Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/23**

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.

© UCLES 2021

[Turn over

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.

© UCLES 2021

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>One mark per bullet point.

• INPUT
• Example input statement in language stated

• PROCESS
• Example process statement in language stated

• Example OUTPUT statement in language stated</td><td>5</td></tr>
<tr><td>1(b)(i)</td><td>One mark per bullet point.

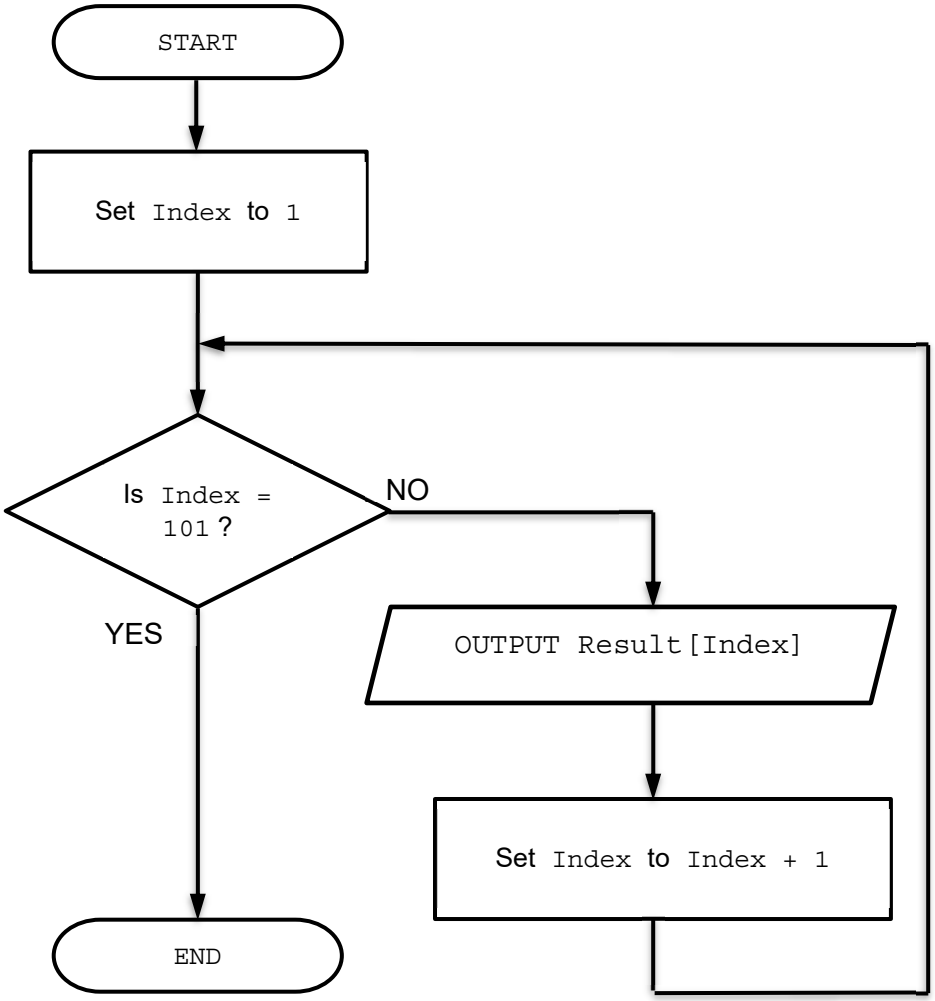
• conditional loop
• the number of iterations is not known</td><td>2</td></tr>
<tr><td>1(b)(ii)</td><td>One mark per bullet point.

• the value is found
• the end of the array is reached (and value not found)</td><td>2</td></tr>
<tr><td>1(c)</td><td><table><tr><th>Statement</th><th>Error</th></tr><tr><td><code>Code ← RIGHT("Cap" * 3, 2)</code></td><td>Cannot multiply a string (by 3)</td></tr><tr><td><code>Valid ← IS_NUM(3.14159)</code></td><td>Parameter should be a string</td></tr><tr><td><code>NextChar ← MID(ThisString, Index), 1</code></td><td>Closing bracket in wrong place</td></tr></table>One mark for each row</td><td>3</td></tr>
</table>

Question	Answer	Marks
2(a)	• (A program fault is) when the program does not do what it is supposed to do / expected to do • … under certain circumstances One mark per point or equivalent	**2**
2(b)(i)	Makes it easier to understand the purpose of each identifier / what the identifier is used for / the purpose of the program	**1**

