Cambridge International AS & A Level

Cambridge Assessment International Education
Cambridge International Advanced Subsidiary and Advanced Level

COMPUTER SCIENCE **9608/23**

Paper 2 Written Paper **October/November 2017**

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.

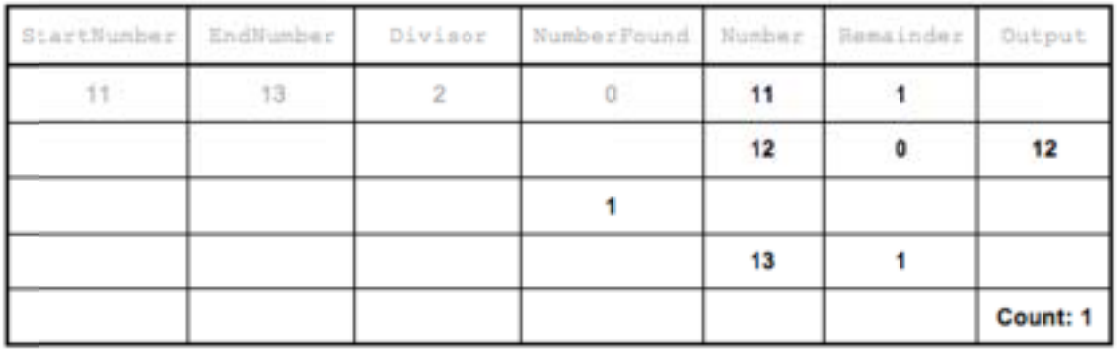
Cambridge International is publishing the mark schemes for the October/November 2017 series for most Cambridge IGCSE®, Cambridge International A and AS Level components and some Cambridge O Level components.

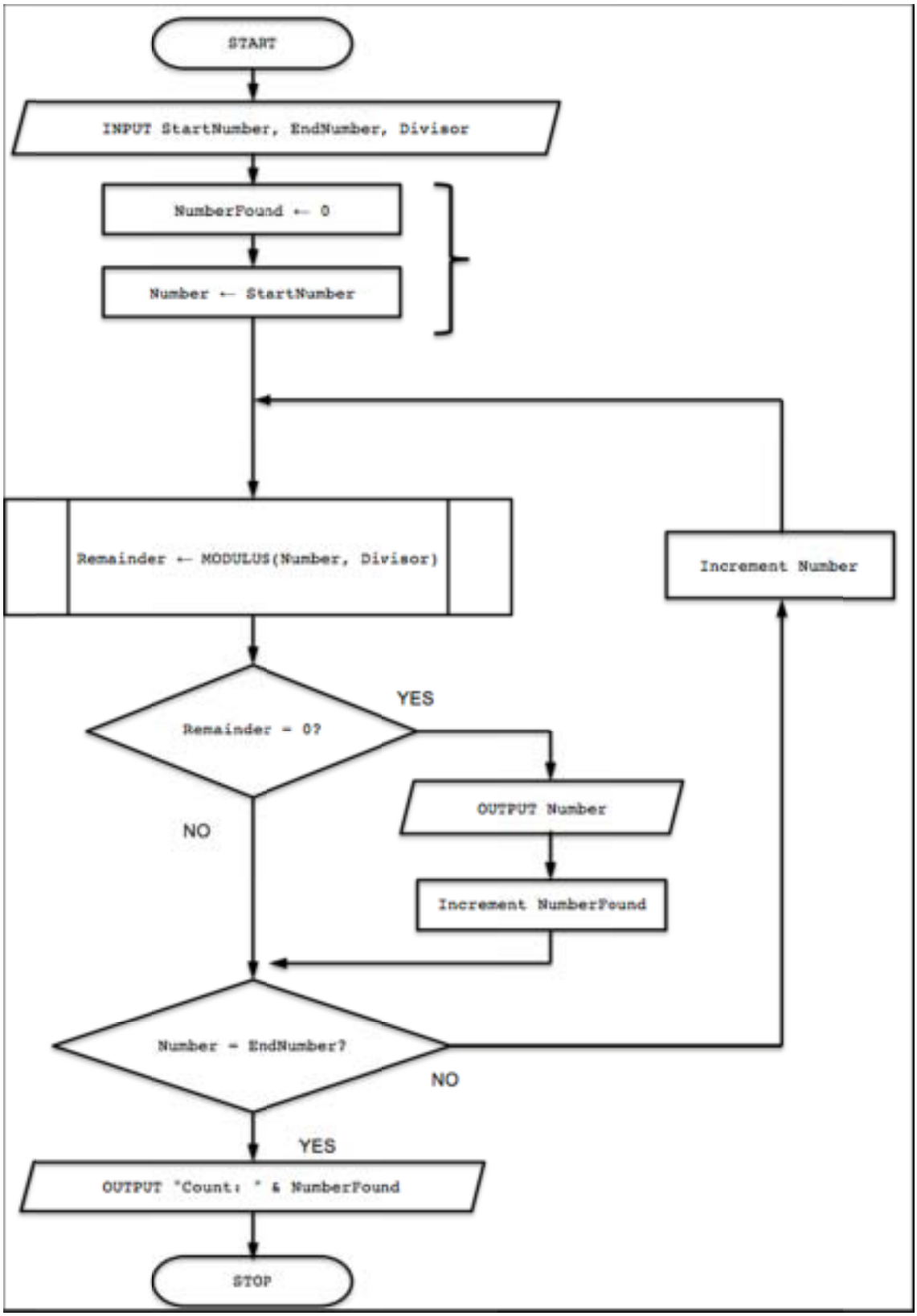
<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)(i)</td><td>
<table>
<tr><th>Data value</th><th>Data type</th></tr>
<tr><td>27</td><td>INTEGER</td></tr>
<tr><td>"27"</td><td>STRING</td></tr>
<tr><td>"27.3"</td><td>STRING</td></tr>
<tr><td>TRUE</td><td>BOOLEAN</td></tr>
<tr><td>27/3/2015</td><td>DATE // DATETIME</td></tr>
<tr><td>27.3</td><td>REAL</td></tr>
</table>
One mark for each data type
Mark first data type given in each case
</td><td>6</td></tr>
<tr><td>1(a)(ii)</td><td>1D Array // 1DList</td><td>2</td></tr>
<tr><td>1(a)(iii)</td><td>• Each character is represented by an unique / corresponding
• binary code / integer / value</td><td>2</td></tr>
<tr><td>1(b)</td><td>• When a section of code would be repeated
• When a piece of code is needed to perform a specific task
• To support modular programming / step wise refinement
• Easier to debug / maintain
• Built-in / library routines are tried and tested

