
COMPUTER SCIENCE

9608/21

Paper 1 Written Paper

October/November 2018

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 2018 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **12** 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)	<table border="1"> <thead> <tr> <th data-bbox="264 259 746 360">Statement</th> <th data-bbox="746 259 935 360">Selection</th> <th data-bbox="935 259 1125 360">Repetition (Iteration)</th> <th data-bbox="1125 259 1335 360">Assignment</th> </tr> </thead> <tbody> <tr> <td data-bbox="264 360 746 427">WHILE Count < 20</td> <td data-bbox="746 360 935 427"></td> <td data-bbox="935 360 1125 427">✓</td> <td data-bbox="1125 360 1335 427"></td> </tr> <tr> <td data-bbox="264 427 746 495">Count ← Count + 1</td> <td data-bbox="746 427 935 495"></td> <td data-bbox="935 427 1125 495"></td> <td data-bbox="1125 427 1335 495">✓</td> </tr> <tr> <td data-bbox="264 495 746 562">If MyGrade <> 'C' THEN</td> <td data-bbox="746 495 935 562">✓</td> <td data-bbox="935 495 1125 562"></td> <td data-bbox="1125 495 1335 562"></td> </tr> <tr> <td data-bbox="264 562 746 663">Mark[Count] ← GetMark(StudentID)</td> <td data-bbox="746 562 935 663"></td> <td data-bbox="935 562 1125 663"></td> <td data-bbox="1125 562 1335 663">✓</td> </tr> <tr> <td data-bbox="264 663 746 730">ELSE OUTPUT "Fail"</td> <td data-bbox="746 663 935 730">✓</td> <td data-bbox="935 663 1125 730"></td> <td data-bbox="1125 663 1335 730"></td> </tr> <tr> <td data-bbox="264 730 746 797">ENDFOR</td> <td data-bbox="746 730 935 797"></td> <td data-bbox="935 730 1125 797">✓</td> <td data-bbox="1125 730 1335 797"></td> </tr> </tbody> </table> <p data-bbox="264 824 568 857">One mark for each row</p>				Statement	Selection	Repetition (Iteration)	Assignment	WHILE Count < 20		✓		Count ← Count + 1			✓	If MyGrade <> 'C' THEN	✓			Mark[Count] ← GetMark(StudentID)			✓	ELSE OUTPUT "Fail"	✓			ENDFOR		✓		6
Statement	Selection	Repetition (Iteration)	Assignment																														
WHILE Count < 20		✓																															
Count ← Count + 1			✓																														
If MyGrade <> 'C' THEN	✓																																
Mark[Count] ← GetMark(StudentID)			✓																														
ELSE OUTPUT "Fail"	✓																																
ENDFOR		✓																															
1(b)(i)	<table border="1"> <thead> <tr> <th data-bbox="264 904 919 972">Statement</th> <th data-bbox="919 904 1335 972">Data type</th> </tr> </thead> <tbody> <tr> <td data-bbox="264 972 919 1039">MyAverage ← 13.5</td> <td data-bbox="919 972 1335 1039">REAL</td> </tr> <tr> <td data-bbox="264 1039 919 1106">ProjectCompleted ← TRUE</td> <td data-bbox="919 1039 1335 1106">BOOLEAN</td> </tr> <tr> <td data-bbox="264 1106 919 1173">Subject ← "Home Economics"</td> <td data-bbox="919 1106 1335 1173">STRING</td> </tr> <tr> <td data-bbox="264 1173 919 1240">MyMark ← 270</td> <td data-bbox="919 1173 1335 1240">INTEGER</td> </tr> <tr> <td data-bbox="264 1240 919 1308">MyGrade ← 'B'</td> <td data-bbox="919 1240 1335 1308">CHAR</td> </tr> </tbody> </table>		Statement	Data type	MyAverage ← 13.5	REAL	ProjectCompleted ← TRUE	BOOLEAN	Subject ← "Home Economics"	STRING	MyMark ← 270	INTEGER	MyGrade ← 'B'	CHAR		5																	
Statement	Data type																																
MyAverage ← 13.5	REAL																																
ProjectCompleted ← TRUE	BOOLEAN																																
Subject ← "Home Economics"	STRING																																
MyMark ← 270	INTEGER																																
MyGrade ← 'B'	CHAR																																
1(b)(ii)	<table border="1"> <thead> <tr> <th data-bbox="264 1368 919 1435">Expression</th> <th data-bbox="919 1368 1335 1435">Evaluates to</th> </tr> </thead> <tbody> <tr> <td data-bbox="264 1435 919 1503">"Air-" & MID(Subject, 7, 3)</td> <td data-bbox="919 1435 1335 1503">"Air-con"</td> </tr> <tr> <td data-bbox="264 1503 919 1570">INT(MyAverage / 2)</td> <td data-bbox="919 1503 1335 1570">6</td> </tr> <tr> <td data-bbox="264 1570 919 1637">ProjectCompleted AND MyMark > 270</td> <td data-bbox="919 1570 1335 1637">FALSE</td> </tr> <tr> <td data-bbox="264 1637 919 1704">ProjectCompleted OR MyMark > 260</td> <td data-bbox="919 1637 1335 1704">TRUE</td> </tr> <tr> <td data-bbox="264 1704 919 1771">ASC(MyGrade / 3)</td> <td data-bbox="919 1704 1335 1771">ERROR</td> </tr> </tbody> </table>		Expression	Evaluates to	"Air-" & MID(Subject, 7, 3)	"Air-con"	INT(MyAverage / 2)	6	ProjectCompleted AND MyMark > 270	FALSE	ProjectCompleted OR MyMark > 260	TRUE	ASC(MyGrade / 3)	ERROR	5																		
Expression	Evaluates to																																
"Air-" & MID(Subject, 7, 3)	"Air-con"																																
INT(MyAverage / 2)	6																																
ProjectCompleted AND MyMark > 270	FALSE																																
ProjectCompleted OR MyMark > 260	TRUE																																
ASC(MyGrade / 3)	ERROR																																