Question	Answer	Marks
2(b)(ii)	One mark per point: 1 The use of modular programming (to avoid repeated code) 2 The use of library / tried and tested subroutines 3 Good formatting to make the code easier to read (indentation, white space) 4 Use of local variables 5 Use of constants 6 Use of comments to explain functionality of code **Max 3 marks**	**3**
2(c)	One mark per point: • white box • dry-run testing / use of trace table / walk through **Max 1 mark**	**1**

Question	Answer	Marks
3(a)	• To break the problem down into sub-tasks • where each sub-task can be implemented by a program module / is easier to solve. One mark for each phrase (or equivalent)	**2**

Question	Answer	Marks
3(b)	 1 mark for each of the following: 1 Initialise Index 2 Test index for 100 elements 3 End when 100 elements output 4 Output, increment and repeat	**4**

Question	Answer	Marks

3(c)	`OnLine ← FALSE`
	
	`WHILE Online = FALSE`
	`  IF Active = TRUE`
	`  THEN`
	`  Call Sync()`
	`  ELSE`
	`  Call Reset()`
	`  IF Active = FALSE`
	`  THEN`
	`  Call Error("No Signal")`
	`  ELSE`
	`  Online ← TRUE`
	`  ENDIF`
	`  ENDIF`
	`  Call ReCheck()`
	`ENDWHILE`
	
	1 mark for each of the following:
	
	1 Initialise `Online`
	2 `WHILE .. ENDWHILE` loop, terminated when `Online = TRUE`
	3 `IF Active = TRUE THEN .. ELSE .. ENDIF`
	4 **Nested** `IF Active = FALSE THEN .. ELSE .. ENDIF`
	5 Call `Sync()` **and** Call `Reset()` **and** Call `Error()` **and** assignment to `Online` in appropriate place in pseudocode
	6 Final call to `ReCheck()` in appropriate place
		6

Question	Answer	Marks
4(a)(i)		**5**

Result	Count	Index	NextChar
0	1		
1		1	‘7’
2		2	‘4’
3		3	‘.’
4		4	‘0’
		5	‘,’
	2		
5		6	‘4’
6		7	‘.’
7		8	‘6’
		9	‘,’
	3		
8		10	‘3’
-1		11	‘x’
0		12	‘2’

Note:
One mark per region indicated
If no marks by zone then mark by column (max 3)
Values in column 4 must be in quotes

4(a)(ii)	0 (zero) Allow FT from final value in ‘Result’ column	**1**

Question	Answer	Marks
4(b)(i)	Final `IN ... ENDIF` errors: • Test needs to check `Count` AND `Result` • Returns `Result` instead of `Count` **OR** `FOR` loop error: • The loop continues after the illegal character is detected • So the error condition (`Result = -1`) can be lost	**2**
4(b)(ii)	• Change `RETURN Result` to `RETURN Count` • Change final if to `IF Count < 3 and Result <> -1` • Terminate the loop as soon as an illegal character is encountered **Max 1 mark**	**1**
4(c)	• Syntax error • Rules of the language are not followed **OR** • Run-time error • Program performs an illegal operation or enters an infinite loop	**2**

Question	Answer	Marks

5(a)	`FUNCTION GroupNum(TelNum, Template : STRING) RETURNS STRING`
	`  DECLARE FString : STRING`
	`  DECLARE Index, ThisDigit, ThisGroup : INTEGER`
	`  CONSTANT SPACE = ' '`
	`  FString ← ""`
	`  ThisDigit ← 1`
	
	`  FOR Index ← 1 TO LENGTH(Template)`
	`  ThisGroup ← STRING_TO_NUM(MID(Template, Index, 1))`
	`  FString ← FString & MID(TelNum, ThisDigit, ThisGroup)`
	`  FString ← FString & SPACE`
	`  ThisDigit ← ThisDigit + ThisGroup`
	`  ENDFOR`
	
	`  FString ← FString & RIGHT(TelNum, LENGTH(TelNum) - ThisDigit)`
	
	`  RETURN FString`
	
	`ENDFUNCTION`
	
	Mark as follows:
	
