CAMBRIDGE
International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/21**

Paper 2 Fundamental Problem-solving and Programming Skills **October/November 2025**

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 2025 series for most Cambridge IGCSE, Cambridge International A and AS Level components, and some Cambridge O Level components.

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 descriptions 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.

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.

Annotations guidance for centres

Examiners use a system of annotations as a shorthand for communicating their marking decisions to one another. Examiners are trained during the standardisation process on how and when to use annotations. The purpose of annotations is to inform the standardisation and monitoring processes and guide the supervising examiners when they are checking the work of examiners within their team. The meaning of annotations and how they are used is specific to each component and is understood by all examiners who mark the component.

We publish annotations in our mark schemes to help centres understand the annotations they may see on copies of scripts. Note that there may not be a direct correlation between the number of annotations on a script and the mark awarded. Similarly, the use of an annotation may not be an indication of the quality of the response.

The annotations listed below were available to examiners marking this component in this series.

Annotations

Annotation	Meaning
	Benefit of the doubt
	To indicate where a key word/phrase/code is missing
	Incorrect
	Follow through
	Indicate a point in an answer
Highlighted text	To draw attention to a particular aspect or to indicate where parts of an answer have been combined
	Ignore
	Not answered question
	No examples or not enough
	Not relevant or used to separate parts of an answer
Off-page comment	Allows comments to be entered at the bottom of the RM marking window and then displayed when the associated question item is navigated to.
	Repetition
	Indicates that work or a page has been seen including blank answer spaces and blank pages.
	Correct

