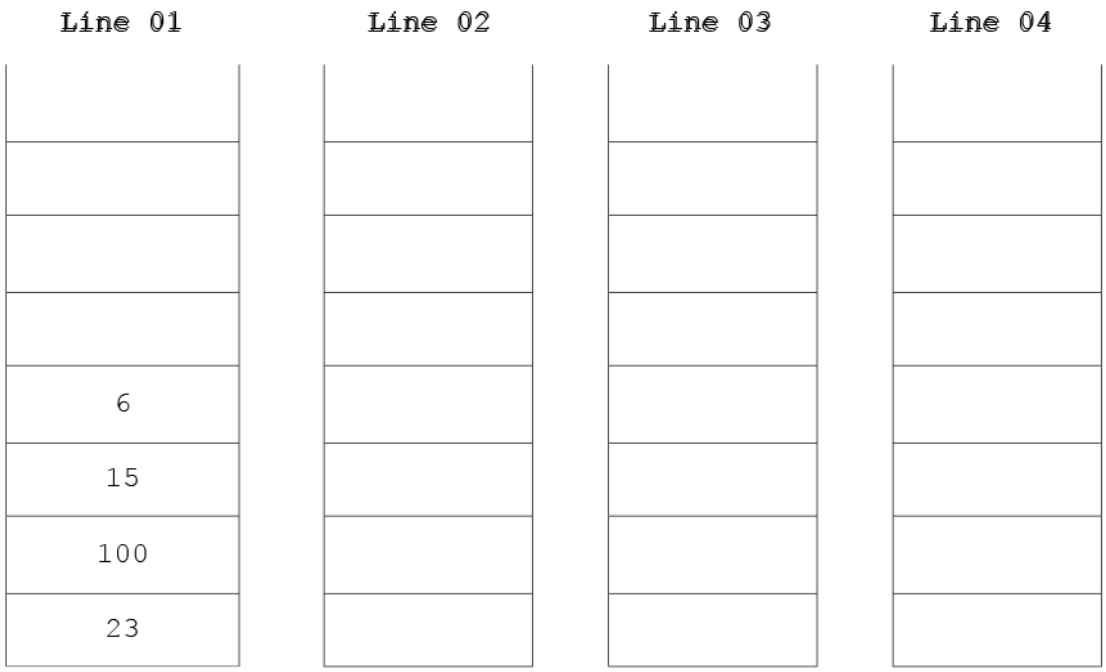
Sub-program	Description
`push()`	The parameter is added to the top of the stack
`pop()`	The parameter is added to the top of the stack

The current contents of `dataItems` are shown:



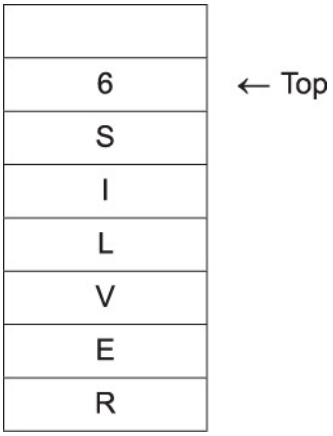
Show the contents of the stack `dataItems` after each line of the following lines of code are run

```
01 push(13)
02 pop()
03 push(10)
04 push(20)
```



[4]

4. A stack, in shared memory, is being used to pass a single variable length ASCII string between two sub-systems. The string is placed in the stack one character at a time in reverse order with the last byte holding the number of characters pushed i.e. the text “SILVER” would be held in the stack as:



Use pseudocode to write a procedure that will take a text string passed to it and push it to the stack in the format defined above. You may assume any given input will fit in the stack.

[6]

5(a). A function, `push`, can be used to add a character to a stack. For example:

```
theStack.push("H")
```

places the character `H` onto the stack, `theStack`.

A procedure, `pushToStack`, takes a string as a parameter and pushes each character of the message onto the stack, `messageStack`.

Complete the procedure below.

Add comments to explain how your code works.

```
procedure pushToStack(message)

----------

----------

----------

----------

----------

----------

----------

----------

endprocedure
```

[5]

(b). An encryption routine reads a line of text from a file, reverses the order of the characters in the string and subtracts 10 from the ASCII value of each letter, then saves the new string into the same file.

The program is split into sub-procedures. Three sub-procedures are described as follows:

- Read string from file
- Push each character of the string onto a stack
- Read and encrypt each character message

Describe the steps that the program would have to take in order to encrypt the characters stored in the stack, and save them in a single variable.

