Question No. 1

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

```
FUNCTION GetStart (WordNum : INTEGER) RETURNS INTEGER
  DECLARE Index, ThisPos, NumFound : INTEGER
  DECLARE ThisChar : Char

  CONSTANT SPACECHAR = ' '

  Index ← -1
  Numfound ← 0
  ThisPos ← 1

  IF WordNum = 1 THEN   // if looking for word 1...
     Index ← 1          // Word 1 always starts at index
                       // position 1
  ELSE
                        // Otherwise start counting spaces...
     WHILE ThisPos <= LENGTH(FNString) AND Index = -1
        ThisChar ← MID(FNString, ThisPos, 1)
        IF ThisChar = SPACECHAR THEN
           NumFound ← NumFound + 1
           IF NumFound = WordNum - 1 THEN
              Index ← ThisPos + 1 // the start of the
                                  // required word
           ENDIF
        ENDIF
        ThisPos ← ThisPos + 1
     ENDWHILE
  ENDIF
RETURN Index

ENDFUNCTION
```

1 mark for each of the following:

1 Function heading, including return type and function end
2 Loop counting spaces until word found or end of `FNString`
3 extract a character from `FNString` **in a loop**
4 compare with `SPACECHAR` **and** increment count if equal **in a loop**
5 compare count with `WordNum - 1` (depending on initialisation value) **in a loop**
6 if equal then set flag or `Index` to `ThisPos + 1` **in a loop**
7 Return Index (correctly in all cases / following a reasonable attempt)
8 Works for special case when looking for word 1

Note: Max 7 marks

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

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>7(b)</td><td>Marks awarded for any reference to each of the following steps of the algorithm:

1 Mention of variable for use as array index
2 Use of a loop (to check through the array)
3 If word is the same as the current array element then return FALSE / set flag
4 If word not already in array, loop to find unused element <i>(second loop)</i>
5 Store word in unused element and return TRUE, otherwise return FALSE

VARIATION:

1 Mention of variable for use as array index
2 Use of a loop (to check through the array)
3 Save index of (first) unused element found
4 If word is the same as the current array element then return FALSE / set flag
5 If word not already in array <b>and</b> unused element available, store word in unused element and return TRUE otherwise return FALSE

<b>Note: Max 4 marks</b></td><td><b>4</b></td></tr>
<tr><td>7(c)</td><td><pre>
FUNCTION GetWord (Index : INTEGER) RETURNS STRING

  DECLARE NextWord : STRING
  DECLARE Done : BOOLEAN
  DECLARE ThisChar : CHAR
  DECLARE Index : INTEGER

  CONSTANT SPACECHAR = ' '

  NextWord ← ""
  Done ← FALSE

  REPEAT
    ThisChar ← MID(FNString, Index, 1)
    IF ThisChar <> SPACECHAR THEN
      NextWord ← NextWord & ThisChar // build up NextWord
    ENDIF
    IF ThisChar = SPACECHAR OR Index = LENGTH(FNString) THEN
        Done ← TRUE
    ENDIF

    Index ← Index + 1

  UNTIL Done = TRUE

RETURN NextWord

ENDFUNCTION
</pre>1 mark for each of the following:

1 Conditional loop
2 Extract char from FNString and compare with SPACECHAR <b>in a loop</b>
3 Concatenate with NextWord if not SPACECHAR <b>in a loop</b>
4 Exit loop when SPACECHAR encountered <b>or</b> when end of FNString reached
5 Return NextWord (after reasonable attempt at forming, and must have been initialised)</td><td><b>5</b></td></tr>
</table>

7(c)

The ‘length and substring’ solution:

```
FUNCTION GetWord (Index : INTEGER) RETURNS STRING
  DECLARE Done : BOOLEAN
  DECLARE ThisChar : CHAR
  DECLARE Index, NextPos : INTEGER

  CONSTANT SPACECHAR = ' '

  Done ← FALSE
  NextPos ← Index // must be at least one character in
                  // the required word

  REPEAT
     ThisChar ← MID(FNString, NextPos, 1)
     IF ThisChar = SPACECHAR OR NextPos =
                            LENGTH(FNString) THEN
        Done ← TRUE
     ELSE
        NextPos ← NextPos + 1
     ENDIF
  UNTIL Done = TRUE

  IF NextPos = LENGTH(FNString) THEN
     NextPos ← NextPos - 1  // special case when last word
  ENDIF

  RETURN MID(FNString, Index, NextPos - Index)

ENDFUNCTION
```

1 mark for each of the following:

1 Conditional loop
2 ...extract char from `FNString` and compare with `SPACECHAR` **in a loop**
3 .. increment count if word continues
4 Exit loop when `SPACECHAR` encountered **or** when end of `FNString` reached
5 Apply substring function and Return

