



Cambridge International AS & A Level

COMPUTER SCIENCE

9608/21

Paper 2 Fundamental Problem-solving and Programming Skills

May/June 2021

MARK SCHEME

Maximum Mark: 75

Published

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

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

Cambridge International is publishing the mark schemes for the May/June 2021 series for most Cambridge IGCSE™, Cambridge International A and AS Level components and some Cambridge O Level components.

This document consists of **17** 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)(i)	One mark for each part statement: <ul style="list-style-type: none"> • Each character is assigned... • ...a unique value • ...using 7 bits Max 2 marks	2												
1(a)(ii)	One mark for two correct, 2 marks for all correct <table border="1" data-bbox="478 589 1153 976" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Memory location</th> <th>ASCII character value</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">100</td> <td style="text-align: center;">70</td> </tr> <tr> <td style="text-align: center;">101</td> <td style="text-align: center;">65</td> </tr> <tr> <td style="text-align: center;">102</td> <td style="text-align: center;">68</td> </tr> <tr> <td style="text-align: center;">103</td> <td style="text-align: center;">69</td> </tr> <tr> <td style="text-align: center;">104</td> <td style="text-align: center;">68</td> </tr> </tbody> </table>	Memory location	ASCII character value	100	70	101	65	102	68	103	69	104	68	2
Memory location	ASCII character value													
100	70													
101	65													
102	68													
103	69													
104	68													
1(b)(i)	One mark for both answers correct. Exact terms only. <ul style="list-style-type: none"> • Lower bound • Upper bound 	1												
1(b)(ii)	index / subscript	1												
1(c)	One mark for each error <table border="1" data-bbox="272 1346 1358 1879" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Statement</th> <th>Error</th> </tr> </thead> <tbody> <tr> <td>Code ← LEFT("Cat", 4)</td> <td>Only 3 characters in string</td> </tr> <tr> <td>Status ← MID("Aardvark", 0, 5)</td> <td>Second parameter should start from 1</td> </tr> <tr> <td>Size ← LENGTH("Password)</td> <td>Missing closing quote / Opening quote should be removed</td> </tr> <tr> <td>Stock[n] ← Stock[n+1]</td> <td>NO ERROR / n may not be integer value / n out of bound</td> </tr> <tr> <td>Result ← 3 OR 4</td> <td>Not Boolean types</td> </tr> </tbody> </table>	Statement	Error	Code ← LEFT("Cat", 4)	Only 3 characters in string	Status ← MID("Aardvark", 0, 5)	Second parameter should start from 1	Size ← LENGTH("Password)	Missing closing quote / Opening quote should be removed	Stock[n] ← Stock[n+1]	NO ERROR / n may not be integer value / n out of bound	Result ← 3 OR 4	Not Boolean types	5
Statement	Error													
Code ← LEFT("Cat", 4)	Only 3 characters in string													
Status ← MID("Aardvark", 0, 5)	Second parameter should start from 1													
Size ← LENGTH("Password)	Missing closing quote / Opening quote should be removed													
Stock[n] ← Stock[n+1]	NO ERROR / n may not be integer value / n out of bound													
Result ← 3 OR 4	Not Boolean types													

Question	Answer	Marks
2	<p>Mark as follows:</p> <p>First mark for START, Initialisation of Index and END</p> <p>Then one mark per area outlined, in correct place.</p> <p>At least one decision box label (YES / NO) must be present.</p> <pre> graph TD Start([START]) --> SetIndex1[SET Index TO 1] SetIndex1 --> Dec1{IS Index <= 26?} Dec1 -- YES --> SetCharCount[SET CharCount[Index] TO 0] SetCharCount --> IncIndex1[INCREMENT Index] IncIndex1 --> Dec1 Dec1 -- NO --> SetIndex2[SET Index TO 1] SetIndex2 --> Dec1 Dec1 --> Dec2{IS Index <= LENGTH(Msg)?} Dec2 -- YES --> SetThisChar1[SET ThisChar TO MID(Msg, Index, 1)] SetThisChar1 --> SetThisChar2[SET ThisChar TO LCASE(ThisChar)] SetThisChar2 --> Dec3{IS ThisChar >= 'a' AND ThisChar <= 'z'?} Dec3 -- YES --> SetThisIndex[SET ThisIndex TO ASC(ThisChar) - 96] SetThisIndex --> IncCharCount[INCREMENT CharCount[ThisIndex]] IncCharCount --> Dec2 Dec3 -- NO --> Dec2 Dec2 -- NO --> End([END]) </pre>	5

