# Cambridge IGCSE™ (9–1)

**COMPUTER SCIENCE**        **0984/21**

Paper 2        **May/June 2021**

MARK SCHEME

Maximum Mark: 50

---

**Published**

---

This document consists of **11** 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: <br><br> Marks must be awarded in line with: <br><br> • the specific content of the mark scheme or the generic level descriptors for the question <br> • the specific skills defined in the mark scheme or in the generic level descriptors for the question <br> • the standard of response required by a candidate as exemplified by the standardisation scripts. |
| GENERIC MARKING PRINCIPLE 2: <br><br> Marks awarded are always **whole marks** (not half marks, or other fractions). |
| GENERIC MARKING PRINCIPLE 3: <br><br> Marks must be awarded **positively**: <br><br> • 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 <br> • marks are awarded when candidates clearly demonstrate what they know and can do <br> • marks are not deducted for errors <br> • marks are not deducted for omissions <br> • 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: <br><br> 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.

**Please note the following further points:**

The words in **bold** in the mark scheme are important text that needs to be present, or some notion of it needs to be present. It does not have to be the exact word, but something close to the meaning.

If a word is underlined, this **exact** word must be present.

A single forward slash means this is an alternative word. A double forward slash means that this is an alternative mark point.

Ellipsis (…) on the end of one-mark point and the start of the next means that the candidate **cannot** get the second mark point without being awarded the first one. If a MP has ellipsis at the beginning, but there is no ellipsis on the MP before it, then this is just a follow-on sentence and **can** be awarded **without** the previous mark point.

| Question | Answer | Marks |
|---|---|---|
| | **Section A** | |
| 1(a)(i) | Many correct answers.  They must be meaningful and related to **Task 1.** The names are examples only.<br><br>**One** mark per mark point<br><br>•   Constant     MaxCandidates<br>•   Value        4<br>•   Use           The value of the maximum number of candidates for the election | **3** |
| 1(a)(ii) | Many correct answers.  They must be meaningful and related to **Task 1.** The names are examples only.<br><br>**One** mark per mark point<br>•   Variable     `NumberCandidates`<br>•   Use           Storing the number of candidates in the election (for a tutor group)<br><br>•   Array         `CandidateNames`<br>•   Use           Storing the names of the candidates for the election | **4** |
| 1(b) | **One** mark per mark point **(Max 4)**<br>MP1      Change the value of the MaxCandidates constant/variable to 8<br>MP2      Change the input message to state the maximum number of candidates is 8 …<br>MP3      …how your program changed the input message<br>MP4      Change the loop limit to up to 8 …<br>MP5      …how your program changed the loop limit<br>MP6      Change the validation to allow input up to 8 …<br>MP7      …how your program changed its validation check<br>MP8      Change the array size(s) to ensure sufficient capacity to store up to 8 names …<br>MP9      …how your program changed the array sizes<br>MP10     Change the counters to ensure votes can be counted for up to 8 candidates …<br>MP11     …how your program changed its counters | **4** |

| Question | Answer | Marks |
|---|---|---|
| 1(c) | Any **five** from:<br>MP1     Input with message to enter unique voter number<br>MP2     Validation of (unique) voter number entered e.g. length check/type check/range check<br>MP3     Attempt to check if voter number input is in list of possible voters<br>MP4     Attempt to check if they have already voted<br>MP5     If voter has already voted, message to warn them they can't vote<br>MP6     Attempt at preventing them from voting<br>MP7     Store voter number in a suitable data structure<br><br>**Example answer**<br>`OUTPUT "Please enter your unique voter number"`<br>`INPUT UniqueVoterNumber`<br>`FoundFlag ← False`<br>`AllNumbersChecked ← False`<br>`Counter ← 0`<br>`WHILE FoundFlag = False AND AllNumbersChecked = False`<br>`  IF StudentNumbers[Counter] = ""`<br>`    THEN`<br>`      AllNumbersChecked = True`<br>`      StudentNumbers[Counter] ← UniqueVoterNumber`<br>`    ELSE`<br>`      IF UniqueVoterNumber = StudentNumbers[Counter]`<br>`        THEN`<br>`          FoundFlag = True`<br>`          PRINT "Sorry, you have already voted"`<br>`        ELSE`<br>`          Counter = Counter + 1`<br>`      ENDIF`<br>`  ENDIF`<br>`ENDWHILE`<br>`IF FoundFlag = False`<br>`  THEN`<br>`    OUTPUT "Please enter the code of your chosen candidate"`<br>`    INPUT Vote`<br>`ENDIF` | 5 |