Question	Answer	Marks
2(a)	<pre> FUNCTION GetDiscountRate(CardNum : STRING) RETURNS REAL DECLARE DRate : REAL DECLARE Points : INTEGER DRate ← 0 Points ← GetPoints(CardNum) IF Points > 199 THEN DRate ← 0.2 ELSE IF Points > 99 THEN DRate ← 0.1 ENDIF ENDIF ENDIF IF Today() = 3 THEN DRate ← DRate * 1.2 ENDIF RETURN DRate ENDFUNCTION </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> 1 Correct FUNCTION heading (as given) and end 2 Declaring local variables for DRate and Points 3 Initialisation of DRate to zero and Points ← GetPoints(CardNum) 4 IF ... THEN ...(ELSE) ... ENDIF with Points > 199 5 (Nested) IF ... THEN ... ENDIF with Points > 99 6 ...correct assignments of DRate to 0.2 and 0.1 7 Checking Today() = 3 and increasing DRate by 20% 8 Return parameter // GetDiscountRate ← DRate <p>Mark points 7 and 8 must not be nested</p>	8
2(b)(i)	<p>Name: Syntax Description: Rules of programming language have not been followed</p> <p>Name: Logic Description: Where the program does not behave as expected / does not give the expected result / an error in the logic of the algorithm</p> <p>1mark for name + 1 mark for corresponding description</p>	2
2(b)(ii)	<p>Name: Stub testing</p> <p>Description: A function could be written for GetPoints() that simply returns a test value or outputs a message (i.e. doesn't do the CardNum lookup)</p>	2

Question	Answer	Marks
2(c)(i)	1 mark for any of the following two values: 0.1 0.2 1.2 99 199 3	1
2(c)(ii)	Example: CONSTANT MinDiscount = 0.1 1 mark for each of the following: <ul style="list-style-type: none"> • meaningful identifier name and corresponding value • correct syntax 	2
2(c)(iii)	1 mark for: <ul style="list-style-type: none"> • The value cannot accidentally get changed // be different in two places • A change to the value requires changing in one place only / don't have to repeatedly write out the same value throughout the program 	2
2(c)(iv)	Tried and tested // pre compiled (contains no syntax errors)	1
2(c)(v)	1 mark for feature (Name) and 1 mark for corresponding description (explanation) Example: Name: Meaningful variable names Explanation: To reduce the risk of referring to the wrong variable / make the code easier to understand Name: Indentation Explanation: To see where loops / selection start / end // indicate program structure Name: Variable type-checking as part of module interface Explanation: Reduces the risk of using an incorrect parameter Name: Pretty-Printing Explanation: Highlights the error / auto-complete / type checking Name: / <u>Dynamic</u> Syntax Checking Explanation: Highlights the error as code is typed in	2