Annotation	Meaning
TV	Too vague

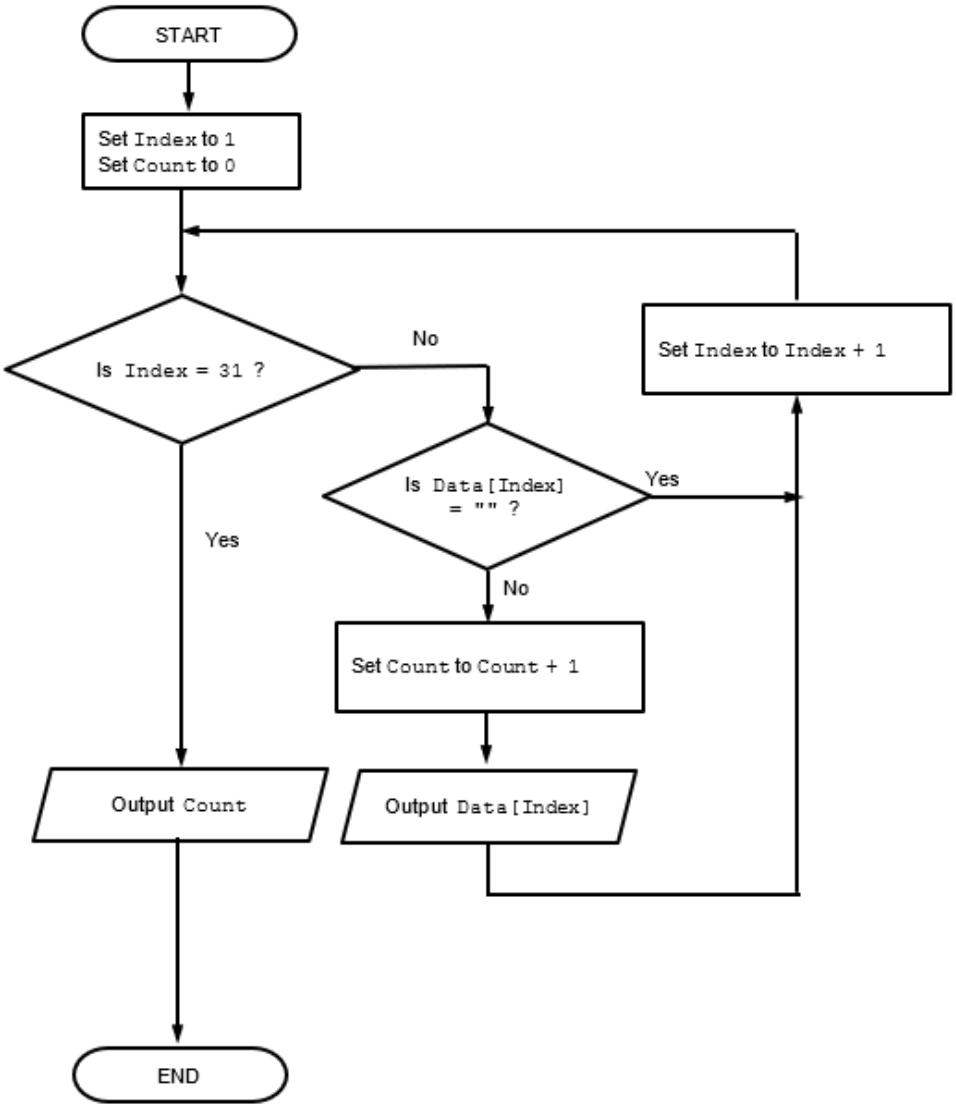
Mark scheme abbreviations

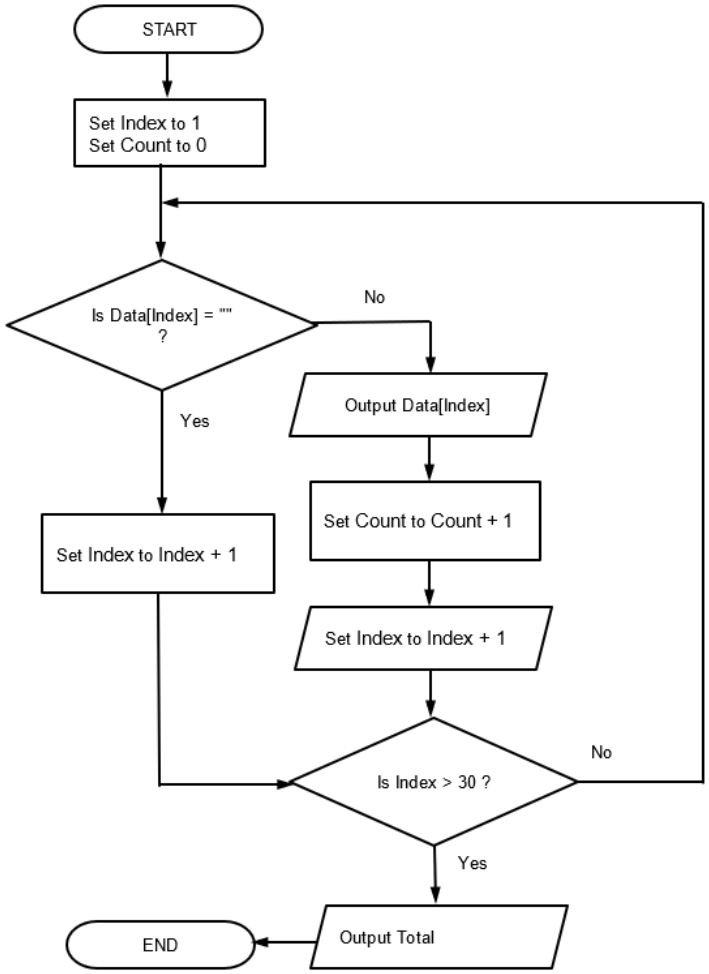
/	separates alternative words / phrases within a marking point
//	separates alternative answers within a marking point
underline	actual word given must be used by candidate (grammatical variants accepted)
max	indicates the maximum number of marks that can be awarded
( )	the word / phrase in brackets is not required, but sets the context

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>
<table>
<tr><th>Expression</th></tr>
<tr><td><code>MyString ← 'X' & MyString</code></td></tr>
<tr><td><code>MyChar ← MID ("ABCD", 1 / 2 / 3 / 4 , 1)</code></td></tr>
<tr><td><code>MyString ← NUM_TO_STR (DAY / MONTH / YEAR / DAYINDEX (MyDOB))</code></td></tr>
<tr><td><code>MyInt ← INT (LENGTH (MyString) / 2)</code></td></tr>
</table>
One mark per row
</td><td>4</td></tr>
<tr><td>1(b)</td><td>
<table>
<tr><th>Test description</th><th>Test method</th></tr>
<tr><td>carried out as soon as a program module has been coded</td><td>alpha</td></tr>
<tr><td>carried out as program modules are being combined</td><td>integration</td></tr>
<tr><td>completed by the developers without referring to the code</td><td>black-box</td></tr>
<tr><td>completed by the customer</td><td>acceptance / beta</td></tr>
</table>
One mark per row (2 to 4)
</td><td>3</td></tr>
<tr><td>1(c)</td><td>One mark per point:

