# QUESTION 7.

**2** **(i)** (Single) software program [1]

Features for:
program editor/writing/editing
translation // interpreter/compiler
testing program code **//** observe outputs } 2 points to score [1]

**(ii)** Syntax checking (on entry)
Structure blocks (e.g. IF structure and loops begin/end highlighted)
General prettyprint features
Automatic indentation
Highlights any undeclared variables
Highlights any unassigned variables
Commenting out/in of blocks of code
Visual collapsing / highlighting of blocks of code
Single stepping
Breakpoints
Variable/expressions report window [MAX 3]

# QUESTION 8.

```
end.
```

**4(e): Python**

```
def ProductCodeSearch(SearchCode):
    # list indexes start at zero
    i = 0
    Found = "no"
    while not(i == 1001 or Found == "yes"):
        if SearchCode == PCode[i]:
            Found = "yes"
```

# QUESTION 9.

Cambridge International AS & A Level

**Cambridge Assessment International Education**
Cambridge International Advanced Subsidiary and Advanced Level

**COMPUTER SCIENCE** **9608/21**

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.

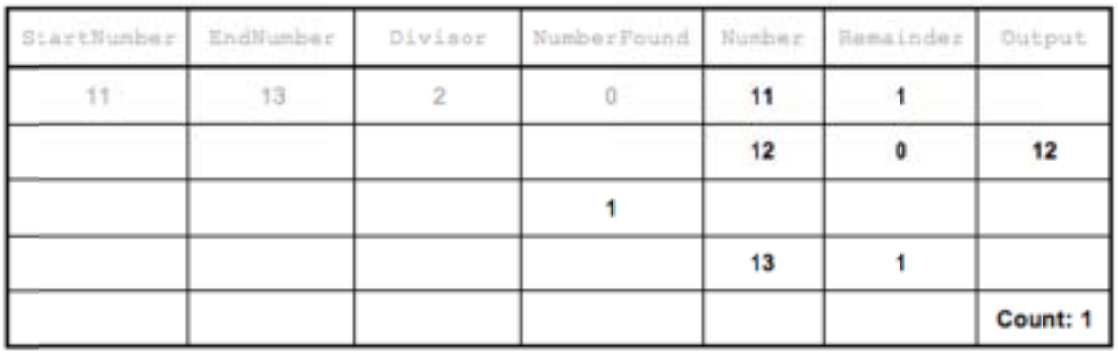
Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

