

CANDIDATE
NAME

--

CENTRE
NUMBER

--	--	--	--	--

CANDIDATE
NUMBER

--	--	--	--



COMPUTING

9691/22

Paper 2

May/June 2016

2 hours

Candidates answer on the Question Paper.

No Additional Materials are required.

READ THESE INSTRUCTIONS FIRST

Write your Centre number, candidate number and name on all the work you hand in.

Write in dark blue or black pen.

You may use an HB pencil for any diagrams, graphs or rough working.

Do not use staples, paper clips, glue or correction fluid.

DO NOT WRITE IN ANY BARCODES.

Answer **all** questions.

At the end of the examination, fasten all your work securely together.

The number of marks is given in brackets [] at the end of each question or part question.

This document consists of **15** printed pages and **1** blank page.

- 1 A high-level programming language has built-in string handling functions specified as follows:

ONECHAR(ThisString : STRING, x : INTEGER) RETURNS STRING

returns a string value consisting of one character from the string ThisString at position x.

For example: ONECHAR("STOP", 4) returns "P"

CONCAT(String1 : STRING, String2 : STRING) RETURNS STRING

returns a string value consisting of String1 followed by String2.

For example: CONCAT("HE", "LLO") returns "HELLO"

Use the pseudocode on the following page to complete the trace table for the function call

Process("011101", "001100")

String1	String2	Position	Digit1	Digit2	Sum	Carry	Result
"011101"	"001100"					"0"	" "

[6]

```

FUNCTION Process(String1, String2 : STRING) RETURNS STRING
  Carry ← "0"
  Result ← ""
  FOR Position ← 6 DOWN TO 1
    Digit1 ← ONECHAR(String1, Position)
    Digit2 ← ONECHAR(String2, Position)
    IF Carry = "0"
      THEN
        CASE OF Digit1
          "0": Sum ← Digit2
          "1": CASE OF Digit2
            "0": Sum ← "1"
            "1": Sum ← "0"
            Carry ← "1"
          ENDCASE
        ENDCASE
      ELSE
        CASE OF Digit1
          "0": CASE OF Digit2
            "0": Sum ← "1"
            Carry ← "0"
            "1": Sum ← "0"
          ENDCASE
          "1": CASE OF Digit2
            "0": Sum ← "0"
            "1": Sum ← "1"
          ENDCASE
        ENDCASE
      ENDIF
    Result ← CONCAT(Sum, Result)
  ENDFOR
  RETURN Result
ENDFUNCTION

```

2 Ryan wrote the following recursive function using pseudocode.

```

01 FUNCTION Power (Number : INTEGER, Exponent : INTEGER) RETURNS INTEGER
02     IF Exponent = 0
03         THEN
04             Result ← 1
05         ELSE
06             Result ← Number * Power (Number, Exponent - 1)
07     ENDIF
08     RETURN Result
09 ENDFUNCTION
    
```

(a) (i) How can you recognise that this function is recursive?

.....
 [1]

(ii) A recursive solution to a problem always has a **base case** (the stopping condition) and a **general case**.

Give the line number of the statement that is executed in the base case.

.....

Give the line number of the statement that is executed in the general case.

..... [2]

(b) (i) State the value returned by `Power(3, 0)`.

..... [1]

(ii) State the value returned by `Power(3, 1)`.

..... [1]

(c) (i) Explain what happens when the call is `Power(3, -1)`.

.....

 [2]

(ii) Describe the changes that will need to be made to function `Power` to address the problem that you described in **part (c)(i)**.

.....
.....
.....
..... [2]

(d) An alternative method for the `Power` function is to use an iterative solution.

Using **pseudocode**, write `Power` as an iterative function.

FUNCTION `Power`(`Number` : INTEGER, `Exponent` : INTEGER) RETURNS INTEGER

.....
.....
.....
.....
.....
.....
.....
.....
..... [4]

(e) A problem may have both an iterative solution and a recursive solution. Each has its own benefits.

State **one** benefit of each solution.

Iterative

.....

Recursive

.....

..... [2]

(f) Ryan uses his recursive function `Power` in a program. He will use debugging tools to check the `Power` function works as expected. The debugging tools include breakpoints, variable watches and stepping.

(i) Identify where it is appropriate to set a breakpoint. Justify your answer.

