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 May/June 2018 series for most Cambridge IGCSE™, Cambridge International A and AS Level and Cambridge Pre-U 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.

<table>
<thead>
<tr>
<th>GENERIC MARKING PRINCIPLE 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marks must be awarded in line with:</td>
</tr>
<tr>
<td>• the specific content of the mark scheme or the generic level descriptors for the question</td>
</tr>
<tr>
<td>• the specific skills defined in the mark scheme or in the generic level descriptors for the question</td>
</tr>
<tr>
<td>• the standard of response required by a candidate as exemplified by the standardisation scripts.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GENERIC MARKING PRINCIPLE 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marks awarded are always <strong>whole marks</strong> (not half marks, or other fractions).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GENERIC MARKING PRINCIPLE 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marks must be awarded <strong>positively</strong>:</td>
</tr>
<tr>
<td>• 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</td>
</tr>
<tr>
<td>• marks are awarded when candidates clearly demonstrate what they know and can do</td>
</tr>
<tr>
<td>• marks are not deducted for errors</td>
</tr>
<tr>
<td>• marks are not deducted for omissions</td>
</tr>
<tr>
<td>• 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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GENERIC MARKING PRINCIPLE 4:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules must be applied consistently e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GENERIC MARKING PRINCIPLE 5:</th>
</tr>
</thead>
<tbody>
<tr>
<td>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).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GENERIC MARKING PRINCIPLE 6:</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>
### Question 1(a)

<table>
<thead>
<tr>
<th>Description of data item</th>
<th>Suitable identifier name</th>
</tr>
</thead>
<tbody>
<tr>
<td>The temperature inside the greenhouse</td>
<td>GreenhouseTemperature</td>
</tr>
<tr>
<td>The temperature outside the greenhouse</td>
<td>OutsideTemperature</td>
</tr>
<tr>
<td>The greenhouse identification number</td>
<td>GreenhouseID</td>
</tr>
<tr>
<td>The time the temperature was measured</td>
<td>SampleTime</td>
</tr>
</tbody>
</table>

The above are examples only.
Names must be meaningful and unambiguous

### Question 1(b)(i)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Evaluates to</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Fas&quot; &amp; MID(Subject, 6, 3)</td>
<td>&quot;Faster&quot;</td>
</tr>
<tr>
<td>LEFT(Mark, 1)</td>
<td>ERROR</td>
</tr>
<tr>
<td>10 + ASC(Grade)</td>
<td>76</td>
</tr>
<tr>
<td>MOD(AverageMark * 2, 3)</td>
<td>0</td>
</tr>
<tr>
<td>CourseCompleted AND (Mark &gt;= 60)</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

### Question 1(b)(ii)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Subject</td>
<td>STRING</td>
</tr>
<tr>
<td>Grade</td>
<td>CHAR</td>
</tr>
<tr>
<td>CourseCompleted</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>AverageMark</td>
<td>REAL</td>
</tr>
</tbody>
</table>

One mark per answer
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
</table>
| 2(a)(i)  | • Keywords in capitals  
|          | • White Space / blank lines / grouping  
|          | • Comments  
|          | • Sensible function names | Max 2 |
| 2(a)(ii) | • Indentation  
|          | • Meaningful identifier names | 2 |
| 2(b)     |        | 8     |