	1 Function header and end including parameters **and** `RETURN` type
	2 Local variable declaration and initialisation of `FString` (return string)
	3 Loop to go through each char of `Template` (each group)
	4 Extract character from template **in a loop**
	5 Use of `STRING_TO_NUM()` on extracted character **in a loop**
	6 Substring statement to pick up current group from `TelNum` and concatenate with `FString` **in a loop**
	7 Concatenate `SPACE` separator **in a loop**
	8 Concatenate final characters from `TelNum` **after the loop** (+ or – 1 characters)
	9 Return the formatted string after reasonable attempt
	Max 8
	Max 7 for not fully working solutions
		8

Question	Answer	Marks
5(b)	One mark for check plus one for corresponding test data example. Test data must be invalid to prove that the check is working. **Telephone number:** • Length check // Check that the telephone number string is at least six characters e.g. number of "127" **OR** • Check it is a number // Check that the telephone number string only contains characters from '0' to '9' e.g. number of "12A" **Template:** • Check it is a number in range 1 to 5 //Check that the template string only contains characters from '1' to '5' e.g. template of "127" **OR** • Check that there are enough characters in the `TelNum` string so that the template can be applied e.g. `Telnum` = "123456", Template = "66"	**4**

Question	Answer	Marks

6(a)	`FUNCTION CheckBackupFile() RETURNS STRING`
	`  DECLARE Filename, FileLine, Response : STRING`
	
	`  Filename ← ""`
	
	`  WHILE Filename = ""`
	`  Filename ← GetValidFilename()`
	`  OPENFILE Filename FOR READ`
	`  READFILE Filename, FileLine`
	`  CLOSEFILE Filename`
	
	`  IF FileLine <> "" //check if data in file`
	`  THEN`
	`  OUTPUT "File already exists - do you want to overwrite? "`
	`  INPUT Response`
	`  IF Response <> "Yes"`
	`  THEN`
	`  Filename ← ""`
	`  OUTPUT "Please input a different filename "`
	`  ENDIF`
	`  ENDIF`
	`  ENDWHILE`
	
	`  RETURN Filename`
	`ENDFUNCTION`
	
	One mark for each of the following:
	
	1 Conditional loop
	2 Use of `GetValidFilename()` **in a loop**
	3 `OPEN` file in `READ` mode **and** `CLOSE` **in a loop**
	4 Test if file not empty (using `EOF()` or `READ` empty string)
	5 If not empty, prompt and input (in case of a non-empty file) **in a loop**
	6 …. and process response
	7 Set loop termination condition by checking for new file or overwrite confirmed **in a loop**
	8 Return `Filename`
		8

Question	Answer	Marks

6(b)	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix
	
	`PROCEDURE GroupReport(Group : STRING)`
	
	`  DECLARE Total : REAL`
	`  DECLARE Count, Index : INTEGER`
	
	`  Total ← 0`
	`  Count ← 0`
	
	`  FOR Index ← 1 TO 10000`
	`  IF LEFT(StockID[Index], 4) = Group`
	`  THEN`
	`  Count ← Count + 1`
	`  Total ← Total + (Quantity[Index] *`
	`  Cost[Index])`
	`  ENDIF`
	`  ENDFOR`
	
	`  IF Count = 0`
	`  THEN`
	`  OUTPUT "There are no items in Group: ", Group`
	`  ELSE`
	`  OUTPUT "Group: ", Group`
	`  OUTPUT "Number of items in Group: ", Count`
	`  OUTPUT "Total value of items in Group: ", Total`
	`  ENDIF`
	
	`ENDPROCEDURE`
	
	1 mark for each of the following:
	
	1 Procedure heading and ending including input parameter
	2 Declare **and** initialise local variables for `Total` and `Count`
	3 Loop through all 10 000 elements (allow LEN(StockID)
	4 Check for required group using substring function **in a loop**
	5 ...Increment `Count` **and** sum `Total` using correct array notation
	6 Generate both sets of output as appropriate **after loop**
		6

Question	Answer	Marks

6(c)	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix
	
	`FUNCTION GroupSummary() RETURNS INTEGER`
	
	`  DECLARE Index, GroupIndex Total : INTEGER`
	`  DECLARE ThisGroup : STRING`
	
	`  Total ← 0`
	
	`  FOR Index ← 1 TO 10000`
	`  IF StockID[Index] <> ""`
	`  THEN`
	`  ThisGroup ← LEFT(StockID[Index], 4)`
	`  GroupIndex ← Lookup(ThisGroup)`
	`  IF GroupIndex = -1  //ThisGroup not found`
	`  THEN`
	`  //add new group`
	`  Summary[Total + 1] ← ThisGroup`
	`  Total ← Total + 1`
	`  ENDIF`
	`  ENDIF`
	`  ENDFOR`
	
