Cambridge Assessment International Education

Cambridge International AS & A Level

CANDIDATE NAME

CENTRE NUMBER | | | | | |

CANDIDATE NUMBER | | | | |

6646749220

COMPUTER SCIENCE **9618/22**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 2022**

2 hours

You must answer on the question paper.

You will need: Insert (enclosed)

INSTRUCTIONS

- Answer **all** questions.
- Use a black or dark blue pen.
- Write your name, centre number and candidate number in the boxes at the top of the page.
- Write your answer to each question in the space provided.
- Do **not** use an erasable pen or correction fluid.
- Do **not** write on any bar codes.
- You may use an HB pencil for any diagrams, graphs or rough working.
- Calculators must **not** be used in this paper.

INFORMATION

- The total mark for this paper is 75.
- The number of marks for each question or part question is shown in brackets [ ].
- No marks will be awarded for using brand names of software packages or hardware.
- The insert contains all the resources referred to in the questions.

This document has **20** pages. Any blank pages are indicated.

Refer to the **insert** for the list of pseudocode functions and operators.

1 **(a)** A programmer is testing a program using an Integrated Development Environment (IDE). The programmer wants the program to stop when it reaches a specific instruction or program statement in order to check the value assigned to a variable.

Give the technical term for the position at which the program stops.

..... Breakpoint ..... [1]

(b) The following table lists some activities from the program development life cycle.

Complete the table by writing the life cycle stage for each activity.

Activity	Life cycle stage
An identifier table is produced.	Design
Syntax errors can occur.	Coding
The developer discusses the program requirements with the customer.	Analysis
A trace table is produced.	Testing

[4]

(c) An identifier table includes the names of identifiers used.

State **two other** pieces of information that the identifier table should contain.

1 .....

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)

[2]

(d) The pseudocode statements in the following table may contain errors.

State the error in each case or write 'NO ERROR' if the statement contains no error.

You can assume that none of the variables referenced are of an incorrect type.

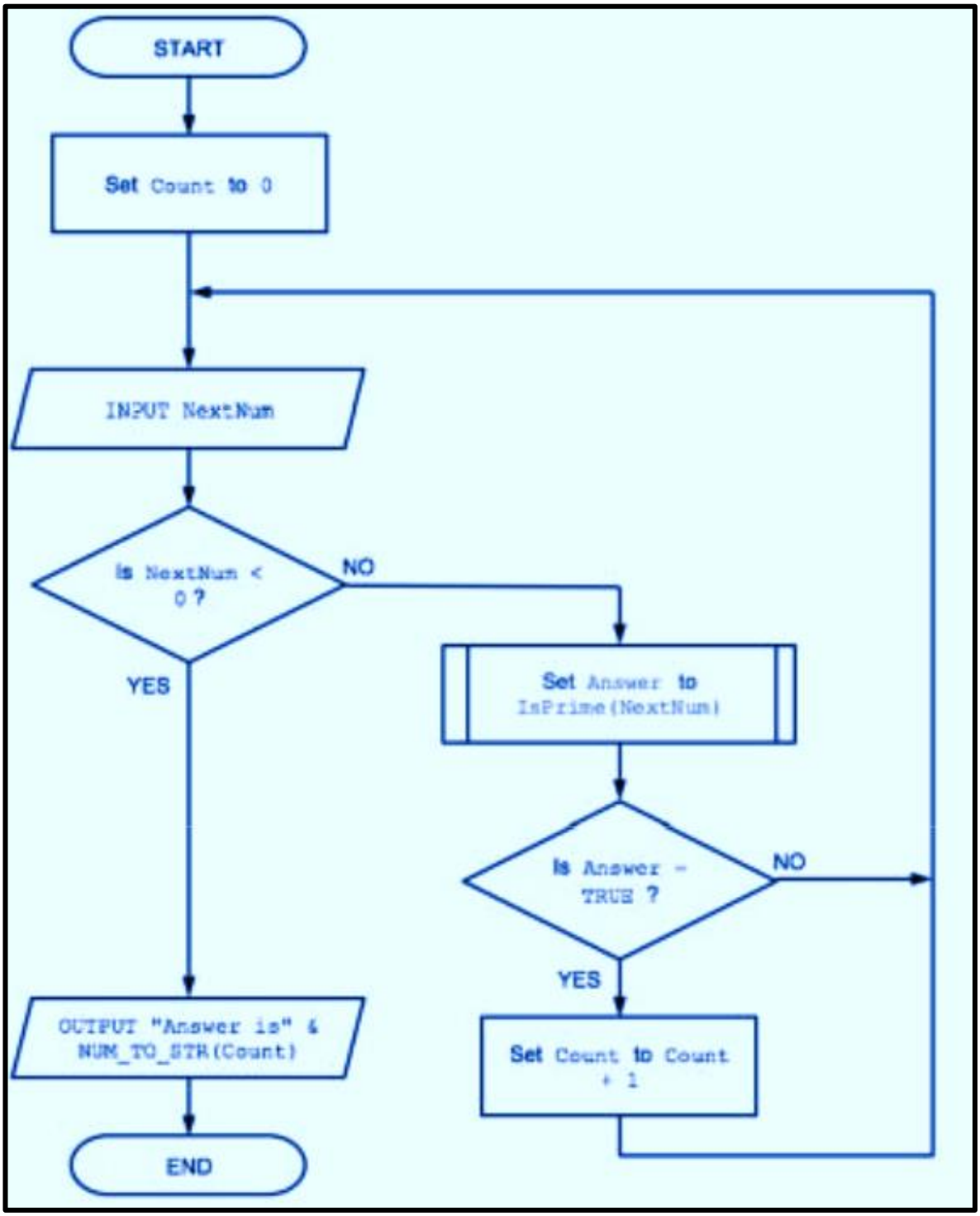
Statement	Error
`Status ← TRUE AND FALSE`	NO ERROR
`IF LENGTH("Password") < "10" THEN`	"10" shouldn't be a string // must be an integer
`Code ← LCASE("Electrical")`	Parameter must be a char // cannot be a string Alternative: LCASE should be TO_LOWER
`Result ← IS_NUM(-27.3)`	Parameter must be a string / char // cannot be a number

