Cambridge Assessment
International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/22**

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 **24** 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.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>One mark for both answers:

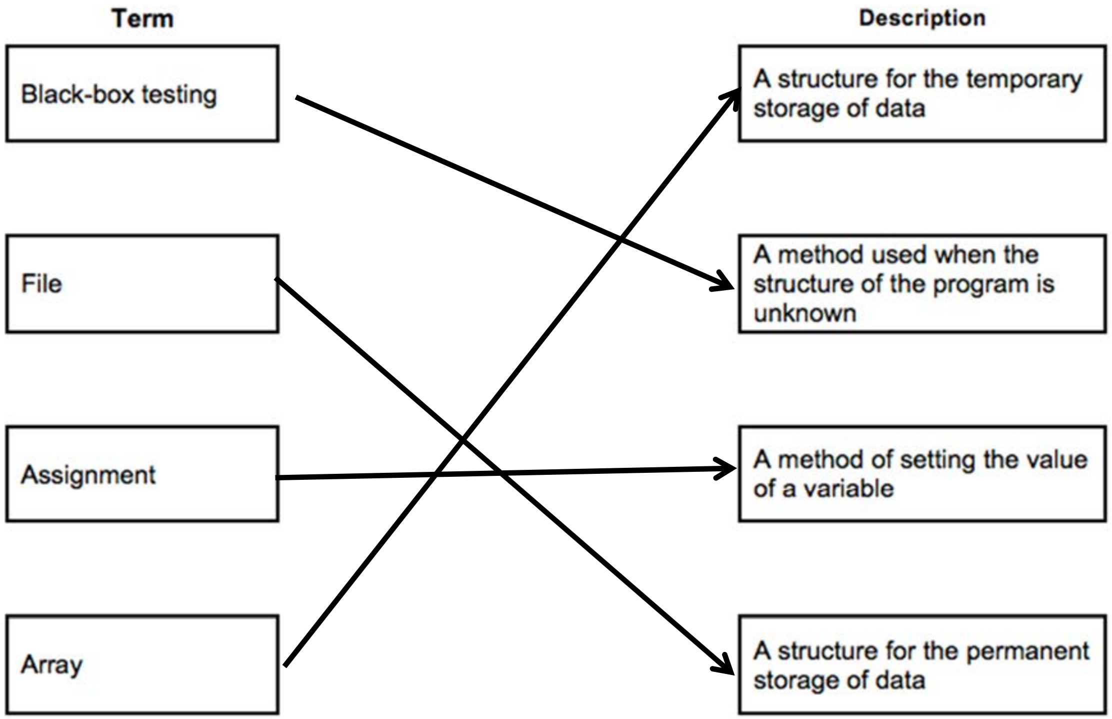
• Process
• Output

Order not important.</td><td>1</td></tr>
<tr><td>1(b)</td><td>One mark per bullet point (or equivalent)

They all represent:

• A solution to a problem / a way to perform a task
• Expressed as a sequence / series of steps / stages / instructions</td><td>2</td></tr>
<tr><td>1(c)</td><td>1 mark per row to max 4 marks

Example answers:
<table>
<tr><th>Data type</th><th>Example data value</th></tr>
<tr><td>BOOLEAN</td><td>FALSE</td></tr>
<tr><td>STRING</td><td>"Happy"</td></tr>
<tr><td>INTEGER</td><td>18</td></tr>
<tr><td>REAL</td><td>31234.56</td></tr>
<tr><td>CHAR</td><td>'H'</td></tr>
<tr><td>DATE</td><td>10/01/2019</td></tr>
</table>
Each row must be a different data type together with an appropriate value</td><td>4</td></tr>
</table>

Question	Answer	Marks
1(d)	Max 3 marks, one mark for each correct line 	**3**

Term
Description
Black-box testing
A structure for the temporary storage of data
File
A method used when the structure of the program is unknown
Assignment
A method of setting the value of a variable
Array
A structure for the permanent storage of data

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(e)</td><td>1 mark for two rows correct, 2 marks for all rows correct.
<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td><code>NOT FlagB AND FlagC</code></td><td><b>TRUE</b></td></tr>
<tr><td><code>NOT (FlagB OR FlagC)</code></td><td><b>FALSE</b></td></tr>
<tr><td><code>(FlagA AND FlagB) OR FlagC</code></td><td><b>TRUE</b></td></tr>
<tr><td><code>NOT (FlagA AND FlagB) OR NOT FlagC</code></td><td><b>TRUE</b></td></tr>
</table>
</td><td><b>2</b></td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>
<pre>
DECLARE A, B, C : INTEGER
DECLARE Average : REAL