<table>
<thead>
<tr>
<th>Feature</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>A line number containing an example of an assignment statement</td>
<td>08 / 09 / 13 / 14 / 15 / 21</td>
</tr>
<tr>
<td>A line number containing the start of a 'pre-condition' loop</td>
<td>11</td>
</tr>
<tr>
<td>A line number containing the end of a 'pre-condition' loop</td>
<td>24</td>
</tr>
<tr>
<td>A line number containing the start of a selection statement</td>
<td>17</td>
</tr>
<tr>
<td>The number of parameters of the LEFT() function</td>
<td>2</td>
</tr>
<tr>
<td>The Boolean operator used</td>
<td>OR</td>
</tr>
<tr>
<td>The number of times the function LEFT() is called from within CountDigits() resulting from the call: Result ← CountDigits(&quot;AB27C4&quot;)</td>
<td>6</td>
</tr>
<tr>
<td>The number of local variables</td>
<td>3</td>
</tr>
<tr>
<td>2(c)(i)</td>
<td></td>
</tr>
</tbody>
</table>
| • Mistake: function header returns a CHAR but last line of code returns an INTEGER  
| • Correction: Function should return an INTEGER  // Change line 26 to return c as CHAR | 2      |
Question | Answer | Marks |
--- | --- | --- |
2(c)(ii) | IF (nc >= '0') AND (nc <= '9') THEN c ← c + 1 ENDIF | 4 |

One mark for each of:
- Single IF THEN ENDIF statement (no ELSE)
- Switching OR to AND
- Lower value comparison
- Upper value comparison

ALTERNATIVE: IF NOT ((nc < '0') OR (nc > '9')) THEN c ← c + 1 ENDIF

One mark for each of:
- Single IF THEN ENDIF statement (no ELSE)
- Inverting test using NOT...
- ... correct use of brackets
- ... both comparisons unchanged
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Example Program Flowchart</td>
<td>10</td>
</tr>
</tbody>
</table>

The flowchart describes a program that initializes variables `Count` and `Index`, then iterates through a list of items in `BarWeight`. If an item's weight exceeds the maximum weight, it increments `Count` and updates `Index`. If `Index` reaches 100, the program checks if `Count` exceeds the threshold. If it does, it outputs a message and calls a function `ServiceCheck()`. The flowchart ends with the `END` statement.
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>One mark for each of:</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>1  START and END / STOP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2  Initialising Count to 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3  Initialising Index to 1 or 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4  Decision box comparing BarWeight[Index] &gt; MaxWeight</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5  Decision box comparing Index to 100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6  Decision box comparing Count &gt; Threshold</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7  Correct increment of Index</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8  Correct increment of Count</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9  Output message (concatenation of text and value) if threshold not exceeded</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 Calling ServiceCheck() if Threshold exceeded (without text message)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>4(a)(i)</td>
<td><img src="image" alt="Diagram" /></td>
<td>2</td>
</tr>
</tbody>
</table>

Mark as follows:

- One mark for three arrows all labelled X (with filled circles)
- One mark for diamond symbol
Question | Answer | Marks
---|---|---
4(a)(ii) | Parameter | Information
| A | BasketID |
| B | BasketID, ItemID, Quantity (In any order) |
| C | (In any order) |
| D | BasketID, DeliveryAddress, PaymentDetails (In any order) |
| E | |
| F | |
| G | |

Mark as follows:
- One mark for parameter A
- One mark for parameters B, C & D
- One mark for parameters E, F & G

Question | Answer | Marks
---|---|---
5(a) | One mark for each of:
- Same data type using a single identifier / more efficient coding / less declaration statements needed
- Access of individual elements (using subscript / index)
- Ability to iterate through the data // easier to search / sort the data
- Code easier to understand / maintain / modify
Max 2

5(b) | One mark for each of:
- (Dynamic) syntax checking / Errors are highlighted / underlined
- Type checking
- Parameter checking
- Identification of unused variables
Max 2
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>5(c)</td>
<td>Example ‘Pseudocode’ solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</td>
<td>4</td>
</tr>
</tbody>
</table>

FUNCTION GetNumber() AS STRING

DECLARE Valid : BOOLEAN
DECLARE MemberNumber : INTEGER

REPEAT
    Valid ← TRUE
    OUTPUT "Please input a valid member number"
    INPUT MemberNumber
    IF (MemberNumber > 9999) OR (MemberNumber < 1111) THEN
        Valid ← FALSE
    ENDIF
UNTIL Valid = TRUE

RETURN STR(MemberNumber)
ENDFUNCTION

1 mark for each of the following:

