
COMPUTER SCIENCE**9608/21**

Paper 2 Written Paper

October/November 2019

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 2019 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **15** printed pages.

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.

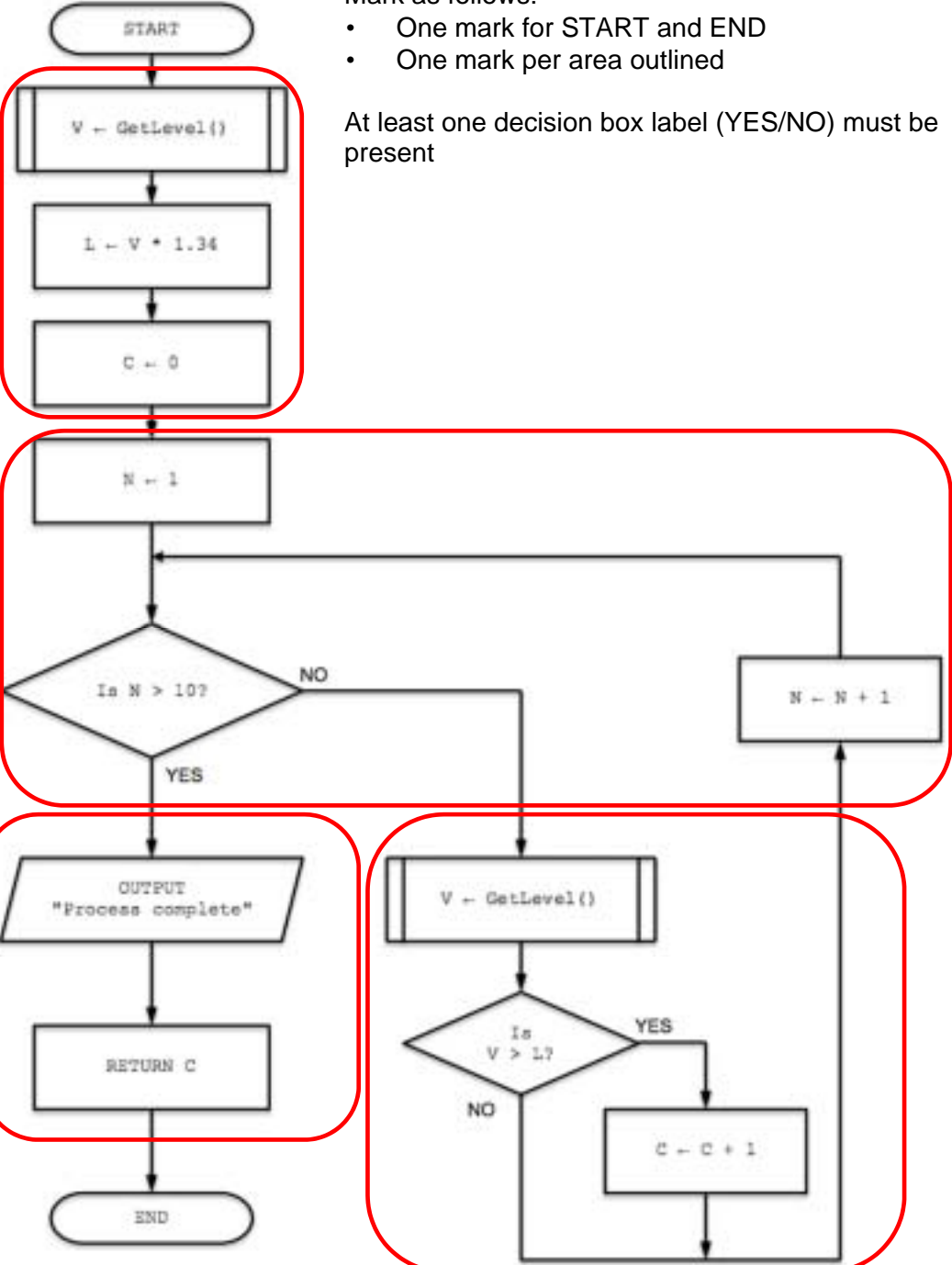
GENERIC MARKING PRINCIPLE 5:

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.

