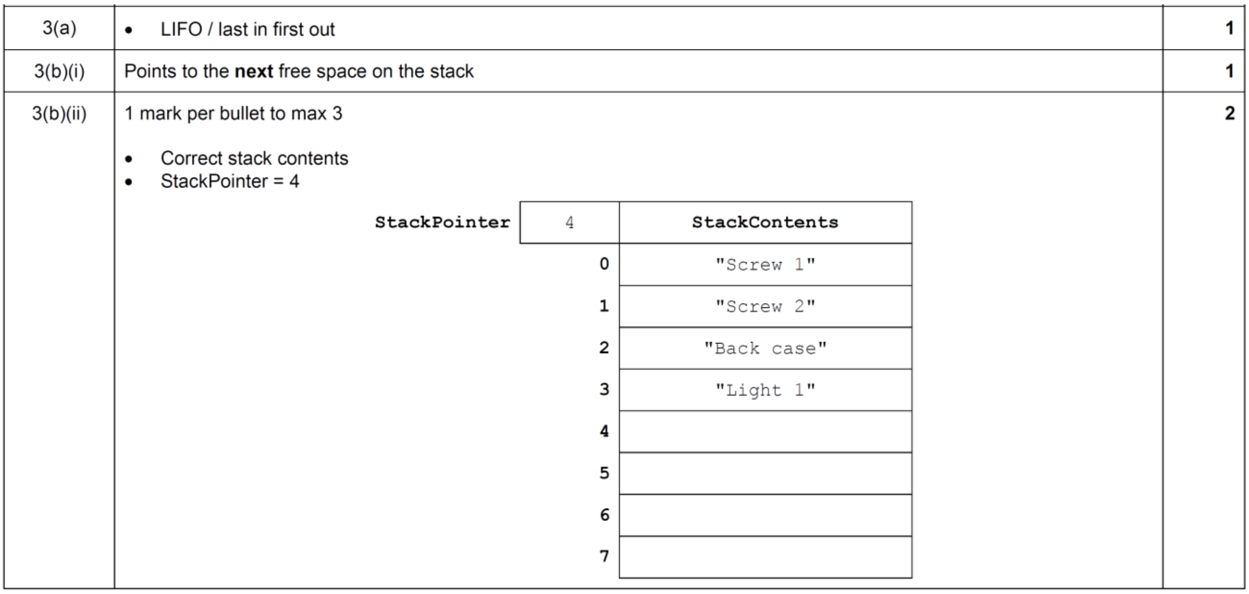
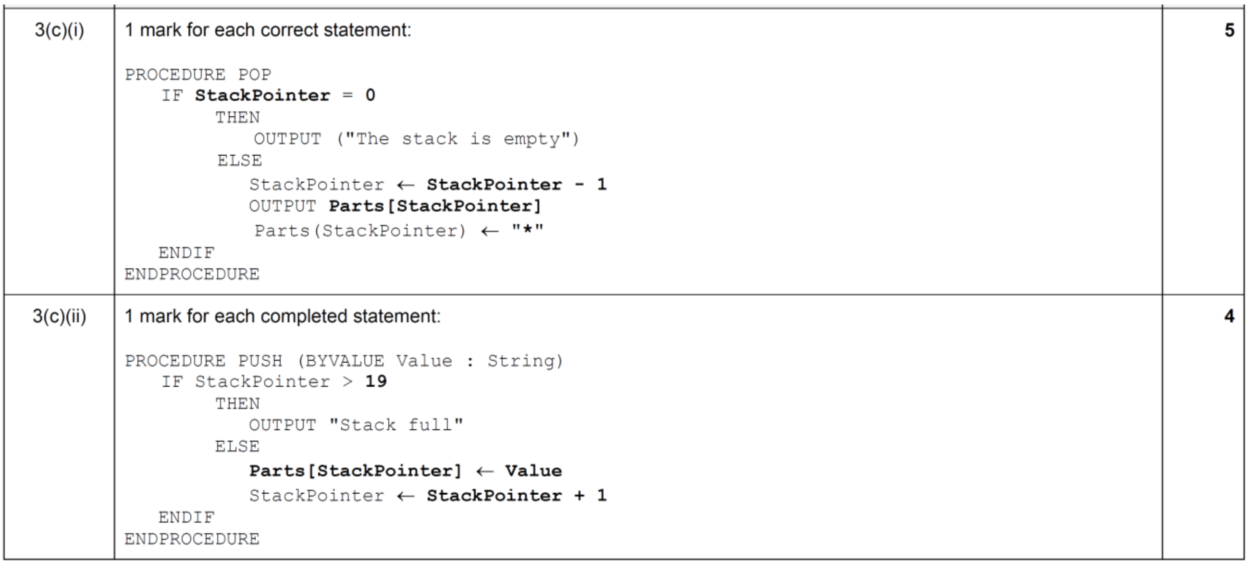
Stack and Queue (ADTs)

