Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/23**

Paper 2 Written Paper **October/November 2020**

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 October/November 2020 series for most Cambridge IGCSE™, Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.

This document consists of **27** 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 bullet point The purpose is: • to express the algorithm in a level of sufficient detail // to split a large task into (smaller) sub-tasks • so that it can be programmed // so that individual tasks are easier to solve // to make the problem more manageable / understandable	**2**

1(b)	Many acceptable answers, must be four different data types together with appropriate values
	One mark per row
	
	For example:
	
	Data type — **Example data value**
	`BOOLEAN` — `FALSE`
	`CHAR` — `'!'`
	`DATE` — `01/01/01`
	`INTEGER` — `27`
	
	Note: STRING and REAL are excluded as these are given in the question.
		4

1(c)(i)	Max 1 mark, features include: • Control Structures / selection statements / iteration statements / IO statements • Modular structure (functions, procedures) • Parameters to / from subroutines • Variable declaration / assignment /data structures / OOP ref	**1**

Question	Answer	Marks
1(c)(ii)	• Transferable skill	**1**
1(d)	Max 3 marks, methods include: • IDE features: breakpoints / single stepping / watch window • Manually check program code / reading error report • Trace table / dry run / White-box testing • Use of appropriate test data • Addition of output statement to follow changes to variables	**3**

Question	Answer	Marks
2(a)	One mark per step (or equivalent): 1 Set `Total` to 0 2 Set `AGradeCount` to 0 3 Input `Mark` 4 Add `Mark` to `Total` 5 If `Mark` > 75 then increment `AGradeCount` 6 Repeat from Step 3 for 30 times 7 Output `AGradeCount` 8 Output `Total` / 30	**8**

Question	Answer	Marks
2(b)	One mark per row: **Statement** / **Error** `Code ← LEFT(3, "Europe")` — Parameters are reversed `Hour ← MID("ALARM:12:02", 7, 6)` — Third param too big (should be max 5) // string too short `Size ← LENGTH(27.5)` — Invalid type – param should be a string `Num ← INT(27/ (Count + 3)` — Missing closing bracket `Result ← "Conditional" AND "Loop"` — Wrong variable types / operator	**5**

Statement	Error
`Code ← LEFT(3, "Europe")`	Parameters are reversed
`Hour ← MID("ALARM:12:02", 7, 6)`	Third param too big (should be max 5) // string too short
`Size ← LENGTH(27.5)`	Invalid type – param should be a string
`Num ← INT(27/ (Count + 3)`	Missing closing bracket
`Result ← "Conditional" AND "Loop"`	Wrong variable types / operator

Question	Answer	Marks

2(c)	‘Pseudocode’ solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`Index ← 0`
	`Status ← FALSE`
	`WHILE Status <> TRUE`
	`   Status ← TopUp()`
	`   Index ← Index + 1`
	`ENDWHILE`
	
	`IF Index > 100`
	`   THEN`
	`      SetLevel("Super")`
	`ENDIF`
	
	Mark as follows:
	
	1 Set `Index` to 0 **and** `Status` to `FALSE`
	2 Pre-condition loop
	3 Assign value of `TopUp()` to `Status` **in a loop**
	4 Increment `Index` **in a loop**
	5 Test `Index` greater than 100 **after loop**
	6 If TRUE then Call to `SetLevel` with param `"Super"`
		6

Question	Answer	Marks

3(a)	'Pseudocode' solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`PROCEDURE BubbleSort()`
	`   DECLARE Temp : INTEGER`
	`   DECLARE NoSwaps : BOOLEAN`
	`   DECLARE Boundary, J : INTEGER`
	
	`   Boundary ← 4999`
	`   REPEAT`
	`      NoSwaps ← TRUE`
	`      FOR J ← 1 TO Boundary`
	`         IF ProdNum[J]> ProdNum[J+1]`
	`            THEN`
	`               Temp ← ProdNum[J]`
	`               ProdNum[J] ← ProdNum[J+1]`
	`               ProdNum[J+1] ← Temp`
	`               NoSwaps ← FALSE`
	`         ENDIF`
	`      ENDFOR`
	`      Boundary ← Boundary - 1`
	`   UNTIL NoSwaps = TRUE`
	
	`ENDPROCEDURE`
		7

Question	Answer	Marks
3(a)	Mark as follows, max 7 marks from 8 possible marks: 1 Procedure heading and ending 2 Conditional outer loop (may be count-controlled but if so must be >= 4999 iterations) 3 An inner loop 4 Correct range for inner loop 5 Comparison (element n with n + 1) **in a loop** 6 Swap array element **in a loop** 7 'No-Swap' mechanism: (**both** needed for mark): o Conditional outer loop including flag reset o Flag set in inner loop to indicate swap 8 Reducing Boundary **in the <u>outer</u> loop**	

