



## Cambridge International AS & A Level

---

**COMPUTER SCIENCE**

**9608/43**

Paper 4 Further Problem-solving and Programming Skills

**October/November 2021**

MARK SCHEME

Maximum Mark: 75

---

**Published**

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the October/November 2021 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

---

This document consists of **19** printed pages.

**PUBLISHED****Generic Marking Principles**

These general marking principles must be applied by all examiners when marking candidate answers. They should be applied alongside the specific content of the mark scheme or generic level descriptors for a question. Each question paper and mark scheme will also comply with these marking principles.

**GENERIC MARKING PRINCIPLE 1:**

Marks must be awarded in line with:

- the specific content of the mark scheme or the generic level descriptors for the question
- the specific skills defined in the mark scheme or in the generic level descriptors for the question
- the standard of response required by a candidate as exemplified by the standardisation scripts.

**GENERIC MARKING PRINCIPLE 2:**

Marks awarded are always **whole marks** (not half marks, or other fractions).

**GENERIC MARKING PRINCIPLE 3:**

Marks must be awarded **positively**:

- marks are awarded for correct/valid answers, as defined in the mark scheme. However, credit is given for valid answers which go beyond the scope of the syllabus and mark scheme, referring to your Team Leader as appropriate
- marks are awarded when candidates clearly demonstrate what they know and can do
- marks are not deducted for errors
- marks are not deducted for omissions
- answers should only be judged on the quality of spelling, punctuation and grammar when these features are specifically assessed by the question as indicated by the mark scheme. The meaning, however, should be unambiguous.

**GENERIC MARKING PRINCIPLE 4:**

Rules must be applied consistently, e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.

**PUBLISHED****GENERIC MARKING PRINCIPLE 5:**

Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

**GENERIC MARKING PRINCIPLE 6:**

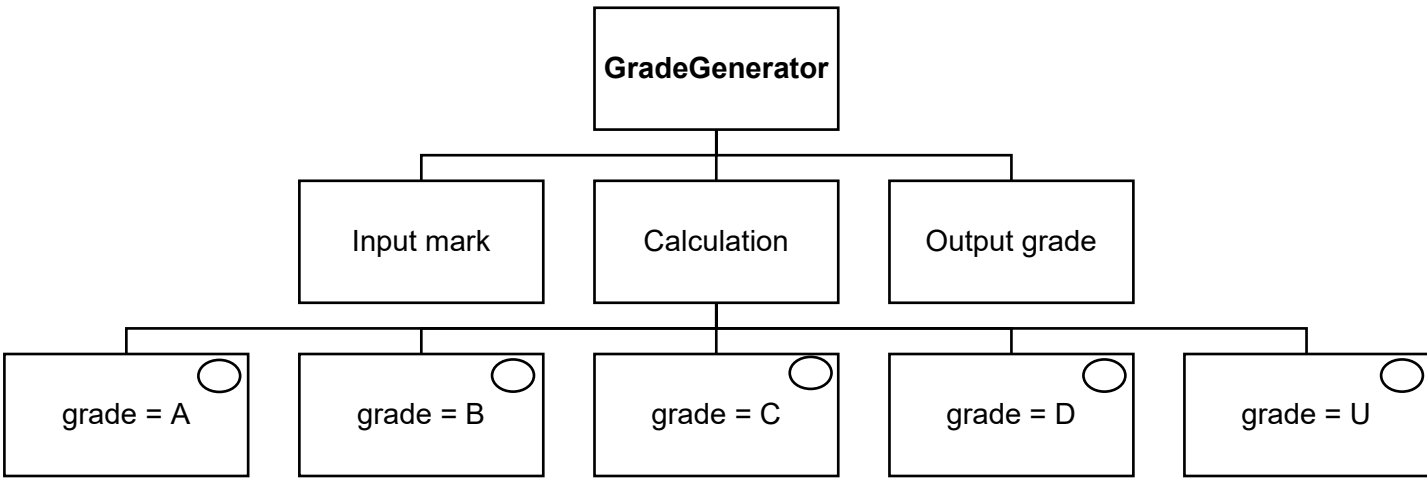
Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