Question No. 1



<table>
<tr><td>3(a)</td><td>• LIFO / last in first out</td><td>1</td></tr>
<tr><td>3(b)(i)</td><td>Points to the <b>next</b> free space on the stack</td><td>1</td></tr>
<tr><td>3(b)(ii)</td><td>1 mark per bullet to max 3
• Correct stack contents
• StackPointer = 4
<table>
<tr><td><b>StackPointer</b></td><td>4</td><td><b>StackContents</b></td></tr>
<tr><td></td><td>0</td><td>"Screw 1"</td></tr>
<tr><td></td><td>1</td><td>"Screw 2"</td></tr>
<tr><td></td><td>2</td><td>"Back case"</td></tr>
<tr><td></td><td>3</td><td>"Light 1"</td></tr>
<tr><td></td><td>4</td><td></td></tr>
<tr><td></td><td>5</td><td></td></tr>
<tr><td></td><td>6</td><td></td></tr>
<tr><td></td><td>7</td><td></td></tr>
</table></td><td>2</td></tr>
</table>



<table>
<tr><td>3(c)(i)</td><td>1 mark for each correct statement:
<pre>
PROCEDURE POP
   IF StackPointer = 0
       THEN
          OUTPUT ("The stack is empty")
       ELSE
          StackPointer ← StackPointer - 1
          OUTPUT Parts[StackPointer]
          Parts(StackPointer) ← "*"
   ENDIF
ENDPROCEDURE
</pre></td><td>5</td></tr>
<tr><td>3(c)(ii)</td><td>1 mark for each completed statement:
<pre>
PROCEDURE PUSH (BYVALUE Value : String)
   IF StackPointer > 19
       THEN
          OUTPUT "Stack full"
       ELSE
          Parts[StackPointer] ← Value
          StackPointer ← StackPointer + 1
   ENDIF
ENDPROCEDURE
</pre></td><td>4</td></tr>
</table>

Question No. 2

2(a)

1 mark for correct tick

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

1

2(b)(i)

1 mark for correct stack

10
35
20

1

2(b)(ii)

1 mark for correct stack

65
50
35
20

1

2(b)(iii)

1 mark for each bullet

- Function Push …
- …taking parameter (returning Boolean)
- Checking if Top = 7 …
- …returning FALSE if full
- …returning TRUE otherwise
- if not full, increment Top
- ... add parameter to Top of ArrayStack

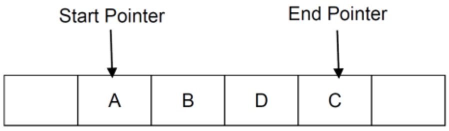
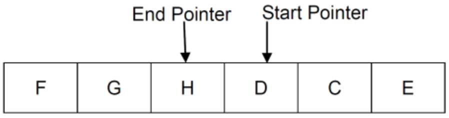
```
FUNCTION Push (BYVALUE DataItem : Integer) (RETURNS Boolean)
    IF Top = 7
        THEN
           RETURN FALSE
        ELSE
           Top ← Top + 1
           ArrayStack[Top] ← DataItem
           RETURN TRUE
    ENDIF
ENDFUNCTION
```