Question No. 2

Question	Answer	Marks

8(a)	`FUNCTION IgnoreWord (ThisWord : STRING) RETURNS BOOLEAN`
	
	`   DECLARE Found : BOOLEAN`
	`   DECLARE Index : INTEGER`
	
	`   Found ← False`
	`   Index ← 1`
	`   ThisWord ← TO_LOWER(ThisWord)`
	
	`   REPEAT`
	`      IF TO_LOWER(IgnoreList[Index]) = ThisWord THEN`
	`         Found ← TRUE`
	`      ENDIF`
	`      Index ← Index + 1`
	`   UNTIL Found = TRUE OR Index > 10`
	
	`   RETURN Found`
	
	`ENDFUNCTION`
	
	1 mark for each of the following:
	
	1 Loop through array elements
	2 Convert both strings to same case
	3 Compare array element with parameter **in a loop**
	4 Set a flag (or similar) if match found (after reasonable attempt at MP3) **in a loop**
	5 Return `TRUE` or `FALSE` in all cases
	
	Note: Max 4 if function declaration incorrect
		5

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>8(b)</td><td>
<pre>
Procedure GetInitials()

  DECLARE NewString, NextWord : STRING
  DECLARE ThisWordNum, Index : INTEGER

  ThisWordNum ← 0
  NewString ← ""

  REPEAT
    ThisWordNum ← ThisWordNum + 1
    Index ← GetStart(ThisWordNum)
    IF Index <> -1 THEN //if there is ThisWordNum
      NextWord ← GetWord(Index)
      IF IgnoreWord(NextWord) = FALSE THEN
       NewString ← NewString & UCASE(LEFT(NextWord, 1))
      ENDIF
    ENDIF

