Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/21**

Paper 2 Problem Solving & Programming Skills **May/June 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 May/June 2021 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 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.

GENERIC MARKING PRINCIPLE 5:

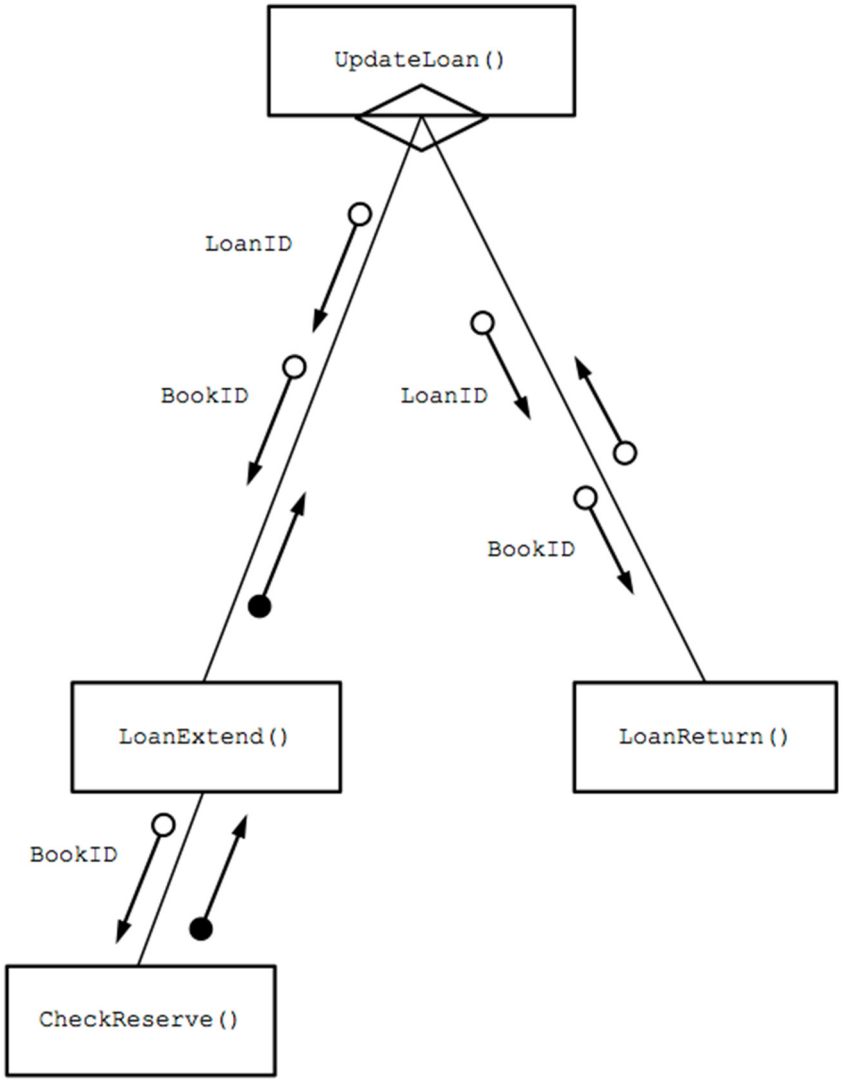
Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

GENERIC MARKING PRINCIPLE 6:

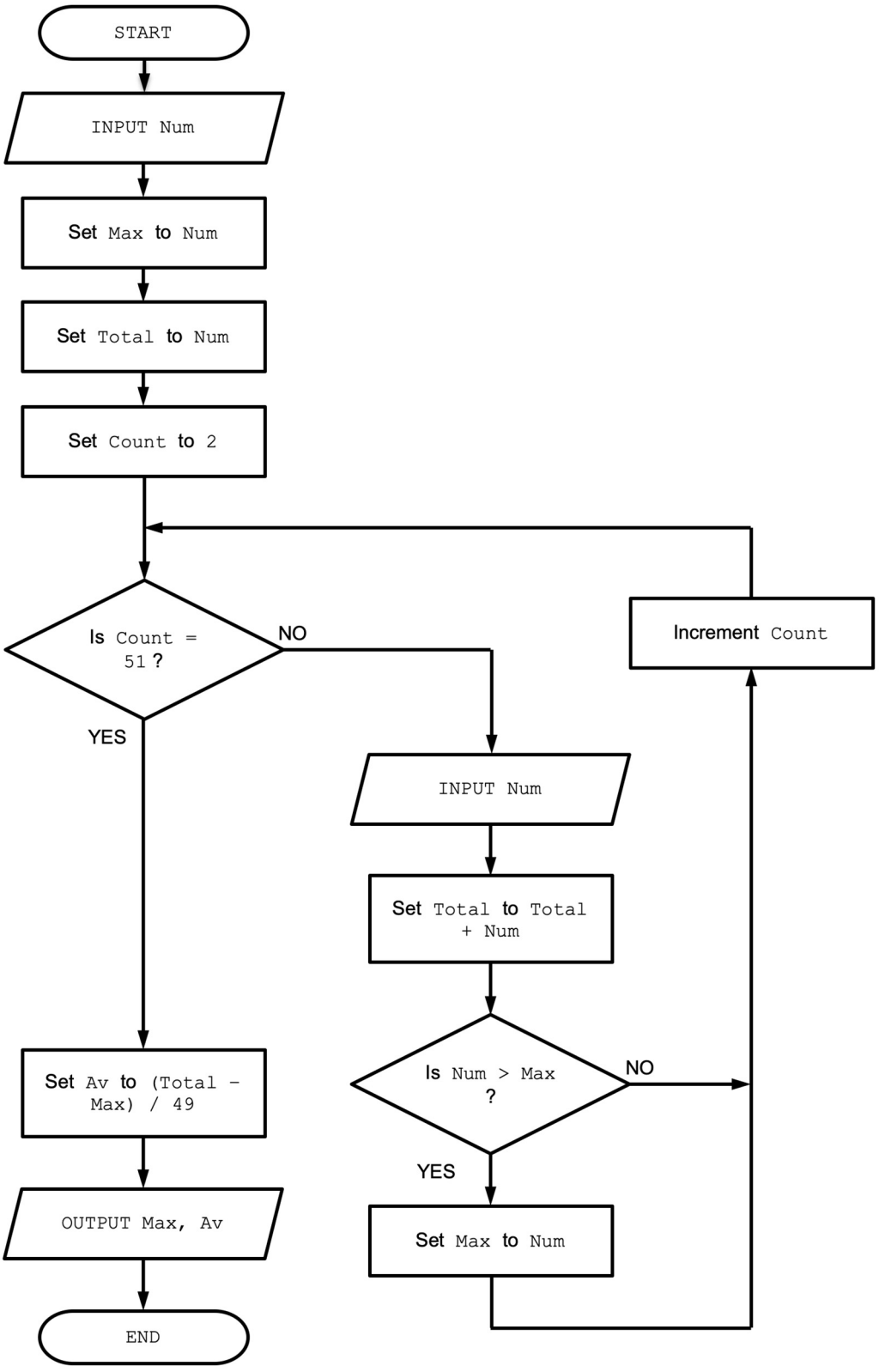
Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>
<table>
<tr><th>Example value</th><th>Explanation</th><th>Variable name</th><th>Data type</th></tr>
<tr><td>"Wong"</td><td>The preferred name of the member joining the football club</td><td>MemberName</td><td>STRING</td></tr>
<tr><td>FALSE</td><td>A value to indicate whether an existing member of the club lives at the same address</td><td>FamilyMember</td><td>BOOLEAN</td></tr>
<tr><td>19/02/1983</td><td>When the member joined the football club</td><td>StartDate</td><td>DATE</td></tr>
<tr><td>1345</td><td>The number of points a member has earned. Members of the club earn points for different activities.</td><td>Points</td><td>INTEGER</td></tr>
</table>
One mark for each appropriate variable name plus data type
</td><td>4</td></tr>
<tr><td>1(b)</td><td>
<table>
<tr><th>Statement</th><th>Error</th></tr>
<tr><td>Result ← 2 & 4</td><td>Should be arithmetic operator (not &) // 2 and 4 should be CHAR / STRING</td></tr>
<tr><td>SubString ← MID("pseudocode", 4, 1)</td><td>NO ERROR</td></tr>
<tr><td>IF x = 3 OR 4 THEN</td><td>Not Boolean values / incorrect operator // Condition incorrect</td></tr>
<tr><td>Result ← Status AND INT(X/2)</td><td>INT(X/2) doesn't evaluate to a boolean value / incorrect operator</td></tr>
<tr><td>Message ← "Done" + LENGTH(MyString)</td><td>Can’t add string to number / "Done" is not a number</td></tr>
</table>
One mark for each row
</td><td>5</td></tr>
</table>

Question	Answer	Marks
1(c)	Structure: • Record Justification: • Allows different data types • to be stored under one identifier Mark as follows: One mark for Structure Two marks for justification. **Alternative** • Two (1D) arrays • One of string, one of integer • Where same index links the name with the score	**3**

Question	Answer	Marks
2(a)	 One mark for: 1 All four boxes correctly labelled and positioned 2 Selection diamond (only on `UpdateLoan` and no iteration arrows) 3 Parameters to and from `LoanExtend()` 4 Parameters to and from `CheckReserve()` 5 Parameters to and from `LoanReturn()`	**5**
2(b)	`PROCEDURE LoanReturn (LoanID, BookID : STRING, BYREF Fine : REAL)` One mark for each underlined part	**2**

