Cambridge Assessment
International Education

Cambridge International AS & A Level

SUBJECT **9618/22**

Paper 22 Fundamental Problem Solving & Programming Skills **May/June 2022**

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 2022 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 **14** 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:

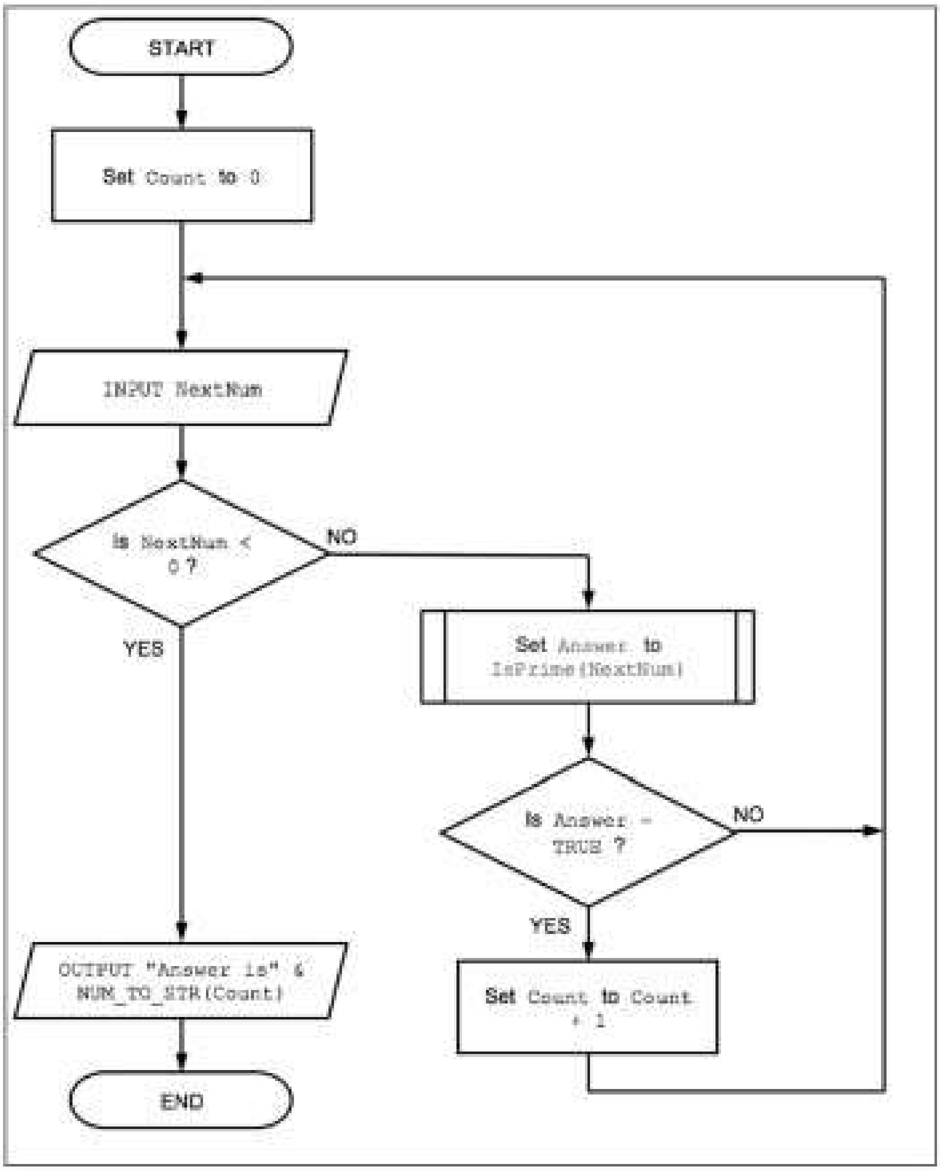
Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).