Question	Answer	Marks															
3(a)	Code has to be in machine code (or equivalent) to be executed	1															
3(b)	<p>1 mark for the name (what you do) and one for description (how)</p> <p>For example:</p> <p>Method:</p> <ul style="list-style-type: none"> • Dry run the code // use of white box testing // trace tables • Trace the contents of variables // trace all possible routes through the program <p>Method:</p> <ul style="list-style-type: none"> • Breakpoints • Run the code to a set point to find error <p>Method:</p> <ul style="list-style-type: none"> • Variable watch • Check the contents of variables at specific points in the program <p>Method:</p> <ul style="list-style-type: none"> • Stepping • Execute the code line by line <p>Method:</p> <ul style="list-style-type: none"> • Include OUTPUT statements in the code • to display the value of variables as the code was running 	4															
3(c)	<table border="1" data-bbox="264 1149 1347 1608"> <thead> <tr> <th data-bbox="264 1149 986 1211">Statement</th> <th data-bbox="986 1149 1166 1211">White-box</th> <th data-bbox="1166 1149 1347 1211">Black-box</th> </tr> </thead> <tbody> <tr> <td data-bbox="264 1211 986 1312">The student does not need to know the structure of the code.</td> <td data-bbox="986 1211 1166 1312"></td> <td data-bbox="1166 1211 1347 1312">✓</td> </tr> <tr> <td data-bbox="264 1312 986 1413">The student chooses data to test every possible path through the code.</td> <td data-bbox="986 1312 1166 1413">✓</td> <td data-bbox="1166 1312 1347 1413"></td> </tr> <tr> <td data-bbox="264 1413 986 1514">The student chooses normal, boundary and erroneous data.</td> <td data-bbox="986 1413 1166 1514">✓</td> <td data-bbox="1166 1413 1347 1514">(✓)</td> </tr> <tr> <td data-bbox="264 1514 986 1608">The student chooses data to test that the program meets the specification.</td> <td data-bbox="986 1514 1166 1608"></td> <td data-bbox="1166 1514 1347 1608">✓</td> </tr> </tbody> </table> <p data-bbox="264 1641 464 1675">1 mark per row</p>	Statement	White-box	Black-box	The student does not need to know the structure of the code.		✓	The student chooses data to test every possible path through the code.	✓		The student chooses normal, boundary and erroneous data.	✓	(✓)	The student chooses data to test that the program meets the specification.		✓	4
Statement	White-box	Black-box															
The student does not need to know the structure of the code.		✓															
The student chooses data to test every possible path through the code.	✓																
The student chooses normal, boundary and erroneous data.	✓	(✓)															
The student chooses data to test that the program meets the specification.		✓															

Question	Answer	Marks								
4(a)(i)	<table border="1"> <tr> <td data-bbox="264 259 1043 327">The identifier name of a global integer referenced</td> <td data-bbox="1043 259 1347 327">NumElements</td> </tr> <tr> <td data-bbox="264 327 1043 394">The identifier name of a user-defined procedure</td> <td data-bbox="1043 327 1347 394">SaveToFile</td> </tr> <tr> <td data-bbox="264 394 1043 461">The line number of an unnecessary statement</td> <td data-bbox="1043 394 1347 461">16</td> </tr> <tr> <td data-bbox="264 461 1043 528">The scope of <code>ArrayString</code></td> <td data-bbox="1043 461 1347 528">Local</td> </tr> </table>	The identifier name of a global integer referenced	NumElements	The identifier name of a user-defined procedure	SaveToFile	The line number of an unnecessary statement	16	The scope of <code>ArrayString</code>	Local	4
The identifier name of a global integer referenced	NumElements									
The identifier name of a user-defined procedure	SaveToFile									
The line number of an unnecessary statement	16									
The scope of <code>ArrayString</code>	Local									
4(a)(ii)	<p>1 mark for each mark point:</p> <ul style="list-style-type: none"> • Loop / repeat / iterate through array <code>ResultArray</code> one element at a time • extract a string from <u>row / column 1</u> of the array • compare the string with <code>SearchString</code> • if they match, call <code>SaveToFile()</code> and increment <code>NumberFound</code> 	4								
4(b)	<p>Pseudocode solution included here for development and clarification of mark scheme. Programming language solutions appear at the end of this mark scheme.</p> <pre> FUNCTION ScanArray(SearchString : STRING) RETURNS INTEGER DECLARE ArrayIndex : INTEGER DECLARE ArrayString : STRING DECLARE NumberFound : INTEGER NumberFound ← 0 FOR ArrayIndex ← 1 TO NumElements ArrayString ← ResultArray[ArrayIndex, 1] IF TO_UPPER(ArrayString) = TO_UPPER(SearchString) THEN CALL SaveToFile(ArrayString) NumberFound ← NumberFound + 1 ENDIF ENDFOR RETURN NumberFound ENDFUNCTION </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> 1 Function header and end including parameter and return 2 Declaration of two local variables as above but NOT <code>NumElements</code> 3 FOR ... ENDFOR loop with range as given 4 Referencing each element from the array 5 Converting both strings to uppercase / lowercase 6 If strings are equal then Call <code>SaveToFile()</code> and increment <code>NumberFound</code> 	6								

