Cambridge International AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

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.

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)(i)</td><td>
<table>
<tr><th>Statement</th><th>Selection</th><th>Repetition (Iteration)</th><th>Assignment</th></tr>
<tr><td>WHILE Count < 20</td><td></td><td>✓</td><td></td></tr>
<tr><td>Count ← Count + 1</td><td></td><td></td><td>✓</td></tr>
<tr><td>If MyGrade <> 'C' THEN</td><td>✓</td><td></td><td></td></tr>
<tr><td>Mark[Count] ← GetMark(StudentID)</td><td></td><td></td><td>✓</td></tr>
<tr><td>ELSE OUTPUT "Fail"</td><td>✓</td><td></td><td></td></tr>
<tr><td>ENDFOR</td><td></td><td>✓</td><td></td></tr>
</table>
One mark for each row
</td><td>6</td></tr>
<tr><td>1(b)(i)</td><td>
<table>
<tr><th>Statement</th><th>Data type</th></tr>
<tr><td>MyAverage ← 13.5</td><td>REAL</td></tr>
<tr><td>ProjectCompleted ← TRUE</td><td>BOOLEAN</td></tr>
<tr><td>Subject ← "Home Economics"</td><td>STRING</td></tr>
<tr><td>MyMark ← 270</td><td>INTEGER</td></tr>
<tr><td>MyGrade ← 'B'</td><td>CHAR</td></tr>
</table>
</td><td>5</td></tr>
<tr><td>1(b)(ii)</td><td>
<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td>"Air-" & MID(Subject, 7, 3)</td><td>"Air-con"</td></tr>
<tr><td>INT(MyAverage / 2)</td><td>6</td></tr>
<tr><td>ProjectCompleted AND MyMark > 270</td><td>FALSE</td></tr>
<tr><td>ProjectCompleted OR MyMark > 260</td><td>TRUE</td></tr>
<tr><td>ASC(MyGrade / 3)</td><td>ERROR</td></tr>
</table>
</td><td>5</td></tr>
</table>

Question	Answer	Marks

2(a)	`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`
	
	`  IF Today() = 3`
	`  THEN`
	`  DRate ← DRate * 1.2`
	`  ENDIF`
	
	`  RETURN DRate`
	
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	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`
	
	Mark points 7 and 8 must not be nested
		8

2(b)(i)	Name: Syntax Description: Rules of programming language have not been followed 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 1mark for name + 1 mark for corresponding description	**2**
2(b)(ii)	Name: Stub testing 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)	**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: • meaningful identifier name **and** corresponding value • correct syntax	**2**
2(c)(iii)	1 mark for: • 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: / Dynamic Syntax Checking Explanation: Highlights the error as code is typed in	**2**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>3(a)</td><td>Code has to be in machine code (or equivalent) to be executed</td><td>1</td></tr>
<tr><td>3(b)</td><td>1 mark for the name (what you do) and one for description (how)

For example:

Method:
• Dry run the code // use of white box testing // trace tables
• Trace the contents of variables // trace all possible routes through the program

Method:
• Breakpoints
• Run the code to a set point to find error

Method:
• Variable watch
• Check the contents of variables at specific points in the program

Method:
• Stepping
• Execute the code line by line

Method:
• Include OUTPUT statements in the code
• to display the value of variables as the code was running</td><td>4</td></tr>
<tr><td>3(c)</td><td><table><tr><th>Statement</th><th>White-box</th><th>Black-box</th></tr><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></table>
1 mark per row</td><td>4</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(a)(i)</td><td>
<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 ArrayString</td><td>Local</td></tr>
</table>
</td><td>4</td></tr>
<tr><td>4(a)(ii)</td><td>1 mark for each mark point:

• Loop / repeat / iterate through array ResultArray one element at a time
• extract a string from row / column 1 of the array
• compare the string with SearchString
• if they match, call SaveToFile() and increment NumberFound</td><td>4</td></tr>
<tr><td>4(b)</td><td>Pseudocode solution included here for development and clarification of mark scheme.
Programming language solutions appear at the end of this mark scheme.

<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>
1 mark for each of the following:

1 Function header and end including parameter and return
2 Declaration of two local variables as above but NOT NumElements
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 SaveToFile() and increment NumberFound</td><td>6</td></tr>
</table>

Question	Answer	Marks
4(c)	1 mark for name; 1 mark for each advantage (max 2) Name: Stepwise refinement // Top-down design // Modularisation // Decomposition Advantage: • 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)	Pseudocode solution included here for development and clarification of mark scheme.
	Programming language solutions appear at the end of this mark scheme.
	
	`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`
	
	One mark for:
	• `ResultArray` declaration / commented in Python
	• assigning to all elements
	• assignment of '*'
		3

Question	Answer	Marks

5	`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`
	
	1 mark for each of the following:
	
	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 `NULL`s
	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;

   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.
```

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
```