



# Cambridge International AS & A Level

---

**COMPUTER SCIENCE**

**9608/21**

Paper 2 Written Paper

**May/June 2020**

**MARK SCHEME**

Maximum Mark: 75

---

<p><b>Published</b></p>
-------------------------

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.

---

This document consists of **15** 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:**

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:**

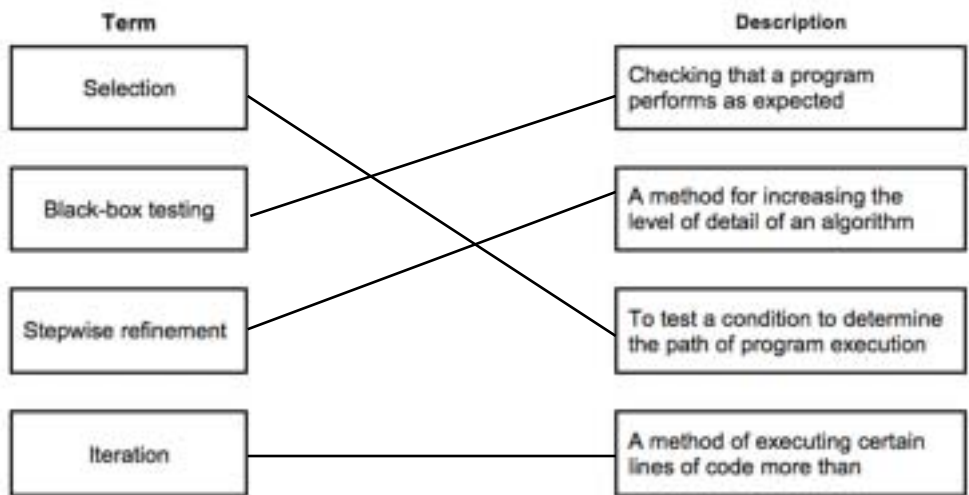
Rules must be applied consistently e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.

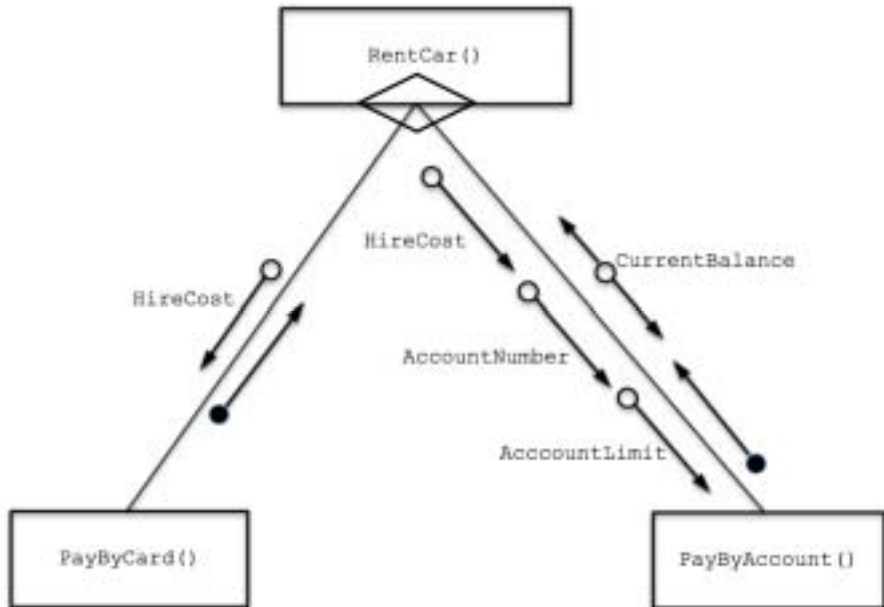
**GENERIC MARKING PRINCIPLE 5:**

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.

Question	Answer	Marks
1(a)	<p>An algorithm is a solution to a problem expressed as:</p> <p>a <u>sequence</u> of defined <u>steps</u> / <u>stages</u> / <u>instructions</u> / <u>lines of code</u></p> <p>1 mark for each underlined term (or equivalent)</p>	2
1(b)	<ul style="list-style-type: none"> <li>Allows the subroutine code to be called from many/multiple places</li> <li>Subroutine code may be (independently) tested and debugged</li> <li>If the subroutine task changes the change needs to be made only once</li> <li>Reduces unnecessary duplication / program lines</li> <li>Enables sharing of development between programmers</li> </ul> <p>Or equivalent points that relate to a PROGRAM (not an algorithm)</p> <p>Max 3</p>	3
1(c)	 <p>One mark for each correct line to max 3</p>	3