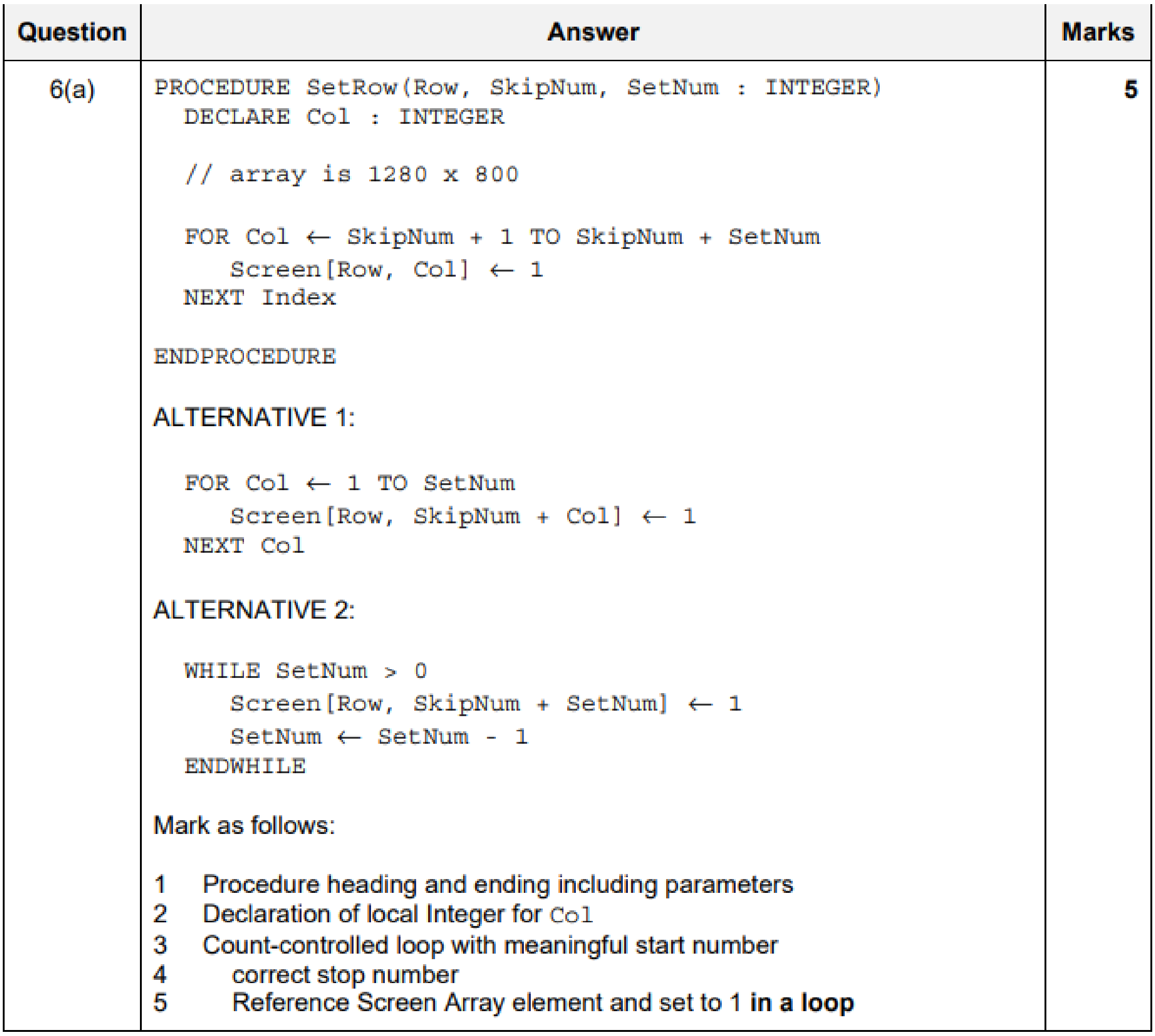
  UNTIL Index = -1

  OUTPUT NewString

ENDPROCEDURE
</pre>
1 mark for each of the following:
1 Declare NewString and initialise to empty string
2 Conditional loop to pick out **all** words from FNString
3 Evaluate result of GetStart() **in a loop**
4 Test result &lt;&gt; -1 and if not:
5 Assign result of GetWord() to a variable **in a loop**
6 Test result of IgnoreWord() **in a loop**
7 If not ignored, add the next initial letter to NewString **in a loop**
8 Increment ThisWordNum (must have been initialised) **in a loop**
9 Output NewString (must be all upper case) **outside loop**

Note: Max 8 marks
</td><td>8</td></tr>
</table>

Question No. 3



<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>6(a)</td><td>

```
PROCEDURE SetRow(Row, SkipNum, SetNum : INTEGER)
  DECLARE Col : INTEGER

  // array is 1280 x 800

  FOR Col ← SkipNum + 1 TO SkipNum + SetNum
     Screen[Row, Col] ← 1
  NEXT Index

ENDPROCEDURE
```

ALTERNATIVE 1:

```
  FOR Col ← 1 TO SetNum
     Screen[Row, SkipNum + Col] ← 1
  NEXT Col
```

ALTERNATIVE 2:

```
  WHILE SetNum > 0
     Screen[Row, SkipNum + SetNum] ← 1
     SetNum ← SetNum - 1
  ENDWHILE
```

Mark as follows:

1 Procedure heading and ending including parameters
2 Declaration of local Integer for `Col`
3 Count-controlled loop with meaningful start number
4 correct stop number
5 Reference Screen Array element and set to 1 **in a loop**

</td><td>5</td></tr>
</table>

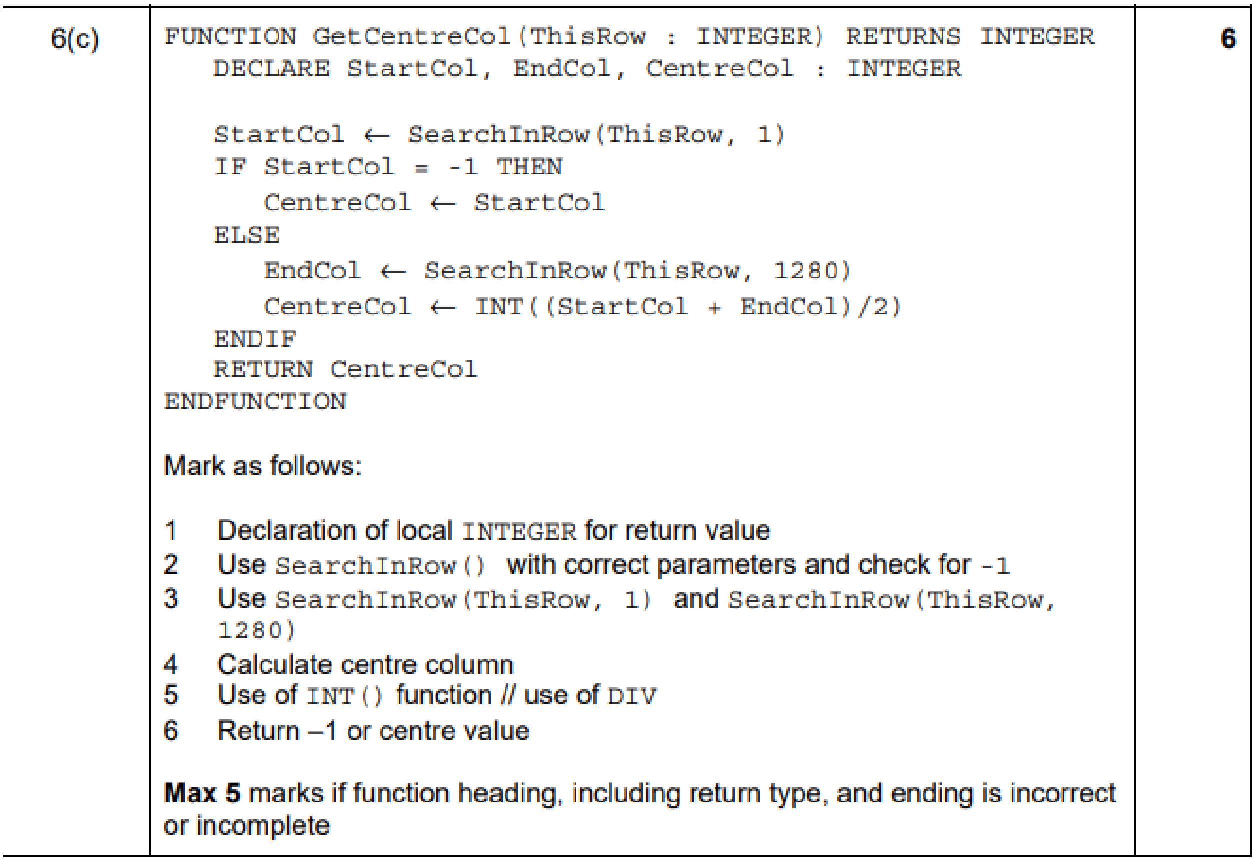
<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>6(b)</td><td>

```
FUNCTION SearchInRow(ThisRow, StartCol : INTEGER) RETURNS
         INTEGER
   DECLARE ThisCol, Step : INTEGER
   DECLARE Found: BOOLEAN

   // array is 1280 x 800

   Found ← FALSE
   ThisCol ← StartCol

   // first decide which way to search
   IF StartCol = 1 THEN
      Step ← 1
      EndCol ← 1281
   ELSE
       Step ← -1
       EndCol ← 0
   ENDIF


   WHILE ThisCol <> EndCol AND Found = FALSE
      IF Screen[ThisRow, ThisCol] <> 1 THEN
         ThisCol ← ThisCol + Step
      ELSE
         Found ← TRUE
      ENDIF
   ENDWHILE

   IF Found = FALSE THEN
      ThisCol ← -1
   ENDIF

   RETURN ThisCol

ENDFUNCTION
```