Question	Answer	Marks
4(c)	<p>1 mark for name; 1 mark for each advantage (max 2)</p> <p>Name: Stepwise refinement // Top-down design // Modularisation // Decomposition</p> <p>Advantage:</p> <ul style="list-style-type: none"> • Makes the problem / task / algorithm easier to understand // reduce program complexity • Smaller modules easier to develop / test / debug • Programmers can work on different modules // different expertise 	3
4(d)	<p>Pseudocode solution included here for development and clarification of mark scheme. Programming language solutions appear at the end of this mark scheme.</p> <pre> DECLARE ResultArray : ARRAY [1:100, 1:2] OF STRING DECLARE i, j : INTEGER FOR i ← 1 to 100 FOR j ← 1 to 2 ResultArray[i, j] ← '*' ENDFOR ENDFOR </pre> <p>One mark for:</p> <ul style="list-style-type: none"> • <code>ResultArray</code> declaration / commented in Python • assigning to all elements • assignment of <code>'*'</code> 	3

Question	Answer	Marks
5	<pre> FUNCTION SaveStatus() RETURNS BOOLEAN DECLARE Time : STRING DECLARE Fuel : STRING DECLARE Distance : STRING DECLARE FileData : STRING DECLARE Tries : INTEGER DECLARE ReturnFlag : BOOLEAN Tries ← 1 ReturnFlag ← TRUE Distance ← GetDistance() Fuel ← GetFuel() Time ← GetTime() WHILE Time = NULL AND Tries < 3 Time ← GetTime() Tries ← Tries + 1 ENDWHILE IF Time = NULL THEN ReturnFlag ← FALSE ELSE FileData ← Time & ',' & Fuel & ',' & Distance OPENFILE "CarStatus.txt" FOR APPEND WRITEFILE "CarStatus.txt", FileData CLOSEFILE "CarStatus.txt" ENDIF RETURN ReturnFlag ENDFUNCTION </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> 1 Function heading as shown 2 Declare Time local variable as STRING 3 Calls GetDistance() and GetFuel() once 4 Loop (up to three times or) until Time <> NULL 5 Call GetTime() in a loop 6 Return FALSE if 3 NULLS 7 Open file in APPEND mode 8 Forming the text string with comma separators and write to the file 9 OPEN ... WRITE ... CLOSE as three lines not separated by loop 10 Return TRUE 	10

Program Code Solutions

Q4 (b): Visual Basic

```
Function ScanArray(SearchString As String) As Integer

    Dim ArrayIndex As Integer
    Dim ArrayString As String
    Dim NumberFound As Integer

    NumberFound = 0

    For ArrayIndex = 1 To NumElements
        ArrayString = ResultArray(ArrayIndex, 1)
        If UCase(ArrayString) = UCase(SearchString) Then
            Call SaveToFile(ArrayString)
            NumberFound = NumberFound + 1
        End If
    Next ArrayIndex
    Return NumberFound

End Function
```

Q4 (b): Pascal

```
function ScanArray(SearchString : String) : Integer;

var
    ArrayIndex : Integer;
    ArrayString : String;
    NumberFound : Integer;

begin
    NumberFound := 0;

    For ArrayIndex := 1 To NumElements do
        begin
            ArrayString := ResultArray[ArrayIndex, 1];
            If ToUpper(ArrayString) = ToUpper(SearchString) then
                begin
                    SaveToFile(ArrayString); // Keyword "Call" not valid
                    NumberFound := NumberFound + 1;
                end;
            end;
        end;

    Result := NumberFound; // ScanArray := NumberFound
end.
```

Q4 (b): Python

```
def ScanArray(SearchString):  
  
    # ArrayIndex : integer  
    # ArrayString : string  
    # NumberFound : integer  
  
    NumberFound = 0  
  
    for ArrayIndex in range(NumElements): # 0 to NumElements-1  
        ArrayString = ResultArray[ArrayIndex][0]  
        if ArrayString.upper == SearchString.upper:  
            SaveToFile(ArrayString) # Keyword "Call" not valid  
            NumberFound = NumberFound + 1  
  
    return NumberFound # ScanArray := NumberFound
```

Q4 (d): Visual Basic

```
Dim ResultArray(100, 2) As String
Dim I, j As Integer

For i = 1 to 100
    For j = 1 to 2
        ResultArray(i, j) = '*'
    Next j
Next i
```

Q4 (d): Pascal

```
var
  ResultArray : array[1..100, 1..2] of string;
  i, j : integer;
begin
  For i := 1 to 100 do
    For j := 1 to 2 do
      begin
        ResultArray[i, j] := '*';
      end;
    end.
end.
```

Q4 (d): Python

```
# ResultArray[1..100, 1..2] : String

ResultArray = [['0'] * 2 for i in range(100)]

for i in range(100):
    for j in range(2):
        ResultArray[i][j] = '*'
```

Q4 (d): Python – alternative 1 of n

```
# ResultArray[1..100, 1..2] : String

ResultArray = [['*'] * 2 for i in range(100)]
```

Q4 (d): Python – alternative 2 of n

```
# ResultArray[1..100, 1..2] : String

ResultArray = [['*'] * 2] * 100
```