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>Correct answer only:
Breakpoint</td><td>1</td></tr>
<tr><td>1(b)</td><td>One mark per row
<table>
<tr><th>Activity</th><th>Life cycle stage</th></tr>
<tr><td>An identifier table is produced.</td><td>Design</td></tr>
<tr><td>Syntax errors can occur.</td><td>Coding</td></tr>
<tr><td>The developer discusses the program requirements with the customer.</td><td>Analysis</td></tr>
<tr><td>A trace table is produced.</td><td>Testing</td></tr>
</table></td><td>4</td></tr>
<tr><td>1(c)</td><td>One mark per bullet point to Max 2
• A description of what the identifier is used for / the purpose of the identifier
• The data type of the identifier
• The number of elements of an array // the length of a string
• An example data value
• Value of any constants used
• The scope of the variable (local or global)</td><td>2</td></tr>
<tr><td>1(d)</td><td>One mark per row
<table>
<tr><th>Statement</th><th>Error</th></tr>
<tr><td>Status ← TRUE AND FALSE</td><td>NO ERROR</td></tr>
<tr><td>IF LENGTH("Password") < "10" THEN</td><td>"10" shouldn’t be a string // must be an integer</td></tr>
<tr><td>Code ← LCASE("Electrical")</td><td>Parameter must be a char // cannot be a string
Alternative:
LCASE should be TO_LOWER</td></tr>
<tr><td>Result ← IS_NUM(-27.3)</td><td>Parameter must be a string / char // cannot be a number</td></tr>
</table></td><td>4</td></tr>
</table>

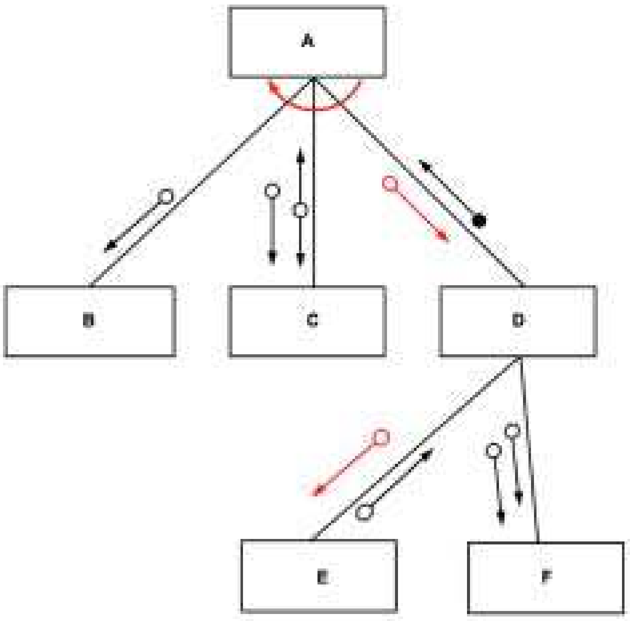
<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2</td><td>



One mark per point:

1 Initialise `Count` before loop **AND** Input `NextNum` **in a loop**
2 Loop until `NextNum` < 0 **AND** OUTPUT statement including `Count` plus a message
3 Use of `IsPrime(NextNum)` as a function (must return a value)
4 Check return value **AND** increment `Count` if appropriate

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

Question	Answer	Marks
3(a)(i)	One mark per red annotation  	**3**
3(a)(ii)	<table><tr><th>Label</th><th>Module name</th></tr><tr><td>**A**</td><td>**Head**</td></tr><tr><td>**B**</td><td>**Mod_W**</td></tr><tr><td>**C**</td><td>**Mod_X**</td></tr><tr><td>**D**</td><td>**Mod_V**</td></tr><tr><td>**E**</td><td>**Mod-Z**</td></tr><tr><td>**F**</td><td>**Mod_Y**</td></tr></table> Marks as follows: • Two rows correct – one mark • Four rows correct – two marks • All rows correct – three marks	**3**
3(b)	One mark per point: • Breaking a complex problem down makes it easier to understand / solve // smaller problems are easier to understand / solve • Smaller problems are easier to program / test / maintain • Sub-problems can be given to different teams / programmers with different expertise // can be solved separately	**3**