.....
.....
.....
..... [2]

(ii) Ryan wants to set up a variable watch window for his function.

```
01 FUNCTION Power (Number : INTEGER, Exponent : INTEGER) RETURNS  
                                INTEGER  
02     IF Exponent = 0  
03         THEN  
04             Result ← 1  
05         ELSE  
06             Result ← Number * Power (Number, Exponent - 1)  
07     ENDIF  
08     RETURN Result  
09 ENDFUNCTION
```

State the variable(s) that would be appropriate to list in the variable watch window. Justify your choice.

.....
.....
.....
..... [2]

(iii) Explain how Ryan uses these debugging tools to check the execution of his recursive function.

.....
.....
.....
.....
..... [3]

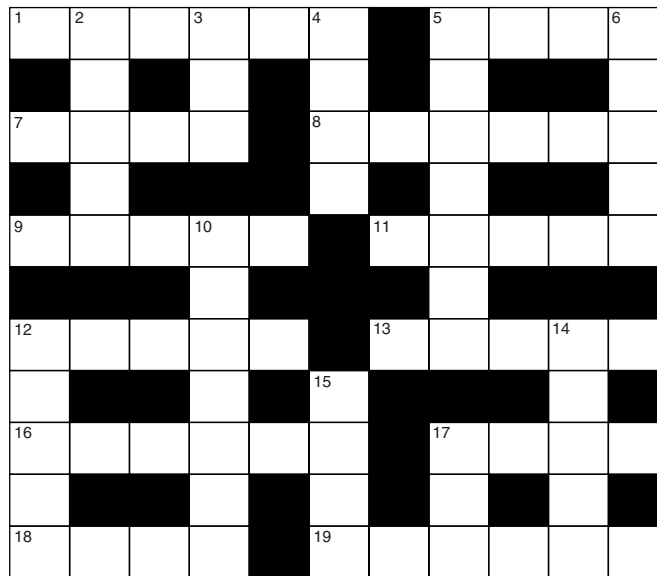
Question 3 begins on page 8.

- 3 A crossword puzzle is based on a grid made up of white and black squares. The white squares are to be filled with letters to form words across the grid and down the grid.

The crossword puzzle designer provides clues for each word. Words consist of at least 2 letters.

The square where a word starts, across or down, is numbered. The numbering starts at the top left of the grid and is consecutive, row by row.

Here is an example of a crossword grid:



- (a) Zara is given a grid design of white and black squares. This design does not yet contain numbers. Zara wants to write a program to set up a data structure for the grid data.

The data for each square will represent either:

- a white square
- a black square
- a white square with a number

- (i) A white square with a number is represented by the integer value of the number.

Suggest a suitable value to represent:

A white square

A black square [2]

- (ii) Write **program code** to declare the data structure with identifier `Puzzle` for a grid design of size 11×11 squares.

Programming language

.....

..... [3]

(iii) Initially, all squares are to be white squares.

Write **program code** to initialise `Puzzle`.

Programming language

Code

.....

.....

.....

.....

.....

..... [3]

(iv) The seventh square in the first row of the example crossword grid is black.

Write **program code** for the assignment statement to set the corresponding element of `Puzzle` to represent black.

Programming language

..... [2]

Where there are other black squares in the design, Zara writes additional assignment statements.

(b) The next stage for Zara is to design a procedure that numbers the relevant white squares.

A word begins in a square if the square is:

- a white square, and
- the first of a sequence of at least 2 white squares
 - across or
 - down

Zara starts to write the pseudocode for the procedure `CheckForStartOfWord`. This will check whether a square at `ThisRow` and `ThisColumn` is the start of a word.

(i) Complete the pseudocode for the global declarations.

CONSTANT WHITE =..... // value from **part (a) (i)**

CONSTANT BLACK =..... // value from **part (a) (i)**
[2]

(ii) Complete the pseudocode:

```

PROCEDURE CheckForStartOfWord(Puzzle, ThisRow, ThisColumn, Across, Down)
  Across ← FALSE // will change to TRUE
                // if a word across starts in this square
  Down ← .....
  IF Puzzle[ThisRow, ThisColumn] = .....
    THEN // this square is white
      // check for sequence across
      IF ThisColumn < 11 // check not in last column
        THEN
          // check this is the first column or a black square to the
          // left
          IF (ThisColumn = 1
            OR Puzzle[ThisRow, ThisColumn - 1] = BLACK)
            // check that the square to the right is white
            AND (Puzzle[.....] = WHITE)
            THEN
              Across ← TRUE
          .....
        ENDIF
      // check for sequence down
      IF ThisRow < 11 // check not in last row
        THEN
          // check this is the first row or a black square above
          IF (ThisRow = .....
            OR Puzzle[.....] = BLACK)
            // check that the square below is white
            AND (.....)
            THEN
              Down ← .....
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDPROCEDURE