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(a)</td><td>
<pre>
FUNCTION Search(SearchString : STRING) RETURNS INTEGER

   DECLARE RetVal : INTEGER
   DECLARE Index : INTEGER

   RetVal ← -1
   Index ← 1

   WHILE Index <= 100 AND RetVal = -1
      IF NameList[Index] = SearchString
         THEN
            RetVal ← Index
      ENDIF
      Index ← Index + 1
   ENDWHILE

   RETURN RetVal

ENDFUNCTION
</pre>
Mark as follows:
1 Function heading and ending including parameter
2 Declaration of integer for <code>Index</code>
3 Initialisation and increment of <code>Index</code> (implied in <code>FOR</code> loop)
4 Conditional loop // <code>FOR</code> loop with immediate <code>RETURN</code> if <code>SearchString</code> found
5 Comparison of array element with <code>SearchString</code> AND assigning just the first occurrence to <code>RetVal</code> OR setting the termination condition
6 Return <code>RetVal</code> (correctly in both cases)
</td><td>6</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(b)</td><td>• Adaptive maintenance</td><td>1</td></tr>
<tr><td>4(c)</td><td>Ma 1 mark, reasons include:

• Program doesn’t perform as expected / does not meet the original specification
• Program contains errors / bugs
• Performance / efficiency needs improving
• New hardware has been introduced</td><td>1</td></tr>
<tr><td>4(d)</td><td>One mark for each value
One mark for each explanation

<table><tr><th>Output</th><th>Explanation</th></tr><tr><td>20</td><td>A copy of the variable itself is passed</td></tr><tr><td>25</td><td>A pointer to / the address of the variable is passed</td></tr></table></td><td>4</td></tr>
<tr><td>4(e)</td><td>Max 2 marks, example answers:

• Allows the module to be called from many / multiple places // re-used
• Module code can be (independently) tested and debugged once and can then be used repeatedly
• If the module task changes the change needs to be made only once
• Reduces unnecessary code duplication
• Allows modules to be shared among many programmers / given to programmers with specific skills
• Makes the program easier to work on / debug / test / etc</td><td>2</td></tr>
</table>

Question	Answer	Marks

5(a)	```FUNCTION AddHashtag (HashTag : STRING) RETURNS BOOLEAN```
	``` DECLARE Index : INTEGER```
	``` DECLARE Added : BOOLEAN```
	``` CONSTANT EMPTY = ""```
	
	``` Added ← FALSE```
	``` Index ← 1 // first element```
	
	``` REPEAT```
	``` IF TagString[Index] = EMPTY```
	``` THEN```
	``` TagString[Index} ← HashTag```
	``` TagCount[Index] ← 1```
	``` Added ← TRUE```
	``` ELSE```
	``` Index ← Index + 1```
	``` ENDIF```
	
	``` UNTIL Index > 10000 OR Added = TRUE```
	
	``` RETURN Added```
	```ENDFUNCTION```
	
	1 mark for each of the following:
	
	1 Declaration of two local variables: Integer for `index` & Boolean for return value (unless immediate Return used)
	2 Conditional loop through all elements until empty element found OR end of array
	3 Test if `TagString` element is empty **in a loop**
	4 If so then assign `HashTag` to `TagString[]` and 1 to `TagCount[]`
	5 Set loop termination
	6 Return Boolean (for both cases)
		6

Question	Answer	Marks

5(b)	‘Pseudocode’ solution included here for development and clarification of mark scheme.
	Programming language example solutions appear in the Appendix.
	
	`FUNCTION CountHashtag (Message : STRING) RETURNS INTEGER`
	`   DECLARE TagNum, StartPos : INTEGER`
	`   DECLARE Found : BOOLEAN`
	
	`   TagNum ← 0`
	`   Found ← TRUE`
	
	`   REPEAT`
	`      StartPos ← GetStart(Message, TagNum + 1)`
	`      IF StartPos = -1`
	`         THEN`
	`            Found ← FALSE`
	`      ELSE`
	`         TagNum ← TagNum + 1`
	`      ENDIF`
	`   UNTIL NOT Found`
	
	`   RETURN TagNum`
	
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Function heading and ending including parameter
	2 Declaration **and** initialisation of local integer for count (`TagNum`)
	3 Conditional loop through message
	4 Use of `GetStart()` **in a loop**
	5 Test `GetStart()` return value for -1 and increment count accordingly **in a loop**
	6 Return integer value
		6

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(c)</td><td>

‘Pseudocode’ solution included here for development and clarification of mark scheme.
Programming language example solutions appear in the Appendix.

