Cambridge Assessment
International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/21**

Paper 2 Fundamental Problem-Solving and Programming Skills **October/November 2021**

MARK SCHEME

Maximum Mark: 75

Published

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)	One mark per row **Variable** / **New identifier name** Var1: `Rainfall / DailyRainfall` Var2: `AvgWindSpeed` Var3: `StationID / WeatherStationID / StationIDNo / WeatherStationIDNo`	**3**

1(b)	One mark per row.
	
	Pseudocode expression / **Evaluates to**
	`LENGTH(HouseCount) > 6` : `"ERROR"`
	`MOD(INT(Turnout2018) * 3, 4)` : `0`
	`ASC(TidalRiskCategory) + Turnout2018` : `87.23`
	`IsConservationArea AND (HouseCount <= 50)` : `FALSE`
	`MID(StationLocationName, 1, 5) & " Eleven"` : `"Ocean Eleven"`
		5

1(c)	1 mark for error: • Function expects a real parameter, but parameter is a string // Data type mismatch (between the parameter and the data passed) 1 mark for the correct function header: `FUNCTION ProcessVars(DataItem : STRING) RETURNS REAL`	**2**
1(d)	1 mark for each description. Breakpoints • Point set where code stops running Report (watch) window • shows the content of all data structures/variables/constants during the execution Single stepping • One **line** of code is run and then it pauses	**3**

1(a) table:

Variable	New identifier name
Var1	`Rainfall / DailyRainfall`
Var2	`AvgWindSpeed`
Var3	`StationID / WeatherStationID / StationIDNo / WeatherStationIDNo`

1(b) table:

Pseudocode expression	Evaluates to
`LENGTH(HouseCount) > 6`	`"ERROR"`
`MOD(INT(Turnout2018) * 3, 4)`	`0`
`ASC(TidalRiskCategory) + Turnout2018`	`87.23`
`IsConservationArea AND (HouseCount <= 50)`	`FALSE`
`MID(StationLocationName, 1, 5) & " Eleven"`	`"Ocean Eleven"`

Question	Answer	Marks
2(a)(i)	Count-controlled loop	**1**
2(a)(ii)	One mark per row.	**4**

The scope of the variable `Message` is	Global
The start and end line numbers of a selection structure	12, 15
The identifier name of a user-defined function is	`CharacterCount`
An arithmetic operator used in the function is	+ // -

Question	Answer	Marks

2(b)	One mark for line number and corrected line.
	• Line 06
	`DECLARE ThisChar : CHAR  /  STRING`
	• Line 08
	`LetterCount ← 0`
	• Line 10
	`FOR Index ← 1 TO LENGTH(Message)`
	`FOR Index ← 0 TO LENGTH(Message)-1`
	• Line 11
	`ThisChar ← MID(Message, Index, 1)`
		4

```
01   DECLARE Message: STRING
02
03   FUNCTION CharacterCount(Letter : CHAR) RETURNS
                                           INTEGER
04
05      DECLARE LetterCount, Index : INTEGER
06      DECLARE ThisChar : CHAR
07
08      LetterCount ← 0
09
10      FOR Index ← 1 TO LENGTH(Message)
11         ThisChar ← MID(Message, Index, 1)
12         IF ThisChar = Letter
13           THEN
14             LetterCount ← LetterCount + 1
15         ENDIF
16      ENDFOR
17      RETURN LetterCount
18   ENDFUNCTION
```

Question	Answer	Marks

2(c)	One mark each to max 5
	1 initialisation of counter data structure for each vowel
	2 prompt **and** input the string
	3 loop through length of input string …
	4 … extract each character in the string **and** use `CASE` structure to increment each counter variable …
	5 … check for both lower case and upper case (by converting to upper/lower or manual check of all)
	6 … output each vowel with its count value **once** at appropriate point
	
	`PROCEDURE Frequency()`
	
	`   DECLARE DataString : STRING`
	`   DECLARE DataCharacter : CHAR`
	`   DECLARE CountA, CountE, CountI, CountO,`
	`           CountU : INTEGER`
	`   DECLARE Index : INTEGER`
	
	`   CountA ← 0`
	`   CountE ← 0`
	`   CountI ← 0`
	`   CountO ← 0`
	`   CountU ← 0`
	