1 reference to a breakpoint
2 reference to single stepping
3 Correct explanation of both e.g. to stop program execution at specific point / line / statement / instruction and to execute one line / statement / instruction at a time to (locate an/the error)

One mark for each bulleted point</td><td>3</td></tr>
</table>

Question	Answer	Marks
2	Example solution  Mark as follows: 1 Initialisation of variables used as **array index** and **count** 2 Loop through exactly 30 elements in the array 3 Test for empty string in current array element // Test for non-empty string 4 Increment count if appropriate **in a loop** 5 Both required outputs under correct conditions with correct outputs	**5**

Question	Answer	Marks
2	Alternative solution Checking for empty string first 	

Question	Answer	Marks

3(a)(i)	Pseudocode:
	
	`TYPE RentalRecord`
	`  DECLARE RentalID : STRING`
	`  DECLARE CarID : INTEGER`
	`  DECLARE DisCode : CHAR`
	`  DECLARE Start : DATE`
	`  DECLARE Duration : INTEGER`
	`  DECLARE Completed : BOOLEAN`
	`ENDTYPE`
	
	Mark as follows:
	
	1 One mark for `TYPE RentalRecord` and `ENDTYPE` statements
	2 One mark for `RentalID` and `CarID` declared correctly
	3 One mark for `DisCode` and `Start` declared correctly
	4 One mark for `Duration` and `Completed` declared correctly
		4

3(a)(ii)	`DECLARE Rental : ARRAY[1:500] OF RentalRecord` One mark per underlined phrase	**2**
3(b)	One mark per point: 1 **Different data types** held (for each rental) under a single identifier 2 Allows for iteration through rentals/records // Can access **rentals/records** in a loop // Can (directly) access a specific rental/record using an (array) index 3 Simplifies the code/program / less code needed / reduces code duplication // Makes **code/program** easier to understand // Makes testing / debugging easier // **Program** maintenance easier	**3**

Question	Answer	Marks

4(a)	Example solution - conditional:
	
	`PROCEDURE Store()`
	`  DECLARE ThisNum, LastNum : REAL`
	`  DECLARE Index : INTEGER // index to global array`
	
	`  INPUT ThisNum`
	`  LastNum ← ThisNum`
	`  Number[1] ← ThisNum`
	
	`  INPUT ThisNum`
	`  Index ← 2`
	
	`  WHILE Index < 21 AND ThisNum > LastNum`
	`  Number[Index] ← ThisNum`
	`  LastNum ← ThisNum`
	`  Index ← Index + 1`
	`  IF Index < 21 THEN`
	`  INPUT ThisNum`
	`  ENDIF`
	`  ENDWHILE`
	
	`  OUTPUT Index - 1, " values were stored in the array"`
	`ENDPROCEDURE`
	
	Alternative loop:
	`  // Loop without using LastNum`
	`  WHILE Index < 21 AND ThisNumNum > Number[Index - 1]`
	`  Number[Index] ← ThisNum`
	`  Index ← Index + 1`
	`  IF Index < 21 THEN  //skip input if array full`
	`  INPUT ThisNum`
	`  ENDIF`
	`  ENDWHILE`
	
	Mark as follows:
	1 Declare local variables `Index` as integer, `ThisNum` and `LastNum` as real
	2 Input value and assign to first element
	3 **Attempt** at conditional loop including **both** tests
	4 **Completely correct** conditional loop including **both** tests
	5 Assign input value to current element if greater than previous element **and** input next value **in a loop**
	6 Final output giving attempted count of elements stored plus message **after a loop**
		6

Question	Answer	Marks

4(a)	Alternative FOR loop example solution:
	
	`PROCEDURE Store()`
	`  DECLARE ThisNum, LastNum : REAL`
	`  DECLARE Index : INTEGER // index to global array`
	`  DECLARE Count : INTEGER`
	`  INPUT ThisNum`
	`  LastNum ← ThisNum`
	`  Number[1] ← ThisNum`
	`  Count ← 1`
	`  INPUT ThisNum`
	`  FOR Index ← 2 TO 20`
	`  IF ThisNum > LastNum THEN`
	`  Number[Index] ← ThisNum`
	`  LastNum ← ThisNum`
	`  Count ← Count + 1`
	`  IF Index < 20 THEN`
	`  INPUT ThisNum`
	`  ENDIF`
	`  ELSE`
	`  BREAK`
	`  ENDIF`
	`  NEXT Index`
	`  OUTPUT Count, " values were stored in the array"`
	`ENDPROCEDURE`
	
