Students did not sit exam papers in the June 2020 series due to the Covid-19 global pandemic.

This mark scheme is published to support teachers and students and should be read together with the question paper. It shows the requirements of the exam. The answer column of the mark scheme shows the proposed basis on which Examiners would award marks for this exam. Where appropriate, this column also provides the most likely acceptable alternative responses expected from students. Examiners usually review the mark scheme after they have seen student responses and update the mark scheme if appropriate. In the June series, Examiners were unable to consider the acceptability of alternative responses, as there were no student responses to consider.

Mark schemes should usually be read together with the Principal Examiner Report for Teachers. However, because students did not sit exam papers, there is no Principal Examiner Report for Teachers for the June 2020 series.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the June 2020 series for most Cambridge IGCSE™ and Cambridge International A & 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.

<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>
<tr>
<td>Question</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>1(a)</td>
</tr>
<tr>
<td>1(b)</td>
</tr>
<tr>
<td>1(c)</td>
</tr>
<tr>
<td>1(d)</td>
</tr>
<tr>
<td>Question</td>
</tr>
<tr>
<td>----------</td>
</tr>
</tbody>
</table>
| 2(a)     | One mark per bullet point:  
• Parameters passed between modules // the **interface** between modules  
• Module Iteration  
• Module selection  
Max 2 | | 2 |
| 2(b)(i)  | Advantages include:  
• Easier to solve / implement / program the solution as online shopping is a complex task  
• Easier to debug / maintain as each module can be tested separately e.g. test **FillBasket()** first then test **Checkout()**  
• Tasks may be shared among a team of programmer. e.g. **Checkout()** and **Search()** modules could be developed in parallel / by teams with different expertise  
Note: Must include reference to given scenario to achieve all 3 marks - Max 2 if no reference. | | 3 |
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>2(b)(ii)</td>
<td><img src="image.png" alt="Diagram" /></td>
<td>6</td>
</tr>
</tbody>
</table>

One Mark for:
1. Three middle row boxes correctly labelled and connected to `Shop()`
2. Two bottom row boxes correctly labelled and connected to `FillBasket()`
3. Iteration arrow on `FillBasket()`
4. Return parameters from `ChooseSlot()` and `Checkout()`
5. Return parameters from `Search()`
6. Two input parameters to `Add()`

Notes:
Parameter types must be as shown but ignore parameter names (if given)
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
</table>
| 3        | FUNCTION CheckCourse(Course : REAL) RETURNS INTEGER  
DECLARE Adjust, Check : INTEGER  
  
  Check ← INT(Deviate(Course))  
  Adjust ← 255  
  
  CASE OF Check  
  -20 to -1: Adjust ← 10  
  0 : Adjust ← 0  
  1 to 20 : Adjust ← -10  
  OTHERWISE CALL Alert()  
ENDCASE  
  
RETURN Adjust  
ENDFUNCTION  
  
1 mark for each of the following:  
1 FUNCTION heading and ending including parameter as given above  
2 Assign value to Check using integer conversion and initialise Adjust to 255  
3 CASE ... ENDCASE  
4 Conditions −20 to −1 and 1 to 20 (and corresponding assignments)  
5 Condition 0 (and corresponding assignment)  
6 OTHERWISE  
7 Return Adjust |
Question | Answer | Marks
---|---|---
4(a) | DECLARE Random : ARRAY [1:10] OF INTEGER  
DECLARE NextNum, Index, Rnum : INTEGER  
DECLARE Exists : BOOLEAN  
NextNum ← 1 // index position for the next random number  
REPEAT  
    Rnum ← INT(RAND(100)) + 1 // from original question  
    Exists ← FALSE  
    FOR Index ← 1 to NextNum - 1 // search for Rnum  
        IF Random[Index] = Rnum  
            THEN  
                Exists ← TRUE  
            ENDIF  
    ENDFOR  
    IF Exists = FALSE  
        THEN  
            Random[NextNum] ← Rnum // store Rnum  
            NextNum ← NextNum + 1 // increment index  
        ENDIF  
UNTIL NextNum > 10  
1 mark for each of the following:  
1 Conditional (outer) loop to generate 10 values  
2 Inner loop to search array for duplicate number  
3 Check for duplicate by comparing number generated with array element in a loop  
4 Avoid checking uninitialised elements // array initialisation to rogue value at start of algorithm  
5 If Rnum is a duplicate then repeat outer loop  
6 If Rnum not a duplicate then assign to array element and Increment index  
Notes:  
Max 5 if statement to generate random number (as given in Q) not present or incorrectly placed. | 6