	`  RETURN Total`
	
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Function heading and ending **and final** return of `Total`
	2 Declare **and** initialise `Total`
	3 Loop through all 10 000 elements
	4 Skip empty `StockID` **in a loop**
	5 Extract `ThisGroup` from `StockID` **and** pass to `Lookup()`
	6 … if `Lookup()` returns −1 (following reasonable attempt at MP5)
	7 …….Store `ThisGroup` to `Summary[Total]` **and** increment `Total`
		7

Program Code Example Solutions

Question 6(b): Visual Basic

```
    Sub GroupReport(Group As String)

        Dim Total As Double
        Dim Count, Index As Integer

        Total = 0
        Count = 0

        For Index = 0 To 3 '1 to 10000
            If Left(StockID(Index), 4) = Group Then
                Count = Count + 1
                Total = Total + (Quantity(Index) * Cost(Index))
            End If
        Next Index

        If Count = 0 Then
            Console.WriteLine("There are no items in Group: " & Group)
        Else
            Console.WriteLine("Group: " & Group)
            Console.WriteLine("Number of items in Group: " & Count)
            Console.WriteLine("Total value of items in Group: " & Total)
        End If

    End Sub
```

Question 6(b): Pascal

```
procedure GroupReport(Group : string);

var
  Total : real;
  Count, Index : integer;

begin
  Total := 0;
  Count := 0;

  for Index := 1 TO 10000 do
  begin
     if LeftStr(StockID[Index], 4) = Group then
     begin
        Count := Count + 1;
        Total := Total + (Quantity[Index] * Cost[Index]);
     end;
  end;

  if Count = 0 then
     writeLn('There are no items in Group: ', Group)
  else
     begin
        writeLn('Group: ', Group);
        writeLn('Number of items in Group: ',  Count);
        writeLn('Total value of items in Group: ', Total);
     end;

end;
```

Question 6(b): Python

```
def GroupReport(Group):

  ## Total As Real
  ## Count, Index As Integer
  ## ThisID As String

  Total = 0
  Count = 0

  for Index in range(1, 10001):
     ThisID = StockID[Index]
     if ThisID[:4] == Group:
        Count = Count + 1
        Total = Total + (Quantity[Index] * Cost[Index])

  if Count == 0:
     print("There are no items in Group: ", Group)
  else:
     print("Group: ", Group)
     print("Number of items in Group: ",  Count)
     print("Total value of items in Group: ", Total)
```

Question 6(c): Visual Basic

```
Function GroupSummary() As Integer

  Dim Index, GroupIndex, Total As Integer
  Dim ThisGroup As String

  Total = 0

  For Index = 1 TO 10000
     If StockID(Index) <> "" Then
        ThisGroup = Left(StockID(Index), 4)
        GroupIndex = Lookup(ThisGroup)
        If GroupIndex = -1 Then               // ThisGroup not found
           Summary(Total + 1) = ThisGroup     // Add new Group
           Total = Total + 1
        End If
     End If
  Next Index

  Return Total

End Function
```

Question 6(c): Pascal

```
function GroupSummary() : Integer;

var
  Index, GroupIndex, Total : Integer;
  ThisGroup : String;

begin
  Total := 0;

  for Index := 1 TO 10000 do
  begin
     if StockID[Index] <> "" then
     begin
        ThisGroup := LeftStr(StockID[Index], 4);
        GroupIndex := Lookup(ThisGroup);
        If GroupIndex = -1 then              //ThisGroup not found
        begin
           Summary[Total + 1] := ThisGroup;  //Add new Group
           Total := Total + 1;
        end;
     end;
  end;

  GroupSummary := Total; // result := Total;

end;
```

Question 6(c): Python

```
def GroupSummary():

  ## Index, GroupIndex, Total As Integer
  ## ThisGroup, ThisID As String

  Total = 0

  for Index in range(1, 10001):
     ThisID = StockID[Index]
     if ThisID != "":
        ThisGroup = ThisId[:4]
        GroupIndex = Lookup(ThisGroup)
        if GroupIndex == -1:                    #ThisGroup not found
           Summary[Total + 1] = ThisGroup       #Add new Group
           Total = Total + 1

  return Total
```