7

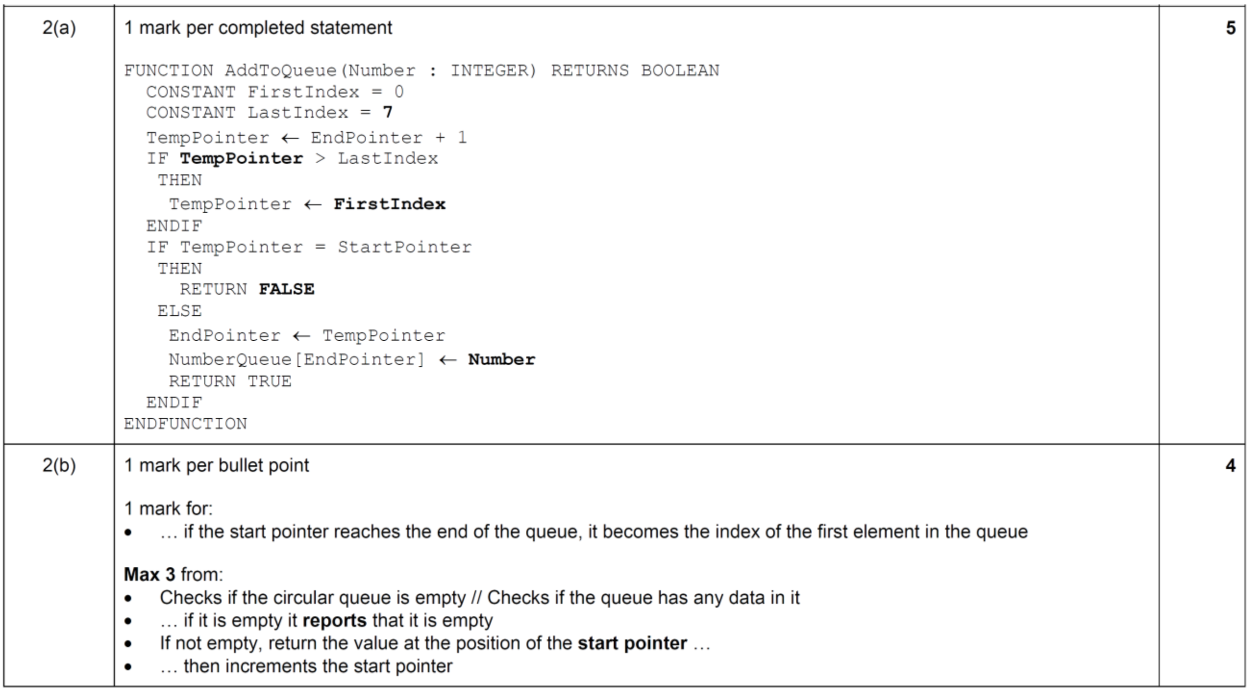
Question No. 3

1(a)(i)	1 mark for correct stack orange purple green blue red	**1**
1(a)(ii)	1 mark for correct stack black green blue red	**1**
1(b)	1 mark per bullet point to max 3 • (Linear) data structure • First in First out // FIFO // An item is added to the end of the queue **and** an item is removed from the front • All items are kept in the order they are entered • It has a head pointer and a tail pointer • Can be static or dynamic • A queue can be circular … • …when the (tail) pointer reaches the last position it returns to the first	**3**

Question No. 4

<table>
<tr><td>1(b)(i)</td><td>A, B, C and D in correct places with no alteration to start and end pointer
 </td><td>1</td></tr>
<tr><td>1(b)(ii)</td><td>1 mark per bullet point
• correct jobs in correct order…
• … correct location of start pointer
• … correction location of new end pointer
 </td><td>3</td></tr>
<tr><td>1(b)(iii)</td><td>1 mark from:
• An error <b>message</b> would be generated</td><td>1</td></tr>
<tr><td>1(b)(iv)</td><td>1 mark for each correct line<pre>
FUNCTION Remove RETURNS STRING
  DECLARE PrintJob : STRING
  IF StartPointer = EndPointer
   THEN
     RETURN "Empty"
   ELSE
     PrintJob ← Queue[StartPointer]
     IF StartPointer = 5
       THEN
        StartPointer ← 0
       ELSE
        StartPointer ← StartPointer + 1
      ENDIF
     RETURN PrintJob
   ENDIF
ENDFUNCTION
</pre></td><td>4</td></tr>
<tr><td>1(b)(v)</td><td>1 mark per bullet point
• A stack is Last In First Out (LIFO) while a queue is First In First Out (FIFO)
• The queue removes and returns the element at <b>start pointer</b> // item is removed from the <b>start/head //</b>
• A stack would remove and return the element at <b>end pointer</b> // item is removed from the <b>end</b></td><td>2</td></tr>
</table>