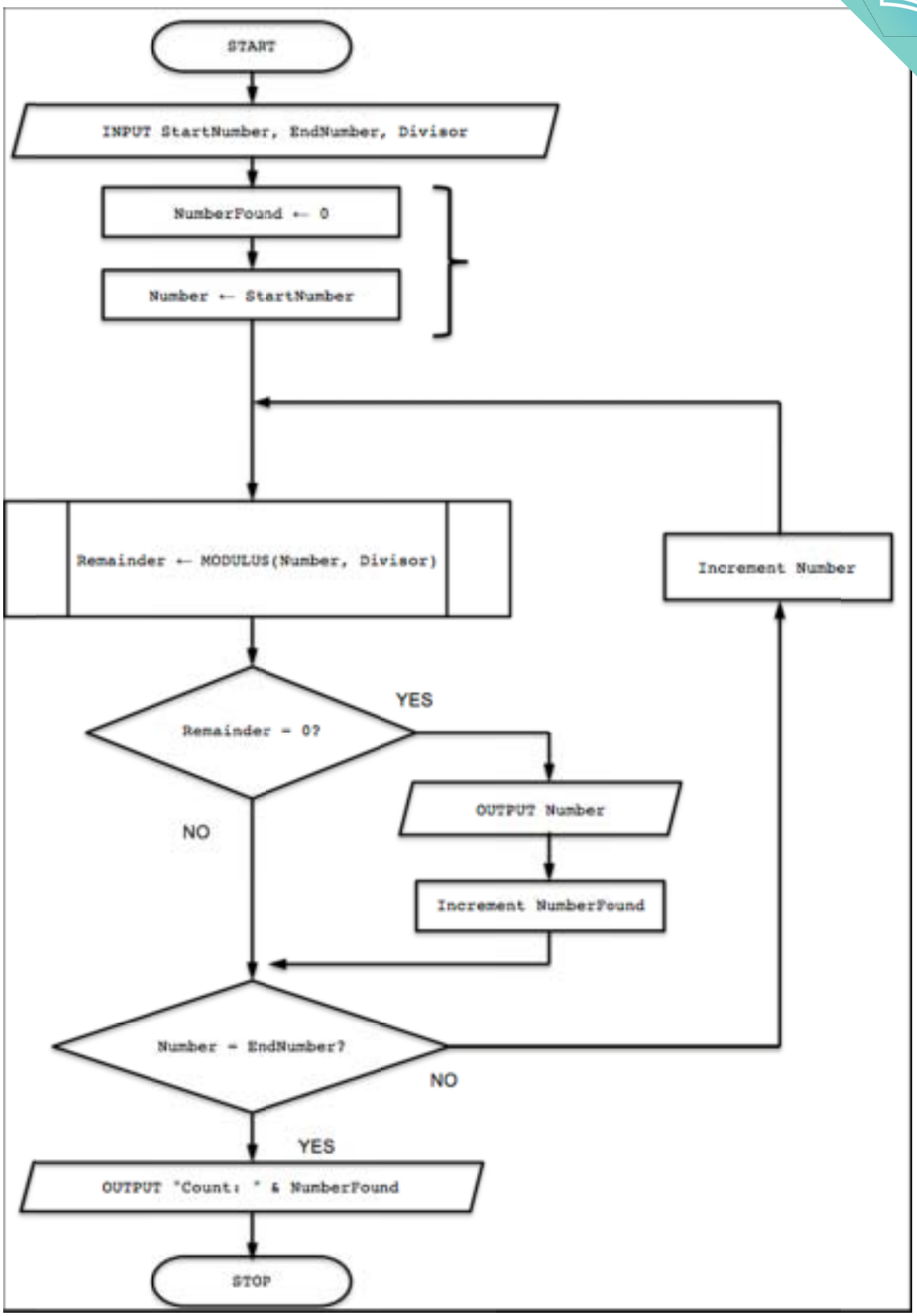
Cambridge International will not enter into discussions about these mark schemes.

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></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<br>Mark first data type given in each case</td><td></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<br>• binary code / integer / value</td><td>2</td></tr>
<tr><td>1(b)</td><td>• When a section of code would be repeated<br>• When a piece of code is needed to perform a specific task<br>• To support modular programming / step wise refinement<br>• Easier to debug / maintain<br>• Built-in / library routines are tried and tested<br><br>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:<br><br>• First line and ENDCASE<br>• All clauses for 1, 2 and 3<br>• 'OTHERWISE' clause<br>• OUTPUT statement</td><td>4</td></tr>
</table>