Question	Answer	Marks
1(a)(i)	One mark for each feature: 1. meaningful / sensible identifier names // use of Camel case for identifier names // use of constants 2. blank lines / white space 3. comments	3

Question	Answer	Marks
1(a)(ii)	<p>Mark as follows:</p> <ul style="list-style-type: none"> • One mark for START and END • One mark per area outlined <p>At least one decision box label (YES/NO) must be present</p>  <pre> graph TD Start([START]) --> GetLevel1[V ← GetLevel()] GetLevel1 --> CalcL[L ← V * 1.34] CalcL --> SetC[C ← 0] SetC --> SetN[N ← 1] SetN --> DecN{Is N > 10?} DecN -- YES --> Output[/OUTPUT "Process complete"/] Output --> ReturnC[RETURN C] ReturnC --> End([END]) DecN -- NO --> GetLevel2[V ← GetLevel()] GetLevel2 --> DecV{Is V > L?} DecV -- YES --> CalcC[C ← C + 1] CalcC --> DecN DecV -- NO --> DecN DecN --> CalcN[N ← N + 1] CalcN --> DecN </pre>	5

Question	Answer	Marks										
1(b)(i)	One mark per row <table border="1" data-bbox="279 304 1134 633"> <thead> <tr> <th data-bbox="279 304 772 369">Example value</th> <th data-bbox="772 304 1134 369">Data type</th> </tr> </thead> <tbody> <tr> <td data-bbox="279 369 772 434">"NOT TRUE"</td> <td data-bbox="772 369 1134 434">STRING</td> </tr> <tr> <td data-bbox="279 434 772 499">–4.5</td> <td data-bbox="772 434 1134 499">REAL</td> </tr> <tr> <td data-bbox="279 499 772 564">NOT FALSE</td> <td data-bbox="772 499 1134 564">BOOLEAN</td> </tr> <tr> <td data-bbox="279 564 772 629">132</td> <td data-bbox="772 564 1134 629">INTEGER</td> </tr> </tbody> </table>	Example value	Data type	"NOT TRUE"	STRING	–4.5	REAL	NOT FALSE	BOOLEAN	132	INTEGER	4
Example value	Data type											
"NOT TRUE"	STRING											
–4.5	REAL											
NOT FALSE	BOOLEAN											
132	INTEGER											
1(b)(ii)	One mark per row <table border="1" data-bbox="279 730 1347 1055"> <thead> <tr> <th data-bbox="279 730 970 795">Expression</th> <th data-bbox="970 730 1347 795">Evaluates to</th> </tr> </thead> <tbody> <tr> <td data-bbox="279 795 970 860">LEFT("Start", 3) & RIGHT("Apple", 3)</td> <td data-bbox="970 795 1347 860">"Staple"</td> </tr> <tr> <td data-bbox="279 860 970 925">MID("sample", 3, 5)</td> <td data-bbox="970 860 1347 925">ERROR</td> </tr> <tr> <td data-bbox="279 925 970 990">NUM_TO_STRING(12.3 * 2)</td> <td data-bbox="970 925 1347 990">"24.6"</td> </tr> <tr> <td data-bbox="279 990 970 1055">INT(STRING_TO_NUM("53.4")) + 7</td> <td data-bbox="970 990 1347 1055">60</td> </tr> </tbody> </table>	Expression	Evaluates to	LEFT("Start", 3) & RIGHT("Apple", 3)	"Staple"	MID("sample", 3, 5)	ERROR	NUM_TO_STRING(12.3 * 2)	"24.6"	INT(STRING_TO_NUM("53.4")) + 7	60	4
Expression	Evaluates to											
LEFT("Start", 3) & RIGHT("Apple", 3)	"Staple"											
MID("sample", 3, 5)	ERROR											
NUM_TO_STRING(12.3 * 2)	"24.6"											
INT(STRING_TO_NUM("53.4")) + 7	60											