Question No. 5



2(a)	1 mark per completed statement
	
	`FUNCTION AddToQueue(Number : INTEGER) RETURNS BOOLEAN`
	`  CONSTANT FirstIndex = 0`
	`  CONSTANT LastIndex = `**`7`**
	`  TempPointer ← EndPointer + 1`
	`  IF `**`TempPointer`**` > LastIndex`
	`   THEN`
	`    TempPointer ← `**`FirstIndex`**
	`  ENDIF`
	`  IF TempPointer = StartPointer`
	`   THEN`
	`     RETURN `**`FALSE`**
	`   ELSE`
	`    EndPointer ← TempPointer`
	`    NumberQueue[EndPointer] ← `**`Number`**
	`    RETURN TRUE`
	`  ENDIF`
	`ENDFUNCTION`
		5

2(b)	1 mark per bullet point 1 mark for: • … if the start pointer reaches the end of the queue, it becomes the index of the first element in the queue **Max 3** from: • Checks if the circular queue is empty // Checks if the queue has any data in it • … if it is empty it **reports** that it is empty • If not empty, return the value at the position of the **start pointer** … • … then increments the start pointer	**4**

Question No. 6

1(a) — 2

1 mark for `TopPointer`
1 mark for correct data in stack

TopPointer 2

Index	Data
[7]	
[6]	
[5]	
[4]	
[3]	(8)
[2]	50
[1]	20
[0]	10

1(b) — 5

1 mark per bullet point

- Function header (and close where appropriate returning an integer)
- Checking if stack is empty …
- … and returning −1 if it is
- If there is data in stack, decrementing `TopPointer`
- (Otherwise) returning the **top** Value

Python

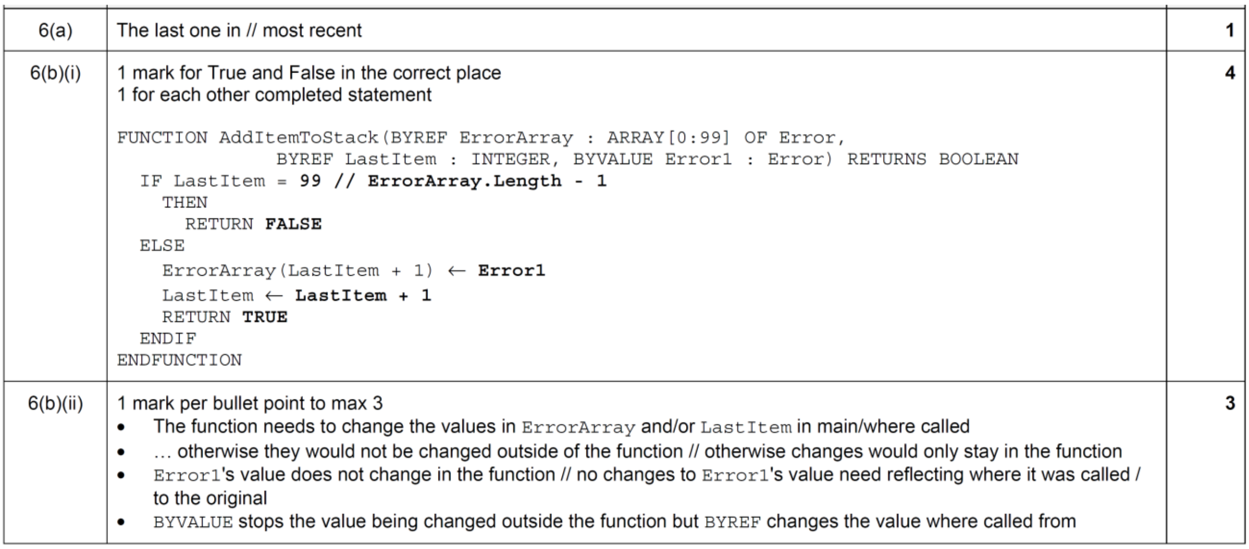
```
def Pop():
  if TopPointer < 0 :
    return -1
  else:
    Value = DataStack(TopPointer)
    TopPointer= TopPointer – 1
    return Value
```