[5]

6. Kamran is writing a program to manipulate the data for a set of items.

For each item, the program needs to store:

- Item name (e.g. Box)
- Cost (e.g. 22.58)
- Date of arrival (e.g. 1/5/2018)
- Transferred (e.g. true)

The items are added to a queue for processing.

The queue is defined as a class, `itemQueue`.

itemQueue

`theItems[10] : Items`
`head : Integer`
`tail : Integer`
`numItems : Integer`

`constructor`
`enqueuer()`
`dequeuer()`
`setnumItems()`
`getnumItems()`

The `head` attribute points to the first element in the queue. The `tail` attribute points to the next available space in the queue. The `numItems` attribute states how many items are currently in the queue.

The array, `theItems`, stores the items in the queue. When the tail of the queue exceeds the last element in the array, it adds a new item to the first element if it is vacant.

For example, in the following queue, the next item to be added would be placed at index 0.

Index	0	1	2	3	4	5	6	7	8	9
Element				Data	Data	Data	Data	Data	Data	Data

(i) Define the term ‘queue’.

---

[2]

(ii) The attributes in `itemQueue` are all declared as private.

Explain how a private attribute improves the integrity of the data.

[2]

(iii) The constructor method creates a new instance of `itemQueue` and sets the `head, tail` and `numItems` attributes to 0.

Write an algorithm, using pseudocode or program code, for the constructor including the initialisation for all attributes.

[2]

(iv) The enqueue method:

- takes as a parameter the item to insert in the queue
- checks if the queue is full
- reports an error and returns false if the queue is full
- does the following if the queue is not full:
  - adds the item to the array at the tail position and adjusts the pointer(s)
  - returns true

The attribute `numItems` stores the number of items currently in the queue.

Write an algorithm, using pseudocode or program code, for the enqueue method.

[6]

(v) Write a programming statement to declare an instance of `itemQueue` called `myItems`.

[1]

(vi) Write a procedure, `insertItems()`, to ask the user to input the data for an item. The item is then added to the queue `myItems`. The user is continually asked to input data items until the queue is full.

[5]

(vii) When the main program ends, the items and the queue no longer exist.

Describe how Kamran could amend the program to make sure the items and queue still exist and are used the next time the program is run.

[2]

7(a). A program stores a queue of mathematical questions to be asked to a user. The questions are asked in the order they are added. Once a question has been asked it cannot be asked again. New questions are continually added to the end of the queue.

The program will use a non-circular queue, `questions`, (implemented using an array) to store the questions.
The pointer, `head`, stores the index of the first element in the queue.
The pointer, `tail`, stores the index of the last element in the queue.

Describe why a queue is a suitable structure for this program.

---

[3]

(b). Complete the following algorithm, to remove, and output, the first element in the queue.

```
procedure remove()
```

```
endprocedure
```

[4]

END OF QUESTION PAPER

Question			Answer/Indicative content	Marks	Guidance
1	a	i	1 mark per bullet, max 2 for insert, max 2 for remove push • Check if the stack is full (pointer = array.length/array.length+1) • If it is not ¡V insert the item • If it is ¡V return/error that the stack is full pop • Check if the stack is empty (pointer = 0/1) • If it is ¡V return/error that the stack is empty • If it is not ¡V return the item	4 AO1.2 (2) AO2.2 (2)	Examiner's Comments A significant number of candidates did not describe a conditional decision clearly and lost marks when merely describing push/pop operations. Where the first part of the question was answered well some candidates then failed to see the second part of the question and did not describe an impact of the condition. Candidates need to be reminded to read and analyse the wording of the whole question to access all marking points.
		ii	1 mark per line, 1 for change • line 02 • Include an OR with variations (e.g. `userAnswer = "PUSH" OR userAnswer = "Push"` etc.)/Convert input to uppercase/lowercase and just compare to equivalent	2 AO2.2 (2)	Examiner's Comments Most candidates answered well, but few gave answers that demonstrated an ability to combine separate logical statements with the OR operator. A significant number of candidates used a .lower() method being mostly familiar with Python syntax and methods.
	b		1 mark per bullet to max 3 • Array size defined • A stack pointer is used to point to the top of the stack • When an item is pushed the stack pointer is incremented • When an item is popped the stack pointer is decremented	3 AO1.2 (1) AO2.1 (1) AO2.2 (1)	Examiner's Comments Those candidates with experience of languages other than Python appreciated that a 1D array is a static structure that needs to be declared with a given size, and that a stack pointer variable would be required. It is of concern that significant numbers of candidates have only had experience of lists and their associated methods in Python. A number of candidates also confused their descriptions with those for a queue rather than a stack.
			Total	9	