Mark as follows:

1 Interpreting `StartCol` parameter to determine direction of search
2 An attempt at searching both up and down
3 Conditional Loop / Count-controlled loop with use of `ThisCol` index
4 Using correct values for `StartCol`, `EndCol` and `Step`
5 Reference a `Screen` element and compare with 1 **in a loop**
6 If equal save column or immediately Return column **in a loop**
7 Return column number or −1
Loop(s) terminate when element with value = 1 found

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



6(c)

```
FUNCTION GetCentreCol(ThisRow : INTEGER) RETURNS INTEGER
   DECLARE StartCol, EndCol, CentreCol : INTEGER

   StartCol ← SearchInRow(ThisRow, 1)
   IF StartCol = -1 THEN
      CentreCol ← StartCol
   ELSE
      EndCol ← SearchInRow(ThisRow, 1280)
      CentreCol ← INT((StartCol + EndCol)/2)
   ENDIF
   RETURN CentreCol
ENDFUNCTION
```

Mark as follows:

1 Declaration of local `INTEGER` for return value
2 Use `SearchInRow()` with correct parameters and check for `-1`
3 Use `SearchInRow(ThisRow, 1)` and `SearchInRow(ThisRow, 1280)`
4 Calculate centre column
5 Use of `INT()` function // use of `DIV`
6 Return –1 or centre value

Max 5 marks if function heading, including return type, and ending is incorrect or incomplete

6

Question No. 4

Question	Answer	Marks

6(a)	`FUNCTION FirstRowSet() RETURNS INTEGER`
	`  DECLARE Row, Col : INTEGER`
	`  DECLARE Found : BOOLEAN`
	
	`  // array is 1280 x 800`
	`  Row ← 1`
	
	`  Found ← FALSE`
	`  WHILE Row <= 800 AND Found = FALSE // top to bottom`
	`     Col ← 1`
	`     WHILE Col <= 1280 AND Found = FALSE // left to right`
	`        IF Screen[Row,Col] = 1 THEN`
	`           Found ← TRUE // end function as soon as first`
	`                        // found`
	`        ENDIF`
	`        Col ← Col + 1`
	`     ENDWHILE`
	`     Row ← Row + 1`
	`  ENDWHILE`
	
	`  IF Found = FALSE THEN  // nothing found`
	`     Row ← 0`
	`  ENDIF`
	`  RETURN Row - 1`
	`ENDFUNCTION`
	
	Mark as follows:
	
	1 Function heading and ending **and** return type
	2 (Conditional) outer loop 1 to 800 (row)
	3 (Conditional) inner loop 1 to 1280 // 1280 to 1 (column)
	4 Reference `Screen` element and test for = 1 // <> 0
	5 and if true save row number **and** exit loops
	6 Increment index variables in both inner **and** outer loop
	7 Return Row number or −1, following a reasonable attempt
		7

6(b)	One mark for: • (A flag is used to) exit the loops // iteration is terminated • as soon as a Screen element with value 1 is found	**2**

Question	Answer	Marks
6(c)(i)	One mark for: • Parameter(s) need to be passed to the module to identify the type of search • Search algorithm is controlled by (global) variables / parameters Alternative: • The search algorithms from the original modules are included in the new module • The new module needs to return / store the four values (the results of the four searches)	2
6(c)(ii)	One mark for advantage and one for disadvantage: Advantage: (max 1) • Only have to change one module if specification changes • Less repetitive code / fewer lines of code • Aids re-usability Disadvantage: (max 1) • Single module more complex / more error prone / more difficult to debug ... • Single module cannot be split among programmers / teams **Max 2**	2

6(d)	`PROCEDURE GetCentre ()`
	`  DECLARE StartRow, EndRow, StartCol, EndCol : INTEGER`
	