```
FUNCTION IncrementHashtag (HashTag : STRING) RETURNS BOOLEAN
   DECLARE Index : INTEGER
   DECLARE Found : BOOLEAN

   Found ← FALSE
   Index ← 1 // first element

   REPEAT
      IF TagString[Index] = HashTag
         THEN
            TagCount[Index] ← TagCount[Index] + 1
            Found ← TRUE
         ELSE
            Index ← Index + 1
      ENDIF
   UNTIL Index > 10000 OR Found = TRUE
   RETURN Found
ENDFUNCTION
```

1 mark for each of the following:

1 Conditional loop until hashtag found **or** end of array
2 Compare element value to parameter **in a loop**
3 If found, increment corresponding `TagCount` element
4 Return Boolean correctly in both cases

</td><td>4</td></tr>
</table>

Question	Answer	Marks

5(d)	```PROCEDURE OutputMostPop()```
	``` DECLARE Index : INTEGER```
	``` DECLARE MostPopTag : STRING```
	``` DECLARE Max : INTEGER // the integer value of the biggest number```
	``` DECLARE Count : INTEGER```
	
	``` CONSTANT EMPTY = ""```
	
	``` Max ← -1```
	
	``` FOR Index ← 1 To 10000```
	``` IF TagCount[Index] > Max```
	``` THEN```
	``` Max ← TagCount[Index]```
	``` Count ← 1 // there is only one max value```
	``` MostPopTag ← TagString[Index]```
	``` ELSE```
	``` IF TagCount[Index] = Max```
	``` THEN```
	``` Count ← Count + 1 // another max value```
	``` ENDIF```
	``` ENDIF```
	``` ENDFOR```
	
	``` IF Count = 1```
	``` THEN```
	``` OUTPUT "The most popular hashtag is: ", MostPopTag, "It occurs: ", Max," times.”```
	``` ELSE```
	``` OUTPUT "The maximum hashtag count is: ",Max,__```
	``` "The number of hashtags with this count is: ", Count```
	``` ENDIF```
	```ENDPROCEDURE```
		8

Question	Answer	Marks

5(d)	1 mark for each of the following:
	
	1 Initialise `Max` to a value less than 1 or to `TagCount[1]`
	2 Loop through all elements
	3 Test if `TagCount` value > `Max` **in a loop**
	4 and if so set `Max` to `TagCount` value
	5 and save `TagString` element (or array index) **and** set `Count to 1 (unless counting is separate)`
	6 ELSE If `TagCount` value = `Max`, increment `Count (or via separate loop)`
	7 Output for single max **after the loop**
	8 Or Output for multiple max **after the loop**
	
	Alternative "two-loop" solution:
	
	`PROCEDURE OutputMostPop()`
	`   DECLARE Index : INTEGER`
	`   DECLARE MostPopTag : STRING`
	`   DECLARE Max : INTEGER //The integer value of the biggest number`
	`   DECLARE MaxCount : INTEGER`
	
	`   CONSTANT EMPTY = ""`
	
	`   Max ← -1`
	
	`   FOR Index ← 1 To 10000`
	`      IF TagCount[Index] > Max`
	`         THEN`
	`            Max ← TagCount[Index]`
	`            MostPopTag ← TagString[Index]`
	`      ENDIF`
	`   ENDFOR`	

Question	Answer	Marks

5(d)	<pre> MaxCount ← 0
	   FOR Index ← 1 To 10000
	      IF TagCount[Index] = Max
	         THEN
	            MaxCount ← MaxCount + 1
	      ENDIF
	   ENDFOR
	
	   IF MaxCount = 1
	      THEN
	        OUTPUT "The most popular hashtag is: ", MostPopTag, ". It occurs: ", Max," times."
	       ELSE
	        OUTPUT "The mamimum value is: ",Max, ". It occurred ", MaxCount, " times."
	   ENDIF
	ENDPROCEDURE</pre>	

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

Program Code Example Solutions

Q2 (c): Visual Basic

```
Index = 0
Status = FALSE
Do While Status <> TRUE
   Status = TopUp()
   Index = Index + 1
Loop

If Index > 100 Then
      SetLevel("Super")
End If
```

Q2 (c): Pascal

```
Index := 0;
Status := FALSE;

while Status <> TRUE do
begin
   Status := TopUp();
   Index := Index + 1;
end;

if Index > 100 then
   SetLevel("Super");
```