<table>
<tr><th colspan="3">Question</th><th>Answer/Indicative content</th><th>Marks</th><th>Guidance</th></tr>
<tr><td>2</td><td></td><td>i</td><td>A procedure does not return a value / a function has to return a value</td><td>1
AO1.2
(1)</td><td><u>Examiner's Comments</u>

Many candidates answered well and understood the difference between functions and procedures, knowing that functions have to return a value.</td></tr>
<tr><td></td><td></td><td>ii</td><td>1 mark per bullet, max 2 for by value, max 2 for by reference by value:

• A local copy of the data is used
• Data is discarded when the subprogram exits
• Does not override/change the original data

by reference:

• Memory location of data is sent
• Changes are made to the original data
• Changes remain after the subprogram exits</td><td>4
AO1.2
(4)</td><td><u>Examiner's Comments</u>

Parameter passing by value and by reference continue to prove problematic to candidates, with many having a poor grasp of the concept. Those candidates who have used a variety of programming languages including those that allow for parameter passing by reference often had the practical experience to draw upon.</td></tr>
<tr><td></td><td></td><td>iii</td><td>1 mark for each completed space to max 5

<pre>function addItem (number)
    if top = 20 then
        return false
    else
        numStack[top] = number
        top = top + 1
        return true
    endif
endfunction</pre></td><td>5
AO2.2
(2)
AO3.2
(3)</td><td>Accept
<code>numStack.length()</code>
instead of <code>20</code>

<u>Examiner's Comments</u>

Candidates are best prepared for this paper by having had practical experience of implementing data structures such as stacks and queues. A number of candidates did not read the whole stem of the question and assumed that the total number of elements was 10 instead of 20. The concept that the stack pointer points to the next space to be used in the stack was poorly understood. Those candidates with practical experience and the ability to read and interpret code did answer well.</td></tr>
</table>

<table>
<tr><th colspan="3">Question</th><th>Answer/Indicative content</th><th>Marks</th><th>Guidance</th></tr>
<tr><td></td><td></td><td>iv</td><td>1 mark for each bullet to max 6

• Initialising <code>total</code> to 0 and <code>add</code> to true
• <code>top</code> = 4 and <code>total</code> = 8
• <code>total</code> = 20 and <code>add</code> = false
• <code>top</code> = 3, <code>total</code> = 14, <code>add</code> = true
• <code>top</code> 2, 1. <code>Total</code> 16, -4, <code>add</code> false, true
• Output = -4

<table>
<tr><th></th><th colspan="6">numStack</th><th></th><th></th><th></th></tr>
<tr><th>top</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>total</th><th>add</th><th>Output</th></tr>
<tr><td>5</td><td>20</td><td>2</td><td>6</td><td>12</td><td>8</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>true</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td><td>8</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>20</td><td>false</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td><td>14</td><td>true</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td>16</td><td>false</td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td>-4</td><td>true</td><td></td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>-4</td></tr>
</table>
</td><td>6
AO1.2
(3)
AO2.2
(3)</td><td><u>Examiner's Comments</u>

Tracing code execution is an area that continues to prove challenging to candidates. Candidates need to have experience of completing dry-runs of code and setting out a trace table in a logical manner. Where candidates did perform well the initialisation of the variables total and add before the main body of the loop was entered was often omitted.</td></tr>
<tr><td></td><td></td><td></td><td>Total</td><td>16</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td>
<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>20</td></tr>
<tr><td>13</td><td></td><td>10</td><td>10</td></tr>
<tr><td>6</td><td>6</td><td>6</td><td>6</td></tr>
<tr><td>15</td><td>15</td><td>15</td><td>15</td></tr>
<tr><td>100</td><td>100</td><td>100</td><td>100</td></tr>
<tr><td>23</td><td>23</td><td>23</td><td>23</td></tr>
</table>
</td><td>4
AO1.2
(2)
AO2.2
(2)</td><td><u>Examiner's Comments</u>