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

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

| Question | Answer |
|---|---|
| 2(c) |  |

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

| Question | Answer |
|---|---|
| 3(a) | |

```
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 |
|---|---|
| 3(b) | Pseudocode solution included here for development and clarification of mark scheme. Programming language example solutions appear in the Appendix.<br><br>`PROCEDURE FindRepeats`<br>`  DECLARE i, RepeatCount: INTEGER`<br>`  DECLARE FirstID, SecondID: STRING`<br>`  RepeatCount ← 0`<br><br>`  FOR i ←  2 TO 100`<br>`    FirstID ←  LEFT(UserNameArray[i – 1],6)`<br>`    SecondID ←  LEFT(UserNameArray[i],6)`<br>`    IF FirstID = SecondID`<br>`      THEN`<br>`        RepeatCount ←  RepeatCount + 1`<br>`        OUTPUT(UserNameArray[i])`<br>`    ENDIF`<br>`  ENDFOR`<br><br>`  IF RepeatCount = 0`<br>`    THEN`<br>`      OUTPUT "The array contains no repeated UserIDs"`<br>`    ELSE`<br>`      OUTPUT "There are " & RepeatCount & " repeated userIDs"`<br>`  ENDIF`<br><br>`ENDPROCEDURE`<br><br>Mark as follows (all must be correct syntax for chosen language):<br><br>1. Procedure heading and ending<br>2. Variable declaration for INTEGER (comment in Python) **and** initialisation for `RepeatCount` (or equivalent name)<br>3. Loop<br>4. Extraction of UserID **in a loop**<br>5. Correct comparison of consecutive elements… **in a loop**<br>6. ...output correct array element (NOT original, only duplicates) **in a loop**<br>7. increment `RepeatCount` following a comparison **in a loop**<br>8. Correct conditional statement checking `RepeatCount` (or equivalent) and then ...<br>... two correct final OUTPUT statements |

| Question | Answer | |
|---|---|---|
| 3(c)(i) | • Problem definition<br>• Design<br>• Coding / programming<br>• Testing<br>• Documentation<br>• Implementation<br>• Maintenance | |
| 3(c)(ii) | Integrated Development Environment or a suitable description | **1** |
| 3(c)(iii) | Examples include:<br><br>• context sensitive prompts<br>• (dynamic) syntax checking<br>• use of colours to highlight key words / pretty printing<br>• Formatting<br>• Single-stepping<br>• Breakpoints<br>• Report / watch window<br>• (UML) modelling<br>• Compiler/interpreter<br>• 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><br>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><br>Currency and ‘punctuation’ symbols must be as shown | **3** |

| Question | Answer | |
|---|---|---|
| 5(a) | `PROCEDURE MakeNewfile`<br>`  DECLARE OldFileLine : STRING`<br>`  DECLARE NewFileLine : STRING`<br><br>`  OPENFILE "EmailDetails" FOR READ`<br>`  OPENFILE "NewEmailDetails" FOR WRITE`<br><br>`  WHILE NOT EOF("EmailDetails")`<br>`  READFILE "EmailDetails", OldFileLine`<br>`  NewFileLine ← "00" & OldFileLine`<br>`  WRITEFILE "NewEmailDetails", NewFileLine`<br>`  ENDWHILE`<br><br>`  CLOSEFILE "EmailDetails"`<br>`  CLOSEFILE "NewEmailDetails"`<br><br>`ENDPROCEDURE`<br><br>Mark as follows:<br>1. Variable declaration of `STRING` for `OldFileLine` (or equivalent)<br>2. Open `EmailDetails` for `READ`<br>3. Open `NewEmailDetails` for `WRITE`<br>4. Correct loop checking for `EOF(EmailDetails)`<br>5. Reading a line from `EmailDetails` **in a loop**<br>6. Correct concatenation **in a loop**<br>7. Writing a line to `NewEmailDetails` **in a loop**<br>Closing both files | |
| 5(b) | **Invalid string examples:**<br><br>A string with nothing **before** '@'<br>A string with nothing **after** '@'<br>A string with 1 or 2 characters after '@'<br>A string with no '@'symbol<br>A string with more than one '@' symbol<br><br>**Explanation**<br>Sensible explanation mapping each given string to an individual rule<br><br>One mark for string<br>One mark for explanation<br>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')
```

# QUESTION 10.

Cambridge International AS & A Level

**Cambridge Assessment International Education**
Cambridge International Advanced Subsidiary and Advanced Level

**COMPUTER SCIENCE** **9608/22**

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.