Question	Answer	Marks
2(a)	One mark for each feature: 1. Module <u>hierarchy</u> 2. The <u>parameters</u> that are passed (between modules) // the module <u>interface</u> 3. Selection / Decisions (which modules are executed) 4. Iteration / Repetition	4
2(b)	One mark for name and one mark for explanation. Example: <ul style="list-style-type: none"> • PrettyPrint // Colour coding • Colour coding of command words / key words • Expand and collapse code blocks • Allows programmer to focus on a section of code // allows quicker navigation of the code • Auto(matic) indentation • Allows the programmer to clearly see the different code sections / easier to see the code structure Accept suitable alternatives	2
2(c)	One mark for identification, one mark for description: <ul style="list-style-type: none"> • By reference / ref • The <u>address</u> of / <u>pointer</u> to the parameter is passed to the subroutine // if the parameter value is changed in the subroutine this changes the original value 	2
2(d)	One mark per bullet point: <ul style="list-style-type: none"> • Changes made to // Updating // Editing a program / algorithm / data structure / software / system • ...as a result of changes to requirements / specification / legislation / available technology 	2

Question	Answer	Marks	
3	One mark per row:	7	
			Answer
	The number of the line containing a variable being		24 / 26 / 28
	The range of line numbers containing a pre-condition loop		20 – 30
	The number of initialisation statements		3
	The number of the line containing a logical operator		20
	The range of line numbers containing a selection statement		22 – 27 / 32 – 37
	The name of a built-in function		MID / LENGTH
	InString / Index		

Question	Answer	Marks
4(a)	<p>One mark for process name, max 3 for structured English.</p> <p>Process:</p> <ul style="list-style-type: none"> • Stepwise Refinement / Top-down design <p>Structured English:</p> <ul style="list-style-type: none"> • Check that character is between 'A' and 'Z' • Produce unique array index for this character • Increment this array element 	4
4(b)	<pre> DECLARE Index : INTEGER DECLARE Count : INTEGER FOR Count ← 1 TO LENGTH(InString) NextChar ← UCASE(MID(InString, Count, 1)) IF NextChar >= 'A' AND NextChar <= 'Z' THEN Index ← ASC(NextChar) - 64 Result[Index] ← Result[Index] + 1 ENDIF ENDFOR FOR Index ← 1 TO 26 OUTPUT "Letter " & CHR(Index + 64) & " : " & NUM_TO_STRING(Result[Index]) ENDFOR </pre> <p>One mark for each of the following (max 7):</p> <ol style="list-style-type: none"> 1 First loop from 1 to length of InString: 2 Extract each character in turn in a loop 3 Check that character is alphabetic (must cater for lower & upper case) in a loop 4 Obtain array index using ASC() - 64 in a loop 5 Increment element of Result array in a loop 6 Second loop from 1 to 26: 7 Attempt to OUTPUT character A to Z and corresponding count in a loop 8 Fully complete OUTPUT including any necessary type conversion in a loop 	7

Question	Answer	Marks
5(a)	One mark for each point. A valid string must contain: <ul style="list-style-type: none">• At least two // more than one upper case character(s)• At least five // more than four lower case character(s)• More digit characters than 'other' characters	3