The vast majority of candidates had no trouble executing a sequence of push/pop instructions successfully.</td></tr>
<tr><td></td><td></td><td></td><td>Total</td><td>4</td><td></td></tr>
</table>

Question			Answer/Indicative content	Marks	Guidance

4			• String length calculated (1)
			• Correct number of characters from passed string taken ... (1)
			• ... in reverse order (1)
			• Characters placed in stack in correct order (1)
			• String length placed in stack at correct point (1)
			• Meaningful variable names used (1) (AO2.1)
			
			Example program
			
			`procedure passToStack(passString)`
			`  stringLen = passString.Length()`
			`  if stringLen == 0 then`
			`  stack[0]=0`
			`  else`
			`  stackPtr = 0`
			`  stringPtr = stringLen - 1`
			`  for i = 1 TO stringLen`
			`  stack[stackPtr] = passString[stringPtr]`
			`  stackPtr = stackPtr + 1`
			`  stringPtr = stringPtr -1`
			`  next i`
			`  stack[stackPtr] = stringLen`
			`  endif`
			`endprocedure`
				6	Allow `StackPtr` to be used instead of `i` in loop, as we would not expect them to know that some compilers do not always increment “loop counter” when they exit loops (i.e. loop counter on exit is undefined)
					
					Accept candidates using built-in stack methods e.g.
					`stack.push(word.substring(i,1))`
					
					Do not penalise for syntax errors if the logic can clearly be followed.
					
					Max 6 mark
					
					Examiner's Comments
					
					Many of the same comments regarding pseudocode as in 4b once again applied in 4d. An encouraging number of able candidates produced quite elegant solutions.

			Total	**6**	

Question			Answer/Indicative content	Marks	Guidance

5	a		1 mark per bullet to max 5
			• Use of appropriate loop
			• Correct end condition (length of `message`)
			• Correct use of `.push` with `messageStack`
			• Accessing `substring` (or equivalent) correctly
			• Appropriate comment(s)
			`procedure pushToStack(message)`
			`  for x = 0 to message.length() //loop through each`
			`  //letter`
			`  messageStack.push(message.substring(x,1)) //take`
			`  //each character and push onto stack`
			`  next x //move to next letter`
			`endprocedure`
				5
				AO2.1
				(2)
				AO3.2
				(3)
					Examiner's Comment:
					Many candidates scored well, but fewer scored full marks. The use of pseudocode rather than Python like syntax would have prevented errors with loop lengths.

	b		1 mark per bullet to max 5 • Pop element from stack • Convert to ASCII value • Subtract 10 from ASCII value • Convert back to character • Append / concatenate with variable	5 AO1.2 (2) AO2.2 (3)	Accept pseudocode equivalent. **Examiner's Comment:** Most candidates scored some of the marks, but fewer appreciated that the characters needed to be popped from the stack initially, and that the converted characters would have to be concatenated into a string at the end of the process.
			Total	10	

Question			Answer/Indicative content	Marks	Guidance
6		i	1 mark per bullet to max 2 • A data structure • FIFO (first in first out)	2 AO1.1 (2)	Examiner's Comments Most candidates had learned the definition for *queue* and answered successfully.
		ii	1 mark per bullet to max 2 • Properties (are encapsulated) and can only be accessed through their methods • Enforce validation through the method // inappropriate data can be caught before entered • Cannot be changed/accessed accidentally	2 AO1.2 (2)	Examiner's Comments Many candidates termed their answers by restating terms in the question. Whilst the term encapsulation was often cited it was less often explained well – few candidates knew that getter() and setter() methods could be used to access the private attributes of a class. Some candidates stated that private attributes could not be changed which demonstrated a clear lack of understanding of the paradigm.

		iii	1 mark per bullet to max
			
			• Constructor method/`new`
			• Setting `head` and `tail` to 0 within constructor method
			
			e.g.
			`public procedure new()`
			`  head = 0`
			`  tail = 0`
			`  numItems = 0`
			`endprocedure`
				2
				AO2.2
				(1)
				AO3.2
				(1)
					Examiner's Comments
					
					Few candidates appeared to have had practical experience of programming in an Object Oriented Programming (OOP) languages. Those that did, answered well. It is advisable to ensure that candidates are prepared for section B of the paper by having implemented programs using the OOP methodology.