1(c) — 2

1 mark per bullet point to max 2

- In a stack the last item in is the first out/LIFO **and** in a queue the first item in is the first out/FIFO
- Queue can be circular, but a stack is linear
- Stack only needs a pointer to the top (and can have a base pointer) **and** a queue needs a pointer to the front and the rear

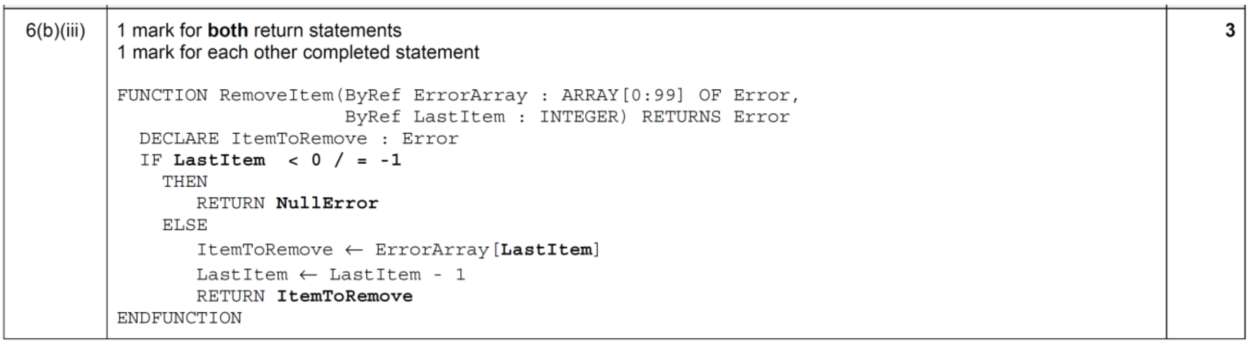
Question No. 7



6(a)	The last one in // most recent	**1**

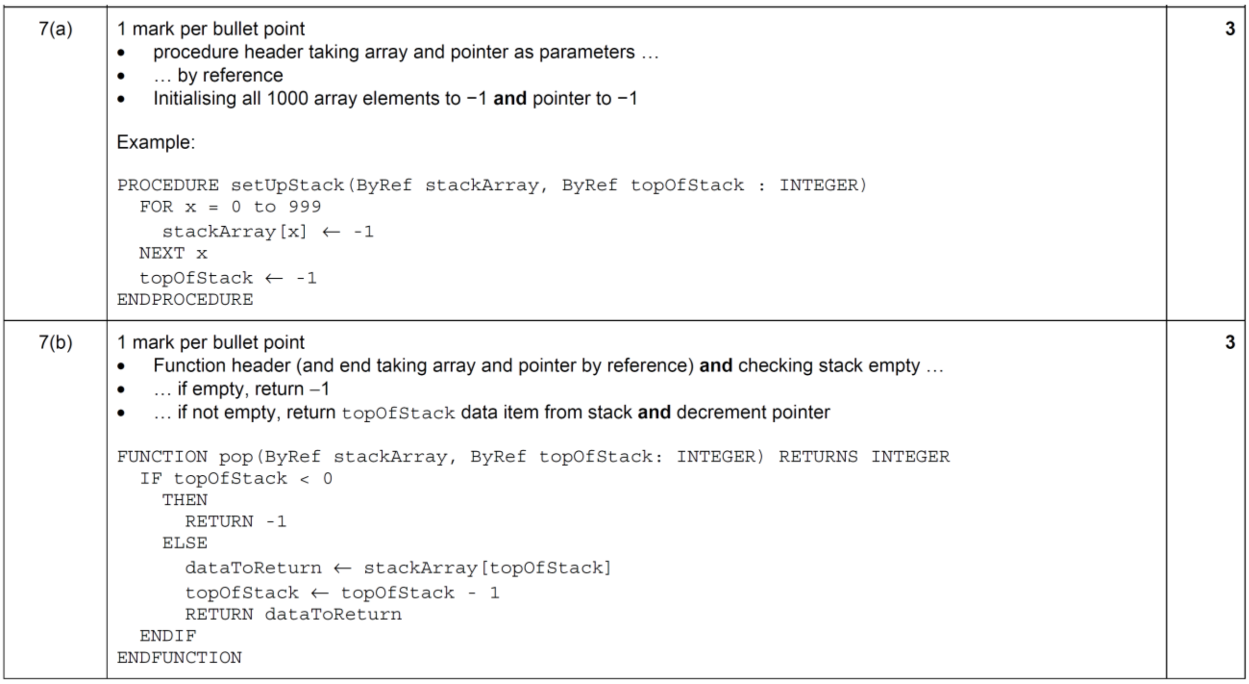
6(b)(i)	1 mark for True and False in the correct place
	1 for each other completed statement
	
