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.

**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>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1(a)</td>
<td>An algorithm is a solution to a problem expressed as: a sequence of defined steps / stages / instructions / lines of code</td>
<td>2</td>
</tr>
<tr>
<td>1(b)</td>
<td>• Allows the subroutine code to be called from many/multiple places&lt;br&gt;• Subroutine code may be (independently) tested and debugged&lt;br&gt;• If the subroutine task changes the change needs to be made only once&lt;br&gt;• Reduces unnecessary duplication / program lines&lt;br&gt;• Enables sharing of development between programmers</td>
<td>3</td>
</tr>
<tr>
<td>1(c)</td>
<td><img src="image.png" alt="Diagram" /> One mark for each correct line to max 3</td>
<td>3</td>
</tr>
<tr>
<td>Question</td>
<td>Answer</td>
<td>Marks</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>2(a)</td>
<td><img src="image.png" alt="Diagram" /></td>
<td>5</td>
</tr>
</tbody>
</table>

Mark as follows:
1. One mark for all three boxes correctly labelled
2. One mark for selection diamond
3. One mark for passing value **and** return Boolean from `PayByCard()`
4. One mark for passing `Value, AccountNumber and AccountLimit` to `PayByAccount()`
5. One mark for passing `CurrentBalance ByRef`

2(b)(i) Trace table shows:
- 'A' is not treated as an upper case character (row 7)
- `NumUpper` not incremented as expected
- Incorrect final value for `NumUpper` (should be 1)

Max 2

2(b)(ii) One mark per point:
- Line number: 33
- Correction: `IF NextChar >= 'A' AND NextChar <= 'Z'`
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
</table>
| 2(b)(iii) | CASE OF NextChar

>= 'a' AND <= 'z' : NumLower ← NumLower + 1

> 'A' AND <= 'Z' : NumUpper ← NumUpper + 1

OTHERWISE NumNonAlpha ← NumNonAlpha + 1

ENDCASE

One mark for CASE OF NextChar ... ENDCASE

One mark for each remaining line

Accept alternative range description. E.g. 'a' to 'z'

Accept corrected version for the second range. | 4 |
### Question 3(a)

PROCEDURE AddCredit(TopUp : REAL, PhoneNum : STRING)  
DEclare Multiple : REAL  
DEclare Balance : REAL  

Multiple ← 1  
Balance ← GetBalance(PhoneNum)  

IF Balance > 10  
THEN  
    Multiple ← 1.125  
ELSE  
    IF Balance > 5  
    THEN  
        Multiple ← 1.1  
    ENDIF  
ENDIF  

TopUp ← TopUp * Multiple  
SetBalance(PhoneNum, Balance + TopUp)  

ENDPROCEDURE

1 mark for each of the following:

1. **PROCEDURE** heading and ending including parameters
2. **Initialise** Multiple
3. **Assign value to** Balance using GetBalance()
4. **Check for** Balance > 10 **and assignment**: Multiple ← 1.25
5. **Check for** Balance > 5 **and assignment**: Multiple ← 1.1
6. **Assignment**: TopUp ← TopUp * Multiple
7. **Calling** SetBalance() **with correct parameters**

Note:
MP6 could be included in MP7 statement
### Question 3(b)

<table>
<thead>
<tr>
<th>Answer</th>
</tr>
</thead>
</table>
| PROCEDURE Search(SearchString : STRING)  
DECLARE Index, Msg : STRING  
  
  Msg ← "Found at:" //initial value  
  
  FOR Index ← 1 TO 100  
    IF NameList[Index, 1] = SearchString  
      AND NameList[Index, 2] = "Active"  
      THEN  
        Msg ← Msg & " " & NUM_TO_STRING(Index)  
      ENDIF  
  ENDFOR  
  
  IF Msg = "Found at:" // no change to initial value  
    THEN  
      OUTPUT "Search String not found"  
    ELSE  
      OUTPUT Msg  
  ENDFOR  
ENDPROCEDURE |

1 mark for each of the following:

1. **PROCEDURE** heading and ending including parameter
2. Declare local variables for `Index` and `Msg` and initialise `Msg` to appropriate string
3. Loop structure
4. Compare `SearchString` to name (column 1)...
5. ... **AND** Compare status to "Active" (column 2) **in a loop**
6. Add `Index` to `Msg` when a match is encountered (using type conversion)
7. Condition to determine which string is output **after loop**
8. Correct output of single message