Question	Answer	Marks
4(a)		6

```
PROCEDURE LastLines(ThisFile : STRING)
   DECLARE ThisLine, LineX, LineY, LineZ : STRING

   OPENFILE ThisFile FOR READ

   LineY ← ""
   LineZ ← ""

   WHILE NOT EOF(ThisFile)
      READFILE Thisfile, ThisLine        // read a line
      LineX ← LineY
      LineY ← LineZ
      LineZ ← ThisLine
   ENDWHILE

   CLOSEFILE ThisFile

   OUTPUT LineX
   OUTPUT LineY
   OUTPUT LineZ

ENDPROCEDURE
```

Marks as follows to **Max 6**:

1 Procedure heading (including parameter) and ending
2 Declaration of local variables for three lines **AND** File `OPEN` in `READ` mode **AND** `CLOSE`
3 Loop until `EOF(ThisFile)`
4 Read line from file... **in a loop**
5 Attempt at a shuffle… **in a loop**
6 Correctly shuffle `LineX`, `LineY` and `LineZ` **in a loop**
7 `OUTPUT` the three lines in correct sequence, following reasonable attempt

Question	Answer	Marks
4(a)	**Alternative (using two loops):**	

```
PROCEDURE LastLines(ThisFile : STRING)
   DECLARE ThisLine, LineX, LineY, LineZ : STRING
   DECLARE Count, Count2 : INTEGER

   Count ← 0
   OPENFILE ThisFile FOR READ

   WHILE NOT EOF(ThisFile)
      READFILE Thisfile, ThisLine      // read a line
      Count ← Count + 1
   ENDWHILE

   CLOSEFILE ThisFile
   OPENFILE ThisFile FOR READ

   FOR Count2 ← 1 TO Count - 3
      READFILE Thisfile, ThisLine     // read a line
   NEXT Count2

   READFILE Thisfile, LineX
   READFILE Thisfile, LineY
   READFILE Thisfile, LineZ

   OUTPUT LineX
   OUTPUT LineY
   OUTPUT LineZ

   CLOSEFILE ThisFile

ENDPROCEDURE
```

Marks as follows to **Max 6**:

1 Procedure heading (including parameter) and ending
2 Declaration of local variables for three lines **AND** (at least one) File `OPEN` in `READ` mode **AND** `CLOSE`
3 Loop until `EOF(ThisFile)`
4 Read line from file and increment `Count` **in a loop**
5 Two separate loops, closing and re-opening the file between loops

6 Read `Count - 3` lines from the file

7 `OUTPUT` the last three lines in correct sequence, following reasonable attempt

Question	Answer	Marks
4(b)	One mark per point to **Max 3**: 1 Change the procedure header to include a (numeric) parameter (as well as the filename) 2 Replace `LineX`, `Y` and `Z` with an array 3 Amend shuffle mechanism 4 Use new parameter to determine first line to output 5 Output the lines in a loop Alternative **'two loop'** solution **to Max 3:** 1 Change the procedure header to include a numeric parameter (as well as the filename) 2 A loop to count the total number of lines in the file 3 Ref use of single variable rather than `LineX`, `LineY` and `LineZ` 4 Close and re-open the file 5 Use the new parameter value to determine first line to output 6 Output the lines in a loop	**3**

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(a)</td><td>One mark for type and one mark for condition:
Independent marks