	`   Index ← 1`
	
	`   OUTPUT "Enter string: "`
	`   INPUT DataString`
	
	`   FOR Index ← 1 to LENGTH(DataString)`
	`      DataCharacter ← UCASE(MID(DataString, Index, 1))`
	`      CASE OF DataCharacter`
	`         'A' : CountA ← CountA + 1`
	`         'E' : CountE ← CountE + 1`
	`         'I' : CountI ← CountI + 1`
	`         'O' : CountO ← CountO + 1`
	`         'U' : CountU ← CountU + 1`
	`      ENDCASE`
	`      Index ← Index + 1`
	`   ENDFOR`
	
	`   OUTPUT "A: " & NUM_TO_STRING(CountA)`
	`   OUTPUT "E: " & NUM_TO_STRING(CountE)`
	`   OUTPUT "I: " & NUM_TO_STRING(CountI)`
	`   OUTPUT "O: " & NUM_TO_STRING(CountO)`
	`   OUTPUT "U: " & NUM_TO_STRING(CountU)`
	
	`ENDPROCEDURE`
		5

Question	Answer	Marks

3(a)	1 mark each to max 8
	1 declaration of appropriate constants for weight // declaration and initialisation of appropriate variable to count cases for the flight
	2 open the file `"HOLD-CARGO.txt"` in `READ` mode **and** close the file
	3 conditional loop until end of file …
	4 … read a line from the file
	5 Extract flight number from **each line in file**..
	6 … compare to **parameter**
	7 Extract weight from **each line in file**
	8 … convert to integer
	9 … check if extracted weight > 50
	10 If correct flight **and** over weight, extract and output Case ID
	11 If correct flight **and** counter for flight is over 300, extract and output Case ID …
	12 … otherwise increment a counter for that flight
	
	`PROCEDURE CheckWeight(FlightNo: STRING)`
	   `CONSTANT FileName = "HOLD-CARGO.txt"`
	   `DECLARE CaseCounter : INTEGER`
	   `DECLARE FlightData, CaseID : STRING`
	   `CaseCounter ← 0`
	   `OPENFILE FileName FOR READ`
	   `WHILE NOT EOF(FileName)`
	     `READFILE FileName, FlightData`
	     `IF LEFT(FlightData, 5) = FlightNo`
	       `THEN`
	         `IF STRING_TO_NUM(RIGHT(FlightData,2)) <= 50 AND CaseCounter <= 300`
	           `THEN`
	              `CaseCounter ← CaseCounter + 1`
	           `ELSE`
	              `CaseID ← MID(FlightData, 6, 3)`
	              `OUTPUT CaseID & " rejected"`
	         `ENDIF`
	      `ENDIF`
	   `ENDWHILE`
	   `CLOSEFILE FileName`
	`ENDPPROCEDURE`
		8

3(b)	One mark each • One change can be reflected throughout the program • The value of the constant cannot be accidentally changed	**2**

Question	Answer	Marks
3(c)(i)	One mark each to max 2 e.g. • Called from several places / reusability • Reduces the length of the overall program code • Less chance of errors as do not need to re-write / re-test • One change in function will be applied in all places used • Can use in multiple programs without rewriting • Can share amongst other programmers to avoid everyone rewriting	**2**
3(c)(ii)	One mark each to max 2 • Allows the use of functions that are difficult to code • They (should) have been more extensively tested // Reduce the time to test your code • Reduce the time to write	**2**
3(d)	One mark for name, two marks for description. Name: • By value Description: • (Copy of) value is passed • Any local changes made are lost when the module terminates // does not overwrite structure being passed	**3**

Question	Answer	Marks
4(a)	1 mark for each underlined part of the pseudocode.	**4**

```
PROCEDURE SafetyCheck()
   DECLARE Count : INTEGER
   DECLARE Index : INTEGER
   CONSTANT TreeCount = 20
   Count ← 0
   FOR Index ← 1 TO TreeCount // 20
      IF TreeAngle[Index] > 36
        THEN
          Count ← Count + 1
      ENDIF
   ENDFOR
   IF Count <= MainTrigger
     THEN
       OUTPUT "Maintenance not needed"
     ELSE
        OUTPUT "Maintain " & NUM_TO_STRING(Count) &
               " trees"
   ENDIF
ENDPROCEDURE
```

Question	Answer	Marks

4(b)	1 mark for each to max 7
	
	1 Declarations of variable/constant/data structures have appropriate data types
	2 Procedure CheckTree taking an integer parameter
	3 Prompt **and** input new angle
	4 … attempt at validation of new angle
	5 Loop 20 times …
	6 … compare TreeAngle[loop counter, 1] with parameter …
	7 … if found, store input value in TreeAngle[loop counter, 2]
	8 … if found, compare new angle to 36
	9 … and check if different to previous angle (one >36 and one is <= 36)
	10 If parameter found (and angle changed), output message saying safety status has changed
	11 If parameter found, output message with reference number and "No match"
	12 If parameter not found in array display a suitable message
	Example:
	`PROCEDURE CheckTree(TreeRef : INTEGER)`
	