One mark per answer</td><td>Max 2</td></tr>
<tr><td>1(c)</td><td>
<pre>
CASE OF MyVar
   1: CALL Proc1()
   2: CALL Proc2()
   3: CALL Proc3()
   OTHERWISE OUTPUT "Error"
ENDCASE
</pre>
One mark for:

• First line and ENDCASE
• All clauses for 1, 2 and 3
• 'OTHERWISE' clause
• OUTPUT statement
</td><td>4</td></tr>
</table>

Question	Answer	Marks
1(d)	Ability to recognise: • selection statement • iteration statement • assignment statements • data declarations / structures / data types / use of variables or objects • modular structure / functions / procedures / subroutines • subroutine parameters • Specific types of statement, e.g. Input, Output, File operations • Code format • Operators Mark as follows: Any two from above, or valid alternative Accept by example	**Max 2**

Question	Answer	Marks
2(a)	 One mark for correct Number column One mark for correct Remainder column One mark for correct Output	**3**
2(b)	Mark as follows: • For a (given) range of values • Counts the number of times one number (numerator) is an exact divisor of the other • Outputs each numerator (only) • Outputs the count Accept by example No mark for: • ...calculate the remainder • ...add one to `NumberFound`	**3**

Question	Answer	Marks
2(c)		**10**

Question	Answer	Marks
2(c)	Mark as follows: • One mark for `START` **and** `STOP` / `END` • One mark for bracketed pair • One mark for each of other labelled boxes (shape must be correct for decision box) Decision box outputs must have two outputs and at least one label (`Yes` / `No`) Different statement categories should not appear in the same symbol (e.g. assignment and I/O) No mark for symbol (or pair) if parent missing or logically incorrect (except for `START`/`END`) Full marks should be awarded for functionally equivalent solutions.	