Question	Answer	Marks
2(b)	<p>One mark per point.</p> <p>Algorithm should mention:</p> <ol style="list-style-type: none"> 1 Initialise variable to hold Max value 2 Loop through 26 elements of array 3 Test if element > Maxand if so set new Max value 4 Method of checking for duplicates 5 Output a message giving alphabetic char with largest count value - needs use of CHR () 6 Output a suitable message if targets count value is shared 	6

Question	Answer	Marks
3(a)	<div style="text-align: center;"> </div> <p>1 mark for each of:</p> <ol style="list-style-type: none"> 1 Iteration arrow 2 Selection diamond 3 Both sets of parameters from Module_X 4 Parameter ByReference to Module_YA 5 Parameter (ByValue) and return Boolean from Module_YB 	5
3(b)(i)	<p>One mark for each statement:</p> <ul style="list-style-type: none"> • it is a function • because it returns a value 	2

Question	Answer	Marks
3(b)(ii)	<p><u>PROCEDURE</u> Module <u>ZB</u> (BYVALUE ParX : REAL, BYREF ParZ : STRING)</p> <p>One mark for:</p> <ul style="list-style-type: none">• Procedure declaration• ParX : REAL and ParZ : STRING• ByRef for ParZ <p>Condone missing BYVALUE for ParX</p>	3

Question	Answer	Marks
4(a)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language solutions appear in the Appendix.</p> <pre> PROCEDURE ScanArray(SearchString STRING) DECLARE Index, Total : INTEGER DECLARE Error : BOOLEAN Index ← 1 Total ← 0 Error ← FALSE WHILE Index <= 1000 AND Error <> TRUE IF LENGTH(ThisArray[Index]) > 5 THEN IF LEFT(ThisArray[Index], 4) = SearchString THEN Total ← Total + LENGTH(ThisArray[Index]) - 5 ENDIF Index ← Index + 1 ELSE Error ← TRUE ENDIF ENDWHILE IF Index > 1 THEN ArrayResult ← INT(Total / (Index - 1)) ENDIF ENDPROCEDURE </pre> <p>Mark as follows:</p> <ol style="list-style-type: none"> 1 Procedure header including parameter and end (where required) 2 Local variable declarations and initialisation of Index, Total and Error but no local declaration of ArrayResult 3 WHILE / ENDWHILE loop 4 Nested IF statement comparing first four character of array element with SearchString 5 Summation of Total using appropriate LENGTH function and subtracting 5 6 Assignment to ArrayResult using appropriate INT function AND check for division by zero 	6
4(b)	<p>One mark for each:</p> <ul style="list-style-type: none"> • The IDE displays hints / choice of keywords / available identifiers • (Appropriate to) the current cursor position / insertion point 	2

Question	Answer	Marks
4(c)(i)	<p>One mark for Name, max 2 for Tasks (one per underlined term):</p> <p>Name: Design Tasks: To define the <u>data structures</u> and <u>algorithms</u> (of the solution)</p> <p>ALTERNATIVE</p> <p>Name: Analysis Tasks: Feasibility study // Problem definition / investigation // Requirement spec</p>	3
4(c)(ii)	Coding / Implementation / Programming	1