	`  StartRow ← FirstRowSet()`
	`  IF StartRow = -1 THEN`
	`     CentreRow ← -1 // no 'touch' detected`
	`  ELSE`
	`     EndRow ← LastRowSet()`
	`     StartCol ← FirstColSet()`
	`     EndCol ← LastColSet()`
	`     CentreRow ← INT((StartRow + EndRow)/2)`
	`     CentreCol ← INT((StartCol + EndCol)/2)`
	`  ENDIF`
	`ENDPROCEDURE`
	
	Mark as follows:
	
	1 Call <any `Set` function> and check for -1 // check for no element set
	2 ...and if so set `CentreRow` to –1
	3 Call all 4 `Set` functions to get 'extremity' values
	4 Calculate centre row and centre column
	5 Use of `INT()` function or `DIV` operator on values from MP4
	6 Assign calculated values to `CentreRow` and `CentreCol`
	
	Note:
	Max 5 if procedure heading and ending missing or incorrect (ignore array if passed as a parameter) **or** any local variables are undefined or of incorrect type
		6

Question No. 5

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>8(a)</td><td><pre>
FUNCTION RandomChar() RETURNS CHAR
  DECLARE ThisRange : INTEGER
  DECLARE ThisChar : CHAR

  //First select the range
  ThisRange ← INT(RAND(3)) + 1 // 1 to 3

  CASE OF ThisRange
    1: ThisChar ← CHR(INT(RAND(26) + 65)) // 65 to 90:
                                              'A' to 'Z'
       ThisChar ← LCASE(ThisChar)   // 'a' to 'z'
    2: ThisChar ← CHR(INT(RAND(26) + 65)) // 65 to 90:
                                              A to Z
    3: ThisChar ← NUM_TO_STR(INT(RAND(10)) // '0' to '9'
  ENDCASE

  RETURN ThisChar
ENDFUNCTION
</pre>
Mark as follows:

1 Generation of <b>any</b> integer random number
2 Randomly decide which of the three ranges to select
3 Selection structure based on range
4 One alphanumeric character range correct
5 All alphanumeric character ranges correct
6 Return ThisChar, following a reasonable attempt to generate a character in each range</td><td>6</td></tr>
</table>

8(b)

```
FUNCTION FindPassword(Name: STRING) RETURNS STRING
  DECLARE Index : INTEGER
  DECLARE Password : STRING

  Password ← ""
  Index ← 1

  WHILE Password = "" AND Index <= 500
     IF Secret[Index, 1] = Name THEN
       Password ← Decrypt(Secret[Index, 2])
     ELSE
       Index ← Index + 1
     ENDIF
  ENDWHILE

  IF Password = "" THEN
     OUTPUT "Domain name not found"
  ENDIF

  RETURN Password

ENDFUNCTION
```

Mark as follows:

1 Declare all local variables used, attempted solution has to be reasonable
2 Conditional loop while not found and not end of array
3 Compare value of element in column 1 with parameter passed into function
4 ...and use Decrypt() with element in column 2 as parameter
5 ...use the return value of `Decrypt()`
6 Output warning message if parameter not found
7 Return `STRING` value

7

8(c)

One mark for the name, one for the description
Name:

- Stub testing

Description:

- A simple module is written to replace each of the modules.
- The simple module will return an expected value // will output a message to show they have been called

3

8(d)

Accept **one** example of a valid password to **Max 2**

One mark for each password example that breaks **one** of the rules due to:

- Length too long // length too short
- Invalid character
- Incorrect grouping (including number of hyphens)
- Duplicated characters

2

8(e)

One mark for each part:

- Generate a random integer divisible by 3
- Split range into 1/3 and set as numeric
- Else alphabetic character

3

Question No. 6

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>8(a)</td><td>

```
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

</td><td>5</td></tr>
</table>

8(b)