[4]

2 An algorithm is described as follows:

1. Input an integer value.
2. Jump to step 6 if the value is less than zero.
3. Call the function `IsPrime()` using the integer value as a parameter.
4. Keep a count of the number of times function `IsPrime()` returns `TRUE`.
5. Repeat from step 1.
6. Output the value of the count with a suitable message.

Draw a program flowchart to represent the algorithm.



[4]

3 **(a)** The module headers for five modules in a program are defined in pseudocode as follows:

Pseudocode module header
`FUNCTION Mod_V(S2 : INTEGER) RETURNS BOOLEAN`
`PROCEDURE Mod_W(P4 : INTEGER)`
`PROCEDURE Mod_X(T4 : INTEGER, BYREF P3 : REAL)`
`PROCEDURE Mod_Y(W3 : REAL, Z8 : INTEGER)`
`FUNCTION Mod_Z(F3 : REAL) RETURNS INTEGER`

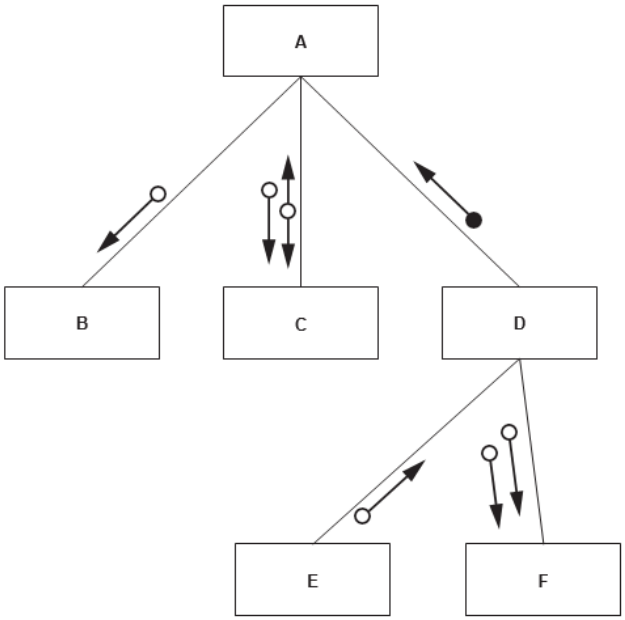
An additional module `Head()` repeatedly calls three of the modules in sequence.