<b>Type</b>: pre-condition
<b>Condition</b>: when the value of <code>ThisNum</code> / the input value is equal to zero</td><td>2</td></tr>
<tr><td>5(b)</td><td>
<table>
<tr><th>ThisNum</th><th>ThisChar</th><th>CountA</th><th>CountB</th><th>Flag</th><th>OUTPUT</th></tr>
<tr><td></td><td></td><td>0</td><td>10</td><td>TRUE</td><td></td></tr>
<tr><td>12</td><td>'1'</td><td>1</td><td></td><td></td><td></td></tr>
<tr><td>24</td><td>'2'</td><td></td><td></td><td></td><td></td></tr>
<tr><td>57</td><td>'5'</td><td></td><td></td><td></td><td>"Ignored"</td></tr>
<tr><td>43</td><td>'4'</td><td></td><td>9</td><td>FALSE</td><td></td></tr>
<tr><td>56</td><td>'5'</td><td>4</td><td></td><td></td><td></td></tr>
<tr><td>22</td><td>'2'</td><td></td><td></td><td>TRUE</td><td>"Flip"</td></tr>
<tr><td>31</td><td>'3'</td><td></td><td>8</td><td></td><td></td></tr>
<tr><td>32</td><td>'3'</td><td></td><td>7</td><td></td><td></td></tr>
<tr><td>47</td><td>'4'</td><td></td><td>6</td><td>FALSE</td><td></td></tr>
<tr><td>99</td><td>'9'</td><td></td><td></td><td>TRUE</td><td>"Flip"</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td>4</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Marks as follows:
One mark per outlined group
If <b>no</b> marks per group then mark by columns (columns 3 to 6) <b>for max 4</b></td><td>6</td></tr>
<tr><td>5(c)</td><td>One mark per point:
• Modules that have already been tested individually
• are combined into a single (sub) program which is then tested as a whole</td><td>2</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>6</td><td>

```
PROCEDURE Parse(InString : STRING)
   DECLARE Count, Total, Index : INTEGER
   DECLARE Average : REAL
   DECLARE NumString : STRING
   DECLARE ThisChar : CHAR

   CONSTANT COMMA = ','

   Count ← 0
   Total ← 0
   NumString ← ""

   FOR Index ← 1 to LENGTH(InString)
      ThisChar ← MID(InString, Index, 1)
      IF ThisChar = COMMA THEN
         Total ← Total + STR_TO_NUM(NumString)
         Count ← Count + 1
         NumString ← ""
      ELSE
         NumString ← NumString & ThisChar  // build the number
                                              string
      ENDIF
   NEXT Index

   // now process the final number
   Total ← Total + STR_TO_NUM(NumString)
   Count ← Count + 1

   Average ← Total / Count
   OUTPUT "The total was ", Total, " and the average was ",
          Average

ENDPROCEDURE
```

Marks as follows:
1 Declare and initialise `Count`, `Total` and `NumString`
2 Loop for number of characters in `InString`
3 Extract a character and test for comma **in a loop**
4 If comma, convert `NumString` to integer and update `Total` and `Count`
5 and reset `NumString`
6 Otherwise append character to `NumString`
7 Calculate average **AND** final output statement(s) **outside the loop**

</td><td>7</td></tr>
</table>

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>7(a)</td><td>

```
FUNCTION MID(InString : STRING, Start, Num : INTEGER)
             RETURNS STRING
   DECLARE MidString : STRING
   DECLARE InStringLen : INTEGER

   InStringLen ← LENGTH(InString)

   // solution for RIGHT() then LEFT()
   MidString ← RIGHT(InString, InStringLen – Start + 1)
   MidString ← LEFT(MidString, Num)

   // alternative solution for LEFT() then RIGHT()
   MidString ← LEFT(InString, Start + Num - 1)
   MidString ← RIGHT(MidString, Num)


   RETURN MidString
ENDFUNCTION
```

Marks as follows:
1 Function heading and ending including parameters and return type
2 Correct use of **one** substring functions
3 Correct use of **both** substring functions (in correct sequence)
4 Return substring after a reasonable attempt</td><td>4</td></tr>
<tr><td>7(b)</td><td>One mark per point
Check that:
• Start and/or Num are >= 1 // positive
• Length of InString is "sufficient" for required operation</td><td>2</td></tr>
</table>

