



Cambridge International AS & A Level

COMPUTER SCIENCE**9608/23**

Paper 2 Written Paper

May/June 2020

MARK SCHEME

Maximum Mark: 75

Published

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 **16** 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)	When: when designing a <u>solution</u> to a problem Purpose: to describe the solution as a <u>sequence of steps / actions</u> One mark per answer Accept equivalent phrase for 'purpose' but reject specific programming references	2
1(b)	Testing <u>a condition</u> <u>to determine the sequence</u> of execution One mark per underlined phrase (or equivalent)	2
1(c)	Explanation: Breaking a problem down into sub tasks Reason: Make the problem easier to solve // to make the solution easier to implement / test / maintain	2
1(d)	<ul style="list-style-type: none"> • Breakpoints • Single-stepping • Watch window Max 2	2

Question	Answer	Marks
2(a)	<ul style="list-style-type: none"> • Sequence of (module) execution • Module Iteration • Module selection One mark per item Max 2	2
2(b)(i)	<u>FUNCTION ModuleB (ParX : INTEGER) RETURNS BOOLEAN</u> One mark for each underlined part Ignore BYVALUE for ParX	3
2(b)(ii)	<u>PROCEDURE ModuleC (BYREF ParW: REAL, BYVALUE ParZ : STRING)</u> One mark for each underlined part BYVALUE not essential for ParZ	3

Question	Answer	Marks
2(c)	<pre> DECLARE Index, Count : INTEGER Count ← 0 FOR Index ← 1 TO 50 IF Item[Index, 1] = SearchString THEN Count ← Count + 1 ENDIF IF Item[Index, 2] = SearchString THEN Count ← Count + 1 ENDIF ENDIFOR OUTPUT "The number of times SearchString found: ", Count Alternative DECLARE I, J, Count : INTEGER Count ← 0 FOR I ← 1 TO 50 FOR J ← 1 TO 2 IF Item[I, J] = SearchString THEN Count ← Count + 1 ENDIF ENDFOR ENDFOR ENDIFOR OUTPUT "The number of times SearchString found: ", Count One mark for each of the following: 1 Initialisation of Count 2 FOR loop 3 Check column 1 element and increment count 4 Check column 2 element and increment count // nested loops 5 OUTPUT Count together with suitable message </pre>	5

Question	Answer	Marks
3(a)	<pre> graph TD Start([START]) --> SetError[SET Error TO value of Delta(Plan, Actual)] SetError --> CaseOfError{CASE OF Error} CaseOfError -- "> 10" --> SteerMinus[SET Steer TO Steer - 10] CaseOfError -- "0" --> ZCount[ZCount + 1] CaseOfError -- "< -10" --> SteerPlus[SET Steer TO Steer + 10] CaseOfError -- OTHERWISE --> Output[/OUTPUT "Unexpected Error"/] SteerMinus --> Merge(()) ZCount --> Merge SteerPlus --> Merge Output --> Merge Merge --> End([END]) </pre> <p>Mark as follows:</p> <ul style="list-style-type: none"> • One mark for START and END • One mark per area outlined <p>All outputs from CASE must be labelled</p>	5
3(b)(i)	<p>Error 1</p> <ul style="list-style-type: none"> • Index stops at 5 – should go to 6. Fails to loop correctly through all characters of string (lines 26 to 38) // Final values for NumUpper is not as expected (because of loop error) <p>Error 2</p> <ul style="list-style-type: none"> • Trace table row 12 – NumOther assigned wrong value. 0 expected not 3 <p>One mark per bullet</p>	2

Question	Answer	Marks
3(b)(ii)	<p>Error 1 Line Number: 26 Correction: FOR Index ← 1 to StrLen</p> <p>Error 2 Line Number: 41 Correction: NumOther ← StrLen - (NumDigit + NumUpper)</p> <p>One mark for each line number with correction</p>	2
3(c)	Data Structure	1

Question	Answer	Marks
4(a)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <pre> FUNCTION Extract(InString : STRING) RETURNS STRING DECLARE Name : STRING DECLARE NextChar : CHAR DECLARE Index : INTEGER CONSTANT COLON = ':' Index ← 1 Name ← "" NextChar ← LEFT(InString, 1) WHILE NextChar <> COLON Name ← Name & NextChar Index ← Index + 1 NextChar ← MID(InString, Index, 1) ENDWHILE RETURN Name ENDFUNCTION </pre> <p>Alternative:</p> <pre> FUNCTION Extract(InString : STRING) RETURNS STRING DECLARE Name : STRING DECLARE Index : INTEGER CONSTANT COLON = ':' Index ← 1 WHILE MID(InString, Index, 1) <> COLON Index ← Index + 1 ENDWHILE Name ← LEFT(InString, Index) RETURN Name ENDFUNCTION </pre> <p>Mark as follows:</p> <ol style="list-style-type: none"> 1 Function heading and ending (where required) including parameters 2 Extract (next) character from InString 3 Conditional loop while character is not colon 4 Append character to Name and increment Index in a loop // calculate substring length and use LEFT() after loop 5 RETURN Name (may be combined with alternative mp 4) 	5