Question	Answer	Marks
2(c)	Example solution:	**6**



One mark for each functional group as listed below:

Question	Answer	Marks
2(c)	Explanation of mark points: 1 Initialise `MAX` to first value input 2 Set `Total` to zero 3 Input 49 more values (or 50 values in total) 4 Sum all values input 5 Set new `MAX` when Input value `>` `MAX` **in a loop** 6 Sum all but largest (or subtract `MAX` from total), calculate and output average	

Question	Answer	Marks
3(a)(i)	One mark per bullet point: • Data from the arrays is written to the files at the end of the day / before the program is terminated / computer is switched off • Data can then be read from the files at the start of the next day and written to / stored in the arrays • No need to (re-)enter the data manually // only need to enter data once **Note: Max 2 marks**	**2**
3(a)(ii)	• The data is retained when the program is terminated / after the computer is switched off // data is stored permanently // non-volatile storage	**1**
3(a)(iii)	One mark per bullet point: • Data items are combined to form a single string / saved as a single line in the file • Data items are separated by a special character // make each data item a fixed length ALTERNATIVE: • Convert all data items / 'number of people' to strings • Consecutive / each line stores a separate data item	**2**

Question	Answer	Marks
3(b)	``` PROCEDURE Preview (ThisFile : STRING) DECLARE LineNum : INTEGER DECLARE ThisLine : STRING OPENFILE ThisFile FOR READ IF EOF(ThisFile) THEN OUTPUT "Warning Message" ELSE LineNum ← 1 WHILE LineNum < 6 AND NOT EOF(ThisFile) READFILE Thisfile, ThisLine OUTPUT ThisLine LineNum ← LineNum + 1 ENDWHILE ENDIF CLOSEFILE ThisFile ENDPROCEDURE ``` Marks as follows: 1 Procedure heading (including parameter) and ending 2 File `OPEN` and subsequently `CLOSE` 3 Check if file is empty **and** output a warning message if it is 4 Conditional Loop 5 Output line (including blank lines) and read next line **in a loop**	**5**

Question	Answer	Marks
4(a)		**5**

Name	Flag	Index	NewName	ThisChar
"∇in∇a∇∇Cup"				
	TRUE	1	""	
				'∇'
			"∇"	
		2		'i'
	FALSE		"∇I"	
		3		'n'
			"∇In"	
		4		'∇'
	TRUE		"∇In∇"	
		5		'a'
	FALSE		"∇In∇A"	
		6		'∇'
	TRUE		"∇In∇A∇"	
		7		'∇'
			"∇In∇A∇∇"	
		8		'C'
	FALSE		"∇In∇A∇∇C"	
		9		'u'
			"∇In∇A∇∇Cu"	
		10		'p'
			"∇In∇A∇∇Cup"	
	FALSE	11	"∇In∇A∇∇Cup"	'p'

Mark as follows:

- One mark for each of columns 2 to 5 (condone missing final 11)
- One mark for final row all correct (including final 11)

Question	Answer	Marks
4(b)	Loop structure: A count-controlled loop Justification: The number of iterations is known One mark per point	**2**
4(c)(i)	A couple of solutions: `24  ThisChar ← LCASE(MID(Name, Index, 1)` ALTERNATIVE: `31  NewName ← NewName & LCASE(ThisChar)` Ignore line number	**1**
4(c)(ii)	One mark for each: Line number: 26 New position: Move to after line 27 / line 28	**2**

Question	Answer	Marks
5	`PROCEDURE Sort()` `  DECLARE Temp : INTEGER` `  DECLARE NoSwaps : BOOLEAN` `  DECLARE Boundary, Row, Col : INTEGER` `  Boundary ← 999` `  REPEAT` `  NoSwaps ← TRUE` `  FOR Row ← 1 TO Boundary` `  IF Result[Row, 2] > Result[Row + 1, 2] THEN` `  FOR Col ← 1 TO 2` `  Temp ← Result [Row, Col]` `  Result [Row, Col] ← Result [Row + 1, Col]` `  Result [Row + 1, Col] ← Temp` `  NEXT Col` `  NoSwaps ← FALSE` `  ENDIF` `  NEXT J` `  Boundary ← Boundary - 1` `  UNTIL NoSwaps = TRUE` `ENDPROCEDURE` Mark as follows: 1 Outer loop 2 Inner loop 3 Correct comparison **in a loop** 4 Correct swap of col1 array elements **in a loop** 5 Correct swap of col2 array elements **in a loop** (via loop or separate statements) 6 `'NoSwap'` mechanism: Conditional outer loop including flag reset 7 `'NoSwap'` mechanism: Set flag in inner loop to indicate swap 8 Reducing Boundary **in the <u>outer</u> loop**	**8**