4(b) | Adaptive Maintenance | 1
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
</table>
| 5(a)     | Mark as follows:  
1  SET Name to ""  
2  SET Index to 1  
3  SELECT the character from input parameter string at Index position  
4  IF character is not colon then concatenate character with Name  
5  …INCREMENT Index  
6  …REPEAT from step 3  
7  RETURN Name  

Alternative Solution:  
Mark as follows:  
1  SET Index to 1  
2  SELECT the character from input parameter string at Index position  
3  IF character is colon then go to 5  
4  Else INCREMENT Index and repeat from 2  
5  Extract a substring from the left of the parameter string (and assign this to variable Name)  
6  …Using Index -1 for the length  
7  RETURN Name  

Note:  
Mark points may be combined for equivalent marks  
e.g a suitable **structured English description** of the pseudocode statement below satisfies MP 5, 6 and 7:  

```
RETURN LEFT(ParamString, Index - 1)
```

| 5(b)(i) | Description:  
- Reduce the number of items to be checked by one after each pass  
- Use a flag variable to stop the outer loop  
- ... after no more swaps made on a single pass of the inner loop  
- ... resetting before the inner loop starts, and setting it whenever a swap is made | 4 |
### Question 5(b)(ii)

'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.

```
PROCEDURE BubbleSort()
    DECLARE Temp : STRING
    DECLARE NoSwaps : BOOLEAN
    DECLARE Boundary, J : INTEGER

    Boundary ← 999
    REPEAT
        NoSwaps ← TRUE
        FOR J ← 1 TO Boundary
            IF Contact[J] > Contact[J+1]
                THEN
                    Temp ← Contact[J]
                    Contact[J] ← Contact[J+1]
                    Contact[J+1] ← Temp
                    NoSwaps ← FALSE
                ENDIF
        ENDFOR
        Boundary ← Boundary - 1
    UNTIL NoSwaps = TRUE

ENDPROCEDURE
```

Mark as follows:

1. Procedure heading and ending
2. Outer loop
3. Inner loop
4. Correct comparison **in a loop**
5. Correct swap of array elements **in a loop**
6. 'NoSwap' mechanism: Post-conditional outer loop including flag reset
7. 'NoSwap' mechanism: Set flag in inner loop to indicate swap
8. Reducing Boundary **in the outer loop**

**Marks:** 8
Question | Answer | Marks
--- | --- | ---
6(a)(i) | FUNCTION AddTime(StartTime : STRING, Duration : INTEGER) _
| DECLARE NewTime : STRING
| DECLARE StartMinutes, StartHours : INTEGER
| DECLARE Total, NewMinutes, NewHours : INTEGER
| StartHours ← STRING_TO_NUM(LEFT(StartTime,2))
| StartMinutes ← STRING_TO_NUM(RIGHT(StartTime, 2))
| Total ← (StartHours * 60) + StartMinutes + Duration
| NewHours ← DIV(Total, 60)
| NewMinutes ← MOD(Total, 60)
| NewTime ← ""
| IF NewHours < 10
| THEN
| NewTime ← '0' // add leading zero to hours
| ENDIF
| NewTime ← NewTime & NUM_TO_STRING(NewHours) & ':'
| IF NewMinutes < 10
| THEN
| NewTime ← NewTime & '0'// add leading zero
| ENDIF
| NewTime ← NewTime & NUM_TO_STRING(NewMinutes)
| RETURN NewTime
| ENDFUNCTION
| 1 mark for each of the following:
| 1 Function heading and ending including parameters
| 2 Extract StartHours and convert to integer
| 3 Extract StartMinutes and convert to integer
| 4 Add Duration to StartTime in minutes
| 5 Use DIV() to extract NewHours
| 6 Use MOD() to extract NewMinutes
| 7 Adding leading zeros when necessary to hours and minutes eg “09:05”
| 8 Return concatenated string
| Note: Accept alternative methods for calculation of NewHours and NewMinutes

6(a)(ii) | To test every path through the algorithm | 1

6(a)(iii) | • Logical error
| • Algorithm is incorrect // program produces unexpected result / incorrect calculation is performed | 2
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>6(b)</td>
<td>Test data: Any string value where hours are &gt; “24” or minutes &gt; “59”</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Explanation: Suitable explanation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note: Accept times that would also be invalid for the given scenario.</td>
<td></td>
</tr>
<tr>
<td>6(c)</td>
<td>‘Pseudocode’ solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</td>
<td>8</td>
</tr>
</tbody>
</table>

PROCEDURE GetTotals()