Question	Answer	Marks
2(a)	 <p>Mark as follows:</p> <ol style="list-style-type: none"> <li>1 One mark for all three boxes correctly labelled</li> <li>2 One mark for selection diamond</li> <li>3 One mark for passing value <b>and</b> return Boolean from PayByCard()</li> <li>4 One mark for passing Value, AccountNumber <b>and</b> AccountLimit to PayByAccount()</li> <li>5 One mark for passing CurrentBalance ByRef</li> </ol>	5
2(b)(i)	<p>Trace table shows:</p> <ul style="list-style-type: none"> <li>• 'A' is not treated as an upper case character (row 7)</li> <li>• NumUpper not incremented as expected</li> <li>• Incorrect final value for NumUpper (should be 1)</li> </ul> <p>Max 2</p>	2
2(b)(ii)	<p>One mark per point:</p> <ul style="list-style-type: none"> <li>• Line number: 33</li> <li>• Correction: IF NextChar &gt;= 'A' AND NextChar &lt;= 'Z'</li> </ul>	2

Question	Answer	Marks
2(b)(iii)	<p>CASE OF NextChar</p> <pre>       &gt;= 'a' AND &lt;= 'z' : NumLower ← NumLower + 1       &gt; 'A' AND &lt;= 'Z' : NumUpper ← NumUpper + 1       OTHERWISE NumNonAlpha ← NumNonAlpha + 1     </pre> <p>ENDCASE</p> <p>One mark for CASE OF NextChar ... ENDCASE One mark for each remaining line</p> <p>Accept alternative range description. E.g. 'a' to 'z'</p> <p>Accept corrected version for the second range.</p>	4

Question	Answer	Marks
3(a)	<pre> PROCEDURE AddCredit(TopUp : REAL, PhoneNum : STRING)   DECLARE Multiple : REAL   DECLARE Balance : REAL    Multiple ← 1   Balance ← GetBalance(PhoneNum)    IF Balance &gt; 10     THEN       Multiple ← 1.125     ELSE       IF Balance &gt; 5         THEN           Multiple ← 1.1         ENDIF       ENDIF   ENDIF    TopUp ← TopUp * Multiple    SetBalance(PhoneNum, Balance + TopUp)  ENDPROCEDURE </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> <li>1 PROCEDURE heading and ending including parameters</li> <li>2 Initialise Multiple</li> <li>3 Assign value to Balance using GetBalance()</li> <li>4 Check for Balance &gt; 10 <b>and</b> assignment: Multiple ← 1.25</li> <li>5 Check for Balance &gt; 5 <b>and</b> assignment: Multiple ← 1.1</li> <li>6 Assignment: TopUp ← TopUp * Multiple</li> <li>7 Calling SetBalance() with correct parameters</li> </ol> <p>Note: MP6 could be included in MP7 statement</p>	7

Question	Answer	Marks
3(b)	<pre> 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 &amp; " " &amp; NUM_TO_STRING(Index)     ENDIF   ENDFOR    IF Msg = "Found at:" // no change to initial value   THEN     OUTPUT "Search String not found"   ELSE     OUTPUT Msg   ENDIF ENDPROCEDURE </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> <li>1 PROCEDURE heading and ending including parameter</li> <li>2 Declare local variables for Index and Msg <b>and</b> initialise Msg to appropriate string</li> <li>3 Loop structure</li> <li>4 Compare SearchString to name (column 1)...</li> <li>5 ... <b>AND</b> Compare status to "Active" (column 2) <b>in a loop</b></li> <li>6 Add Index to Msg when a match is encountered (using type conversion)</li> <li>7 Condition to determine which string is output <b>after loop</b></li> <li>8 Correct output of single message</li> </ol> <p>Note: Credit alternative solutions for forming and checking a single output string</p>	8

Question	Answer	Marks
4(a)	<p>A program fault is something that makes the program <u>not do what it is supposed to do</u> under <u>certain circumstances</u></p> <p>One mark per underlined phrase or equivalent</p>	2

Question	Answer	Marks
4(b)	<p>Answers include the use of:</p> <ul style="list-style-type: none"><li>• Tried and tested (library) subroutines / code</li><li>• Modular programming techniques (to break the problem down and make it easier to solve)</li><li>• Good programming practice (formatting, sensible variable names, comments etc)</li><li>• IDE features (parameter type-checking, auto-complete)</li></ul> <p>Max 3</p>	<b>3</b>
4(c)	<p>Syntax error: A construct / statement in the source code that breaks the rules of the language</p> <p>Logic Error: An error in the algorithm that causes the program not to behave as intended</p> <p>Run-time: A program performs an invalid operation / tries to divide by zero // enters an infinite loop / stops unexpectedly</p>	<b>3</b>