| Question | Answer | Marks |
|---|---|---|
| 1(d) | Explanation of how the program does the following:<br>Any **four** from:<br>MP1    Find out how many votes in total (for all candidates) were cast in the election.<br>MP2    For each candidate<br>MP3    … calculate the percentage of votes<br>MP4    … excluding abstentions.<br>MP5    Display the name of each candidate, the number of votes and the percentage of votes they received with appropriate messages.<br>MP6    Display the number of votes cast and the number of abstentions with appropriate message. | **4** |

| Question | Answer | Marks |
|---|---|---|
| | **Section B** | |

| Question | Answer | Marks |
|---|---|---|
| 2 | **One** mark per correct column | 3 |

| Statement | Validation | Verification | Both |
|---|---|---|---|
| Entering the data twice to check if both entries are the same. | | ✓ | |
| Automatically checking that only numeric data has been entered. | ✓ | | |
| Checking data entered into a computer system before it is stored or processed. | | | ✓ |
| Visually checking that no errors have been introduced during data entry. | | ✓ | |

| Question | Answer | Marks |
|---|---|---|
| 3 | **One** mark per bullet point<br>37<br>• Data type name         Integer<br>• Data type description     (Any) **whole number**<br><br>Cambridge2021<br>• Data type name         String<br>• Data type description     A group of characters/text<br><br>47.86<br>• Data type name         Real<br>• Data type description     (Any real) number that could be a whole number or a fraction | 6 |

| Question | Answer | Marks |
|---|---|---|
| 4(a) | **One** mark per mark point **(Max 3)**<br>MP1    Marks input are **stored in the array** `Score[]`<br>MP2    Marks are checked against a range of boundaries // allow example<br>MP3    … and a **matching grade** is assigned to each mark that has been input<br>MP4    … then **stored in the array** `Grade[]`…<br>MP5    … at the same index as the mark input<br>MP6    The algorithm finishes after 30 marks have been input // allows 30 scores to be entered | **3** |
| 4(b) | **One** mark per mark point **(Max 3)**<br>MP1    Correct loop, including counter if not a FOR loop<br>MP2    Correct output of `Score[]`<br>MP3    Correct output of `Grade[]`<br>MP4    Suitable messages/text in output for both arrays<br><br>Example answers<br><br>```<br>Count ← 0<br>REPEAT<br>  PRINT "Student: ", Count, " Mark: ", Score[Count], " Grade: ",Grade[Count]<br>  Count ← Count + 1<br>UNTIL Count = 30<br><br>Count ← 0<br>WHILE Count < 30 DO<br>  PRINT "Student: ", Count, " Mark: ", Score[Count], " Grade: ",Grade[Count]<br>  Count ← Count + 1<br>ENDWHILE<br><br>FOR Count ← 0 TO 29<br>  PRINT "Student: ", Count, " Mark: ", Score[Count], " Grade: ", Grade[Count]<br>NEXT<br>``` | **3** |

**PUBLISHED**

| Question | Answer | Marks |
|---|---|---|
| 4(c) | Any **three** correct statements **(Max 3)** e.g.<br>MP1   Add an input facility to allow teachers to enter the class size<br>MP2   Add a variable to store the input class size<br>MP3   Use the class size variable as the terminating condition for the loop<br>MP4   Make sure the arrays are sufficiently large to accommodate the largest possible class size | **3** |

| Question | Answer | Marks |
|---|---|---|
| 5(a) | **One** mark for each correct column **(Max 4)** | **4** |

| Value | Diff1 | Diff2 | OUTPUT |
|---|---|---|---|
| 50 | 50 | 0 | Accept: Extreme |
| 75 | 25 | 25 | Accept: Normal |
| 99 | 1 | 49 | Accept: Normal |
| 28 | | | Reject: Abnormal |
| 82 | 18 | 32 | Accept: Normal |
| 150 | | | Reject: Abnormal |
| −1 | | | |
| | | | |
| | | | |
| | | | |

| Question | Answer | Marks |
|---|---|---|
| 5(b) | **One** mark per bullet point **(Max 2)**<br>• To output the type of test data<br>• … by performing a range check //… by checking if numbers are within the range 50 and 100 (inclusive) (or not). | **2** |

| Question | Answer | Marks |
|---|---|---|
| 6(a) | The data in the ID column/field is unique/not repeated in each row/record | **1** |
| 6(b) | 18 | **1** |
| 6(c) |  | **4** |

| Field: | ID | GenreName | Overdue | | |
|---|---|---|---|---|---|
| Table: | GENRE | GENRE | GENRE | | |
| Sort: | | | Descending | | |
| Show: | ☑ | ☑ | ☑ | ☐ | ☐ |
| Criteria: | | | >0 | | |
| or: | | | | | |

**One** mark for the correct fields present and correctly named
**One** mark for correct table name and show box in all columns
**One** mark for correct sorting
**One** mark for correct search criterion