Question	Answer	Marks
5(a)(i)	<pre> PROCEDURE GuessNum() DECLARE Count: INTEGER DECLARE RndNumber : INTEGER DECLARE MyGuess : INTEGER RndNumber ← 1 + INT(RAND(20)) Count ← 1 REPEAT OUTPUT "Input your guess" INPUT MyGuess IF MyGuess <> RndNumber THEN Count ← Count + 1 OUTPUT "Incorrect - try again" ENDIF UNTIL MyGuess = RndNumber OUTPUT "You took ", Count, " guesses." ENDPROCEDURE </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> 1 Use of RAND () to generate an integer between 1 and 20 2 Conditional loop until random number is guessed 3 Prompt and input a guess ...in a loop 4 Comparison and increment Count and 'Try again' output message...in a loop 5 Final output message...not in a loop 	5

Question	Answer	Marks
5(a)(ii)	<p>One mark per point.</p> <p>Check for:</p> <ul style="list-style-type: none"> Integer / number out of range <1 OR > 20 Real number entered Non-numeric value entered <p>Max 2 marks</p>	2
5(b)(i)	<ul style="list-style-type: none"> Stub testing 	1
5(b)(ii)	<p>One mark for each:</p> <ul style="list-style-type: none"> A simplified version of <code>Status()</code> / a dummy function is written that returns a typical / expected value. 	2
5(b)(iii)	A compiler is used to translate / convert the source code / program / high-level language code into <u>object code</u> / <u>machine code</u> / <u>an executable file</u>	1

Question	Answer	Marks
6(a)	<p>'Pseudocode' solution included here for development and clarification of the mark scheme. Programming language example solutions appear in the Appendix.</p> <pre> FUNCTION Check (Index : INTEGER) RETURNS BOOLEAN IF LENGTH (StockID[Index]) <> 8 OR ___ Description[Index]) = "" OR ___ Quantity[Index] < 0 THEN RETURN FALSE ELSE RETURN TRUE ENDIF ENDFUNCTION </pre> <p>One mark for each of the following:</p> <ol style="list-style-type: none"> Function heading and ending (where appropriate) Three comparisons... ... connected by logical OR // AND / correct nested IF RETURN value in both cases 	4

Question	Answer	Marks
6(b)	<pre> FUNCTION Backup() RETURNS BOOLEAN DECLARE Index : INTEGER DECLARE FileName, FileLine : STRING DECLARE ALLOK : BOOLEAN CONSTANT ASTERISK = '*' ALLOK ← TRUE FileName ← GetValidFileName() OPENFILE FileName FOR WRITE OPENFILE "ERRORLOG.TXT" FOR WRITE FOR Index ← 1 TO 10000 IF StockID[Index] <> "" THEN FileLine ← StockID[Index] & ASTERISK FileLine ← FileLine & Description[Index] & ASTERISK FileLine ← FileLine & NUM_TO_STRING(Quantity[Index]) & ASTERISK FileLine ← FileLine & NUM_TO_STRING(Cost[Index]) WRITEFILE FileName, FileLine //now check for sensible data IF Check(Index) <> TRUE THEN WRITEFILE, "ERRORLOG.TXT", FileLine ALLOK ← FALSE ENDIF ENDIF ENDFOR CLOSEFILE FileName CLOSEFILE "ERRORLOG.TXT" RETURN ALLOK ENDFUNCTION </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> 1 Declare local variable for backup filename and index 2 Call to function GetValidFileName() 3 OPEN and CLOSE both files 4 Loop all 10 000 elements 5 Form FileLine using at least one correct array index expression and asterisk 6 Use of NUM_TO_STRING() to convert at least one of QUANTITY or COST 7 Write line to backup file 8 Call Check() to determine whether values are valid and if not, write to ERRORLOG.TXT 9 Return ALLOK <p>Max 8 marks from possible 10 mark points</p>	8