Question	Answer	Marks																											
1(a)	<p>1 mark for TopPointer                      1 mark for correct data in stack</p> <p>TopPointer <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>2</td></tr></table></p> <table style="margin-left: 200px;"> <thead> <tr> <th>Index</th> <th>Data</th> </tr> </thead> <tbody> <tr><td>[7]</td><td><table border="1"><tr><td> </td></tr></table></td></tr> <tr><td>[6]</td><td><table border="1"><tr><td> </td></tr></table></td></tr> <tr><td>[5]</td><td><table border="1"><tr><td> </td></tr></table></td></tr> <tr><td>[4]</td><td><table border="1"><tr><td> </td></tr></table></td></tr> <tr><td>[3]</td><td><table border="1"><tr><td>(8)</td></tr></table></td></tr> <tr><td>[2]</td><td><table border="1"><tr><td>50</td></tr></table></td></tr> <tr><td>[1]</td><td><table border="1"><tr><td>20</td></tr></table></td></tr> <tr><td>[0]</td><td><table border="1"><tr><td>10</td></tr></table></td></tr> </tbody> </table>	2	Index	Data	[7]	<table border="1"><tr><td> </td></tr></table>		[6]	<table border="1"><tr><td> </td></tr></table>		[5]	<table border="1"><tr><td> </td></tr></table>		[4]	<table border="1"><tr><td> </td></tr></table>		[3]	<table border="1"><tr><td>(8)</td></tr></table>	(8)	[2]	<table border="1"><tr><td>50</td></tr></table>	50	[1]	<table border="1"><tr><td>20</td></tr></table>	20	[0]	<table border="1"><tr><td>10</td></tr></table>	10	<b>2</b>
2																													
Index	Data																												
[7]	<table border="1"><tr><td> </td></tr></table>																												
[6]	<table border="1"><tr><td> </td></tr></table>																												
[5]	<table border="1"><tr><td> </td></tr></table>																												
[4]	<table border="1"><tr><td> </td></tr></table>																												
[3]	<table border="1"><tr><td>(8)</td></tr></table>	(8)																											
(8)																													
[2]	<table border="1"><tr><td>50</td></tr></table>	50																											
50																													
[1]	<table border="1"><tr><td>20</td></tr></table>	20																											
20																													
[0]	<table border="1"><tr><td>10</td></tr></table>	10																											
10																													

Question	Answer	Marks
1(b)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• Function header (and close where appropriate returning an integer)</li> <li>• Checking if stack is empty ...</li> <li>• ... and returning -1 if it is</li> <li>• If there is data in stack, decrementing <code>TopPointer</code></li> <li>• (Otherwise) returning the <b>top</b> Value</li> </ul> <p>Example code:</p> <p><b>VB.NET</b></p> <pre>Function Pop()     Dim Value as Integer     If TopPointer &lt; 0 Then         Return -1     Else         Value = DataStack(TopPointer)         TopPointer = TopPointer - 1         Return Value     End if End Function</pre> <p><b>Python</b></p> <pre>def Pop():     if TopPointer &lt; 0 :         return -1     else:         Value = DataStack(TopPointer)         TopPointer= TopPointer - 1         return Value</pre>	<b>5</b>

**PUBLISHED**

<b>Question</b>	<b>Answer</b>	<b>Marks</b>
1(b)	<p><b>Pascal</b></p> <pre>Function Pop(): integer; var   Value : integer; begin   if TopPointer &lt; 0 then     Pop := -1   else     Value := DataStack(TopPointer);     TopPointer := TopPointer - 1;     Pop := Value end;</pre>	
1(c)	<p>1 mark per bullet point to max 2</p> <ul style="list-style-type: none"> <li>• In a stack the last item in is the first out/LIFO <b>and</b> in a queue the first item in is the first out/FIFO</li> <li>• Queue can be circular, but a stack is linear</li> <li>• Stack only needs a pointer to the top (and can have a base pointer) <b>and</b> a queue needs a pointer to the front and the rear</li> </ul>	<b>2</b>