	`   DECLARE Index : INTEGER`
	`   DECLARE PreviousAngle, Angle : INTEGER`
	`   DECLARE PreviousStatus, NewStatus: STRING`
	`   DECLARE Found : BOOLEAN`
	
	`   CONSTANT TreeCount = 20`
	`   CONSTANT SafeLimit = 36`
	
	`   Found ← FALSE`
	
	`   FOR Index ← 1 TO TreeCount`
	`      IF TreeAngle[Index, 1] = TreeRef`
	`        THEN`
	`          Found ← TRUE`
	`          PreviousAngle ← TreeAngle[Index, 2]`
	`          OUTPUT "Tree angle: "`
	`          INPUT Angle`
	`          TreeAngle[Index, 2] ← Angle`
	
	`          IF PreviousAngle <= SafeLimit`
	`             THEN`
	`                PreviousStatus ← "SAFE"`
	`             ELSE`
	`                PreviousStatus ← "UNSAFE"`
	`          ENDIF`
	`          IF Angle <= SafeLimit`
	`             THEN`
	`                NewStatus ← "SAFE"`
	`             ELSE`
	`                NewStatus ← "UNSAFE"`
	`          ENDIF`
		7

Question	Answer	Marks

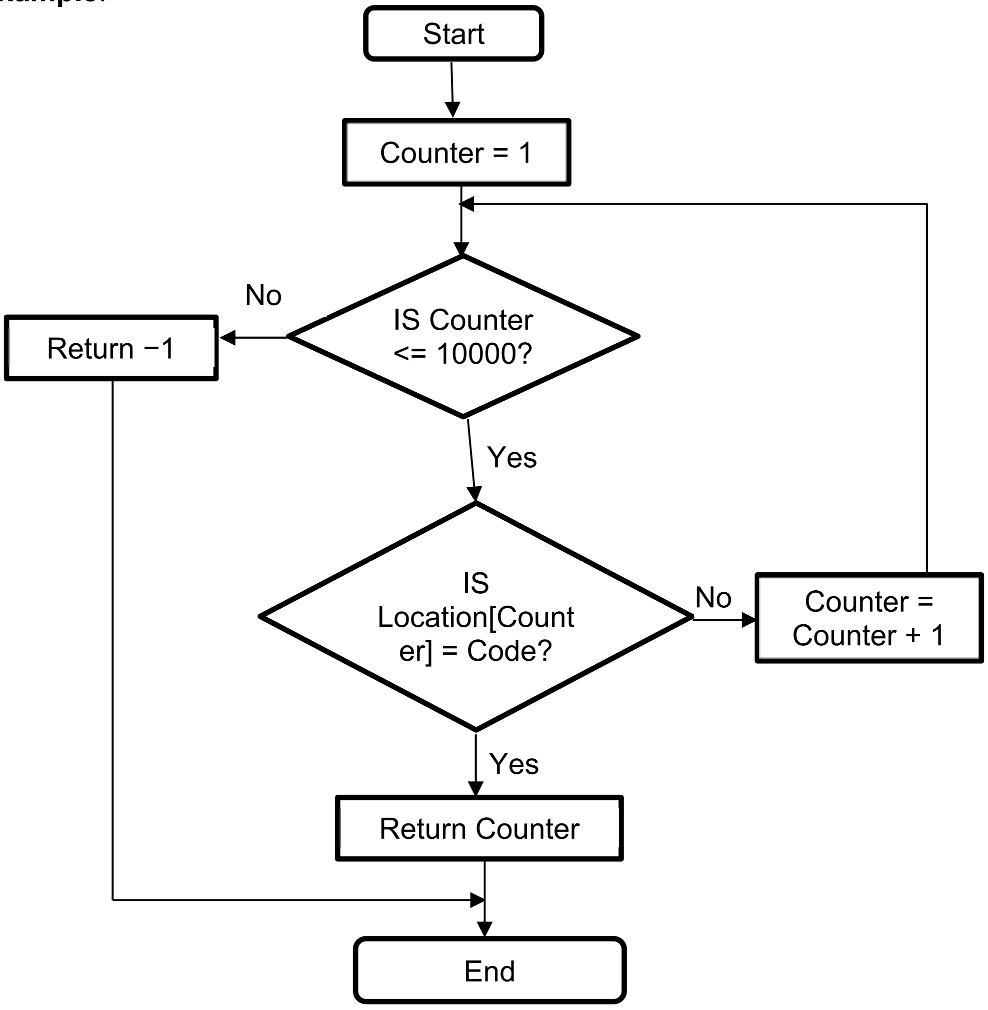
4(b)	
	```
	         // check if safety status has changed
	         IF PreviousStatus <> NewStatus
	           THEN
	             OUTPUT "Safety status has changed"
	         ENDIF

	      ENDIF
	   ENDFOR

	   // output "No match" if not found
	   IF Found = FALSE
	      THEN
	         OUTPUT NUM_TO_STRING(TreeRef) & " No match"
	   ENDIF

	ENDPROCEDURE
	```