Question			Answer/Indicative content	Marks	Guidance

		iv	1 mark per bullet to max 6
			
			• Function declaration, taking item as a parameter
			• Checking if the queue is full…
			• …outputting/reporting error and returning `false`
			• Adding the item to the `tail` position
			• Correctly updating the `tail` pointer (either before or after addition)
			• Incrementing `numItems` and returning `true` if successful
			
			e.g.
			`public function enqueue(newItem : items) : boolean`
			`  if numItems = 10 then`
			`  print("Error: The queue is full")`
			`  return false`
			`  else`
			`  theItems[tail] = newItem`
			`  if tail = 9 then`
			`  tail = 0`
			`  else`
			`  tail += 1`
			`  endif`
			`  numItems += 1`
			`  return true`
			`  endif`
			`endprocedure`
				6
				AO2.2
				(3)
				AO3.1
				(1)
				AO3.2
				(2)
					Examiner's Comments
					
					Candidates struggled to apply all the information given in the stem of question 6. The stem defined the class *itemQueue* with the private attributes and methods it held. Candidates who did not use this information were unable to produce correct solutions. Only a few of the strong candidates realised that it was a circular queue that was implemented, and therefore checked the increment of the tail pointer.

		v	e.g. `myItems = (new) itemQueue()`	1 AO2.1 (1)	Allow follow through if they have parameters in 6(b)(iii) Examiner's Comments Lack of practical experience and lack of correct solutions for 6b(iii) often led to incorrect answers for this question.

Question			Answer/Indicative content	Marks	Guidance

		vi	1 mark per bullet to max 5
			
			• Procedure declaration for insertItems
			• Asking for input of data items for a new item …..
			• …using record structure correctly
			• Use of myItems.enqueue
			• Looping while the queue is not full
			
			e.g.
			`procedure insertItems()`
			
			`newItem : Items`
			`itemCount = myItems.getnumItems()`
			
			`while itemCount < 10`
			`newItem.itemName = input("Enter the item name")`
			`newItem.cost = input("Enter the item cost")`
			`newItem.dateArrival = input("Enter the date of arrival")`
			`newItem.transferred = input("Has it been transferred?")`
			`myItems.enqueue(newItem)`
			`itemCount = itemCount + 1`
			`endwhile`
			
			`myItems.setnumItems(itemCount)`
			
			`endprocedure`
				5
				AO2.2
				(2)
				AO3.1
				(1)
				AO3.2
				(2)
					Examiner's Comments
					
					Few candidates realised that you cannot use a single input statement to read in four data items, and only stronger candidates realised that they should use the enqueue method created in b(iv) to add the inputted items into the queue.

		vii	1 mark per bullet to max 2 • Store the items and queue to an external file (when the program closes) • Load the items and queue from the file when it starts	2 AO2.1 (1) AO2.2 (1)	Examiner's Comments Most candidates answered this well, realising that the data in the queue had to be written and retrieved from secondary storage.
			Total	20	

<table>
<tr><th colspan="3">Question</th><th>Answer/Indicative content</th><th>Marks</th><th>Guidance</th></tr>
<tr><td>7</td><td>a</td><td></td><td>1 mark per bullet to max 3
e.g.

• A queue is First In First Out (FIFO) [1]
• The questions are retrieved in the order they are stored [1]
• Questions can be added to the end [1]
• Dynamic structure… [1]
• …expands to take more questions [1]</td><td>3
AO1.2

(2)
AO2.1
(1)</td><td>Examiner's Comment:
Many candidates understood that a queue was a FIFO structure, but fewer could then go on to explain in context why this would then be a suitable data structure for the problem in context.</td></tr>
<tr><td></td><td>b</td><td></td><td>1 mark for pseudocode/code that meets each bullet

• Checking if queue is empty [1]
• …outputting message/reporting error [1]
• Outputting element in <code>questions</code> at index <code>head</code> [1]
• Increment <code>head</code> [1]

e.g.
<pre>procedure remove()
    if head == tail + 1 then
        print("No questions")
    else
        print(questions[head])
        head = head + 1
    endif
endprocedure</pre></td><td>4

AO3.2
(4)</td><td>Examiner's Comment:
Again, the use of pseudocode posed problems for many candidates. Those who had a wider programming experience were apparent from the well-crafted solutions. Those who gained credit generally gained two marks for understanding how the pointers were updated and how data was added/removed. Fewer scored full marks by also performing error checking.</td></tr>
<tr><td></td><td></td><td></td><td>Total</td><td>7</td><td></td></tr>
</table>