Question	Answer	Marks
2	<p>1 mark per bullet point</p> <ul style="list-style-type: none"><li>• Input <b>mark</b>, calculate grade and output <b>grade</b> on level 1...</li><li>• ... in correct order</li><li>• All grades below calculation</li><li>• Selection only on the grades and no other iteration/selection anywhere</li></ul>  <pre>graph TD;   GG[GradeGenerator] --- IM[Input mark];   GG --- C[Calculation];   GG --- OG[Output grade];   C --- GA["grade = A"];   C --- GB["grade = B"];   C --- GC["grade = C"];   C --- GD["grade = D"];   C --- GU["grade = U"];   style GA fill:#fff,stroke:#000,stroke-width:1px;   style GB fill:#fff,stroke:#000,stroke-width:1px;   style GC fill:#fff,stroke:#000,stroke-width:1px;   style GD fill:#fff,stroke:#000,stroke-width:1px;   style GU fill:#fff,stroke:#000,stroke-width:1px;</pre>	4

Question	Answer	Marks
3	<p>1 mark for each completed statement</p> <pre> FUNCTION BinarySearch(ThisArray, LowerBound, UpperBound, SearchItem: INTEGER)   RETURNS INTEGER    DECLARE Flag : BOOLEAN   DECLARE Mid : INTEGER   Flag ← -2   WHILE Flag &lt;&gt; -1     Mid ← LowerBound + ((UpperBound - LowerBound) DIV 2)     IF <b>UpperBound &lt; LowerBound</b>       THEN         RETURN -1       ELSE         IF ThisArray[Mid] &gt; SearchItem           THEN             UpperBound ← Mid - 1           ELSE             IF ThisArray[Mid] &lt; SearchItem               THEN                 LowerBound ← Mid + 1               ELSE                 RETURN <b>Mid</b>             ENDIF           ENDIF         ENDIF       ENDWHILE     ENDFUNCTION </pre>	6

Question	Answer	Marks
4(a)	<p>1 mark per clause  teacher(fred)  busy(fred, tuesday, 1)</p>	2



**PUBLISHED**

<b>Question</b>	<b>Answer</b>	<b>Marks</b>
4(b)	1 mark for 1 correct 1 mark for all 3 days of the week correct busy(jill, monday, 1) busy(jill, tuesday, 1) busy(jill, wednesday, 1)  1 mark for 1 correct 1 for the other 2 with OR busy(jill, monday, 1) OR busy(jill, tuesday, 1) OR busy(jill, wednesday, 1)	<b>2</b>
4(c)	1 mark busy(X, monday, 3)	<b>1</b>
4(d)	1 mark per bullet point <ul style="list-style-type: none"> <li>• Checking X is a teacher</li> <li>• Checking Y is a timeslot, Z is a day</li> <li>• NOT(busy(X, Z, Y))</li> <li>• All included, linked with ANDs and nothing superfluous</li> </ul> teacher(X) AND timeslot(Y) AND day(Z) AND NOT(busy(X, Z, Y))	<b>4</b>

Question	Answer	Marks																				
5(a)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• Output = 21</li> <li>• Function calls with Recursion(104, 102) and Recursion(103, 102)</li> <li>• Function calls with 102 and 102, and 92 and 102</li> <li>• Unwinding the return values 5+5+10+1</li> </ul> <table border="1" data-bbox="338 427 1178 853"> <thead> <tr> <th>Function call</th> <th>A</th> <th>B</th> <th>Return value</th> </tr> </thead> <tbody> <tr> <td>Recursion(104, 102)</td> <td>104</td> <td>102</td> <td>5 + Recursion(103, 102) 5 + 16</td> </tr> <tr> <td>Recursion(103, 102)</td> <td>103</td> <td>102</td> <td>5 + Recursion(102, 102) 5 + 11</td> </tr> <tr> <td>Recursion(102, 102)</td> <td>102</td> <td>102</td> <td>10 + Recursion(92, 102) 10 + 1</td> </tr> <tr> <td>Recursion(92, 102)</td> <td>92</td> <td>102</td> <td>1</td> </tr> </tbody> </table>	Function call	A	B	Return value	Recursion(104, 102)	104	102	5 + Recursion(103, 102) 5 + 16	Recursion(103, 102)	103	102	5 + Recursion(102, 102) 5 + 11	Recursion(102, 102)	102	102	10 + Recursion(92, 102) 10 + 1	Recursion(92, 102)	92	102	1	<b>4</b>
Function call	A	B	Return value																			
Recursion(104, 102)	104	102	5 + Recursion(103, 102) 5 + 16																			
Recursion(103, 102)	103	102	5 + Recursion(102, 102) 5 + 11																			
Recursion(102, 102)	102	102	10 + Recursion(92, 102) 10 + 1																			
Recursion(92, 102)	92	102	1																			