```
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**

8

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**

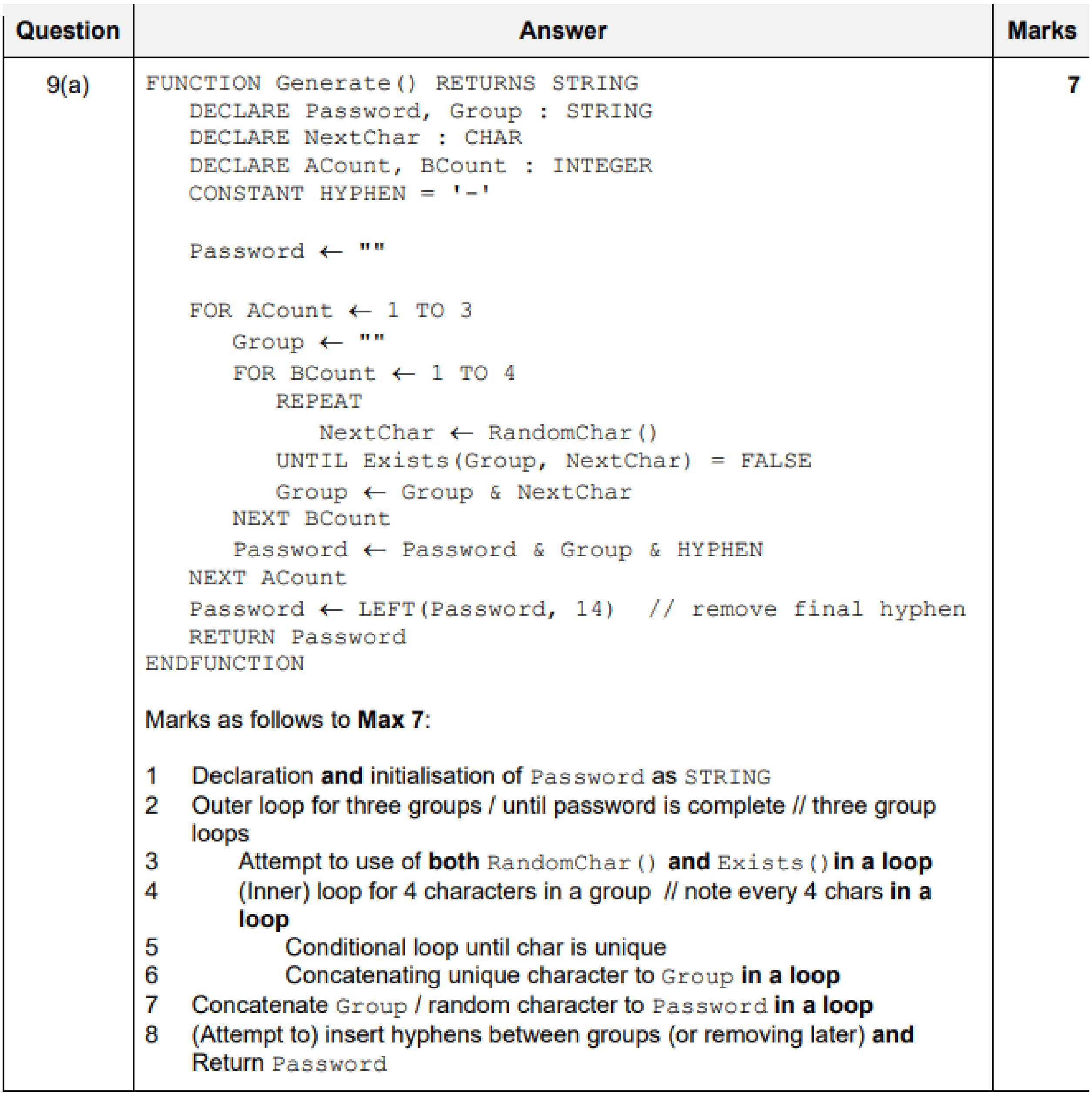
4 Reject character if `Exists()` returns `TRUE`, otherwise form string **in a loop**

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

Question No. 7



Question | Answer | Marks

9(a)

```
FUNCTION Generate() RETURNS STRING
   DECLARE Password, Group : STRING
   DECLARE NextChar : CHAR
   DECLARE ACount, BCount : INTEGER
   CONSTANT HYPHEN = '-'

   Password ← ""

   FOR ACount ← 1 TO 3
      Group ← ""
      FOR BCount ← 1 TO 4
         REPEAT
            NextChar ← RandomChar()
         UNTIL Exists(Group, NextChar) = FALSE
         Group ← Group & NextChar
      NEXT BCount
      Password ← Password & Group & HYPHEN
   NEXT ACount
   Password ← LEFT(Password, 14)   // remove final hyphen
   RETURN Password
ENDFUNCTION
```

Marks as follows to **Max 7**:

1 Declaration **and** initialisation of `Password` as `STRING`

2 Outer loop for three groups / until password is complete // three group loops

3 Attempt to use of **both** `RandomChar()` **and** `Exists()` **in a loop**

4 (Inner) loop for 4 characters in a group // note every 4 chars **in a loop**

5 Conditional loop until char is unique

6 Concatenating unique character to `Group` **in a loop**

7 Concatenate `Group` / random character to `Password` **in a loop**

8 (Attempt to) insert hyphens between groups (or removing later) **and** Return `Password`

7

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>9(b)</td><td>

```
FUNCTION AddPassword(Name, Password : STRING)
         RETURNS BOOLEAN
   DECLARE Index : INTEGER
   DECLARE Added : BOOLEAN

   Added ← FALSE
   Index ← 1

   IF FindPassword(Name) = "" THEN // Domain name not in
                                   // array
      WHILE Added = FALSE AND Index <= 500
         IF Secret[Index, 1] = "" THEN
            Secret[Index, 1] ← Name
            Secret[Index, 2] ← Encrypt(Password)
            Added ← TRUE
         ELSE
            Index ← Index + 1
         ENDIF
      ENDWHILE
   ENDIF

   RETURN Added

ENDFUNCTION
```

Marks as follows:

1 Check that the website domain name isn't already in array using `FindPassword()` / linear search, otherwise:
2 (Conditional) loop while not added and not end of array
3 Check for unused element by testing value in column 1 **in a loop**
4 If unused, write parameter values to column 1 and 2 **and** set flag / variable
5 ...having used `Encrypt()` on the password
6 Return `BOOLEAN` value (correctly in all cases)

</td><td>6</td></tr>
<tr><td>9(c)</td><td>

One mark per point to **Max 3**.