Question	Answer	Marks
6(c)	<p>'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.</p> <pre> PROCEDURE Unpack(Index : INTEGER, FileLine : STRING) DECLARE Pointer : INTEGER DECLARE NextChar : CHAR DECLARE TempString : STRING CONSTANT ASTERISK = '*' StockID[Index] ← LEFT(FileLine, 8) // the only fixed length field Pointer ← 10 // Point to start of Description (skip the '*') NextChar ← MID(FileLine, Pointer, 1) TempString ← "" WHILE NextChar <> ASTERISK TempString ← TempString & NextChar Pointer ← Pointer + 1 NextChar ← MID(FileLine, Pointer, 1) ENDWHILE Description[Index] ← TempString Pointer ← Pointer + 1 NextChar ← MID(FileLine, Pointer, 1) TempString ← "" WHILE NextChar <> ASTERISK TempString ← TempString & NextChar Pointer ← Pointer + 1 NextChar ← MID(FileLine, Pointer, 1) ENDWHILE Quantity[Index] ← STRING_TO_NUM(TempString) TempString ← RIGHT(FileLine, LENGTH(FileLine) - Pointer) Cost[Index] ← STRING_TO_NUM(TempString) ENDFUNCTION </pre> <p>1 mark for each of the following:</p> <ol style="list-style-type: none"> 1 Procedure heading with parameters 2 Extract first 8 chars of FileLine 3 Assign to StockID 4 Search for asterisk for place separator 5 Extract Description string and assign to Description array 6 Extract Quantity string, and assign to Quantity array... 7 Extract Cost string and assign to Cost array 8 Type conversion for Cost and Quantity 	8

*** End of Mark Scheme – example program code solutions follow ***

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

```
Sub ScanArray(SearchString As String)

    Dim Index, Total As Integer
    Dim Error As Boolean

    Index = 1
    Total = 0
    Error = FALSE

    While Index <= 1000 And Error <> TRUE
        If Len(ThisArray(Index)) > 5 Then
            If Left(ThisArray(Index), 4) = SearchString Then
                Total = Total + Len(ThisArray(Index)) - 5
            End If
            Index = Index + 1
        Else
            Error = TRUE
        End If
    End While

    If Index > 1 Then
        ArrayResult = Int(Total / (Index - 1))
    End If

End Sub
```

Q4 (a): Pascal

```
procedure ScanArray(SearchString : string);

var
  Index, Total : integer;
  Error : boolean;

begin
  Index := 1;
  Total := 0;
  Error := FALSE;

  while Index <= 1000 And Error <> TRUE do
    begin
      if Length(ThisArray[Index]) > 5 then
        begin
          if LeftStr(ThisArray[Index], 4) = SearchString then
            Total := Total + Length(ThisArray[Index]) - 5;

            Index := Index + 1;
          else
            Error := TRUE;
          end;
        end;

      if Index > 1 then
        ArrayResult := int(Total / (Index - 1));
      end;
    end;
  end;
```

Q4(a): Python

```
def ScanArray(SearchString):

    ## Index, Total As Integer
    ## Error As Boolean

    Index = 1
    Total = 0
    Error = FALSE

    while Index <= 1000 and Error <> TRUE:
        if len(ThisArray[Index]) > 5:
            ThisElement = ThisArray[Index]
            if ThisElement[:4] == SearchString:
                Total = Total + len(ThisArray[Index]) - 5
                Index = Index + 1
            else:
                Error = TRUE

    if Index > 1:
        ArrayResult = int(Total / (Index - 1))
```

Q6 (a): Visual Basic

```
Function Check(Index As Integer) As Boolean

    If Len(StockID(Index)) <> 8 Or _
        Description(Index) = "" Or _
        Quantity(Index) < 0 Then
        Return FALSE
    Else
        Return TRUE
    End If

End Function
```