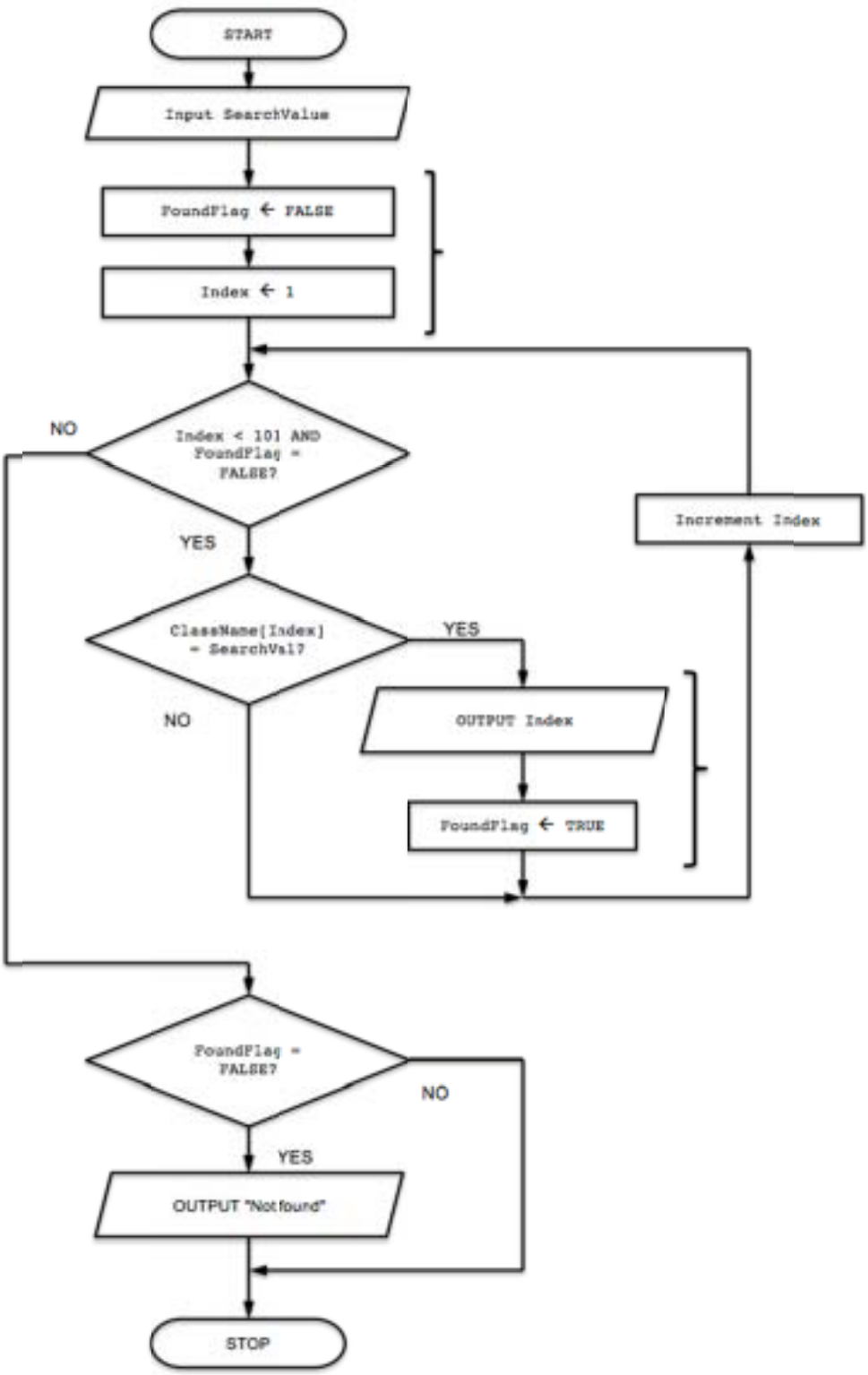
Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

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></tr>
<tr><td>1(a)(i)</td><td>
<table>
<tr><th>Data value</th><th>Data type</th></tr>
<tr><td>FALSE</td><td>BOOLEAN</td></tr>
<tr><td>03/03/2013</td><td>DATE // DATETIME</td></tr>
<tr><td>35</td><td>INTEGER</td></tr>
<tr><td>"INTEGER"</td><td>STRING</td></tr>
<tr><td>3.5</td><td>REAL</td></tr>
<tr><td>"35"</td><td>STRING</td></tr>
</table>
<br>One mark for each data type<br>Mark first data type given in each case</td><td></td></tr>
<tr><td>1(a)(ii)</td><td>1D Array // 1D List</td><td>1</td></tr>
<tr><td>1(a)(iii)</td><td>Ability to recognise:<br>• selection statement<br>• iteration statement<br>• assignment statements<br>• data declarations / structures / data types / use of variables or objects<br>• modular structure / functions / procedures / subroutines<br>• subroutine parameters<br>• Specific types of statement e.g. Input, Output, File operations<br>• Code format<br>• Operators<br><br>Mark as follows:<br>Any two from above, or valid alternative<br>Accept by example</td><td>2</td></tr>
<tr><td>1(b)(i)</td><td>
<table>
<tr><th>Data</th></tr>
<tr><td></td></tr>
<tr><td>67 // 0100 0011 // 043h</td></tr>
<tr><td>65 // 0100 0001 // 041h</td></tr>
<tr><td>71 // 0100 0111 // 047h</td></tr>
<tr><td>69 // 0100 0101 // 045h</td></tr>
<tr><td></td></tr>
</table>
<br>One mark for 67 and 65<br>One mark for 71 and 69<br>Accept binary, denary or hex values (hex must be clearly indicated)<br>Max one mark if blank cell anywhere in sequence<br>Ignore any data values before or after the four characters</td><td>2</td></tr>
</table>