Question	Answer	Marks
5(b)	<p>1 mark per bullet point to max 4</p> <ul style="list-style-type: none"> <li>• Function header takes two parameters, returns the calculated value accurately (outside/end loop and in all cases)</li> <li>• Initialising variable to 1 outside loop (or adds 1 before returning)</li> <li>• Looping while A &gt; 100 // looping until A &lt;= 100 (or equivalent) ...</li> <li>• ... checking if A &gt; B inside loop <b>and</b> if true, add 5 to variable and decrement A</li> <li>• ... checking if A &lt;= B in loop <b>and</b> if true, add 10 to variable and A – 10</li> </ul> <p>Example pseudocode:</p> <pre> FUNCTION Recursion(A, B : INTEGER) RETURNS INTEGER   DECLARE Value : INTEGER   Value ← 1   WHILE A &gt; 100     IF A &gt; B       THEN         Value ← Value + 5         A ← A - 1       ELSE         Value ← Value + 10         A ← A - 10     ENDIF   ENDWHILE   RETURN Value ENDFUNCTION </pre>	<b>4</b>

Question	Answer	Marks
6(a)(i)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• Data structure to store <b>multiple pieces of data</b> (under one identifier)</li> <li>• ... (stores data) of that can be different data types</li> </ul>	<b>2</b>

**PUBLISHED**

Question	Answer	Marks
6(a)(ii)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>record declaration named <code>CustomerData</code> ...</li> <li>... all 3 correct data items with suitable data types (and identifiers)</li> </ul> <pre> TYPE CustomerData   DECLARE CustomerID : INTEGER   DECLARE FirstName : STRING   DECLARE SecondName : STRING ENDTYPE </pre>	<b>2</b>
6(b)	<p>1 mark per completed statement</p> <pre> PROCEDURE StoreRecord(NewData : <b>CustomerData</b>)   HashValue ← CustomerHash(NewData.CustomerID)   Filename ← "CustomerRecords.dat"   OPENFILE Filename FOR <b>RANDOM</b>   SEEK Filename, <b>HashValue</b>   PUTRECORD Filename, <b>NewData</b>   <b>CLOSE</b> Filename ENDPROCEDURE </pre>	<b>5</b>
6(c)	<p>1 mark for naming a feature, 1 for description. Max 2 for each feature</p> <p>Example:</p> <ul style="list-style-type: none"> <li>Breakpoint</li> <li>Stop the program at a set point and check the variables</li> <li>Stepping/step-through etc.</li> <li>Execute the program one line at a time to check the values</li> <li>Variable watch window</li> <li>Displays the variable values whilst the program is running so Kobi can make sure they are changed correctly</li> </ul>	<b>4</b>

**PUBLISHED**

Question	Answer	Marks
6(d)	1 mark for benefit, 1 for drawback Benefit Example: <ul style="list-style-type: none"> <li>• Saves time because does not have to write own code // write program faster</li> <li>• Programmer can have limited skills and still produce complex programs</li> </ul> Drawback Example: <ul style="list-style-type: none"> <li>• May not perform the tasks exactly as required</li> <li>• Solution is likely to be inefficient</li> <li>• Might produce errors</li> <li>• The programmer may not understand the solution and hence cannot edit/change</li> </ul>	<b>2</b>

Question	Answer	Marks
7(a)	1 mark per bullet point to max 2 <ul style="list-style-type: none"> <li>• To stop the program crashing ...</li> <li>• To stop a run-time error ...</li> <li>• ... to make sure the input is the correct data type // other reasonable example</li> </ul>	<b>2</b>