INPUT A
REPEAT
   INPUT B
UNTIL B <> A

REPEAT
   INPUT C
UNTIL C <> A AND C <> B

Average ← (A + B + C) / 3
OUTPUT Average

IF A > B AND A > C
   THEN
      OUTPUT A
   ELSE
        IF B > A AND B > C
           THEN
              OUTPUT B
           ELSE
              OUTPUT C
        ENDIF
 ENDIF
</pre>
Mark as follows:
1 Declaration of <b>all</b> variables used (at least A, B and C)
2 Uniqueness test on A, B and C
3 Loop(s) to repeat until three unique values have been entered
4 Calculation of average value
5 Determine the largest value
6 Output of average value <b>and</b> largest value
</td><td><b>6</b></td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(b)</td><td>One mark per correct row
(Completed parts shown in bold)
<table>
<tr><th>Expression</th><th>Evaluates to</th></tr>
<tr><td>"ALARM: " & **RIGHT**("Time: 1202",**4**)</td><td>"ALARM: 1202"</td></tr>
<tr><td>**MID**("Stepwise.",**5**, **4**)</td><td>"wise"</td></tr>
<tr><td>1.5 * **LENGTH**("OnePointFive")</td><td>18</td></tr>
<tr><td>**NUM_TO_STRING**(27.5)</td><td>"27.5"</td></tr>
<tr><td>**DIV**(9, 4)</td><td>2</td></tr>
</table>
</td><td>**5**</td></tr>
<tr><td>2(c)</td><td>One mark per point, example points:

1 Subtasks make the solution more manageable // make the algorithm easier to follow
2 A subtask makes the problem easier to solve / design / program than the whole task
3 A subtask is useful when a part of the algorithm is repeated</td><td>**3**</td></tr>
</table>

Question	Answer	Marks
3(a)	'Pseudocode' solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.	**8**

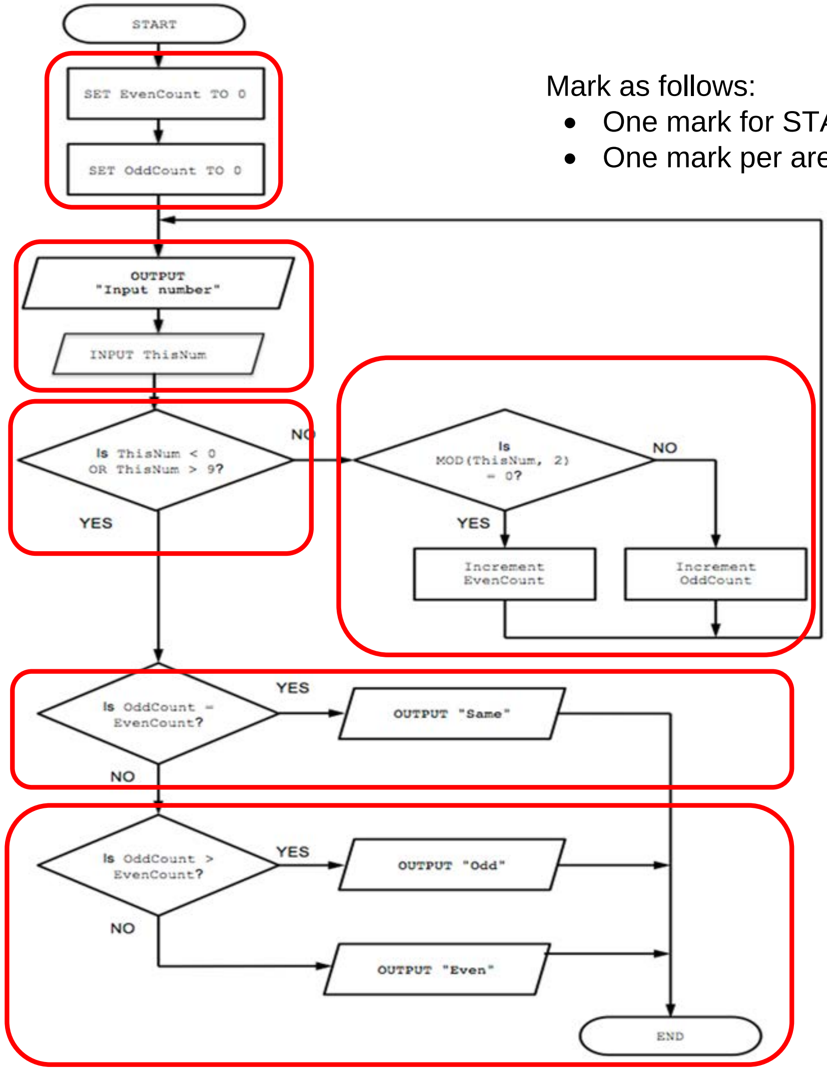
```
FUNCTION CheckSkid() RETURNS BOOLEAN
   DECLARE Rot : ARRAY[1:4] OF INTEGER
   DECLARE Average : REAL
   DECLARE ThisRot : INTEGER
   DECLARE Danger : BOOLEAN

   FOR Index ← 1 TO 4
      REPEAT
         OUTPUT "Input Rotation speed for wheel ",Index
         INPUT ThisRot
      UNTIL ThisRot >= 0 AND ThisRot <= 1000
      Rot[Index] ← ThisRot
   ENDFOR

   Average ← (Rot[1] + Rot[2] + Rot[3] + Rot[4]) / 4

   Danger ← FALSE
   FOR Index ← 1 TO 4
      IF Rot[Index] > (Average * 1.1) OR Rot[Index] < (Average * 0.9)
         THEN
            Danger ← TRUE
      ENDIF
   ENDFOR

   IF Danger = TRUE
      THEN
         OUTPUT "Skid Danger"
   ENDIF

   RETURN Danger

ENDFUNCTION
```