Note:
Credit alternative solutions for forming and checking a single output string

### Question 4(a)

<table>
<thead>
<tr>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>A program fault is something that makes the program <strong>not do what it is supposed to do</strong> under certain circumstances</td>
</tr>
</tbody>
</table>

One mark per underlined phrase or equivalent
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>4(b)</td>
<td>Answers include the use of:</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>• Tried and tested (library) subroutines / code</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Modular programming techniques (to break the problem down and make it easier to solve)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Good programming practice (formatting, sensible variable names, comments etc)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• IDE features (parameter type-checking, auto-complete)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max 3</td>
<td></td>
</tr>
<tr>
<td>4(c)</td>
<td>Syntax error: A construct / statement in the source code that breaks the rules of the language</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Logic Error: An error in the algorithm that causes the program not to behave as intended</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run-time: A program performs an invalid operation / tries to divide by zero // enters an infinite loop / stops unexpectedly</td>
<td></td>
</tr>
</tbody>
</table>
### Question 5(a)(i)

PROCEEDURE SortContacts()
DECLARE Temp : STRING
DECLARE FirstName, SecondName : STRING
DECLARE NoSwaps : BOOLEAN
DECLARE Boundary, J : INTEGER
Boundary ← 999
REPEAT
  NoSwaps ← TRUE
  FOR J ← 1 TO Boundary
    FirstName ← RIGHT(Directory[J], LENGTH(Directory[J]) - 4)
    SecondName ← RIGHT(Directory[J + 1], LENGTH(Directory[J + 1]) - 4)
    IF FirstName > SecondName THEN
      Temp ← Directory[J]
      Directory[J + 1] ← Temp
      NoSwaps ← FALSE
    ENDIF
  ENDFOR
  Boundary ← Boundary - 1
UNTIL NoSwaps = TRUE
ENDPROCEDURE

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
</table>

### Question 5(b)

**Description:**
- uses a flag variable to stop the outer loop
- after no more swaps made during one pass of the inner loop
- the flag is reset before the inner loop starts, and set whenever a swap is made
- decreases the loop size at end of inner loop (Boundary decremented)

Max 3 for description

**Effective because:**
- It prevents unnecessary iterations / passes through the array (i.e. when the array is already sorted) // terminates the algorithm when all elements are in order // reduces the number of unnecessary comparisons

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>5(b)</td>
<td>Description: uses a flag variable to stop the outer loop after no more swaps made during one pass of the inner loop the flag is reset before the inner loop starts, and set whenever a swap is made decreases the loop size at end of inner loop (Boundary decremented) Effective because: It prevents unnecessary iterations / passes through the array (i.e. when the array is already sorted) // terminates the algorithm when all elements are in order // reduces the number of unnecessary comparisons</td>
<td>4</td>
</tr>
</tbody>
</table>
Question | Answer | Marks
--- | --- | ---
6(a) | PROCEDURE ListAvailable(StartTime : STRING)  
DECLARE NumAvailable, Index : INTEGER  
DECLARE TimeBack : STRING  
DECLARE Available : BOOLEAN  

NumAvailable ← 0  

FOR Index ← 1 TO 10  
Available ← FALSE // initialise  
IF HireTime[Index] = "Available" // not on hire  
THEN  
Available ← TRUE // available now  
ELSE  
TimeBack ← AddTime(HireTime[Index],__  
Duration[Index])  
IF TimeBack < StartTime // < or <=  
THEN  
Available ← TRUE // will be available  
ENDIF  
ENDIF  

IF Available = TRUE  
THEN  
OUTPUT "Boat ", Index ", " is available"  
NumAvailable ← NumAvailable + 1  
ENDIF  
ENDFOR  

IF NumAvailable > 0  
THEN  
OUTPUT "There are ", NumAvailable ,"  
" boats available."  
ELSE  
OUTPUT "Sorry, there are no boats available"  
ENDIF  
ENDPROCEDURE  

1 mark for each of the following:

1 Procedure heading and ending including input parameter  
2 Declare local variable for the count of available boats and initialise to 0  
3 Loop through all 10 boats  
4 Use of AddTime() to calculate TimeBack  
5 Check for boats that are not on hire OR those due back in time in a loop  
6 Increment count for number of available boats in a loop  
7 Output a message for each available boat in a loop  
8 Output both messages as appropriate outside a loop
### Question 6(b)