Question	Answer	Marks
7(b)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• Using try (and close where appropriate) followed by the input</li> <li>• Catching exception</li> <li>• Outputting appropriate message (built-in or otherwise)</li> </ul> <p>Example program code:</p> <p><b>VB.NET</b></p> <pre>Try   Dim Value As Integer   Console.WriteLine("Enter a number")   Value = Console.ReadLine() Catch ex As Exception   Console.WriteLine(ex.Message) End Try</pre> <p><b>Python</b></p> <pre>try:     Value = int(input("Enter a number")) except:     print("Invalid number")</pre> <p><b>Pascal:</b></p> <pre>begin   try     readln(Value);   except     On E : Exception do writeln("Invalid number");   end;</pre>	<b>3</b>
7(c)	<p>1 mark per example</p> <ul style="list-style-type: none"> <li>• Check file exists</li> <li>• No input</li> <li>• No data in file</li> <li>• Array out of bounds</li> <li>• Calculation / division by 0</li> </ul>	<b>2</b>

Question	Answer	Marks																																																																								
8(a)	<p>1 mark for rows with index 0, 1 and 3                      1 mark for null pointers set to -1</p> <table border="1" data-bbox="336 316 1379 1104"> <tr> <td data-bbox="336 383 504 443">RootNode</td> <td data-bbox="504 383 600 443" style="border: 1px solid black; text-align: center;">0</td> <td data-bbox="696 331 792 363">Index</td> <td data-bbox="792 319 1021 379">LeftPointer</td> <td data-bbox="1021 319 1140 379">Data</td> <td data-bbox="1140 319 1379 379">RightPointer</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 395 792 427">[0]</td> <td data-bbox="792 383 1021 443">3</td> <td data-bbox="1021 383 1140 443">50</td> <td data-bbox="1140 383 1379 443">1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 459 792 491">[1]</td> <td data-bbox="792 443 1021 504">6</td> <td data-bbox="1021 443 1140 504">67</td> <td data-bbox="1140 443 1379 504">2</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 523 792 555">[2]</td> <td data-bbox="792 504 1021 564">-1</td> <td data-bbox="1021 504 1140 564">77</td> <td data-bbox="1140 504 1379 564">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 587 792 619">[3]</td> <td data-bbox="792 564 1021 625">4</td> <td data-bbox="1021 564 1140 625">35</td> <td data-bbox="1140 564 1379 625">5</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 651 792 683">[4]</td> <td data-bbox="792 625 1021 686">-1</td> <td data-bbox="1021 625 1140 686">2</td> <td data-bbox="1140 625 1379 686">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 715 792 746">[5]</td> <td data-bbox="792 686 1021 746">-1</td> <td data-bbox="1021 686 1140 746">43</td> <td data-bbox="1140 686 1379 746">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 778 792 810">[6]</td> <td data-bbox="792 746 1021 807">-1</td> <td data-bbox="1021 746 1140 807">52</td> <td data-bbox="1140 746 1379 807">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 842 792 874">[7]</td> <td data-bbox="792 807 1021 868">-1</td> <td data-bbox="1021 807 1140 868"></td> <td data-bbox="1140 807 1379 868">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 906 792 938">[8]</td> <td data-bbox="792 868 1021 928">-1</td> <td data-bbox="1021 868 1140 928"></td> <td data-bbox="1140 868 1379 928">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 970 792 1002">[9]</td> <td data-bbox="792 928 1021 989">-1</td> <td data-bbox="1021 928 1140 989"></td> <td data-bbox="1140 928 1379 989">-1</td> </tr> <tr> <td></td> <td></td> <td data-bbox="696 1034 792 1066">[10]</td> <td data-bbox="792 989 1021 1050">-1</td> <td data-bbox="1021 989 1140 1050"></td> <td data-bbox="1140 989 1379 1050">-1</td> </tr> </table>	RootNode	0	Index	LeftPointer	Data	RightPointer			[0]	3	50	1			[1]	6	67	2			[2]	-1	77	-1			[3]	4	35	5			[4]	-1	2	-1			[5]	-1	43	-1			[6]	-1	52	-1			[7]	-1		-1			[8]	-1		-1			[9]	-1		-1			[10]	-1		-1	<b>2</b>
RootNode	0	Index	LeftPointer	Data	RightPointer																																																																					
		[0]	3	50	1																																																																					
		[1]	6	67	2																																																																					
		[2]	-1	77	-1																																																																					
		[3]	4	35	5																																																																					
		[4]	-1	2	-1																																																																					
		[5]	-1	43	-1																																																																					
		[6]	-1	52	-1																																																																					
		[7]	-1		-1																																																																					
		[8]	-1		-1																																																																					
		[9]	-1		-1																																																																					
		[10]	-1		-1																																																																					