	`FUNCTION AddItemToStack(BYREF ErrorArray : ARRAY[0:99] OF Error,`
	`              BYREF LastItem : INTEGER, BYVALUE Error1 : Error) RETURNS BOOLEAN`
	`  IF LastItem = `**`99 // ErrorArray.Length - 1`**
	`    THEN`
	`      RETURN `**`FALSE`**
	`  ELSE`
	`    ErrorArray(LastItem + 1) ← `**`Error1`**
	`    LastItem ← `**`LastItem + 1`**
	`    RETURN `**`TRUE`**
	`  ENDIF`
	`ENDFUNCTION`
		4

6(b)(ii)	1 mark per bullet point to max 3 • The function needs to change the values in `ErrorArray` and/or `LastItem` in main/where called • … otherwise they would not be changed outside of the function // otherwise changes would only stay in the function • `Error1`'s value does not change in the function // no changes to `Error1`'s value need reflecting where it was called / to the original • `BYVALUE` stops the value being changed outside the function but `BYREF` changes the value where called from	**3**



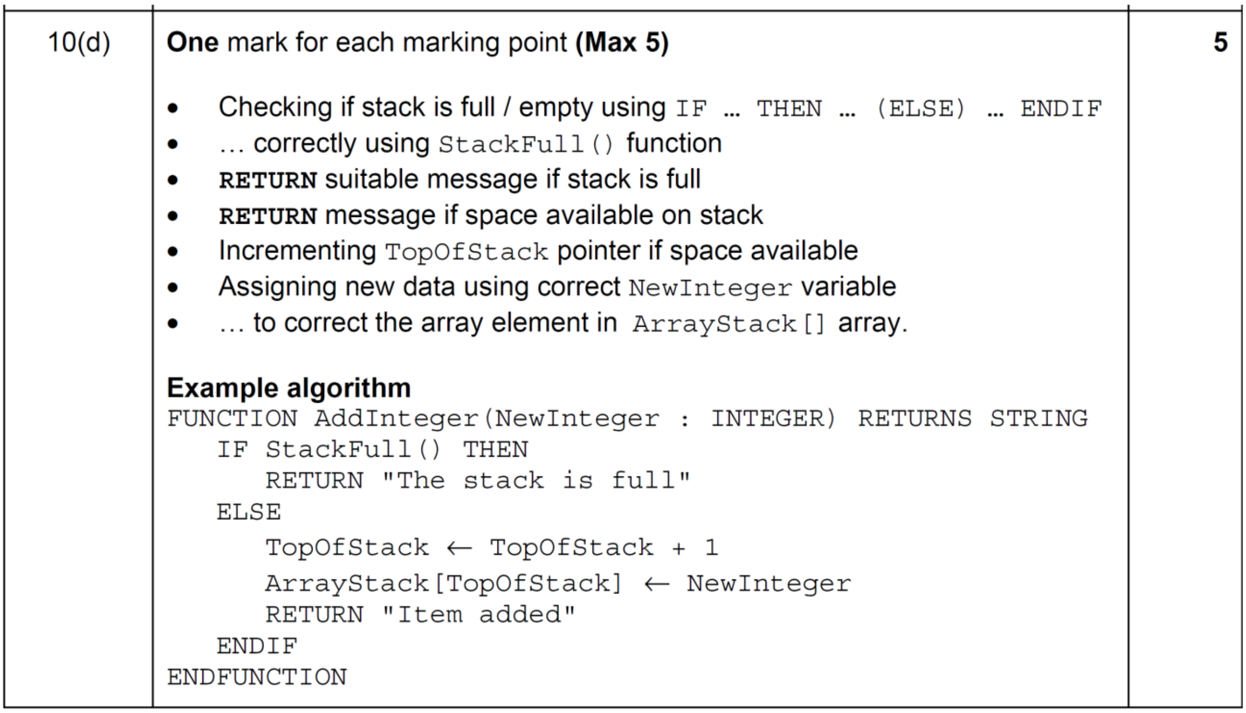
6(b)(iii)	1 mark for **both** return statements
	1 mark for each other completed statement
	