Question	Answer	Marks
5(a)	One mark each to max 2 • Shows module hierarchy / relationships • Shows parameters passed between modules • Shows module names • Shows sequence of the modules	**2**
5(b)	One mark for each row.	**5**

Parameter identifier	Parameter letter
Quantity	C // D
BookingID	A
ItemCost	D // C
TotalCost	E
BookingDate	B

Question	Answer	Marks

6(a)	One mark each:
	• Location declared as array, 10 000 elements of type string
	• Loops 10000 times …
	• … assign each index `"22+VV"`
	
	Pseudocode solution:
	`DECLARE Location : ARRAY [1:10000] OF STRING`
	`DECLARE Index : INTEGER`
	
	`FOR Index ← 1 TO 10000`
	`   Location[Index] ← "22+VV"`
	`ENDFOR`
		3

6(b)	One mark each: • loop 10 000 times • compare variable with each Location index • if variable found in array, return index (stop loop) • not found after checking **all** records, return −1 **Example**:  Start Counter = 1 IS Counter <= 10000? No Return −1 Yes IS Location[Counter] = Code? No Counter = Counter + 1 Yes Return Counter End	**4**

Question	Answer	Marks

6(c)	1 mark for each to max 6
	1 Function heading **and** ending (where appropriate) including **two** parameters (string and integer)
	2 Loop until end of message (or " " or "." found)
	3 Extract the character at the integer parameter start position
	4 Compare **each** character to " " and "."
	5 … if equal, break out of loop and return
	6 … extracting geocode
	7 Returning the **extracted** geocode
	
	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`FUNCTION RetrieveCode(EmailMsg : STRING,`
	`                      StartPos : INTEGER) RETURNS STRING`
	
	`   DECLARE Index : INTEGER`
	`   DECLARE GeoCode : STRING`
	`   DECLARE NextChar : CHAR`
	`   DECLARE EndOfGeoCode : BOOLEAN`
	
	`   Index ← StartPos`
	`   GeoCode ← ""`
	`   EndOfGeoCode ← FALSE`
	
	`   WHILE Index <= LENGTH(EmailMsg) AND`
	`         EndOfGeoCode = FALSE`
	`      NextChar ← MID(EmailMsg, Index, 1)`
	`      IF (NextChar = ' ' OR NextChar = '.')`
	`        THEN`
	`          EndOfGeoCode ← TRUE`
	`        ELSE`
	`          GeoCode ← GeoCode & NextChar`
	`      ENDIF`
	`      Index ← Index + 1`
	`   ENDWHILE`
	
	`   RETURN GeoCode`
	
	`ENDFUNCTION`
		6

Program Code Example Solutions

Q4 (b): Visual Basic

```
        Sub CheckTree(TreeRef As Integer)
        Dim Index As Integer
        Dim PreviousAngle, Angle As Integer
        Dim PreviousStatus, NewStatus As String
        Dim Found As Boolean

        Const TREECOUNT = 20
        Const SAFELIMIT = 36

        Found = False

        For Index = 1 To TREECOUNT
            If TreeAngle(Index, 0) = TreeRef Then
                Found = True
                PreviousAngle = TreeAngle(Index, 1)
                Console.Write("Tree angle: ")
                Angle = Console.ReadLine()
                TreeAngle(Index, 2) = Angle

                If PreviousAngle <= SAFELIMIT Then
                    PreviousStatus = "SAFE"
                Else
                    PreviousStatus = "UNSAFE"
                End If

                If Angle <= SAFELIMIT Then
                    NewStatus = "SAFE"
                Else
                    NewStatus = "UNSAFE"
                End If

                ' check if safety status has changed
                If PreviousStatus <> NewStatus Then
                    Console.WriteLine("Safety status has changed")
                End If
            End If
        Next

        If Found = False Then
            Console.WriteLine(CStr(TreeRef) & " No match")
        End If
    End Sub
```