**PUBLISHED**

Question	Answer	Marks
8(b)	<p>1 mark for each completed statement</p> <pre> PROCEDURE PostOrder(<b>RootNode</b> : INTEGER)   IF BinaryTree[RootNode, 0] &lt;&gt; -1 THEN     <b>PostOrder</b>(BinaryTree[RootNode, 0])   ENDIF   IF BinaryTree[RootNode, 2] &lt;&gt; -1 THEN     <b>PostOrder</b>(BinaryTree[RootNode, 2])   ENDIF   OUTPUT(BinaryTree[RootNode, 1]) ENDPROCEDURE </pre>	<b>5</b>

Question	Answer	Marks
9(a)	<p>1 mark per bullet point</p> <ul style="list-style-type: none"> <li>• array named <code>StoredData</code> of type integer</li> <li>• with 10 000 elements, index 0–9999</li> <li>• All elements initialised with -1</li> </ul> <p>Example pseudocode</p> <pre> DECLARE <code>StoredData</code> : ARRAY[0:9999] OF INTEGER FOR X ← 0 to 9999   <code>StoredData</code>[X] ← -1 NEXT X </pre>	<b>3</b>
9(b)	1 mark per bullet point to max 7	<b>7</b>



Question	Answer	Marks
9(b)	<ul style="list-style-type: none"> <li>• Function declaration (and end where appropriate) taking data as (integer) parameter (returns Boolean)</li> <li>• Calculate hash: <b>parameter</b> mod 1000 + 6</li>   <li>• Check if StoredData[hashed value] = -1 ...</li>   <li>• ... if it is -1, store data at <b>hash</b> ...</li> <li>• ... and return true</li>   <li>• ... if not -1, increment/decrement hashed value by 1 ...</li> <li>• ... if reached index 9999 return to index 0 // checking and going to 9999 if not at 0</li> <li>• ... repeatedly decrement until either <b>found or all elements checked</b> ...</li> <li>• ... returning False if <b>full and</b> True when stored</li> </ul> <p>Example program code</p> <p><b>VB.NET</b></p> <pre>Function AddItem(DataToAdd)     Dim Location As Integer     Dim Found As Boolean     Dim Counter As Integer      Location = (DataToAdd Mod 1000) + 6     If StoredData(Location) &lt;&gt; -1 Then         Found = False         Counter = 0         While Found = False And Counter &lt; 9999             Location = Location + 1             If Location &gt; 9999 Then                 Location = 0             End If             If StoredData(Location) = -1 Then                 Found = True             End If             Counter = Counter + 1         End While</pre>	

Question	Answer	Marks
9(b)	<pre> If Found = True Then     StoredData(Location) = DataToAdd     Return True Else     Return False End If Else     StoredData(Location) = DataToAdd     Return True End If End Function  <b>Python</b> def AddItem(DataToAdd):     Location = (DataToAdd % 1000) + 6     if StoredData[Location] &lt;&gt; -1:         Found = False         Counter = 0          while Found == False and Counter &lt; 9999:             Location = Location + 1             if Location &gt; 9999:                 Location = 0             if StoredData[Location] == -1:                 Found = True                 Counter = Counter + 1              if Found == True:                 StoredData[Location] = DataToAdd                 return True             else:                 return False         else:             StoredData[Location] = DataToAdd             return True </pre>	

Question	Answer	Marks
9(b)	<p><b>Pascal</b></p> <pre> function AddItem(DataToAdd:Integer):Boolean; begin   Location := (DataToAdd mod 1000) + 6;   if StoredData[Location] &lt;&gt; -1 then     begin       Found := false;       Counter := 0;        while (Found = false) and (Counter &lt; 9999) do         begin           Location := Location + 1;           if Location &gt; 9999 then             Location := 0;           if StoredData[Location] = -1 then             found := true;             Counter := Counter + 1;           end;           if Found = true then             begin               StoredData[Location] := DataToAdd;               AddItem := True;             end           Else             begin               AddItem := False;             end;           end         end       else         begin           StoredData[Location] := DataToAdd;           AddItem := True;         end;       end;     end;   end; end; </pre>	