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>3(a)</td><td>1 mark for each of the following:

1 Function heading and ending
2 Declare local integers for 4 rotation values and a real for the average / tolerance
3 Prompt and input four rotation values
4 Validate each input value **in a loop**
5 Calculate average rotation **AND** calculate acceptable max and min (or single tolerance, or alternative method)
6 Compare rotational value of each wheel
7 Test if rotational value of (each) wheel is within the acceptable range
8 Output a warning message **and** return the correct value in all cases</td><td></td></tr>
<tr><td>3(b)</td><td>Example answers:

Test1 – No Skid detected
<table>
<tr><td>Value 1</td><td>Value2</td><td>Value 3</td><td>Value 4</td></tr>
<tr><td>100</td><td>100</td><td>100</td><td>100</td></tr>
</table>
One of:

Test2 – Skid detected (one wheel too fast)
<table>
<tr><td>Value 1</td><td>Value2</td><td>Value 3</td><td>Value 4</td></tr>
<tr><td>100</td><td>100</td><td>100</td><td>160</td></tr>
</table>
Test2 – Skid detected (one wheel too slow)
<table>
<tr><td>Value 1</td><td>Value2</td><td>Value 3</td><td>Value 4</td></tr>
<tr><td>100</td><td>100</td><td>100</td><td>40</td></tr>
</table>
Independent marks: one mark each for Test1 and Test 2</td><td>**2**</td></tr>
</table>

Question	Answer	Marks
4(a)	 START SET EvenCount TO 0 SET OddCount TO 0 OUTPUT "Input number" INPUT ThisNum Is ThisNum < 0 OR ThisNum > 9? NO YES Is MOD(ThisNum, 2) = 0? NO YES Increment EvenCount Increment OddCount Is OddCount = EvenCount? YES OUTPUT "Same" NO Is OddCount > EvenCount? YES OUTPUT "Odd" NO OUTPUT "Even" END Mark as follows: • One mark for START and END • One mark per area outlined Outputs from conditional diamond must have at least one label	**7**

Question	Answer	Marks
4(b)(i)	One mark per region as indicated.	**5**

String1	String2	Len1	RetFlag	x	Len2	NextChar	New	y
"SUB"	"BUS"	3	TRUE	1				
					3	'S'	""	
							"B"	1
							"BU"	2
								3
	"BU"			2				
					2	'U'	""	
								1
							"B"	2
	"B"			3				
					1	'B'	""	
								1
	""							

Question	Answer	Marks
4(b)(ii)	TRUE	**1**
4(b)(iii)	One mark for explanation of problem, one mark for test strings Problem: • The inner FOR loop removes ALL characters from String2 that match the current character from String1 and not just one instance Test Strings: • ‘SAME’ and ‘MASS’ (for example)	**2**
4(b)(iv)	The inner FOR loop should only remove <u>one</u> instance of the character from String2	**1**
4(b)(v)	• Dry run // White-box testing	**1**
4(b)(vi)	Max 2 marks, features include: • Single stepping • Breakpoints • Variable and expressions report window • <u>Syntax</u> error highlighting	**2**