| Question | Answer | |
|---|---|---|
| 1(b)(ii) | • A value representing the number of characters ... ... stored at beginning of string<br>OR<br>• Terminator / special character ... ... stored to indicate the end of string<br><br>One mark for each phrase or equivalent. | |
| 1(c) | Explanation includes:<br>• to pass values to/from the subroutine<br>• to produce re-useable code<br>• to avoid global variables<br>• to allow recursion<br><br>One mark per answer | **Max 3** |
| 1(d)(i) | 27: `MyGrade` assigned the value `"Fail"`<br>101: Output the text `"Invalid Value Entered"`<br>Ignore minor spelling mistakes | **2** |
| 1(d)(ii) | `IF MyMark >= 75 AND MyMark <=100`<br>`  THEN`<br>`  MyGrade ← "Distinction"`<br>`  ELSE`<br>`  IF MyMark >= 35 AND MyMark <=74`<br>`  THEN`<br>`  MyGrade ← "Pass"`<br>`  ELSE`<br>`  IF MyMark >= 0 AND MyMark <=34`<br>`  THEN`<br>`  MyGrade ← "Fail"`<br>`  ELSE`<br>`  OUTPUT "Invalid value entered"`<br>`  ENDIF`<br>`  ENDIF`<br>`ENDIF`<br><br>One mark for each of:<br>• One correct range test<br>• 'IF' equivalent (nested or not) to three CASE range tests...<br>• ... with three corresponding assignments<br>• Equivalent of CASE `OTHERWISE` with corresponding `OUTPUT` statement<br>• Matching (three) `ENDIF`s (Or one if `ELSIF`s used)<br><br>Max 4 if solution doesn't work under all circumstances // is not functionally equivalent to CASE | **5** |

| Question | Answer | |
|---|---|---|
| 2(a) | Mark as follows:<br><br>• To search for a given value in the array<br>• and output the position in the array if the search value is found<br>• if search value not found then output “Not found” | |
| 2(b) |  | 9 |

| Question | Answer |
|---|---|
| 2(b) | Mark as follows:<br>• One mark for `START` **and** `STOP` / `END`<br>• One mark for each bracketed pair<br>• One mark for each of other labelled symbol (decision box shape must be correct)<br>• Allow F/T from incorrect decision symbol<br><br>Full marks should be awarded for functionally equivalent solutions. |

| Question | Answer | Marks |
|---|---|---|
| 3 | | **Max 8** |

| Line number | Error | Correction |
|---|---|---|
| 01 | Wrong procedure name – "SortArray" | PROCEDURE ArraySort |
| 02 | Wrong data type - `CHAR` | `DECLARE Temp: STRING` |
| 03 | Variables undefined | `DECLARE FirstID, SecondID, I, J : INTEGER` |
| 04 | Wrong 'Value2' of 100 | `FOR I ← 1 TO 99` |
| 05 | Wrong range | `FOR J ← 1 TO (100 - I)` |
| 06/07 | Wrong function - `MODULUS` | Replace `MODULUS` with `TONUM`:<br>`FirstID ← TONUM(LEFT(Product[J],` |
| 06/07 | Wrong value of 6 | Should be 4:<br>`FirstID ← TONUM(LEFT(Product[J],` |
| 10 | Assigning wrong value to `Temp` | `Temp ← Product[J]` |
| 11 | Assigning wrong value to `Product[I]` | `Product[J] ← Product[J + 1]` |
| 13/14 | Lines reversed | `13 ENDIF`<br>`14 ENDFOR` |