	Alternative loop:
	`  Count ← 1`
	`  INPUT ThisNum`
	`  FOR Index ← 2 TO 20`
	`  IF ThisNum > Number[Index - 1] THEN`
	`  Number[Index] ← ThisNum`
	`  Count ← Count + 1`
	`  IF Index < 20 THEN`
	`  INPUT ThisNum`
	`  ENDIF`
	`  ELSE`
	`  BREAK`
	`  ENDIF`
	`  NEXT Index`
	
	Mark as follows:
	1 Declare local variables `Index` as integer, `ThisNum` and `LastNum` as real
	2 Input value and assign to first element
	3 Count-controlled loop
	4 Count-controlled loop **with BREAK** if current value greater than previous value
	5 ... Assign input value to current element **following correct comparison in a loop**
	6 Final output giving count of elements stored plus message following a reasonable attempt **after loop**	

Question	Answer	Marks
4(b)(i)	One mark per point: 1 The amount of values **stored** (in a text file) is **not fixed** / **not limited** (other than by secondary storage available) // The amount values **stored** is **not limited** to the size of the array 2 The data is stored **permanently** / **non-volatile** // Data can be restored / reused without **re-entering** values	**2**
4(b)(ii)	Each (real) value would have to be **converted** to a string	**1**

Question	Answer	Marks
5(a)(i)	It is a value that is **invalid as an** (array) **index** // not in the **range** of valid (index) values	**1**
5(a)(ii)	To test whether the **current element / value at index** matches the **given / search** string / data / **parameter**	**1**
5(a)(iii)	The value of `FoundAt` is only updated to the (current) index if it currently stores the initial value / -1 // `FoundAt` is not updated when it currently stores any other value than -1	**1**
5(b)(i)	The loop continues after the (first instance) of the **search** value / string / **matching element** has been found Comparisons/tests continue to be made even if the **search** value **/ string / matching element** has been found	**1**
5(b)(ii)	One mark per point Construct: A conditional loop Justification: The loop / search can be terminated as soon as the (first occurrence of) **`SearchString`** / **required** value / **required** string / **matching** element is found	**2**

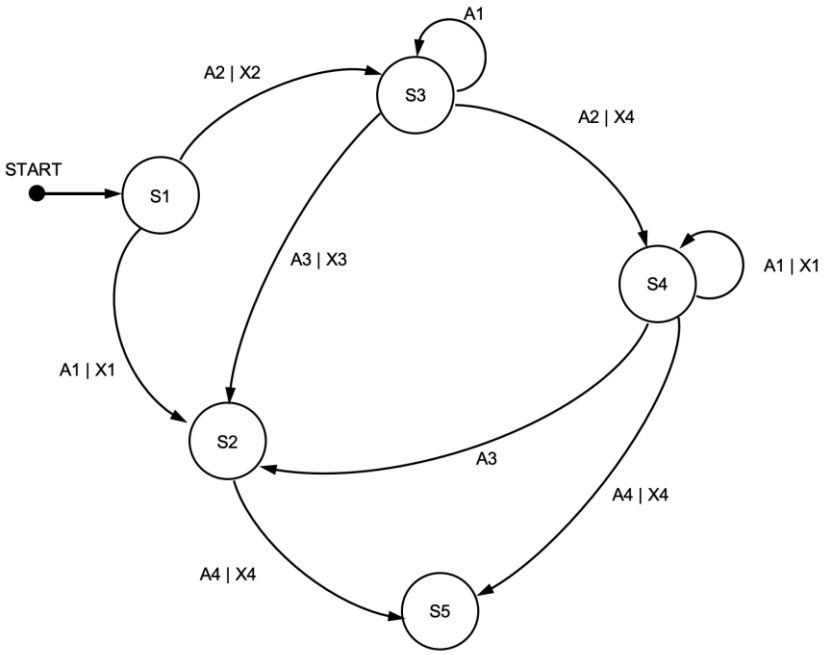
Question	Answer	Marks
6(a)	Number: Any number where the first two digits sum to 5 / 15 **and** the last three digits are 623 Explanation: The remainder is the same / 3 // The sum of the (first four) digits is the same / 13	**2**