Q2 (c): Python

```
Index = 0
Status = FALSE
while Status <> TRUE:
   Status = TopUp()
   Index = Index + 1

if Index > 100:
   SetLevel("Super")
```

Q3: Visual Basic

```
Sub BubbleSort()
  Dim Temp As Integer
  Dim NoSwaps As Boolean
  Dim Boundary, J As Integer

  Boundary = 4998
  Do
     NoSwaps = TRUE
     For J = 0 To Boundary
        If ProdNum(J)> ProdNum(J+1)Then
           Temp = ProdNum(J)
           ProdNum(J) = ProdNum(J+1)
           ProdNum(J+1) = Temp
           NoSwaps = FALSE
        End If
     Next
     Boundary = Boundary - 1
  Loop Until NoSwaps = TRUE

End Sub
```

Q3: Pascal

```
Peocedure BubbleSort();
var
  Temp: Integer;
  NoSwaps : Boolean;
  Boundary, J : Integer;

begin
  Boundary := 4999;
  repeat
     NoSwaps := TRUE;
     for J := 1 To Boundary do
        begin
           if ProdNum[J] > ProdNum[J+1] then
              begin
                 Temp := ProdNum[J];
                 ProdNum[J] := ProdNum[J+1];
                 ProdNum[J+1] := Temp;
                 NoSwaps := FALSE;
              end;
        end;
     Boundary := Boundary - 1;
  until NoSwaps = TRUE;

end;
```

Q3: Python

```
def BubbleSort():
  # Temp As Integer
  # NoSwaps As Boolean
  # Boundary, J As Integer

   NoSwaps = False
   Boundary = 4999

  while not NoSwaps:
     NoSwaps = True
     for J in range(Boundary):
        if ProdNum[J]> ProdNum[J+1]:
           Temp = ProdNum[J]
           ProdNum[J]  = ProdNum[J+1]
           ProdNum[J+1]  = Temp
           NoSwaps = FALSE

     Boundary = Boundary - 1
```

Q5 (b): Visual Basic

```
Function CountHashtag (Message As STRING) As INTEGER
  Dim TagNum As INTEGER
  Dim StartPos As INTEGER
  Dim Found As BOOLEAN

  TagNum = 0
  Found = TRUE

  Do
     StartPos = GetStart(Message, TagNum + 1)
     If StartPos = -1 Then
        Found = FALSE
     Else
        TagNum = TagNum + 1
     End If
  Loop Until No Found

  Return TagNum

End Function
```

Q5 (b): Pascal

```
Function CountHashtag (Message : STRING) : INTEGER;
var
  TagNum : Integer;
  StartPos : Integer;
  Found : Boolean;

begin
  TagNum := 0;
  Found:= TRUE;

  repeat
     StartPos := GetStart(Message, TagNum + 1);
     if StartPos = -1 then
        Found := FALSE
     else
        TagNum := TagNum + 1;

  until Not Found;

  CountHashtag := TagNum;

end;
```

Q5 (b): Python

```
def CountHashtag (Message)
  # TagNum, StartPos As INTEGER
  # Found As BOOLEAN

  TagNum = 0
  Found = TRUE

  while Found:
     StartPos = GetStart(Message, TagNum + 1)
     if StartPos == -1:
        Found = FALSE
     else:
        TagNum = TagNum + 1


  return TagNum
```

Q 5 (c): Visual Basic

```
Function IncrementHashtag (HashTag As String) As Boolean
  Dim Index As Integer
  Dim Found As Boolean

  Found = False
  Index = 1 'First element

  Do
     If TagString(Index) = HashTag Then
        TagCount(Index) = TagCount(Index) + 1
        Found = True
     Else
        Index = Index + 1
     End If
  Loop Until Index > 10000 Or Found = True

  Return Found
End Function
```

Q 5 (c): Pascal

```
Function IncrementHashtag (HashTag : String) : Boolean;
var
  Index : Integer;
  Found : Boolean

begin
  Found := FALSE;
  Index := 1; //First element

  repeat
     If TagString[Index] = HashTag then
        begin
           TagCount[Index] := TagCount[Index] + 1;
           Found := TRUE;
        end
     else
        Index := Index + 1;

  until Index > 10000 OR Found = TRUE;

  IncrementHashtag := Found;

end;
```

Q 5 (c): Python

```
def IncrementHashtag (HashTag):
  # Index As Integer
  # Found As Boolean

  Found = FALSE
  Index = 0 #First element

  while not Found and Index < 10000:
     if TagString[Index] == HashTag:
        TagCount[Index] = TagCount[Index] + 1
        Found = TRUE
     else:
        Index = Index + 1

  Return Found
```