DECLARE BoatNum : INTEGER
DECLARE Paid : REAL
DECLARE FileLine : STRING

FOR BoatNum ← 1 TO 17
    Total[BoatNum] ← 0
ENDFOR

OPENFILE "Hirelog.txt" FOR READ

WHILE NOT EOF("Hirelog.txt")
    READFILE "Hirelog.txt", FileLine
    BoatNum ← STRING_TO_NUM(LEFT(FileLine, 2))
    Paid ← STRING_TO_NUM (RIGHT(FileLine, LENGTH(FileLine) – 8))
    Total[BoatNum] ← Total[BoatNum] + Paid
ENDDO

CLOSEFILE "Hirelog.txt"

ENDPROCEDURE

One mark for each of the following:
1 Procedure heading and ending (where appropriate) with no parameters
2 Initialisation of elements in Total array
3 OPEN "Hirelog.txt" in read mode and CLOSE after use
4 Loop until EOF()
5 Read line from file in a loop
6 Extract and convert BoatNum
7 Extract and convert Paid
8 Update appropriate array total in a loop
Program Code Example Solutions
To be reviewed at STM

Q5(b)(i): Visual Basic

Sub BubbleSort()
    Dim Temp As String
    Dim NoSwaps As Boolean
    Dim Boundary, J As Integer

    Boundary = 999
    Do
        NoSwaps = TRUE
        For J = 1 To Boundary
            If Contact(J) > Contact(J + 1) Then
                Temp = Contact(J)
                Contact(J) = Contact(J + 1)
                Contact(J + 1) = Temp
                NoSwaps = FALSE
            End If
        Next
        Boundary = Boundary - 1
    Loop Until NoSwaps = TRUE
End Sub

Q5(b)(i): Pascal

procedre BubbleSort()
var
    Temp : String;
    NoSwaps : Boolean;
    Boundary, J : Integer;

    Boundary := 999
repeat
    begin
        NoSwaps := TRUE
        for J := 1 to Boundary do
            begin
                if Contact[J] > Contact[J+1]then
                    begin
                        Temp := Contact[J];
                        Contact[J] := Contact[J+1];
                        Contact[J+1] := Temp;
                        NoSwaps := FALSE;
                    end;
                end;

        Boundary := Boundary - 1
    end;
until NoSwaps = TRUE;
End Sub
Q5(b)(i): Python

def BubbleSort():
    # Temp : String
    # NoSwaps : Boolean
    # Boundary, J : Integer

    Boundary = 999
    NoSwaps = TRUE

    while NoSwaps == TRUE:
        NoSwaps = TRUE
        For J in range(Boundary + 1):
            If Contact[J] > Contact[J+1]:
                Temp = Contact[J]
                Contact[J] = Contact[J+1]
                Contact[J+1] = Temp
                NoSwaps = FALSE
        Boundary = Boundary - 1

End Sub

Q6(c): Visual Basic

Sub GetTotals()
    Dim BoatNum As Integer
    Dim Paid As Real
    Dim File As StreamReader("Hirelog.txt")

    For BoatNum = 1 To 17
        Total(BoatNum) = 0
    Next

    Do While File.Peek >= 0
        FileLine = File.ReadLine()
        BoatNum = CInt(Left(FileLine, 2))
        Paid = CSng(Right(FileLine, Len(FileLine) – 8))
        Total(boatnumber) = Total(boatnumber) + Paid
    Loop

    File.Close()

End Sub
Q6(c): Pascal

procedure GetTotals()

var
    BoatNum : Integer;
    Paid : Real;
    MyFile : testfile;

    for BoatNum := 1 to 17 do
        Total[BoatNum] := 0;

    assignFile(MyFile, "Hirelog.txt");
    reset(MyFile);

    while not eof(MyFile) do
    begin
        readln(MyFile, FileLine);
        BoatNum = StrToInt(copy(FileLine, 1, 2));
        Paid = StrToFloat(copy(FileLine, 9, length(Fileline) – 8));
        Total(boatnumber) = Total(boatnumber) + Paid;
    end;

    close(MyFile)
end;

Alternative FreePascal string functions):

    BoatNum := LeftStr(FileLine, 2);

    Paid := StrToFloat(RightStr(FileLine, length(Fileline) – 8));
Q6(c): Python

def GetTotals():
    # BoatNum : Integer
    # Paid : Real
    # File : File Handle
    # FileData : String

    For BoatNum in range (1, 18):
        Total[BoatNum] = 0
    Next

    File = open("Hirelog.txt", "r")
    FileData = File.readline()
    while FileData !="":
        FileLine = File.ReadLine()
        BoatNum = int(FileLine[1, 3])
        Paid = float(FileLine[8, len(FileLine) - 7])
        Total[boatnumber] = Total[boatnumber] + Paid
        FileData = File.readline()

    File.Close()