Question	Answer	Marks
3(a)		**8**

```
PROCEDURE BubbleSort
  DECLARE Temp : STRING
  DECLARE FirstID, SecondID : INTEGER
  DECLARE NoSwaps : BOOLEAN
  DECLARE Boundary : INTEGER
  Declare J : INTEGER

  Boundary ← 99
  REPEAT
    NoSwaps ← TRUE
    FOR J ← 1 TO Boundary
      FirstID ← UserNameArray[J]
      SecondID ← UserNameArray[J + 1]
      IF FirstID > SecondID
        THEN
          Temp ← UserNameArray[J]
          UserNameArray[J] ← UserNameArray[J + 1]
          UserNameArray[J + 1] ← Temp
          NoSwaps ← FALSE
      ENDIF
    ENDFOR
    Boundary ← Boundary - 1
  UNTIL NoSwaps = TRUE
ENDPROCEDURE
```

Mark as follows:

1. Procedure heading and ending (allow array as input parameter)
2. Variable declaration for counter / index (integer) **or** temp (string)
3. Outer working loop
4. Inner loop with suitable range
5. Correct comparison **in a loop**
6. Correct swap of complete array element **in a loop**
7. Set flag to indicate swap **in inner loop and** resetting **in outer loop**
8. Reducing Boundary **in a loop**

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

```
PROCEDURE FindRepeats
  DECLARE i, RepeatCount: INTEGER
  DECLARE FirstID, SecondID: STRING
  RepeatCount ← 0

  FOR i ←  2 TO 100
    FirstID ←  LEFT(UserNameArray[i – 1],6)
    SecondID ←  LEFT(UserNameArray[i],6)
    IF FirstID = SecondID
      THEN
        RepeatCount ←  RepeatCount + 1
        OUTPUT(UserNameArray[i])
    ENDIF
  ENDFOR

  IF RepeatCount = 0
    THEN
      OUTPUT "The array contains no repeated UserIDs"
    ELSE
      OUTPUT "There are " & RepeatCount & " repeated userIDs"
  ENDIF

ENDPROCEDURE
```

Mark as follows (all must be correct syntax for chosen language):

1. Procedure heading and ending
2. Variable declaration for INTEGER (comment in Python) **and** initialisation for `RepeatCount` (or equivalent name)
3. Loop
4. Extraction of UserID **in a loop**
5. Correct comparison of consecutive elements… **in a loop**
6. ...output correct array element (NOT original, only duplicates) **in a loop**
7. increment `RepeatCount` following a comparison **in a loop**
8. Correct conditional statement checking `RepeatCount` (or equivalent) and then ... ... two correct final OUTPUT statements

Question	Answer	Marks
3(c)(i)	• Problem definition • Design • Coding / programming • Testing • Documentation • Implementation • Maintenance	**3**
3(c)(ii)	Integrated Development Environment or a suitable description	**1**
3(c)(iii)	Examples include: • context sensitive prompts • (dynamic) syntax checking • use of colours to highlight key words / pretty printing • Formatting • Single-stepping • Breakpoints • Report / watch window • (UML) modelling • Compiler/interpreter • Text editor	**Max 2**
3(c)(iv)	Run-time	**1**

Question	Answer	Marks
4(a)	<table><tr><th>Value</th><th>Formatted String</th></tr><tr><td>1327.5</td><td>"□1327.50"</td></tr><tr><td>1234</td><td>"□1234.00"</td></tr><tr><td>7.456</td><td>"□□□07.45"</td></tr></table> Leading spaces must be present	**2**
4(b)	<table><tr><th>Value</th><th>Required output</th><th>Mask</th></tr><tr><td>1234.00</td><td>"1,234.00"</td><td>"0,000.00"</td></tr><tr><td>3445.66</td><td>"£3,445.66"</td><td>"£0,000.00"</td></tr><tr><td>10345.56</td><td>"$□□10,345"</td><td><b>"$##00,000"</td></tr></table> Currency and ‘punctuation’ symbols must be as shown	**3**

Question	Answer	Marks

5(a)	`PROCEDURE MakeNewfile`
	`  DECLARE OldFileLine : STRING`
	`  DECLARE NewFileLine : STRING`
	
	`  OPENFILE "EmailDetails" FOR READ`
	`  OPENFILE "NewEmailDetails" FOR WRITE`
	
	`  WHILE NOT EOF("EmailDetails")`
	`  READFILE "EmailDetails", OldFileLine`
	`  NewFileLine ← "00" & OldFileLine`
	`  WRITEFILE "NewEmailDetails", NewFileLine`
	`  ENDWHILE`
	