Question	Answer	Marks
8(a)		**5**

```
FUNCTION Exists(ThisString : STRING, Search : CHAR)
                RETURNS BOOLEAN
   DECLARE Found : BOOLEAN
   DECLARE Index : INTEGER

   Found ← FALSE
   Index ← 1

   WHILE Found = FALSE AND Index <= LENGTH(ThisString)
      IF MID(ThisString, Index, 1) = Search THEN
         Found ← TRUE
      ELSE
         Index ← Index + 1
      ENDIF
   ENDWHILE

   RETURN Found

ENDFUNCTION
```

Marks as follows (Conditional loop solution):

1 Conditional loop while character not found and not end of string
2 Extract a char **in a loop**
3 Compare with parameter **without** case conversion **in a loop**
4 If match found, set termination logic **in a loop**
5 Return `BOOLEAN` value

ALTERNATIVE (Using Count-controlled loop):

```
  FOR Index ← 1 TO LENGTH(ThisString)
     IF MID(ThisString, Index, 1) = Search THEN
        RETURN TRUE
     ENDIF
   NEXT Index
   RETURN FALSE
```

Marks as follows (Count-controlled loop variant):

1 Loop for length of `ThisString` (allow from 0 or 1)
2 Extract a char **in a loop**
3 Compare with parameter without case conversion **in a loop**
4 If match found, immediate RETURN of TRUE
5 Return `FALSE` after the loop // Return Boolean if no immediate RETURN

Question	Answer	Marks
8(b)		8

```
PROCEDURE SearchDuplicates()
  DECLARE IndexA, IndexB : INTEGER
  DECLARE ThisPassword, ThisValue : STRING
  DECLARE Duplicates : BOOLEAN

  Duplicates ← FALSE
  IndexA ← 1

  WHILE Duplicates = FALSE AND IndexA < 500
     ThisValue ← Secret[IndexA, 2]
     IF ThisValue <> "" THEN
        ThisPassword ← Decrypt(ThisValue)
        FOR IndexB ← IndexA + 1 TO 500 //
           IF Secret[IndexB, 2] <> "" THEN
              IF Decrypt(Secret[IndexB, 2]) = ThisPassword
                 THEN
                 OUTPUT "Password for " & Secret[IndexA, 1] &
                        "also used for " & Secret[IndexB, 1]
                 Duplicates ← TRUE
              ENDIF
           ENDIF
        NEXT IndexB
     ENDIF
     IndexA ← IndexA + 1
  ENDWHILE

  IF Duplicates = FALSE THEN
     OUTPUT "No duplicate passwords found"
  ENDIF

ENDPROCEDURE
```

Marks as follows to **Max 8**:

1. (Any) conditional loop...
2. ... from 1 to 499 while (attempt at) no duplicate
3. Skip unused password
4. Use `Decrypt()` and assign return value to `ThisPassword`
5. Inner loop from outer loop index + 1 to 500 searching for duplicates
6. Compare `ThisPassword` with subsequent passwords (**after** use of `Decrypt()`)
7. If match found, set outer loop termination
8. and attempt an Output message giving duplicate
9. Output 'No duplicate passwords found' message if no duplicates found **after the loop**

Question	Answer	Marks
8(c)	One mark for each point that is referenced: 1 Initialise password to empty string at the start **and** return (attempted) password at the end of the function 2 Two loops to generate 3 groups of 4 characters // One loop to generate 12 / 14 characters 3 Use of `RandomChar()` to generate a character **in a loop** 4 Reject character if `Exists()` returns `TRUE`, otherwise form string **in a loop** 5 (Attempt to) use hyphens to link three groups 6 Three groups of four characters generated correctly with hyphens and without duplication (completely working algorithm)	**6**