Question	Answer	Marks																																																																																										
5(b)(i)	<p data-bbox="277 248 751 282">One mark for each area as outlined:</p> <table border="1" data-bbox="277 315 1350 1283"> <thead> <tr> <th data-bbox="277 315 408 376">Index</th> <th data-bbox="408 315 587 376">NextChar</th> <th data-bbox="587 315 719 376">Upper</th> <th data-bbox="719 315 852 376">Lower</th> <th data-bbox="852 315 1082 376">Digit</th> <th data-bbox="1082 315 1350 376">Other</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>'J'</td> <td>1</td> <td></td> <td></td> <td></td> </tr> <tr> <td>2</td> <td>'i'</td> <td></td> <td>1</td> <td></td> <td></td> </tr> <tr> <td>3</td> <td>'m'</td> <td></td> <td>2</td> <td></td> <td></td> </tr> <tr> <td>4</td> <td>'+'</td> <td></td> <td></td> <td></td> <td>1</td> </tr> <tr> <td>5</td> <td>'S'</td> <td>2</td> <td></td> <td></td> <td></td> </tr> <tr> <td>6</td> <td>'m'</td> <td></td> <td>3</td> <td></td> <td></td> </tr> <tr> <td>7</td> <td>'i'</td> <td></td> <td>4</td> <td></td> <td></td> </tr> <tr> <td>8</td> <td>'t'</td> <td></td> <td>5</td> <td></td> <td></td> </tr> <tr> <td>9</td> <td>'h'</td> <td></td> <td>6</td> <td></td> <td></td> </tr> <tr> <td>10</td> <td>'*'</td> <td></td> <td></td> <td></td> <td>2</td> </tr> <tr> <td>11</td> <td>'9'</td> <td></td> <td></td> <td>1</td> <td></td> </tr> <tr> <td>12</td> <td>'9'</td> <td></td> <td></td> <td>2</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>	Index	NextChar	Upper	Lower	Digit	Other			0	0	0	0	1	'J'	1				2	'i'		1			3	'm'		2			4	'+'				1	5	'S'	2				6	'm'		3			7	'i'		4			8	't'		5			9	'h'		6			10	'*'				2	11	'9'			1		12	'9'			2								5
Index	NextChar	Upper	Lower	Digit	Other																																																																																							
		0	0	0	0																																																																																							
1	'J'	1																																																																																										
2	'i'		1																																																																																									
3	'm'		2																																																																																									
4	'+'				1																																																																																							
5	'S'	2																																																																																										
6	'm'		3																																																																																									
7	'i'		4																																																																																									
8	't'		5																																																																																									
9	'h'		6																																																																																									
10	'*'				2																																																																																							
11	'9'			1																																																																																								
12	'9'			2																																																																																								
5(b)(ii)	<p data-bbox="277 1312 620 1346">One mark per bullet point:</p> <ul data-bbox="277 1384 1267 1487" style="list-style-type: none"> • Returned value is FALSE • Digit - Other is not greater than zero // Number of Digit same as Other 	2																																																																																										

Question	Answer	Marks
6(a)	<p>To retain data when the computer is shut down / turned off // after the program ends</p> <p>Accept equivalent answer.</p>	1
6(b)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <pre> FUNCTION SearchFileNtoZ(AccNum : STRING) RETURNS BOOLEAN DECLARE FileData : STRING DECLARE Found : BOOLEAN CONSTANT SearchFile = "UserListNtoZ.txt" Found ← FALSE OPENFILE SearchFile FOR READ WHILE NOT EOF(SearchFile) AND NOT Found READFILE SearchFile, FileData IF AccNum & '*' = LEFT(FileData, LENGTH(AccNum)+ 1) THEN Found ← TRUE ENDIF ENDWHILE CLOSEFILE SearchFile RETURN Found ENDFUNCTION </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> 1. Function heading and ending, (ignore parameter) and returned BOOLEAN 2. File OPEN UserListNtoZ.txt in READ mode and CLOSE 3. Conditional loop repeating until EOF() or 'Found' 4. Read a line from the file in a loop 5. Compare the correct number of characters with AccNum in a loop 6. Set termination logic if found in a loop 7. Return Boolean value 	7