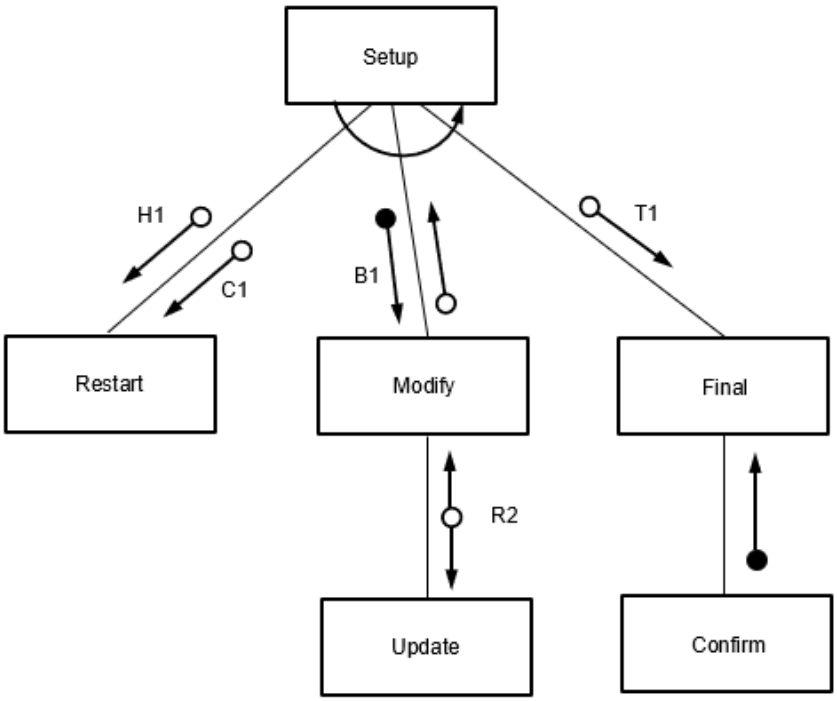
Question	Answer	Marks

6(b)	Example solution – conversion of `Number` to a string
	
	`FUNCTION Generate(Number : INTEGER) RETURNS INTEGER`
	`  DECLARE Total, Index, CheckDigit : INTEGER`
	`  DECLARE NumString : STRING`
	
	`  Total ← 0`
	`  NumString ← NUM_TO_STR(Number)`
	
	`  FOR Index ← 1 TO LENGTH(NumString)`
	`  Total ← Total + STR_TO_NUM(MID(NumString, Index, 1))`
	`  NEXT Index`
	
	`  CheckDigit ← Total MOD 10`
	
	`  Number ← Number * 10 + CheckDigit`
	`  // NumString ← Numstring & NUM_TO_STR(CheckDigit)`
	`  RETURN Number // STR_TO_NUM(NumString)`
	
	`ENDFUNCTION`
	
	Mark as follows:
	
	1 Function heading, parameters return type and ending
	2 Initialise `Total`
	3 Convert `Number` to a string **and** loop for length of converted `Number`
	4 attempt to form sum of digits **in a loop**
	5 completely correct sum of digits **in a loop**
	6 Calculate `CheckDigit`
	7 Add `CheckDigit` to original number multiplied by 10 **and** return number
	//
	Convert `CheckDigit` to a character and appended to `NumString` **and** then convert to a number and return
		7

Question	Answer	Marks

6(b)	Example Solution – no conversion to string
	
	`FUNCTION Generate(Number : INTEGER) RETURNS INTEGER`
	`  DECLARE Total, CheckDigit, Remainder, Num: INTEGER`
	
	`  Num ← Number`
	`  Total ← 0`
	`  WHILE Num > 0`
	`  Remainder ← Num MOD 10`
	`  Num ← Num DIV 10`
	`  Total ← Total + Remainder`
	`  ENDWHILE`
	
	`  CheckDigit ← Total MOD 10`
	
	`  Number ← Number * 10 + CheckDigit`
	
	`  RETURN Number`
	
	`ENDFUNCTION`
	
	Mark as follows:
	
	1 Function heading, parameters return type and ending
	2 Initialise `Total`
	3 Loop while `Num > 0`
	4 attempt to use MOD and DIV to sum successive digits in `Number`
	5 completely correct sum of digits
	6 Calculate `CheckDigit`
	7 Add `CheckDigit` to original number **and** return number	

Question	Answer	Marks
7(a)	 One mark for each: 1 Label A1 | X1 added on line from S1 to state labelled S2 2 States S3, S4 and S5 labelled 3 Line from S2 to S5 with event label A4|X4 4 Lines from S3 with correct event labels including 'loop' label 5 Lines from S4 with correct event labels including 'loop'	**5**