1. Function heading (as above) and ending
2. Conditional loop structure
3. Condition to check valid membership number
4. Returning string value

PROCEDURE OutputLowestScore()
    DECLARE FileData, FileMembershipNumber : STRING
    DECLARE FileScore, LowestScore : INTEGER
    DECLARE LowestScoreDate, MembershipNumber : STRING

    MembershipNumber ← GetNumber()

    OPENFILE "ScoreDetails.txt" FOR READ
    LowestScore ← 100

    WHILE NOT EOF("ScoreDetails.txt")
        READFILE "ScoreDetails.txt", FileData
        FileMembershipNumber ← LEFT(FileData, 4)
        IF FileMembershipNumber = MembershipNumber THEN
            FileScore ← INT(RIGHT(FileData, 2))
            IF FileScore < LowestScore THEN
                LowestScore ← FileScore
                LowestScoreDate ← MID(FileData(5, 6))
            ENDIF
        ENDIF
    ENDWHILE

    OUTPUT ("The lowest score was " & LowestScore & " on ")
    & _
    LowestScoreDate)
    CLOSEFILE("ScoreDetails.txt ")

ENDPROCEDURE

1 mark for each of the following:

1 Declare variables to store LowestScore as INTEGER and FileData as STRING (commented in Python) (variable names may be different)
2 Function call GetNumber() for membership number
3 Initialisation of LowestScore to 99 or above
4 Open file in READ mode
5 Loop until EOF() (in a loop
6 Read a line from the file
7 Use of substring operations to extract at least one data item
8 Compare the membership numbers
9 Convert score to INTEGER
10 Compare and assign (if appropriate) new value to LowestScore
11 Output the lowest score message including lowest score and date (outside the loop)
12 Close the file
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
</table>
| **6(a)** | • lower bound  
• upper bound | **2** |
| **6(b)** | **Example using single temp variable:**  
PROCEDURE Flip()  
//Use of Single temp value  
DECLARE temp : INTEGER  
DECLARE i : INTEGER  //i is the row  
DECLARE j : INTEGER  //j is the column  
FOR i ← 1 TO 5  
    FOR j ← 1 to 4  //swap element 1&8, 2&7, 3&6, 4&5  
        temp ← Picture[i,j]  
        Picture[i,j] ← Picture[i, 9 - j]  
        Picture[i, 9 - j] ← temp  
    ENDFOR  
ENDFOR  
ENDPROCEDURE  
**Alternative Solution – Use of temp array row:**  
PROCEDURE Flip()  
//Use of temproary row (8 elements)  
DECLARE temp : ARRAY[1:8] OF INTEGER  
DECLARE i : INTEGER  //i is the row  
DECLARE j : INTEGER  //j is the column  
FOR i ← 1 to 5  
    FOR j ← 1 to 8  
        temp[j] ← Picture[i, 9 - j]  //temp is row i reversed  
    ENDFOR  
    FOR j ← 1 to 8  
        Picture[i,j] ← temp[j]  //copy temp back to row i  
    ENDFOR  
ENDFOR  
ENDPROCEDURE | **8** |
6(b) Alternative Solution – Use of new array:

PROCEDURE Flip()
  //Flip to New array
  
  DECLARE NewPic : ARRAY[1:5, 1:8] OF INTEGER
  DECLARE i : INTEGER  //i is the row
  DECLARE j : INTEGER  //j is the column
  
  FOR i ← 1 to 5
    FOR j ← 1 to 8
      NewPic[i, 9 - j] ← Picture[i, j]  //NewPic row is Pic row flipped
    ENDFOR
  ENDFOR
END PROCEDURE

1 mark for each of the following (all methods):

1 Correct procedure heading and ending
2 Declaring local variables for loop counter(s)
3 Declaring a temporary storage variable for swap or new duplicate 2D array
4 A nested loop including attempt at flip operation ...
5 ... Correct number of iterations
6 Assign element to temp (single var or temp array) or to new array
7 Selection of correct source element (row, column)
8 Selection of correct destination element (row, column)