```

[8]

(iii) The procedure

CheckForStartOfWord(Puzzle, ThisRow, ThisColumn, Across, Down)
 has five parameters. Parameters can be passed **by reference** or **by value**.

For each parameter, tick (✓) to show how it should be passed.

Parameter	By reference	By value
Puzzle		
ThisRow		
ThisColumn		
Across		
Down		

[3]

(c) Zara wants her program to number the squares where a word starts. She also wants the program to create two lists, `AcrossList` and `DownList`.

`AcrossList` will contain the numbers of the starting squares of words going across.

`DownList` will contain the numbers of the starting squares of words going down.

Note that a square can be a starting square for a word going across **and** a word going down. The same number would therefore be contained in both lists.

For example, the given puzzle has these lists. Squares numbered 5, 12 and 17 are starting squares for words going across and words going down.

1	2		3		4		5			6
7					8					
9			10			11				
12						13			14	
					15					
16							17			
18					19					

AcrossList
1
5
7
8
9
11
12
13
16
17
18
19

DownList
2
3
4
5
6
10
12
14
15
17

Zara designs this part of the program by writing pseudocode.

Convert this pseudocode into **program code**.

```

NextNumber ← 1 // first number for numbering starting squares
a ← 1 // pointer to first element of AcrossList
d ← 1 // pointer to first element of DownList
FOR ThisRow ← 1 TO 11
  FOR ThisColumn ← 1 TO 11
    CALL CheckForStartOfWord(Puzzle, ThisRow, ThisColumn, Across, Down)
    IF Across = TRUE
      THEN
        AcrossList[a] ← NextNumber // update AcrossList
        a ← a + 1
      ENDIF
    IF Down = TRUE
      THEN
        DownList[d] ← NextNumber // update DownList
        d ← d + 1
      ENDIF
    IF Across = True OR Down = TRUE
      THEN
        Puzzle[ThisRow, ThisColumn] ← NextNumber // number the square
        NextNumber ← NextNumber + 1
      ENDIF
    ENDFOR
  ENDFOR
ENDFOR

```


- (d) The pseudocode Zara wrote in **part (b)** and **part (c)** has some features that make it easier to read and understand.

State **four** such features.

1
.....

2
.....

3
.....

4
..... [4]

- (e) Zara wants to save the contents of the data structure `Puzzle` to a sequential file `Puzzle.TXT`

Write **program code** for the procedure `SavePuzzleToFile(Puzzle)`.

Programming language

Code

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
..... [5]

- (f) One of Zara’s friends suggests that the number of squares for a word should be counted.

Zara decides that she needs a function to count the number of squares for a word going across. She needs another function to count the number of squares for a word going down.

The program will call the function `CountSquaresAcross` when `ThisRow` and `ThisColumn` refer to a square that is the start of a word across.

Write **pseudocode** for the function `CountSquaresAcross`. Declare any local variables required.

FUNCTION `CountSquaresAcross(Puzzle, ThisRow, ThisColumn)` RETURNS
INTEGER

.....
.....
.....
.....
.....
.....
.....
.....
.....
..... [7]

BLANK PAGE

Permission to reproduce items where third-party owned material protected by copyright is included has been sought and cleared where possible. Every reasonable effort has been made by the publisher (UCLES) to trace copyright holders, but if any items requiring clearance have unwittingly been included, the publisher will be pleased to make amends at the earliest possible opportunity.

To avoid the issue of disclosure of answer-related information to candidates, all copyright acknowledgements are reproduced online in the Cambridge International Examinations Copyright Acknowledgements Booklet. This is produced for each series of examinations and is freely available to download at www.cie.org.uk after the live examination series.

Cambridge International Examinations is part of the Cambridge Assessment Group. Cambridge Assessment is the brand name of University of Cambridge Local Examinations Syndicate (UCLES), which is itself a department of the University of Cambridge.