Question	Answer	Marks
5(a)(i)	<pre> PROCEDURE SortContacts() DECLARE Temp : STRING DECLARE FirstName, SecondName : STRING DECLARE NoSwaps : <b>BOOLEAN</b> DECLARE Boundary, J : INTEGER Boundary ← <b>999</b> REPEAT     NoSwaps ← TRUE     FOR J ← 1 TO Boundary         FirstName ← <b>RIGHT</b>(Directory[J], <b>__</b>                                 LENGTH(Directory[J]) - <b>4</b>)         SecondName ← <b>RIGHT</b>(Directory[J + 1], <b>__</b>                                 LENGTH(Directory[J + 1]) - 4)         IF FirstName <b>&gt; SecondName</b>             THEN                 Temp ← Directory[J]                 Directory[J] ← Directory[<b>J + 1</b>]                 Directory[J + 1] ← Temp                 NoSwaps ← <b>FALSE</b>             ENDIF     ENDFOR     Boundary ← <b>Boundary - 1</b> UNTIL NoSwaps = TRUE ENDPROCEDURE </pre> <p>One mark per highlighted phrase</p>	8
5(b)	<p><b>Description:</b></p> <ul style="list-style-type: none"> <li>• uses a flag variable to stop the outer loop</li> <li>• after no more swaps made during one pass of the inner loop</li> <li>• the flag is reset before the inner loop starts, and set whenever a swap is made</li> <li>• decreases the loop size at end of inner loop (Boundary decremented)</li> </ul> <p>Max 3 for description</p> <p><b>Effective because:</b></p> <ul style="list-style-type: none"> <li>• 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</li> </ul>	4

Question	Answer	Marks
6(a)	<pre> 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 &lt; StartTime // &lt; or &lt;=           THEN             Available ← TRUE // will be available           ENDIF         ENDIF       ENDIF      IF Available = TRUE       THEN         OUTPUT "Boat " , Index , " is available"         NumAvailable ← NumAvailable + 1       ENDIF     ENDFOR      IF NumAvailable &gt; 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 <b>and</b> 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 <b>OR</b> those due back in time <b>in a loop</b> 6  Increment count for number of available boats <b>in a loop</b> 7  Output a message for each available boat <b>in a loop</b> 8  Output both messages as appropriate <b>outside a loop</b> </pre>	8

Question	Answer	Marks
6(b)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <pre> 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) &amp; Comma     FileLine ← FileLine &amp; HTime &amp; Comma     FileLine ← FileLine &amp; NUM_TO_STRING(HDuration)     FileLine ← FileLine &amp; Comma &amp; NUM_TO_STRING(HCost)      WRITEFILE "HireLog.txt", FileLine      CLOSEFILE "HireLog.txt"  ENDFUNCTION </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> <li>1 Procedure heading and ending (where appropriate), including input parameters (order not essential)</li> <li>2 Updating the three arrays from parameter values</li> <li>3 Totalling DailyTakings</li> <li>4 OPEN "HireLog.txt" in append mode</li> <li>5 Creating file text line including separators</li> <li>6 ....making use of type conversion as required</li> <li>7 Writing the line to the file</li> <li>8 Closing the file</li> </ol> <p>Solutions may combine mark points 5 and 6 (and 7)</p> <p>Max 7</p>	7

Question	Answer	Marks
6(c)(i)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <p><u>EndTime</u> ← <u>Addtime</u> (<u>BeginTime</u>, 60)</p> <p>One mark per underlined section (Space before bracket for mark scheme clarification only)</p>	2
6(c)(ii)	<p>One mark for each test:</p> <p>For example:</p> <p><b>Test 1</b> Start time value "10:00", Duration value 30 Expected new time value "10:30"</p> <p><b>Test 2</b> Start time value "10:45", Duration value 30 Expected new time value "11:15"</p> <p>String values (time) must be enclosed in quotation marks, integer values (duration) must not. Penalise once then FT.</p>	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**

```
EndTime = Addtime(BeginTime, 60)
```

**Q6(c)(i): Pascal**

```
EndTime := Addtime(BeginTime, 60)
```

**Q6(c)(i): Python**

```
EndTime = Addtime(BeginTime, 60)
```