*** End of Mark Scheme – program code solutions follow ***
Appendix

Program Code Example Solutions

Q5 (c): Visual Basic

Function GetNumber() As String
    Dim Valid As Boolean
    Dim MemberNumber As Integer
    Do
        Valid = True
        Console.Write("Please enter a valid member number: ")
        MemberNumber = Console.ReadLine()
        If MemberNumber > 9999 Or MemberNumber < 1111 Then
            Valid = False
        End If
    Loop Until Valid = True
    Return MemberNumber.ToString()
End Function

Q5 (c): Pascal

function GetNumber() : string;
var
    Valid : boolean;
    MemberNumber : integer;
begin
    repeat
        Valid := true;
        write('Please enter a valid member number: '); readln(MemberNumber);
        if (MemberNumber > 9999) or (MemberNumber < 1111) then
            Valid := false;
        until Valid = true;
        GetNumber := IntToStr(MemberNumber);
    end;

Q5 (c): Python

def GetNumber() :
    # Valid : boolean
    # MemberNumber : integer
    Valid = False
    while not Valid :
        MemberNumber = int(input("Please enter a valid member number: "))
        if MemberNumber > 9999 or MemberNumber < 1111 :
            Valid = False
    return str(MemberNumber)
Q5 (d): Visual Basic

Sub OutputLowestScore()
    Dim FileData As String
    Dim FileMembershipNumber As String
    Dim FileScore As Integer
    Dim LowestScore As Integer
    Dim LowestScoreDate As String
    Dim MembershipNumber As String
    MembershipNumber = GetNumber()
    FileOpen(1, "ScoreDetails.txt", OpenMode.Input)
    LowestScore = 100
    While Not EOF(1)
        FileData = LineInput(1)
        FileMembershipNumber = Left(FileData, 4)
        If FileMembershipNumber = MembershipNumber Then
            FileScore = Integer.Parse(Right(FileData, 2))
            If FileScore < LowestScore Then
                LowestScore = FileScore
                LowestScoreDate = Mid(FileData, 5, 6)
            End If
        End If
    End While
    Console.WriteLine("The lowest score was " & LowestScore & " on " & LowestScoreDate)
    FileClose(1)
End Sub
Q5 (d): Pascal

procedure OutputLowestScore();
var
  FileData : string;
  FileMembershipNumber : string;
  FileScore : integer;
  LowestScore : integer;
  LowestScoreDate : string;
  MembershipNumber : string;
  Scores : textFile;
begin
  MembershipNumber := GetNumber();
  assignFile(Scores, 'ScoreDetails.txt');
  reset(Scores);
  LowestScore := 100;
  while not eof(Scores) do
  begin
    readln(Scores, FileData);
    FileMembershipNumber := copy(FileData, 1, 4);
    if FileMembershipNumber = MembershipNumber then
      begin
        FileScore := StrToInt(copy(FileData, 11, 2));
        if FileScore < LowestScore then
          begin
            LowestScore := FileScore;
            LowestScoreDate := copy(FileData, 5, 6)
          end;
      end;
  writeln('The lowest score was ', LowestScore, ' on ', LowestScoreDate);
  close(Scores);
end;
Q5 (d): Python

def OutputLowestScore() :
    # FileData : string
    # FileMembershipNumber : string
    # FileScore : integer
    # LowestScore : integer
    # LowestScoreDate : string
    # MembershipNumber : string
    # File : file handle
    MembershipNumber = GetNumber()
    File = open("ScoreDetails.txt", "r")
    LowestScore = 100
    FileData = File.readline()
    while FileData != "":
        FileMembershipNumber = FileData[:4]
        if FileMembershipNumber == MembershipNumber :
            FileScore = int(FileData[10:12])
            if FileScore < LowestScore :
                LowestScore = FileScore
                LowestScoreDate = FileData[4:10]
        FileData = File.readline()
    print("The lowest score was ", LowestScore, " on ", LowestScoreDate)
    File.close()