One mark for each correct row

| Question | Answer | |
|---|---|---|
| 4(a) | Pseudocode solution included here for development and clarification of mark schem.<br>Programming language solutions appear in the Appendix.<br><br>`PROCEDURE TestRandom (Repetitions : INTEGER)`<br>`  DECLARE Frequency : ARRAY [1 : 10] OF INTEGER`<br>`  DECLARE Expected : REAL / INTEGER //allow either`<br>`  DECLARE NextRandom : INTEGER`<br>`  DECLARE N : INTEGER`<br><br>`  FOR N ← 1 TO 10`<br>`  Frequency[N] ← 0`<br>`  ENDFOR`<br><br>`  Expected ← INT(Repetitions / 10)`<br><br>`  CALL RANDOMIZE()  //Set random seed`<br><br>`  FOR N ← 1 TO Repetitions`<br>`  NextRandom ← INT(RND() * 10) + 1`<br>`  Frequency[NextRandom] ← Frequency[NextRandom] + 1`<br>`  ENDFOR`<br><br>`  OUTPUT "The expected frequency is " & Expected`<br><br>`  OUTPUT "Number  Frequency  Difference"`<br><br>`  FOR N ← 1 TO 10`<br>`  OUTPUT N & "  " & Frequency[N] & "  " & Frequency[N] –`<br>`  Expected`<br>`  ENDFOR`<br>`ENDPROCEDURE`<br><br>Mark as follows:<br>1. Procedure heading (including parameter)<br>2. Array declaration – 10 or 11 elements<br>3. Array declaration – data type<br>4. Variable declaration for a loop counter (integer) or expected frequency (integer or real)<br>5. Variable declaration for next random value<br><br>(For Python solutions, mark points 1 to 4 may be gained by suitable comments)<br><br>6. Initialise all elements of array<br>7. To set all elements to zero<br>8. Calculate expected frequency | |

| Question | Answer | |
|---|---|---|
| 4(a) | 9. Loop to generate required number of random values<br>10. Use of relevant `RANDOM()` function **in a loop**<br>11. Generate random integer value in the range 1 to 10 **in a loop**<br>12. Increment (array) element **in a loop**<br><br>13. Output expected frequency message **not in any loop**<br>14. Output column header text<br>15. (Loop to) output each row<br>16. ... including three correct values (spaces optional) | |
| 4(b) | • Single-stepping<br>– to allow program statements to be executed one at a time<br>• Breakpoints<br>– to pause / stop the program at a specific line / statement<br>• Variable / expression watch window<br>– to monitor the value of variables / expressions as the program is run<br><br>One mark for each Feature (text as above or equivalent) + 1 for meaningful explanation of use in context. | **6** |
| 4(c) | Program is probably working correctly if:<br>• Header is present giving frequency as 20<br>• Column headers are present<br>• All rows are present (1 to 10)<br>• The difference is calculated correctly<br>• Output is formatted correctly<br>• Total differences should be zero<br>• Sum of Frequencies should be 200 | **Max 2** |