Q4 (b): Pascal

```
procedure CheckTree(TreeRef: integer);
const
   TREECOUNT = 20;
   SAFELIMIT = 36;
var
   Index: integer;
   PreviousAngle, Angle: integer;
   PreviousStatus, NewStatus: string;
   Found: boolean;
begin
   Found := false;
   for Index := 1 to TREECOUNT do
   begin
      if TreeAngle[Index,0] = TreeRef then
      begin
         Found := True;
         PreviousAngle := TreeAngle[Index, 1];
         write ('Tree angle: ');
         readln(Angle);
         TreeAngle[Index, 1] := Angle;

         if PreviousAngle <= SAFELIMIT then
            PreviousStatus := 'SAFE'
         else
            PreviousStatus := 'UNSAFE';
         if Angle <= SAFELIMIT then
            NewStatus := 'SAFE'
         else
            NewStatus := 'UNSAFE';

         // check if safety status has changed
         if PreviousStatus <> NewStatus then
            writeln('Safety status has changed');
      end;
   end; //for

   // output "No match" if not found
   if Found = False then
      writeln(TreeRef,' No match');
end;
```

Q4 (b): Python

```
def CheckTree(TreeRef):
    #DECLARE Index : INTEGER
    #DECLARE PreviousAngle, Angle : INTEGER
    #DECLARE PreviousStatus, NewStatus: STRING
    #DECLARE Found : BOOLEAN

    TREECOUNT = 20
    SAFELIMIT = 36

    Found = False

    for Index in range(1, TREECOUNT):
        if TreeAngle[Index][0] == TreeRef:
            Found = True
            PreviousAngle = TreeAngle[Index][1]
            Angle = int(input("Tree angle:"))
            TreeAngle[Index][1] = Angle

            if PreviousAngle <= SAFELIMIT:
                PreviousStatus = "SAFE"
            else:
                PreviousStatus = "UNSAFE"

            if Angle <= SAFELIMIT:
                NewStatus = "SAFE"
            else:
                NewStatus = "UNSAFE"

            #check if safety status has changed
            if PreviousStatus != NewStatus:
                print("Safety status has changed")

    #output "No match" if not found
    if Found == False:
        print(str(TreeRef) + " No match")
```

Q6 (c): Visual Basic

```
Function RetrieveCode(EmailMsg As String, StartPos As Integer) As String
    Dim Index As Integer
    Dim GeoCode As String
    Dim NextChar As Char
    Dim EndOfGeoCode As Boolean

    Index = StartPos
    GeoCode = ""
    EndOfGeoCode = False

    Do While (Index <= EmailMsg.Length) And (EndOfGeoCode = False)
        NextChar = EmailMsg.SubString(Index, 1)
        If NextChar = " " Or NextChar = "." Then
            EndOfGeoCode = True
        Else
            GeoCode = GeoCode & NextChar
        End If
        Index = Index + 1
    Loop
    Return GeoCode
End Function
```

Q6 (c): Pascal

```
function RetrieveCode(EmailMsg: string; StartPos: integer): string;
var
   Index: integer;
   GeoCode: string;
   NextChar: string[1]; //char
   EndOfGeoCode: boolean;

begin
   Index := StartPos;
   GeoCode := '';
   EndOfGeoCode := False;

   while (Index<=Length(EmailMsg)) and (EndOfGeoCode=False) do
   begin
      NextChar := MidStr(EmailMsg, Index, 1);
      if (NextChar=' ') or (NextChar='.') then
         EndOfGeoCode := True
      else
         GeoCode := GeoCode + NextChar;
      Index := Index + 1;
   end;
   RetrieveCode := GeoCode;
end;
```

Q6 (c): Python

```
def RetrieveCode(EmailMsg, StartPos):
    #DECLARE Index : INTEGER
    #DECLARE GeoCode : STRING
    #DECLARE NextChar : CHAR
    #DECLARE EndOfGeoCode : BOOLEAN

    Index = StartPos
    GeoCode = ""
    EndOfGeoCode = False

    while Index <= len(EmailMsg) and EndOfGeoCode == False:
        NextChar = EmailMsg[Index:Index+1]
        if NextChar == " " or NextChar == ".":
            EndOfGeoCode = True
        else:
            GeoCode = GeoCode + NextChar
        Index += 1

    return GeoCode
```