Question	Answer	Marks

5(a)	`PROCEDURE InitArrays()`
	`   DECLARE Index : INTEGER`
	`   FOR Index ← 1 TO 10000`
	`      TagString[Index] ← ""`
	`      TagCount[Index] ← 0`
	`   ENDFOR`
	`ENDPROCEDURE`
	
	1 mark for each of the following:
	
	1 Procedure heading **and** ending (as shown)
	2 Declaration of `Index` (e.g.) as integer
	3 Loop for 10000 iterations
	4 Initialise TagString element to ""
		4

Question	Answer	Marks
5(b)		8

```
FUNCTION SaveArrays() RETURNS INTEGER

   DECLARE Index, NumUnused : INTEGER
   DECLARE FileString : STRING
   CONSTANT COMMA = ','

   NumUnused ← 0

   OPEN "Backup.txt" FOR WRITE
   FOR Index ← 1 to 10000
      IF TagString[Index] <> ""
         THEN
            FileString ← TagString[Index] & COMMA & NUM_TO_STRING(TagCount[Index])
            WRITEFILE "Backup.txt", FileString
         ELSE
            NumUnused ← NumUnused + 1
      ENDIF
   ENDFOR
   CLOSEFILE "Backup.txt"

   RETURN NumUnused
ENDFUNCTION
```

1 mark for each of the following:

1 Function heading and ending
2 Open the file `Backup.txt` in write mode **and** close file
3 Loop through 10000 elements
4 Test if `TagString[Index]` is "" **in a loop**
5 If not then form `FileString` from array elements **with** separator **and** using `NUM_TO_STRING()`**in a loop**
6 Write string to file **in a loop**
7 Count the number of unused elements
8 Return `NumUnused` **not in a loop**

<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.
Max 8 marks from 9 available mark points

```
FUNCTION LoadArrrays() RETURNS INTEGER

   DECLARE ArrayIndex, Index, CountLen, Count : INTEGER
   DECLARE FileString, HashTag : STRING
   CONSTANT COMMA = ','

   ArrayIndex ← 0                  //  first element

   OPEN "Backup.txt" FOR READ
   WHILE NOT EOF("Backup.txt")
      READFILE "Backup.txt", FileString
      Index ← 1
      HashTag ← ""
      WHILE MID(FileString, Index, 1) <> COMMA             // hashtag
         HashTag ← HashTag & MID(FileString, Index, 1)
         Index ← Index + 1
      ENDWHILE
      TagString[ArrayIndex] ← HashTag
      CountLen ← LENGTH(FileString) - LENGTH(HashTag) - 1
      Count ← STR_TO_NUM(RIGHT(FileString, CountLen))      // count
      TagCount[ArrayIndex] ← Count
      ArrayIndex ← ArrayIndex + 1
   ENDWHILE

   CLOSE "Backup.txt"

   RETURN ArrayIndex
ENDFUNCTION
```

</td><td>**8**</td></tr>
</table>

Question	Answer	Marks
5(c)	1 mark for each of the following: 1 Function heading and ending 2 Declare and initialise `ArrayIndex` (or equivalent name) 3 Open the file `Backup.txt` in read mode **and** close the file 4 Loop until end of the `Backup.txt` file // string read is null 5 Read a line from the file **in a loop** 6 Extract hashtag and count **in a loop** 7 Store hashtag in `TagString` array and count in `TagCount` array **after type conversion** 8 Increment `ArrayIndex` **in a loop** 9 Return number of array elements	

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

Appendix: Program Code Example Solutions

Q3 (a): Visual Basic

```
Function CheckSkid() As Boolean

  Dim Rot(3) As Integer
  Dim Average As Double
  Dim ThisRot As Integer
  Dim Danger As Boolean

  For Index = 0 To 3
     Do
        Console.Writeline("Enter Wheel Rotation Speed: "
        ThisRot = Console.Readline()
     Loop Until ThisRot >= 0 And ThisRot <= 1000
     Rot(Index) = ThisRot
  Next

  Average = (Rot(0) + Rot(1) + Rot(2) + Rot(3)) / 4

  Danger = FALSE
  For Index = 0 TO 3
     If Rot(Index) > (Average * 1.1) OR Rot(Index) < (Average * 0.9) Then
        Danger = TRUE
     End If
  Next


 If Danger = TRUE Then
    Console.Writeline("Skid Danger")
 Else
    Console.Writeline("No Skid Danger")
 End if

RETURN Danger

End Function
```