Question	Answer	Marks
7(b)	 One mark per point: 1 All boxes correctly labelled 2 Iteration arrow 3 Parameter to Restart 4 Return value from Modify **and** parameter to Final 5 BYREF parameter to Update	**5**

Question	Answer	Marks

8(a)	Example solution:
	
	`PROCEDURE OKToBorrow(ThisStudentID : STRING)`
	`  DECLARE Index, Count : INTEGER`
	
	`  CONSTANT Max = 5`
	`  Count ← 0`
	`  Index ← 1 / 0`
	
	`  WHILE Index <= 5000 / Index <= 4999 AND Count < Max`
	`  IF Loan[Index].StudentID = ThisStudentID AND __`
	`  Loan[Index].OnLoan =`
	`TRUE THEN`
	`  Count ← Count + 1`
	`  ENDIF`
	`  Index ← Index + 1`
	`  ENDWHILE`
	
	`  IF Count < Max THEN`
	`  OUTPUT "Student may borrow another book"`
	`  ELSE`
	`  OUTPUT "Student may not borrow another book"`
	`  ENDIF`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	1 Procedure heading, parameter and ending
	2 Loop through **all elements** in `Loan` array
	3 ... terminating if `Max` reached
	
	4 Test for correct `StudentID` field **in a loop**
	5 ... and that `OnLoan = TRUE` **in a loop**
	... if so, then increment `Count` **in a loop**
	6 Output an appropriate **message** in both cases **after the loop**
		7

Question	Answer	Marks

8(b)	Example solution:
	
	`FUNCTION ReturnBook(ThisStudentID, ThisBookID : STRING) RETURNS BOOLEAN`
	`  DECLARE Index : INTEGER`
	
	`  Index ← 1`
	
	`  WHILE Index < 5001`
	`  IF Loan[Index].StudentID = ThisStudentID AND __`
	`  Loan[Index].BookID = ThisBookID THEN`
	`  IF Loan[Index].OnLoan = TRUE THEN // Not returned`
	`  Loan[Index].OnLoan ← FALSE // Mark as returned now`
	`  RETURN TRUE //Found and not already returned`
	`  ELSE`
	`  RETURN FALSE // Found and already returned`
	`  ENDIF`
	`  ENDIF`
	`  Index ← Index + 1`
	`  ENDWHILE`
	
	`  RETURN False // loan not found`
	`ENDFUNCTION`
	
	Mark as follows:
	1 Loop through **all elements** in `Loan` array
	2 ...and book loan not found
	3 Attempt to reference an individual data item **in a loop**
	4 Correct test for `StudentID` **and** correct `BookID` **in a loop**
	5 **...** If book not returned yet, then set `OnLoan` field to `FALSE`
	6 RETURN TRUE if book loan Found and not already returned
	7 RETURN FALSE if book loan Found and loan already returned
	8 RETURN FALSE if book loan not found
	
	Alterantive example solution – not immediate return
	
	`FUNCTION ReturnBook(ThisStudentID, ThisBookID : STRING) RETURNS BOOLEAN`
	`  DECLARE Index : INTEGER`
	`  DECLARE Found, OK : BOOLEAN`
	
	`  Index ← 1`
	`  Found ← FALSE`
	`  OK ← TRUE`
		8

Question	Answer	Marks

8(b)	`WHILE Index < 5001 AND NOT Found`
	`  IF Loan[Index].StudentID = ThisStudentID  AND __`
	`  Loan[Index].BookID = ThisBookID THEN`
	`  Found ← TRUE`
	`  IF Loan[Index].OnLoan = FALSE THEN // Already returned`
	`  OK ← FALSE`
	`  ELSE`
	`  Loan[Index].OnLoan ← FALSE // Mark as returned now`
	`  ENDIF`
	`  ENDIF`
	`  Index ← Index + 1`
	`ENDWHILE`
	
	`RETURN Found AND OK`
	`ENDFUNCTION`	

8(c)	Award marks for reference to following design features: Change to existing Record: 1 Store date due back as a new data item / in the loan record 2 Store date of loan **and** the loan length category as new data items / in the loan record **Max 1** New Record 3 **New** Records with Category (and Loan length field) Change to Program: 4 (Use the loan length category of a book to) calculate the date due date 5 Compare due date with today's date (to check if overdue / to calculate fine) 6 Map loan length category to length of loan / date due back **Max 1** **Max 2 marks**	**2**