Solution based on field length:

- Convert the length of the website domain name (either field) …
- … to a string of fixed length
- Form a string by concatenate this string with the other two (and write as one line of the file)

Solution based on use of separator character:

- Select a (separator) character that cannot occur in the domain name (e.g. space)
- Create a string from the domain name followed by the separator
- ...Concatenate the encrypted password (and write as one line of the file)

</td><td>3</td></tr>
</table>

Question No. 8

Question	Answer	Marks

8(a)	One mark for each point (**Max 7**) as follows:
	1 Function heading and ending **including** parameter and return type
	2 Declaration **and** initialisation of local Integer for `Count`
	3 OPEN in READ mode and CLOSE
	4 Conditional loop until `EOF()`
	5 Read a line **in a loop**
	6 If non-blank, increment count **in a loop**
	7 Terminate loop when 10 non-blank lines have been read
	8 Return Boolean in both cases
	
	`FUNCTION CheckFile(Thisfile : STRING) RETURNS BOOLEAN`
	   `DECLARE Valid : BOOLEAN`
	   `DECLARE ThisLine : STRING`
	   `DECLARE Count : INTEGER`
	
	   `Count ← 0`
	   `Valid ← FALSE`
	   `OPEN ThisFile FOR READ`
	
	   `WHILE NOT EOF(ThisFile) AND Valid = FALSE`
	      `READFILE ThisFile, ThisLine`
	      `IF ThisLine <> "" THEN`
	         `Count ← Count + 1`
	         `IF Count > 9 THEN`
	            `Valid ← TRUE`
	         `ENDIF`
	      `ENDIF`
	   `ENDWHILE`
	
	   `CLOSEFILE ThisFile`
	   `RETURN Valid`
	
	`ENDFUNCTION`
		7

8(b)	`CALL CountErrors("Jim01Prog.txt", 20)` One mark for each: 1 Module name, at least one parameter in brackets and one parameter correct 2 Completely correct statement	2

8(c)

Mark as follows:

1 Procedure heading and ending **including** parameters
2 Declaration and initialisation of local Integer value for `ErrCount`
3 Use of `CheckFile()`, output message **and** terminate if it returns `FALSE`
4 Conditional loop until `EOF()`
5 ...or `ErrCount` > `MaxErrors`
6 Read line and use as parameter to `CheckLine()` **in a loop**
7 Test return value and increment `ErrCount` if non-zero **in a loop**
8 Output either message once only as appropriate

```
PROCEDURE CountErrors(ThisFile : STRING, MaxErrors :
INTEGER)
   DECLARE ErrCount, ThisError : INTEGER
   DECLARE ThisLine : STRING

   ErrCount ← 0

   IF CheckFile(ThisFile) = FALSE THEN
      OUTPUT "That program file is not valid"
   ELSE
      OPEN ThisFile FOR READ

      REPEAT
         READFILE, ThisFile, ThisLine
         ThisError ← CheckLine(ThisLine)
         IF ThisError <> 0 THEN
            ErrCount ← ErrCount + 1
         ENDIF
      UNTIL ErrCount > MaxErrors OR EOF(ThisFile)

      IF EOF(ThisFile) = FALSE THEN
         OUTPUT "Check terminated – too many errors"
      ELSE
         OUTPUT "There were ", ErrCount, " errors."
      ENDIF

   CLOSEFILE ThisFile
   ENDIF

ENDPROCEDURE
```

8

8(d)

One mark for each (**Max 2**):

Examples:

1 Incorrect block structure. Missing keyword denoting part of block (for example `ENDPROCEDURE`, `ENDFUNCTION`, `ENDTYPE`)
2 Data type errors, for example, assigning an integer value to a string
3 Identifier used before it is declared
4 Incorrect parameter use

2

Question No. 9

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

One mark per point (**Max 6**):

1 Procedure heading and ending **including** parameters
2 Conditional loop containing incrementing `Index`...
3 ...terminating when `ErrNum` found
4 ...terminating when `ErrCode[Index] > ErrNum` (i.e. `ErrNum` not found)
5 ... **OR** after element 500 tested
6 Test if found and OUTPUT 'Found' message
7 ...otherwise OUTPUT 'Not Found' message

```
PROCEDURE OutputError(LineNum, ErrNum : INTEGER)
  DECLARE Index : INTEGER

  Index ← 0

  // Search until ErrNum found OR not present OR end of
array

  REPEAT
     Index ← Index + 1
  UNTIL ErrCode[Index] >= ErrNum OR Index = 500

  IF ErrCode[Index] = ErrNum THEN
     OUTPUT ErrText[Index], " on line ", LineNum
//Found
  ELSE
     OUTPUT "Unknown error on line ", LineNum     //Not
found
  ENDIF

ENDPROCEURE
```

</td><td>6</td></tr>
</table>

Question	Answer	Marks

7(b)	One mark per point (**Max 8**):
	