| Question | Answer |
|---|---|
| 5 | ```<br>PROCEDURE RemoveDetails<br>  DECLARE FileLine: STRING<br>  DECLARE MemberToDelete: STRING<br><br>  OPENFILE "EmailDetails.txt" FOR READ<br>  OPENFILE "NewEmailDetails.txt" FOR WRITE<br><br>  INPUT MembershipNumber<br>  WHILE NOT EOF("EmailDetails.txt")<br>  READFILE "EmailDetails.txt", FileLine<br>  IF LEFT(FileLine, 4) <> MembershipNumber<br>  THEN<br>  WRITEFILE "NewEmailDetails.txt", FileLine<br>  ENDIF<br>  ENDWHILE<br><br>  CLOSEFILE "EmailDetails.txt"<br>  CLOSEFILE "NewEmailDetails.txt"<br><br>ENDPROCEDURE<br>```<br><br>Mark as follows:<br>1. Procedure declaration and end. No parameters.<br>2. Variable declaration of `STRING` for variable `FileLine` (or similar)<br>3. Input the `MembershipNumber` of the person who has left<br>4. Open `EmailDetails` for READ<br>5. Open `NewEmailDetails` for `WRITE`<br>6. Correct loop checking for `EOF(EmailDetails)`<br>7. Reading a line from `EmailDetails.txt` **in a loop**<br>8. Correct check for `MemberToDelete` **in a loop**<br>9. Writing a line to `NewEmailDetails.txt` **in a loop**<br>10. Closing both files (not in a loop) |

## Appendix - Program Code Example Solutions

### Q4 (a): Visual Basic

```
Dim random As New Random()

Sub TestRandom(ByVal repetitions As Integer)
   Dim randinrange As Integer
   Dim i As Integer
   Dim num(1 To 10) As Integer
   Dim freq As Integer
   Dim difference As Integer

   freq = repetitions / 10   'calculate expected frequency

   For i = 1 To 10           'initialise array to store total frequencies
      num(i) = 0
   Next i

   For i = 1 To repetitions 'generate random numbers & increment
appropriate freq
      randinrange = random.Next(1, 11)
      num(randinrange) = num(randinrange) + 1
   Next i

   Console.WriteLine("The expected frequency is " & freq)   'report header
   Console.WriteLine("Number   Frequency    Difference")    'column headers

   For i = 1 To 10   'calc & display difference between expected and actual
freq
      difference = num(i) - freq
      Console.WriteLine(i & "       " & num(i) & "        " & difference)
   Next i

End Sub
```

**Other possible ways of calculating a random number in VB include:**

```
   randinrange = CInt(Math.Floor((upperbound - lowerbound + 1) * Rnd())) +
lowerbound

   randinrange = math.round((Rnd()*9)+1)

   randinrange = CInt(Math.Ceiling(Rnd() * 9
```

**Q4 (a): Pascal**

```
procedure TestRandom(var Repetitions : integer);
   var
      Frequency : array[1..10] :  integer;
      Expected, NextRandom, N  : integer;

   begin
      Expected := Round(Repetitions/10);
      for N := 1 to 10 do
         Frequency[N] := 0;

      for N := 1 to Repetitions do
      begin
         NextRandom := random(10)+1;
         Frequency[NextRandom] := Frequency[NextRandom]+1;
      end;

      writeln ('The expected frequency is ', Expected);
      writeln ('Number Frequency Difference');

      for N := 1 to 10 do
         writeln ('   ',N,'      ',Frequency[N],'       ',Frequency[N]-
Expected);

   end;
```

**Q4 (a): Python**

```
# frequency is an array from 1 to 10 of type integer;
# nextNumber is an integer which stores the created random number
# expected is an integer which stores the expected frequency of each nu

def TestRandom (repetitions):
   import random
   frequency = [0 for i in range(1,11)]  # initialise each frequency count
to 0

   expected = repetitions / 10

   for i in range(1, repetitions + 1):
      nextNumber = random.randint(1,10)
      frequency[nextNumber] = frequency[nextNumber]+ 1

   print ("The expected frequency is ",  expected)
   print("   Number  Frequency  Difference")

   for i in range(1,11):
     print ("    ", i, "    ", frequency[i],"    ", frequency[i] -
expected)
```

**Alternative:**

```
def TestRandom (repetitions):
   expected = repetitions / 10     ## initialised as real/integer
                                   ## NextRandom and N defined as integers
   frequency =[0,0,0,0,0,0,0,0,0,0]  ## defined as an array and
initialised to zero

   for n in range (0,repetitions):
      nextNumber = randint(1, 10)
      frequency[nextNumber] += 1

   print ('The expected frequency is ', expected)

   print ('Number  Frequency  Difference')
   for n in range (1, 11):
     print (n,'       ',frequency[n],'          ',frequency[n] - expected)
```