Q6(a): Pascal

```
function Check(Index : Integer) : boolean;

begin

    if Length(StockID[Index]) <> 8 Or
        Description[Index] = "" Or
        Quantity[Index] < 0 then
        Check := FALSE // result := FALSE
    else
        Check := TRUE // result := TRUE
    end;

end
```

Q6(a): Python

```
def Check(Index):

    if len(StockID[Index]) <> 8 or \
        Description[Index] == "" or \
        Quantity[Index] < 0:
        return FALSE
    else:
        return TRUE
```

Q6(c): Visual Basic

```

Sub Unpack(Index As Integer, FileLine As String)

    Dim Pointer As Integer
    Dim NextChar As Char
    Dim TempString As String
    Const ASTERISK = '*'

    StockID(Index) = LEFT(FileLine, 8)

    Pointer = 10          'point to start of Description (skip the '*')
    NextChar = Mid(FileLine, Pointer, 1)
    TempString = ""
    While NextChar <> ASTERISK
        TempString = TempString & NextChar
        Pointer = Pointer + 1
        NextChar = Mid(FileLine, Pointer, 1)
    End While

    Description(Index) = TempString
    Pointer = Pointer + 1
    NextChar = Mid(FileLine, Pointer, 1)

    TempString = ""
    While NextChar <> ASTERISK
        TempString = TempString & NextChar
        Pointer = Pointer + 1
        NextChar = Mid(FileLine, Pointer, 1)
    End While

    Quantity(Index) = CInt(TempString)
    TempString = Right(FileLine, Len(FileLine) - Pointer)
    Cost(Index) = CDec(TempString)

End Sub

```

Q6(c): Pascal

```

procedure Unpack(Index : Integer, FileLine : String);

var
    Pointer : integer;
    NextChar : char;
    TempString : string;

const
    ASTERISK = '*';

begin
    StockID[Index] := LeftStr(FileLine, 8);

    Pointer := 10;          //point to start of Description (skip the '*')
    NextChar := MidStr(FileLine, Pointer, 1);
    TempString := "";
    while NextChar <> ASTERISK do

```

```

begin
  TempString := TempString & NextChar;
  Pointer := Pointer + 1;
  NextChar := MidStr(FileLine, Pointer, 1);
end;

Description[Index] := TempString;
Pointer := Pointer + 1;
NextChar := MidStr(FileLine, Pointer, 1);

TempString := "";
while NextChar <> ASTERISK do
begin
  TempString := TempString & NextChar;
  Pointer := Pointer + 1;
  NextChar := MidStr(FileLine, Pointer, 1);
end;

Quantity[Index] := StrToInt(TempString);
TempString := RightStr(FileLine, Length(FileLine) - Pointer);
Cost[Index] := StrToFloat(TempString);

end;

```

Q6(c): Python

```

def Unpack(Index, FileLine):

    ## Pointer As Integer
    ## NextChar As Char
    ## TempString As String
    ASTERISK = '*'

    StockID[Index] = FileLine[:8] #characters 0 to 7

    Pointer = 9          #point to start of Description (skip the '*')
    NextChar = FileLine[Pointer]
    TempString = ""

    while NextChar <> ASTERISK:
        TempString = TempString + NextChar
        Pointer = Pointer + 1
        NextChar = FileLine[Pointer]

    Description[Index] = TempString
    Pointer = Pointer + 1
    NextChar = FileLine[Pointer]

    TempString = ""

    while NextChar <> ASTERISK:
        TempString = TempString + NextChar
        Pointer = Pointer + 1
        NextChar = FileLine[Pointer]

    Quantity[Index] = int(TempString)
    TempString = FileLine[Len(FileLine) - Pointer - 1:]

```

```
Cost[Index] = float(TempString)
```

Alternative

```
def Unpack(Index, FileLine):  
    ## TempString As String  
  
    StockID[Index] = FileLine[:8]  
    TempString = FileLine[8:] // remove first 8 characters  
    Description[Index], Quantity[Index], Cost[Index] =  
    (TempString.split('*'))
```