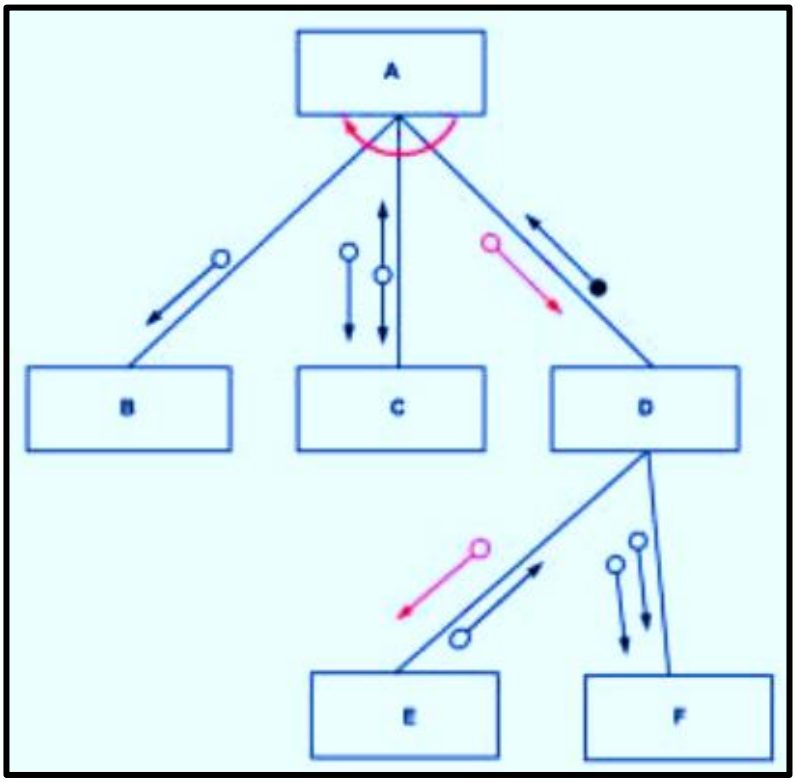
A structure chart has been partially completed.

(i) Complete the structure chart to include the information given about the six modules.

Do **not** label the parameters and do **not** write the module names.

[3]





(ii) Complete the table using the information in **part 3(a)** by writing each module name to replace the labels **A** to **F**.

Label	Module name
A	Head
B	Mod_W
C	Mod_X
D	Mod_V
E	Mod-Z
F	Mod_Y

[3]

(b) The structure chart represents part of a complex problem. The process of decomposition is used to break down the complex problem into sub-problems.

Describe **three** benefits of this approach.

1 .....

- 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

2 .....

3 .....

.....

[3]

4 **(a)** A procedure `LastLines()` will:

- take the name of a text file as a parameter
- output the **last** three lines from that file, in the same order as they appear in the file.

Note:

- Use local variables `LineX`, `LineY` and `LineZ` to store the three lines from the file.
- You may assume the file exists and contains **at least** three lines.

Write pseudocode for the procedure `LastLines()`.

```
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
```

[6]

(b) The algorithm in **part (a)** is to be amended. The calling program will pass the number of lines to be output as well as the name of the text file.

The number of lines could be any value from 1 to 30.

It can be assumed that the file contains **at least** the number of lines passed.

Outline **three** changes that would be needed.

1 .....

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

2 .....

.....

.....

3 .....

.....

.....

[3]

5 Study the following pseudocode. Line numbers are for reference only.

```
10 PROCEDURE Encode()
11   DECLARE CountA, CountB, ThisNum : INTEGER
12   DECLARE ThisChar : CHAR
13   DECLARE Flag : BOOLEAN
14   CountA ← 0
15   CountB ← 10
16   Flag ← TRUE
17   INPUT ThisNum
18   WHILE ThisNum <> 0
19      ThisChar ← LEFT(NUM_TO_STR(ThisNum), 1)
20      IF Flag = TRUE THEN
21         CASE OF ThisChar
22            '1' : CountA ← CountA + 1
23            '2' : IF CountB < 10 THEN
24                     CountA ← CountA + 1
25                  ENDIF
26            '3' : CountB ← CountB - 1
27            '4' : CountB ← CountB - 1
28                  Flag ← FALSE
29            OTHERWISE : OUTPUT "Ignored"
30         ENDCASE
31      ELSE
32         IF CountA > 2 THEN
33            Flag ← NOT Flag
34            OUTPUT "Flip"
35         ELSE
36            CountA ← 4
37         ENDIF
38      ENDIF
39      INPUT ThisNum
40   ENDWHILE
41   OUTPUT CountA
42 ENDPROCEDURE
```

(a) Procedure `Encode()` contains a loop structure.

Identify the type of loop **and** state the condition that ends the loop.

Do **not** include pseudocode statements in your answer.

Type ..... pre-condition .....

Condition ..... when the value of `ThisNum` / the input value is equal to zero .....

.....

[2]

(b) Complete the trace table below by dry running the procedure `Encode()` when the following values are input:

`12, 24, 57, 43, 56, 22, 31, 32, 47, 99, 0`

The first row is already complete.

ThisNum	ThisChar	CountA	CountB	Flag	OUTPUT
		0	10	TRUE	

[6]

ThisNum	ThisChar	CountA	CountB	Flag	OUTPUT
		0	10	TRUE	
12	'1'	1			
24	'2'				
57	'5'				"Ignored"
43	'4'		9	FALSE	
56	'5'	4			
22	'2'			TRUE	"Flip"
31	'3'		8		
32	'3'		7		
47	'4'		6	FALSE	
99	'9'			TRUE	"Flip"
0					4

(c) Procedure `Encode()` is part of a modular program. Integration testing is to be carried out on the program.