**Alternative:**

```
   frequency =[0]*11        ## alternate way to initialise array to zero
   frequency =[]            ## empty array/list
```

**Alternative:**

```
    for n in range (1,11):
       frequency[n-1] = 0       ##alternate way to initialise array to zero
```

**Alternative:**

```
for n in range (0,11):  ##alternate way to initialise array to 
    frequency.append(0)
```

# QUESTION 11.

| Question | Answer | Marks |
|---|---|---|
| 2(a) | One mark for each point:<br><br>• Initialise a count to zero<br>• loop 100 times // loop through all of the array<br>• compare an element with "Empty" **in a loop**<br>• increment the count if equal **in a loop**<br>• Output a message together with the count **not inside a loop** | **5** |
| 2(b) | One mark for each point:<br><br>• The breaking down of an algorithm / task / problem<br>• to a level of (sufficient) detail // into smaller parts / sub-tasks<br>• from which it can be programmed // which are easier to program | **3** |

| Question | Answer | |
|---|---|---|
| 2(c) | **Mode**: READ<br>**Description**: Used to read data from a file // input data from a file to the program // only allows you to read data from the file / can't change the data<br><br>**Mode**: APPEND<br>**Description**: Used to add data // write to the end of the text file // output data from the program to the end of a file (without changing / deleting anything)<br><br>One mark for mode; one mark for corresponding description | |
| 2(d) | **Write:**<br>Use an editor to write the source code / program / high-level language code.<br><br>*Or by example of feature:*<br>An editor provides (features such as) context-sensitive prompts / dynamic syntax checking / PrettyPrint / auto-indentation etc.<br><br>**Translate:**<br>A translator (compiler) will convert the source code / program / high-level language code into object code / machine code / an executable file<br><br>A translator (interpreter) is used to translate the source code / program / high-level language code line by line // Or by example: identify syntax errors<br><br>*Or by example of feature:*<br>A translator will identify errors<br><br>**Test:**<br>A debugger is used to find / (help to) correct errors.<br><br>*Or by example of feature:*<br>e.g. single-step, break-points, watch-window…<br><br>One mark per category (Write, Translate, Test) for each reference to a specific 'feature'. | **3** |

# QUESTION 12.

| | | |
|---|---|---|
| | 4. Iteration / Repetition | |
| 2(b) | One mark for name and one mark for explanation.<br><br>Example:<br><br>• PrettyPrint // Colour coding<br>• Colour coding of command words / key words<br><br>• Expand and collapse code blocks<br>• Allows programmer to focus on a section of code // allows quicker navigation of the code<br><br>• Auto(matic) indentation<br>• Allows the programmer to clearly see the different code sections / easier to see the code structure<br><br>Accept suitable alternatives | **2** |
| 2(c) | One mark for identification, one mark for description:<br><br>• By reference / ref<br>• The address of / pointer to the parameter is passed to the subroutine // if the parameter value is changed in the subroutine this changes the original value | **2** |
| 2(d) | One mark per bullet point:<br><br>• Changes made to // Updating // Editing a program / algorithm / data structure / software / system<br>• ...as a result of changes to requirements / specification / legislation / available technology | **2** |

# QUESTION 13.

| Question | Answer | Marks |
|---|---|---|
| 2(a) | One mark per point:<br><br>1. The source code represents a solution / design / algorithm expressed in a high-level language<br>2. The Object code is produced (by the compiler) during the translation stage // The Object code is produced by translating the source code (NOT produced by Interpreter)<br>3. Corrective maintenance occurs when testing reveals a fault (or error) in the program **and** this is corrected // Corrective maintenance is when errors are found and fixed // Corrective maintenance is when a program is debugged<br><br>Accept alternative answers provided they relate to the **program development cycle** | **3** |
| 2(b) | Any **three** from:<br><br>1. Dynamic syntax checking // Identification of syntax errors<br>2. Highlighting undeclared variables // incorrect variable usage...<br>3. Parameter checking<br>4. Type checking<br>5. Auto-indentation<br>6. PrettyPrint | **3** |