Question	Answer	Marks
4(b)	<p>Two alternatives:</p> <p>Alternative #1</p> <p>Header: <u>PROCEDURE Extract (BYREF Name : STRING, BYREF Email : STRING, BYVALUE DataItem : STRING)</u></p> <p>Explanation: Could additionally pass the name and email address to the procedure using BYREF. Procedure would extract name and email and assign values to BYREF parameters; these would then be available to calling program.</p> <p>Alternative #2</p> <p>Header: <u>PROCEDURE Extract (DataItem : STRING)</u></p> <p>Explanation: Declare new global variables for the name and email address. These could be assigned values within the new procedure and these values would be used by the calling program.</p> <p>Mark as follows:</p> <ul style="list-style-type: none"> • Two marks for header – must be a Procedure not a Function (but see note below) • Max Two marks for explanation <p>Note: Allow solution based on user-defined record type / record item returned from modified function.</p> <p>Max 3</p>	3

Question	Answer	Marks
5(a)	<pre> FUNCTION GetLastService(BoatNum : STRING) RETURNS STRING DECLARE LastService, FileData : STRING LastService ← "" OPEN "ServiceLog.txt" FOR READ WHILE NOT EOF("ServiceLog.txt") READFILE "ServiceLog.txt", FileData IF LEFT(FileData, 2) = BoatNum THEN LastService ← RIGHT(FileData, 8) ENDIF ENDWHILE Close "ServiceLog.txt" RETURN LastService // Most recent service will be the last one read ENDFUNCTION </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> 1 Function heading and ending including parameters 2 Declaration of variables for LastService and FileData 3 Open file in read mode and close 4 Conditional loop - while not EOF() 5 Read line from file in a loop 6 Compare BoatNum with data from file in a loop 7 If matched, assign value to LastService in a loop 8 Return LastService 	8

Question	Answer	Marks
5(b)(i)	<pre> FUNCTION GetHours (BNum, SDate, : STRING) RETURNS REAL DECLARE FileData, HString : STRING DECLARE TotHours : REAL DECLARE HLength : INTEGER TotHours ← 0 OPEN "HireLog.txt" FOR READ WHILE NOT EOF("HireLog.txt") READFILE "HireLog.txt" FileData IF MID(FileData, 9, 2) = BNum THEN IF LEFT(FileData, 8) > SDate THEN HLength ← LENGTH(Filedata) - 10 HString ← RIGHT(Filedata, HLength) TotHours ← TotHours + STRING_TO_NUM(HString) ENDFIF ENDFIF ENDFUNCTION ENDFUNCTION Close "HireLog.txt" RETURN TotHours ENDFUNCTION </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> 1 Function heading and ending including parameters 2 Declaration of TotHours as integer 3 ...and initialisation to zero 4 Extract and compare BoatNum ... 5 ... extract date and compare to ServData and if later then... 6 ... extract HireDuration and ... 7 ... convert HireDuration to REAL and sum TotHours 8 Return TotHours 	8

Question	Answer	Marks
5(b)(ii)	<p>Three <i>different</i> tests</p> <p>String example 1</p> <ul style="list-style-type: none">• String: "007"• Reason: Too many characters <p>String example 2</p> <ul style="list-style-type: none">• String: "4x"• Reason: String does not represent a numeric value / contains an illegal character <p>String example 3</p> <ul style="list-style-type: none">• String: "25"• Reason: number outside range 1 to 20 <p>Illegal character can only be used in one test</p> <p>One mark for string plus one for corresponding explanation</p>	6