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

```plaintext
PROCEDURE RecordHire(HBoatNumber, HDuration : INTEGER, __
                   HTime : STRING, HCost : REAL)

DECLARE FileLine : STRING
CONSTANT Comma = ','

HireTime[HBoatNumber] ← HTime
Duration[HBoatNumber] ← HDuration
Cost[HBoatNumber] ← HCost

DailyTakings ← DailyTakings + HCost

OPENFILE "HireLog.txt" FOR APPEND

FileLine ← NUM_TO_STRING(HBoatNumber) & Comma
FileLine ← FileLine & HTime & Comma
FileLine ← FileLine & NUM_TO_STRING(HDuration)
FileLine ← FileLine & Comma & NUM_TO_STRING(HCost)

WRITEFILE "HireLog.txt", FileLine

CLOSEFILE "HireLog.txt"

ENDFUNCTION
```

One mark for each of the following:

1. Procedure heading and ending (where appropriate), including input parameters (order not essential)
2. Updating the three arrays from parameter values
3. Totalling DailyTakings
4. OPEN "HireLog.txt" in append mode
5. Creating file text line including separators
6. ...making use of type conversion as required
7. Writing the line to the file
8. Closing the file

Solutions may combine mark points 5 and 6 (and 7)

Max 7

---

© UCLES 2020
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Marks</th>
</tr>
</thead>
</table>
| 6(c)(i)  | 'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.  
\[
\text{EndTime} \leftarrow \text{Addtime} (\text{BeginTime}, 60)
\]  
One mark per underlined section (Space before bracket for mark scheme clarification only) | 2 |
| 6(c)(ii) | One mark for each test:  
For example:  
**Test 1**  
Start time value "10:00", Duration value 30  
Expected new time value "10:30"  
**Test 2**  
Start time value "10:45", Duration value 30  
Expected new time value "11:15"  
String values (time) must be enclosed in quotation marks, integer values (duration) must not. Penalise once then FT. | 2 |
Program Code Example Solutions  
To be reviewed at STM

Q6(b)(i): Visual Basic

Sub RecordHire(HBoatNumber, HDuration As Integer, HTime As String, HCost As Real)

    Dim FileLine As String
    Const Comma = ',,'

    HireTime(HBoatNumber) = HTime
    Duration(HBoatNumber) = HDuration
    Cost(HBoatNumber) = HCost

    DailyTakings = DailyTakings + HCost

    FileOpen(1, "HireLog.txt", OpenMode.Append)

    FileLine = CStr(HBoatNumber) & Comma
    FileLine = FileLine & HTime & Comma
    FileLine = FileLine & CStr(HDuration) & Comma
    FileLine = FileLine & CStr(HCost)

    Print(1, FileLine)
    PrintLine(1)

    Fileclose(1)

End Sub
Q6(b)(i): Pascal

procedure RecordHire(HBoatNumber, HDuration : integer; HTime : string;
HCost : Real);

var
    Fileline : string;
    ThisFile: TextFile;

const Comma = ',,';

begin

    HireTime[HBoatNumber] := HTime;
    Duration[HBoatNumber] := HDuration;
    Cost[HBoatNumber] := HCost;

    DailyTakings := DailyTakings + HCost;

    AssignFile(Thisfile, "HireLog.txt");

    FileLine := IntToStr(HBoatNumber) + Comma;
    FileLine := FileLine + HTime + Comma;
    FileLine := FileLine + IntToStr (HDuration) + Comma;
    FileLine := FileLine + IntToStr (HCost);

    writeln(ThisFile, FileLine);

    CloseFile(ThisFile);
end;

Q6(b)(i): Python

def RecordHire(HBoatNumber, HDuration, HTime, HCost)

    # FileLine : String
    # File : File handle

    Comma = ',,'

    HireTime[HBoatNumber] = HTime
    Duration[HBoatNumber] = HDuration
    Cost[HBoatNumber] = HCost

    DailyTakings = DailyTakings + HCost

    File = Open("HireLog.txt", "a")

    FileLine = Str(HBoatNumber) + Comma
    FileLine = FileLine + HTime + Comma
    FileLine = FileLine + Str(HDuration) + Comma
    FileLine = FileLine + Str(HCost)

    File.write(FileLine)

    File.close
Q6(c)(i): Visual Basic

End Time = Addtime(Begin Time, 60)

Q6(c)(i): Pascal

End Time := Addtime(Begin Time, 60)

Q6(c)(i): Python

End Time = Addtime(Begin Time, 60)