	`  CLOSEFILE "EmailDetails"`
	`  CLOSEFILE "NewEmailDetails"`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	1. Variable declaration of `STRING` for `OldFileLine` (or equivalent)
	2. Open `EmailDetails` for `READ`
	3. Open `NewEmailDetails` for `WRITE`
	4. Correct loop checking for `EOF(EmailDetails)`
	5. Reading a line from `EmailDetails` **in a loop**
	6. Correct concatenation **in a loop**
	7. Writing a line to `NewEmailDetails` **in a loop**
	Closing both files
		8

5(b)	**Invalid string examples:** A string with nothing **before** '@' A string with nothing **after** '@' A string with 1 or 2 characters after '@' A string with no '@'symbol A string with more than one '@' symbol **Explanation** Sensible explanation mapping each given string to an individual rule One mark for string One mark for explanation Each rule should be tested once only	**6**

Programming Example Solutions

Q3(b): Visual Basic

```
Sub FindRepeats()
   Dim Repeats As Integer
   Dim i As Integer
   Dim FirstID As String
   Dim SecondID As String

   Repeats = 0
   For i = 1 To 99
      FirstID = Left(UserNameArray(i), 6)
      SecondID = Left(UserNameArray(i + 1), 6)
      If FirstID = SecondID Then
         Console.WriteLine(UserNameArray(i + 1))
         Repeats = Repeats + 1
      End If
   Next i

   If Repeats = 0 Then
     Console.WriteLine("The array contains no repeated UserIDs")
   Else
     Console.WriteLine("There are " & Repeats & " repeated UserIDs")
   End If

End Sub
```

Alternative:

```
Sub FindRepeats ()

   Dim RepeatCount, i As Integer
   Dim FirstID, SecondID As String

   RepeatCount = 0
   For i = 1 to 99
      FirstID = Left(UserNameArray(i-1),6)
      SecondID = Left(UserNameArray(i),6)
      If FirstID = SecondID then
         Console.WriteLine (UserNameArray(i))
         RepeatCount = RepeatCount + 1
      End If
   Next i

   If RepeatCount = 0 then
      Console.WriteLine ("The array contains no repeated UserIDs")
   Else
      Console.WriteLine ("There are "& RepeatCount & " repeated UserIDs")
   End If
End Sub
```

Q3(b): Pascal

```
procedure FindRepeats ();

var
   RepeatCount, i : integer;
   FirstID, SecondID : string;

   begin
      RepeatCount := 0;
      for i := 1 to 99 do
      begin
         FirstID := Copy(UserNameArray[i-1],1,6);
         SecondID := Copy(UserNameArray[i],1,6);
         if FirstID = SecondID then
         begin
            writeln (UserNameArray[i]);
            RepeatCount := RepeatCount + 1;
         end;
      end;

      if RepeatCount = 0 then
         writeln ('The array contains no repeated UserIDs')
      else
         writeln ('There are ', RepeatCount,' repeated UserIDs')
   end;
```

Q3(b): Python

```
def FindRepeats():
   #Repeats, i Integer
   #FirstID, SecondID string
   Repeats = 0
   for i in range(0, len(UserNameArray)-1):
      FirstID = (UserNameArray[i])[:6]
      SecondID = (UserNameArray[i+1])[:6]
      if FirstID == SecondID:
         print(UserNameArray[i+1])
         Repeats = Repeats + 1
      if Repeats == 0:
         print("The array contains no repeated UserIDs")
      else:
         print("There are ", Repeats, " repeated UserIDs")
```

Alternative:

```
def FindRepeats ():

   RepeatCount = 0                          ## Defined as an integer

   for i in range (1,100):                  ## depending on next two
lines(0,99) (2,101)
      FirstID = UserNameArray[i-1]          ## Defined as string
      SecondID = UserNameArray[i]           ## Defined as string
      if FirstID[0:6] == SecondID[0:6]:     ## Using split
         print (UserNameArray[i])
         RepeatCount += 1

   if repeatCount == 0:
      print ('The array contains no repeated UserIDs')
   else:
      print ('There are ', RepeatCount,' repeated UserIDs')
```