Question	Answer	Marks
5(c)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <pre> PROCEDURE ServiceList (MaxHours : INTEGER) DECLARE BoatNum, Hours : INTEGER DECLARE LastService : STRING DECLARE Due : BOOLEAN Due ← FALSE OUTPUT "Boat Service List" FOR BoatNum ← 1 TO 20 LastService ← GetLastService (NUM_TO_STRING (BoatNum)) Hours ← GetHours (NUM_TO_STRING (BoatNum), LastService) IF Hours > MaxHours THEN OUTPUT NUM_TO_STRING (Boatnum) & ": " ___ & NUM_TO_STRING (Hours) Due ← TRUE ENDIF ENDFOR IF Due = TRUE THEN OUTPUT "No boats are due to be serviced" ENDIF ENDPROCEDURE </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> 1 Procedure heading and ending (where appropriate) including parameter 2 Output "report header" line not in a loop 3 Loop from 1 to 20 4 Call GetLastService 5 Call GetHours 6 Check return value from GetHours > MaxHours 7 Suitable Output statement in a loop 8 Mechanism to initialise, count and produce final output if no boats to be serviced 	8
5(d)(i)	<p>One mark per bullet point</p> <ul style="list-style-type: none"> • The use of tried and tested (library) subroutines • The use of modular programming (to break the problem down and make it easier to solve) • The use of programming practice to make the code easier to read (e.g. format, use of sensible variable names) <p>Max 2</p>	2

Question	Answer	Marks
5(d)(ii)	One mark per bullet point <ul style="list-style-type: none">• Testing may be carried out before the modules are developed // not ready for full testing• Module stubs contain simple code to provide a known response // temporary replacement for a called module / return a fixed value / output a message to confirm the module has been called	2
5(d)(iii)	One mark per bullet point <ul style="list-style-type: none">• Executes a line of code at a time• Used to trace the path of execution (sequence)• Track variable values using a watch window Max 2	2

Program Code Example Solutions**Q4(a): Visual Basic**

```

Function Extract(InString As String) As String
    Dim Name As String
    Dim NextChar As Char
    Dim Index As Integer
    Const COLON = ':'

    Index = 1
    Name = ""
    NextChar = Left(InString, 1)

    While NextChar <> COLON
        Name = Name & NextChar
        Index = Index + 1
        NextChar = Mid(InString, Index, 1)
    Loop

    Return Name

End Function

Function Extract(InString As String) As String
    Return Mid(InString, 1, InString.IndexOf(":"))
End Function

```

Q4(a): Pascal

```

function extract(InString : string) : string;
var
    Name : string;
    NextChar : char;
    Index : integer;

const
    COLON = ':';

    Index := 1;
    Name := "";
    NextChar := copy(InString, 1, 1);

    While NextChar <> COLON do
    begin
        Name := Name & NextChar;
        Index := Index + 1;
        NextChar := copy(InString, Index, 1);
    end;

    extract := Name;

end;

```

Q4(a): Python

```
def Extract(InString)
    # Name : String
    # NextChar : Char
    # Index : Integer

    COLON = ':'

    Index = 1
    Name = ""
    NextChar = InString[1:2]

    While NextChar <> COLON:
        Name = Name + NextChar
        Index = Index + 1
        NextChar = Instring[Index, Index + 1]

    Return Name
```

Q5(c): Visual Basic

```
Sub ServiceList(MaxHours As Integer)

    Dim BoatNum, Due As Integer
    Dim LastService As String

    Due = 0
    console.writeline("Boat Service List")

    For BoatNum = 1 To 20
        LastService = GetLastService(CStr(BoatNum))
        Hours = GetHours(CStr(BoatNum), LastService)
        If Hours > MaxHours Then
            Console.Writeline(CStr(Boatnum) & ": " & CStr(Hours))
            Due = Due + 1
        End If
    Next

    If Due = 0 Then
        Console.Writeline("No boats are due to be serviced")
    End If

End Sub
```

Q5(c): Pascal

```
procedure ServiceList(MaxHours : Integer)

var
  BoatNum, Due : integer;
  LastService : string;

  Due := 0
  print("Boat Service List")

  for BoatNum := 1 to 20 do
  begin
    LastService := GetLastService(CInt(BoatNum));
    Hours := GetHours(BoatNum, LastService);
    If Hours > MaxHours then
      begin
        writeln(IntToStr(Boatnum) & ": " & IntToStr(Hours));
        Due := Due + 1;
      end;
    end;

  If Due = 0 then
    writeln ("No boats are due to be serviced")
  End If
End Sub
```

Q5(c): Python

```
def ServiceList(MaxHours)

  # BoatNum, Due : Integer
  # LastService : String

  Due = 0
  print("Boat Service List")

  For BoatNum in range(1, 21):
    LastService = GetLastService(str(BoatNum))
    Hours = GetHours(BoatNum, LastService)
    if Hours > MaxHours:
      print(str(Boatnum) & ": " & str(Hours))
      Due = Due + 1

  if Due == 0:
    print("No boats are due to be serviced")
```