	1 Procedure heading and ending as shown
	2 Conditional loop correctly terminated
	3 An inner loop
	4 Correct range for inner loop
	5 Comparison (element J with J+1) **in a loop**
	6 Swap elements in **both** arrays **in a loop**
	7 'No-Swap' mechanism:
	• Conditional **outer** loop including flag reset
	• Flag set in **inner** loop to indicate swap
	8 Efficiency (this scenario): terminate inner loop when `ErrCode = 999`
	9 Reducing `Boundary` **in the <u>outer</u> loop**
	
	`PROCEDURE SortArrays()`
	`  DECLARE TempInt, J, Boundary : INTEGER`
	`  DECLARE TempStr : STRING`
	`  DECLARE NoSwaps : BOOLEAN`
	
	`  Boundary ← 499`
	
	`  REPEAT`
	`     NoSwaps ← TRUE`
	`     FOR J ← 1 TO Boundary`
	`        IF ErrCode[J]> ErrCode[J+1] THEN`
	`           //first swap ErrCode elements`
	`           TempInt ← ErrCode[J]`
	`           ErrCode[J] ← ErrCode[J+1]`
	`           ErrCode[J+1] ← TempInt`
	`           //now swap corresponding ErrText elements`
	`           TempStr ← ErrText[J]`
	`           ErrText[J] ← ErrText[J+1]`
	`           ErrText[J+1] ← TempStr`
	`           NoSwaps ← FALSE`
	`        ENDIF`
	`     NEXT J`
	`     Boundary ← Boundary - 1`
	`  UNTIL NoSwaps = TRUE`
	
	`ENDPROCEDURE`
		8

7(c)(i)	`ErrCode` should be an `INTEGER` // `ErrCode` should not be a `STRING`	**1**
7(c)(ii)	Benefits include: 1 Array of records can store mixed data types / multiple data types under a single identifer 2 Tighter / closer association between `ErrCode` and `ErrText` // simpler code as fields may be referenced together // values cannot get out of step as with two arrays 3 Program easier to design / write / debug / test / maintain / understand One mark per point **Note: Max 2 marks**	**2**
7(c)(iii)	`DECLARE Error : ARRAY[1:500] OF ErrorRec`	**1**

Question No. 10

Question	Answer	Marks
7(a)	One mark per point (**Max 8**):	**8**

1 Declaration and initialisation of local integer for `Count`
2 Appropriate prompt and two inputs
3 (Conditional) loop while error number input is in range // error code 999 reached
4 ...and not end of array
5 Check if this `ErrCode` needs to be output **in a loop**
6 if so check for blank error text **in a loop**
7 Output in both cases
8 ....and increment count **in a loop**
9 OUTPUT of header and summary including count

```
PROCEDURE OutputRange()
  DECLARE First, Last, Count, Index, ThisErr : INTEGER
  DECLARE ThisMess : STRING
  DECLARE PastLast: BOOLEAN

  Count ← 0
  Index ← 1
  PastLast ← FALSE

  OUTPUT "Please input first error number: "
  INPUT First
  OUTPUT "Please input last error number: "
  INPUT Last

  OUTPUT "List of error numbers from ", First, " to ",
Last

  WHILE Index < 501 AND NOT PastLast
     ThisErr ← ErrCode[Index]
     IF ThisErr > Last THEN
        PastLast ← TRUE
     ELSE
        IF ThisErr >= First THEN
           ThisMess ← ErrText[Index]
           IF ThisMess = "" THEN
              ThisMess ← "Error Text Missing"
           ENDIF
           OUTPUT ThisErr, " : ", ThisMess
           Count ← Count + 1
        ENDIF
     ENDIF
     Index ← Index + 1
   ENDWHILE

  OUTPUT Count, " error numbers output"

ENDPROCEDURE
```

7(b)(i)

One mark per point:

1 (Conditional) loop terminating when item added **OR** end of array reached
2 Test for unused element **in a loop**
3 Assignment of values to arrays // save index of first blank location and assign after loop
4 Set loop termination if empty element found in a loop
5 Call `SortArrays()` **once**
6 Calculation of remaining unused elements **and** return Integer value (for both cases)

```
FUNCTION AddError(ErrNum : INTEGER, ErrMess : STRING)
RETURNS INTEGER
  DECLARE Index, Remaining : INTEGER
  CONSTANT Unused = 999

  Index ← 1
  Remaining ← -1

  REPEAT
     IF ErrCode[Index] = Unused THEN
        ErrCode[Index] ← ErrNum
        ErrText[Index] ← ErrMess
        CALL SortArrays()
        Remaining ← 500 - Index
     ENDIF
     Index ← Index + 1
  UNTIL Remaining <> -1 OR Index > 500

  RETURN Remaining

ENDFUNCTION
```

6

7(b)(ii)

One mark per point (**Max 3**):

1. Loop through 500 elements (while error number not found)
2. Compare `ErrCode` for current element with the error number
3. If same, set element value to 999 (and terminate loop)
4. ... and call `SortArrays()` (to move 999 to the end) – once only

3