Question	Answer	Marks
6(c)	<pre> PROCEDURE FindDuplicates() DECLARE Index : INTEGER DECLARE FileData : STRING DECLARE Continue : BOOLEAN DECLARE AccNum : STRING Index ← 1 // assuming array is [1:100] Continue ← TRUE OPENFILE "UserListAtOM.txt" FOR READ WHILE NOT EOF("UserListAtOM.txt") AND Continue = TRUE READFILE "UserListAtOM.txt", FileData IF MID(FileData, 7, 1) = '*' // six character reference THEN AccNum ← LEFT(FileData, 6) ELSE AccNum ← LEFT(FileData, 9) ENDIF IF SearchFileNtoZ(AccNum) = TRUE THEN IF Index = 101 // is the array already full? THEN OUTPUT "Error - Array Full" Continue ← FALSE ELSE Duplicates[Index] ← AccNum Index ← Index + 1 ENDIF ENDIF ENDIF ENDWHILE CLOSEFILE "UserListAtOM.txt" ENDPROCEDURE </pre> <p>One mark for each of the following (max 8):</p> <ol style="list-style-type: none"> 1. Declaration and Initialisation of Index and used to index array Duplicates 2. OPEN file UserListAtOM.txt in READ mode and CLOSE 3. Pre-Condition loop to go through the file until EOF() and early termination if array full 4. Read line from file and extract account number (AccNum) in a loop 5. Call SearchFileNtoZ (with AccNum) following an attempt at MP4 in a loop 6. Check if return value is TRUE and if so: in a loop 7. store AccNum in correct array element 8. increment array index following an attempt at MP7 9. If array overflow OUTPUT error message 	8

Question	Answer	Marks
6(d)(i)	<pre>PROCEDURE ClearArray(BYREF ThisArray : ARRAY, NumElements : INTEGER, InitVal : STRING) DECLARE Index : INTEGER FOR Index ← 1 TO NumElements ThisArray[Index] ← InitVal ENDFOR ENDPROCEDURE</pre> <p>Mark as follows:</p> <ul style="list-style-type: none"> • Procedure header • Loop • Assignment within loop 	3
6(d)(ii)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <pre><u>CALL ClearArray(Duplicates, 100, "Empty")</u></pre> <p>Mark as follows:</p> <ul style="list-style-type: none"> • Procedure call • Parameter list (in brackets) 	2

Program Code Example Solutions**Question 6(b): Visual Basic**

```

Function SearchFileNtoZ(ByVal SearchString As String) As Boolean
    Dim FileData As String
    Dim Found As Boolean

    Found = FALSE

    FileOpen(1, "UserListNtoZ.txt", OpenMode.Input)

    While Not EOF(1) And Not Found

        Filedata = LineInput(1)
        If SearchString & '*' = Left(FileData, Len(SearchString)+1) Then
            Found = TRUE
        End If

    End While

    FileClose(1)

    Return Found

End Function

```

Question 6(b): Pascal

```

function SearchFileNtoZ (SearchString : string): boolean;
var
    FileData : string;
    Found : boolean;
    MyFile : text;

begin
    Found := FALSE;

    assign(MyFile, "UserListNtoZ.txt");
    reset (Myfile);

    while Not EOF(MyFile) And Not Found do
        begin
            readLn(MyFile, FileData);
            if SearchString + '*' = LeftStr(FileData, length(SearchString)+1)
then
                Found := TRUE;

            end;

        close(MyFile);

        result := Found; // SearchFileB := Found;

    end;

```

Question 6(b): Python

```
def SearchFileNtoZ(SearchString):  
    ## FileData : String  
    ## Found : Boolean  
    ## MyFile : Text  
  
    Found = False  
  
    MyFile = open("UserListNtoZ.txt", 'r')  
    FileData = MyFile.readline()  
    while FileData != "" and not Found :  
        if SearchString + '*' == FileData[0: len(SearchString)+1]:  
            Found = True  
  
        FileData = MyFile.readline()  
  
    MyFile.close  
  
    return(Found)
```

Question 6(d)(ii): Visual Basic

Call ClearArray(Duplicates, 100, "Empty") 'Call optional

Question 6(d)(ii): Pascal

ClearArray(Duplicates, 100, 'Empty');

Question 6(d)(ii): Python

ClearArray(Duplicates, 100, "Empty")