Describe **integration testing**.

- Modules that have already been tested individually
- are combined into a single (sub) program which is then tested as a whole

.....

..... [2]

6 A string represents a series of whole numbers, separated by commas.

For example:

"12,13,451,22"

Assume that:

- the comma character ',' is used as a separator
- the string contains only the characters '0' to '9' and the comma character ','.

A procedure `Parse` will:

- take the string as a parameter
- extract each number in turn
- calculate the total value and average value of all the numbers
- output the total and average values with a suitable message.

Write pseudocode for the procedure.

```
PROCEDURE Parse(InString : STRING)
```

```
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
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```

[7]

7 A programming language has string functions equivalent to those given in the **insert**.

The language includes a `LEFT()` and a `RIGHT()` function, but it does **not** have a `MID()` function.

(a) Write pseudocode for an algorithm to implement your own version of the `MID()` function which will operate in the same way as that shown in the **insert**.

Do **not** use the `MID()` function given in the **insert**, but you may use any of the other functions.

Assume that the values passed to the function will be correct.

```
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
```

[4]

(b) The values passed to your `MID()` function in **part (a)** need to be validated.

Assume that the values are of the correct data type.

State **two** checks that could be applied to the values passed to the function.

1 • `Start` and/or `Num` are >= 1 // positive

2 • Length of `InString` is "sufficient" for required operation

[2]

BLANK PAGE

8 A program allows a user to save passwords used to log in to websites. A stored password is then inserted automatically when the user logs in to the corresponding website.

A global 2D array `Secret` of type `STRING` stores the passwords together with the website domain name where they are used. `Secret` contains 1000 elements organised as 500 rows by 2 columns.

Unused elements contain the empty string (`""`). These may occur anywhere in the array.

An example of a part of the array is:

Array element	Value
`Secret[27, 1]`	"thiswebsite.com"
`Secret[27, 2]`	"••••••••••••"
`Secret[28, 1]`	"thatwebsite.com"
`Secret[28, 2]`	"••••••••••••"

Note:

- For security, the passwords are stored in an encrypted form, shown as "••••••••••••" in the example.
- The passwords cannot be used without being decrypted.
- You may assume that the encrypted form of a password will **NOT** be an empty string.

The programmer has started to define program modules as follows:

Module	Description

`Exists()`	• Takes two parameters:
	  ◦ a string
	  ◦ a character
	• Performs a case-sensitive search for the character in the string
	• Returns `TRUE` if the character occurs in the string, otherwise returns `FALSE`

`Encrypt()`	• Takes a password as a parameter of type string • Returns the encrypted form of the password as a string
`Decrypt()`	• Takes an encrypted password as a parameter of type string • Returns the decrypted form of the password as a string

Note: in a case-sensitive comparison, 'a' is not the same as 'A'.

(a) Write pseudocode for the module `Exists()`.

```
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
```

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
```

[5]

(b) A new module `SearchDuplicates()` will:

- search for the **first** password that occurs more than once in the array and output a message each time a duplicate is found.

  For example, if the same password was used for the three websites ThisWebsite.com, website27.net and websiteZ99.org, then the following messages will be output:

```
"Password for ThisWebsite.com also used for website27.net"
"Password for ThisWebsite.com also used for websiteZ99.org"
```

- end once all messages have been output.

The module will output a message if no duplicates are found.
For example:

```
"No duplicate passwords found"
```

Write efficient pseudocode for the module `SearchDuplicates()`. `Encrypt()` and `Decrypt()` functions have been written.

Note: It is necessary to decrypt each password before checking its value.

```
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
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [8]

(c) A password has a fixed format, consisting of **three groups of four** alphanumeric characters, separated by the hyphen character '-'.

An example of a password is:

"FxAf-3haV-Tq49"

Each password must:

- be 14 characters long
- be organised as three groups of four alphanumeric characters. The groups are separated by hyphen characters
- not include any duplicated characters, except for the hyphen characters.

An algorithm is needed for a new function `GeneratePassword()`, which will generate and return a password in this format.

Assume that the following modules have already been written:

Module	Description

`Exists()`	• Takes two parameters:
	  ◦ a string
	  ◦ a character
	• Performs a case-sensitive search for the character in the string
	• Returns `TRUE` if the character occurs in the string, otherwise returns `FALSE`

`RandomChar()`	• Generates a single random character from within one of the following ranges:
	  ◦ 'a' to 'z'
	  ◦ 'A' to 'Z'
	  ◦ '0' to '9'
	• Returns the character

Note: in a case-sensitive comparison, 'a' is not the same as 'A'.

Describe the algorithm for `GeneratePassword()`.

Do **not** use pseudocode statements in your answer.

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]

BLANK PAGE