Q3 (a): Pascal

```
Function CheckSkid() : Boolean;

  var
  Rot : array [1..4] of integer;
  Average : Real;
  ThisRot : Integer;
  Index : Integer;
  Danger : Boolean;

  For Index := 1 to 4 do
     begin
        repeat
           write('Enter rotation speed : ');
        readln(ThisRot);
        until (ThisRot >= 0) And (ThisRot <= 1000);
     Rot[Index] := ThisRot;
     end;

  Average := (Rot[1] + Rot[2] + Rot[3] + Rot[4]) / 4;

  Danger := FALSE;
  For Index := 1 to 4 do
     begin
     If (Rot[Index] > (Average * 1.1)) OR (Rot[Index] < (Average * 0.9)) then
        Danger := TRUE;
     end;


 If Danger = TRUE then
    writeln('Skid Danger')
 Else
    writeln('No Skid Danger');

CheckSkid := Danger;

end;
```

Q3 (a): Python

```
def CheckSkid():

  # Rot[3] As Integer
  # Average As Real
  # ThisRot As Integer
  # Danger As Boolean

  Rot = [0, 0, 0, 0]
  for Index in range(0, 4):
     while True:
        ThisRot = float(input("Enter the rotation speed of the wheel: "))
        if ThisRot >= 0 and ThisRot <= 1000:
           break
     Rot[Index] = ThisRot
  Next

  Average = (Rot[0] + Rot[1] + Rot[2] + Rot[3]) / 4

  Danger = False
  for Index in range(0, 4):
     if Rot[Index] > (Average * 1.1) or Rot[Index] < (Average * 0.9):
        Danger = True


  If Danger == True:
     print("Skid Danger")
  else:
     print("No Skid Danger")

  return Danger
```

Q5 (c): Visual Basic

```
Function LoadArrrays () As Integer

  Dim ArrayIndex, Index, CountLen, Count As Integer
  Dim FileString, HashTag As String
  Dim File As New StreamReader("Backup.txt")

  Const COMMA = ','

  ArrayIndex = 0 ' First element

  Do While File.Peek <> -1
     FileString = File.ReadLine()
     Index = 1
     HashTag = ""
     Do While Mid(FileString, Index, 1) <> COMMA             ' the hashtag
        HashTag = HashTag & MID(FileString, Index, 1)
        Index = Index + 1
     Loop

     TagString(arrayIndex) = HashTag
     CountLen = Len(fileString) – Len(HashTag) – 1
     Count = CInt(Right(FileString, CountLen))              ' the count
     TagCount(ArrayIndex) = Count
     ArrayIndex = ArrayIndex + 1
  Loop

  File.Close

  Return ArrayIndex
End Function
```

Q5 (c): Pascal

```
Function LoadArrrays () : Integer;

  var
  ArrayIndex, Index, CountLen, Count : Integer;
  FileData, HashTag : String;
  Backup : Textfile;

  const
  COMMA = ',';

  begin
    assignfile(Backup, 'Backup.txt');
    reset(File);


  ArrayIndex := 0; //First element

  while not EOF(File) do
     begin
        readln(Backup, FileData);
        Index := 1;
        HashTag := "";
        while midstr(FileData, Index, 1) <> COMMA do            // the hashtag
           begin
              HashTag := HashTag + midstr(FileData, Index, 1);
              Index := Index + 1;
           end;

        TagString[ArrayIndex] := HashTag;
        CountLen := length(FileData) – length(HashTag) – 1;
        Count := strtoint(RightStr(FileData, CountLen));       // the count
        TagCount[ArrayIndex] := Count;
        ArrayIndex := ArrayIndex + 1;
     end;

  closefile(File);
  LoadArrays := ArrayIndex;

end;
```

Q5 (c): Python

```
def LoadArrays ():

  # ArrayIndex, Index, ~~CountLen~~, Count As Integer
  # FileString, HashTag As String
  # File As StreamReader("Backup.txt")

  COMMA = ','

  File = open("Backup.txt", "r")
  ArrayIndex = 0 #First element

  for FileString in File:
     Index = 0
     HashTag = ""
     while FileString[Index] != COMMA:             # the hashtag
        HashTag = HashTag + FileString[Index]
        Index = Index + 1

     TagString[ArrayIndex] = HashTag
     Count = int(FileString[Index+1:])            # the count
     TagCount[ArrayIndex] = Count
     ArrayIndex = ArrayIndex + 1


  File.close()

  return ArrayIndex
```