Question	Answer	Marks
6(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 start pointer 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**

Question	Answer	Marks
6(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**

Question	Answer	Marks
7(a)	``` FUNCTION GetStart (WordNum : INTEGER) RETURNS INTEGER DECLARE Index, ThisPos, NumFound : INTEGER DECLARE ThisChar : Char CONSTANT SPACECHAR = ' ' Index ← -1 Numfound ← 0 ThisPos ← 1 IF WordNum = 1 THEN // if looking for word 1... Index ← 1 // Word 1 always starts at index // position 1 ELSE // Otherwise start counting spaces... WHILE ThisPos <= LENGTH(FNString) AND Index = -1 ThisChar ← MID(FNString, ThisPos, 1) IF ThisChar = SPACECHAR THEN NumFound ← NumFound + 1 IF NumFound = WordNum - 1 THEN Index ← ThisPos + 1 // the start of the // required word ENDIF ENDIF ThisPos ← ThisPos + 1 ENDWHILE ENDIF RETURN Index ENDFUNCTION ```	**7**

1 mark for each of the following:

1 Function heading, including return type and function end
2 Loop counting spaces until word found or end of `FNString`
3 extract a character from `FNString` **in a loop**
4 compare with `SPACECHAR` **and** increment count if equal **in a loop**
5 compare count with `WordNum - 1` (depending on initialisation value) **in a loop**
6 if equal then set flag or `Index` to `ThisPos + 1` **in a loop**
7 Return Index (correctly in all cases / following a reasonable attempt)
8 Works for special case when looking for word 1

Note: Max 7 marks

Question	Answer	Marks
7(b)	Marks awarded for any reference to each of the following steps of the algorithm: 1 Mention of variable for use as array index 2 Use of a loop (to check through the array) 3 If word is the same as the current array element then return `FALSE` / set flag 4 If word not already in array, loop to find unused element *(second loop)* 5 Store word in unused element and return `TRUE`, otherwise return `FALSE` VARIATION: 1 Mention of variable for use as array index 2 Use of a loop (to check through the array) 3 Save index of (first) unused element found 4 If word is the same as the current array element then return `FALSE` / set flag 5 If word not already in array **and** unused element available, store word in unused element and return `TRUE` otherwise return `FALSE` **Note: Max 4 marks**	**4**

Question	Answer	Marks
7(c)	`FUNCTION GetWord (Index : INTEGER) RETURNS STRING` `  DECLARE NextWord : STRING` `  DECLARE Done : BOOLEAN` `  DECLARE ThisChar : CHAR` `  DECLARE Index : INTEGER` `  CONSTANT SPACECHAR = ' '` `  NextWord ← ""` `  Done ← FALSE` `  REPEAT` `    ThisChar ← MID(FNString, Index, 1)` `    IF ThisChar <> SPACECHAR THEN` `      NextWord ← NextWord & ThisChar // build up NextWord` `    ENDIF` `    IF ThisChar = SPACECHAR OR Index = LENGTH(FNString) THEN` `        Done ← TRUE` `    ENDIF` `    Index ← Index + 1` `  UNTIL Done = TRUE` `RETURN NextWord` `ENDFUNCTION` 1 mark for each of the following: 1 Conditional loop 2 Extract char from `FNString` and compare with `SPACECHAR` **in a loop** 3 Concatenate with `NextWord` if not `SPACECHAR` **in a loop** 4 Exit loop when `SPACECHAR` encountered **or** when end of `FNString` reached 5 Return `NextWord` (after reasonable attempt at forming, and must have been initialised)	**5**

Question	Answer	Marks
7(c)	The 'length and substring' solution: `FUNCTION GetWord (Index : INTEGER) RETURNS STRING` `  DECLARE Done : BOOLEAN` `  DECLARE ThisChar : CHAR` `  DECLARE Index, NextPos : INTEGER` `  CONSTANT SPACECHAR = ' '` `  Done ← FALSE` `  NextPos ← Index // must be at least one character in` `  // the required word` `  REPEAT` `  ThisChar ← MID(FNString, NextPos, 1)` `  IF ThisChar = SPACECHAR OR NextPos =` `  LENGTH(FNString) THEN` `  Done ← TRUE` `  ELSE` `  NextPos ← NextPos + 1` `  ENDIF` `  UNTIL Done = TRUE` `  IF NextPos = LENGTH(FNString) THEN` `  NextPos ← NextPos - 1  // special case when last word` `  ENDIF` `  RETURN MID(FNString, Index, NextPos - Index)` `ENDFUNCTION` 1 mark for each of the following: 1 Conditional loop 2 ...extract char from `FNString` and compare with `SPACECHAR` **in a loop** 3 .. increment count if word continues 4 Exit loop when `SPACECHAR` encountered **or** when end of `FNString` reached 5 Apply substring function and Return	