

---

**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)	Statement	Selection	Repetition (Iteration)	Assignment	6
	WHILE Count < 20		✓		
	Count ← Count + 1			✓	
	If MyGrade <> 'C' THEN	✓			
	Mark[Count] ← GetMark(StudentID)			✓	
	ELSE OUTPUT "Fail"	✓			
	ENDFOR		✓		
	One mark for each row				
1(b)(i)	Statement		Data type		5
	MyAverage ← 13.5		REAL		
	ProjectCompleted ← TRUE		BOOLEAN		
	Subject ← "Home Economics"		STRING		
	MyMark ← 270		INTEGER		
	MyGrade ← 'B'		CHAR		
1(b)(ii)	Expression		Evaluates to		5
	"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 &gt; 199     THEN       DRate ← 0.2     ELSE       IF Points &gt; 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"> <li>1 Correct FUNCTION heading (as given) and end</li> <li>2 Declaring local variables for DRate and Points</li> <li>3 Initialisation of DRate to zero and Points ← GetPoints(CardNum)</li> <li>4 IF ... THEN ...(ELSE) ... ENDIF with Points &gt; 199</li> <li>5 (Nested) IF ... THEN ... ENDIF with Points &gt; 99</li> <li>6 ...correct assignments of DRate to 0.2 and 0.1</li> <li>7 Checking Today() = 3 and increasing DRate by 20%</li> <li>8 Return parameter // GetDiscountRate ← DRate</li> </ol> <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 <b>two</b> 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"> <li>meaningful identifier name <b>and</b> corresponding value</li> <li>correct syntax</li> </ul>	2
2(c)(iii)	1 mark for:  <ul style="list-style-type: none"> <li>The value cannot accidentally get changed // be different in two places</li> <li>A change to the value requires changing in one place only / don't have to repeatedly write out the same value throughout the program</li> </ul>	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"> <li>• Dry run the code // use of white box testing // trace tables</li> <li>• Trace the contents of variables // trace all possible routes through the program</li> </ul> <p>Method:</p> <ul style="list-style-type: none"> <li>• Breakpoints</li> <li>• Run the code to a set point to find error</li> </ul> <p>Method:</p> <ul style="list-style-type: none"> <li>• Variable watch</li> <li>• Check the contents of variables at specific points in the program</li> </ul> <p>Method:</p> <ul style="list-style-type: none"> <li>• Stepping</li> <li>• Execute the code line by line</li> </ul> <p>Method:</p> <ul style="list-style-type: none"> <li>• Include OUTPUT statements in the code</li> <li>• to display the value of variables as the code was running</li> </ul>	4															
3(c)	<table border="1"> <thead> <tr> <th>Statement</th><th>White-box</th><th>Black-box</th></tr> </thead> <tbody> <tr> <td>The student does not need to know the structure of the code.</td><td></td><td>✓</td></tr> <tr> <td>The student chooses data to test every possible path through the code.</td><td>✓</td><td></td></tr> <tr> <td>The student chooses normal, boundary and erroneous data.</td><td>✓</td><td>(✓)</td></tr> <tr> <td>The student chooses data to test that the program meets the specification.</td><td></td><td>✓</td></tr> </tbody> </table> <p>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><tr><td>The identifier name of a global integer referenced</td><td>NumElements</td></tr><tr><td>The identifier name of a user-defined procedure</td><td>SaveToFile</td></tr><tr><td>The line number of an unnecessary statement</td><td>16</td></tr><tr><td>The scope of <code>ArrayString</code></td><td>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"><li>• Loop / repeat / iterate through array <code>ResultArray</code> one element at a time</li><li>• extract a string from <u>row / column 1</u> of the array</li><li>• compare the string with <code>SearchString</code></li><li>• if they match, call <code>SaveToFile()</code> <b>and</b> increment <code>NumberFound</code></li></ul>	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"><li>1 Function header and end including parameter <b>and</b> return</li><li>2 Declaration of <b>two</b> local variables as above but <b>NOT</b> <code>NumElements</code></li><li>3 <code>FOR ... ENDFOR</code> loop with range as given</li><li>4 Referencing each element from the array</li><li>5 Converting both strings to uppercase / lowercase</li><li>6 If strings are equal then Call <code>SaveToFile()</code> <b>and</b> increment <code>NumberFound</code></li></ol>	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"> <li>• Makes the problem / task / algorithm easier to understand // reduce program complexity</li> <li>• Smaller modules easier to develop / test / debug</li> <li>• Programmers can work on different modules // different expertise</li> </ul>	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"> <li>• ResultArray declaration / commented in Python</li> <li>• assigning to all elements</li> <li>• assignment of '*'</li> </ul>	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 &lt; 3         Time ← GetTime()         Tries ← Tries + 1     ENDWHILE      IF Time = NULL         THEN             ReturnFlag ← FALSE         ELSE             FileData ← Time &amp; ',' &amp; Fuel &amp; ',' &amp; 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"> <li>Function heading as shown</li> <li>Declare Time local variable as STRING</li> <li>Calls GetDistance() and GetFuel() <b>once</b></li> <li>Loop (up to three times or) until Time &lt;&gt; NULL</li> <li>Call GetTime() <b>in a loop</b></li> <li>Return FALSE if 3 NULLS</li> <li>Open file in APPEND mode</li> <li>Forming the text string with comma separators <b>and</b> write to the file</li> <li>OPEN ... WRITE ... CLOSE as three lines not separated by loop</li> <li>Return TRUE</li> </ol>	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

```