	`FUNCTION RemoveItem(ByRef ErrorArray : ARRAY[0:99] OF Error,`
	`                    ByRef LastItem : INTEGER) RETURNS Error`
	`  DECLARE ItemToRemove : Error`
	`  IF `**`LastItem  < 0 / = -1`**
	`    THEN`
	`       RETURN `**`NullError`**
	`    ELSE`
	`       ItemToRemove ← ErrorArray[`**`LastItem`**`]`
	`       LastItem ← LastItem - 1`
	`       RETURN `**`ItemToRemove`**
	`ENDFUNCTION`
		3

Question No. 8



<table>
<tr>
<td>7(a)</td>
<td>1 mark per bullet point
<ul>
<li>procedure header taking array and pointer as parameters …</li>
<li>… by reference</li>
<li>Initialising all 1000 array elements to −1 <b>and</b> pointer to −1</li>
</ul>
Example:
<pre>
PROCEDURE setUpStack(ByRef stackArray, ByRef topOfStack : INTEGER)
  FOR x = 0 to 999
    stackArray[x] ← -1
  NEXT x
  topOfStack ← -1
ENDPROCEDURE
</pre>
</td>
<td><b>3</b></td>
</tr>
<tr>
<td>7(b)</td>
<td>1 mark per bullet point
<ul>
<li>Function header (and end taking array and pointer by reference) <b>and</b> checking stack empty …</li>
<li>… if empty, return –1</li>
<li>… if not empty, return <code>topOfStack</code> data item from stack <b>and</b> decrement pointer</li>
</ul>
<pre>
FUNCTION pop(ByRef stackArray, ByRef topOfStack: INTEGER) RETURNS INTEGER
  IF topOfStack &lt; 0
    THEN
      RETURN -1
    ELSE
      dataToReturn ← stackArray[topOfStack]
      topOfStack ← topOfStack - 1
      RETURN dataToReturn
  ENDIF
ENDFUNCTION
</pre>
</td>
<td><b>3</b></td>
</tr>
</table>

Question No. 9



10(d)	**One** mark for each marking point **(Max 5)**
	• Checking if stack is full / empty using `IF … THEN … (ELSE) … ENDIF`
	• … correctly using `StackFull()` function
	• **`RETURN`** suitable message if stack is full
	• **`RETURN`** message if space available on stack
	• Incrementing `TopOfStack` pointer if space available
	• Assigning new data using correct `NewInteger` variable
	• … to correct the array element in `ArrayStack[]` array.
	Example algorithm
	`FUNCTION AddInteger(NewInteger : INTEGER) RETURNS STRING`
	`   IF StackFull() THEN`
	`      RETURN "The stack is full"`
	`   ELSE`
	`      TopOfStack ← TopOfStack + 1`
	`      ArrayStack[TopOfStack] ← NewInteger`
	`      RETURN "Item added"`
	`   ENDIF`
	`ENDFUNCTION`
		5