GradeMax

* 0019655333801 *

Cambridge Assessment International Education

Cambridge International AS & A Level

CANDIDATE NAME

CENTRE NUMBER

CANDIDATE NUMBER

* 6 8 0 6 8 0 7 8 2 2 *

COMPUTER SCIENCE **9618/23**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 2024**

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 **24** pages. Any blank pages are indicated.

DC (KS/CGW) 329352/4
© UCLES 2024

GradeMax

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

1 A program uses many complex algorithms.

One algorithm is repeated in several places. The code for the algorithm is the same wherever it is used, but the calculations within the algorithm may operate on different data.

The result of each calculation is used by the code that follows it.

It is decided to modify the program and implement the algorithm as a separate module.

(a) (i) State **two** benefits of this modification to the existing program.

1 .....

.....

2 .....

.....

[2]

(ii) Describe how the modification would be implemented.

.....

.....

.....

.....

.....

.....

.....

..... [3]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

(b) Four of the expressions used in the program are represented by pseudocode in the table.

Complete each pseudocode expression with a function or operator so that it evaluates to the value shown.

Any functions and operators used must be defined in the **insert**.

Pseudocode expression	Evaluates to
..... ("Random", 2, 3)	"and"
5 + ... (10/11/2023)	15
..... ("45000")	TRUE
(20 ..... 3) + 1	3

[4]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

2 **(a)** A program uses a global 1D array of type string and a text file.

An algorithm that forms part of the program is expressed as follows:
- copy the first line from the file into the first element of the array
- copy the second line from the file into the second element of the array
- continue until all lines in the file have been copied into the array.

Stepwise refinement is applied to the algorithm.

Outline **five** steps for this algorithm that could be used to produce pseudocode.

Assume there are more elements in the array than lines in the file.

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

Step 1 .....

.....

Step 2 .....

.....

Step 3 .....

.....

Step 4 .....

.....

Step 5 .....

.....

[5]

(b) Sequence is one programming construct.

Identify **one other** programming construct that will be required when the algorithm from part **(a)** is converted into pseudocode **and** explain its use.

Construct .....

.....

Use .....

.....

[2]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

* 0019655333805 *



BLANK PAGE

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

3 A record structure is declared to hold data relating to components being produced in a factory:

```
TYPE Component
   DECLARE Item_ID : STRING
   DECLARE Reject : BOOLEAN
   DECLARE Weight : REAL
ENDTYPE
```

The factory normally produces a batch (or set) of 1000 components at a time. A global array is declared to store 1000 records for a batch:

```
DECLARE Batch : ARRAY [1:1000] OF Component
```

Two global variables contain the minimum and maximum acceptable weight for each component. The values represent an inclusive range and are declared as:

```
DECLARE Min, Max : REAL
```

(a) (i) A program uses a variable `ThisIndex` as the array index to access a record.

Write a pseudocode clause to check whether or **not** the weight of an individual component is within the acceptable range.

.....

.....

..... [3]

(ii) When batches of less than 1000 components are processed, it is necessary to indicate that certain elements in the array are unused.

Suggest how an unused array element could be indicated.

.....

..... [1]

(b) A module `InRange()` will:

- be called with an integer parameter representing an index value of a record in the `Batch` array
- check if the weight of the indexed component is within the acceptable range
- return `TRUE` if the weight is in the range and `FALSE` if it is **not**.

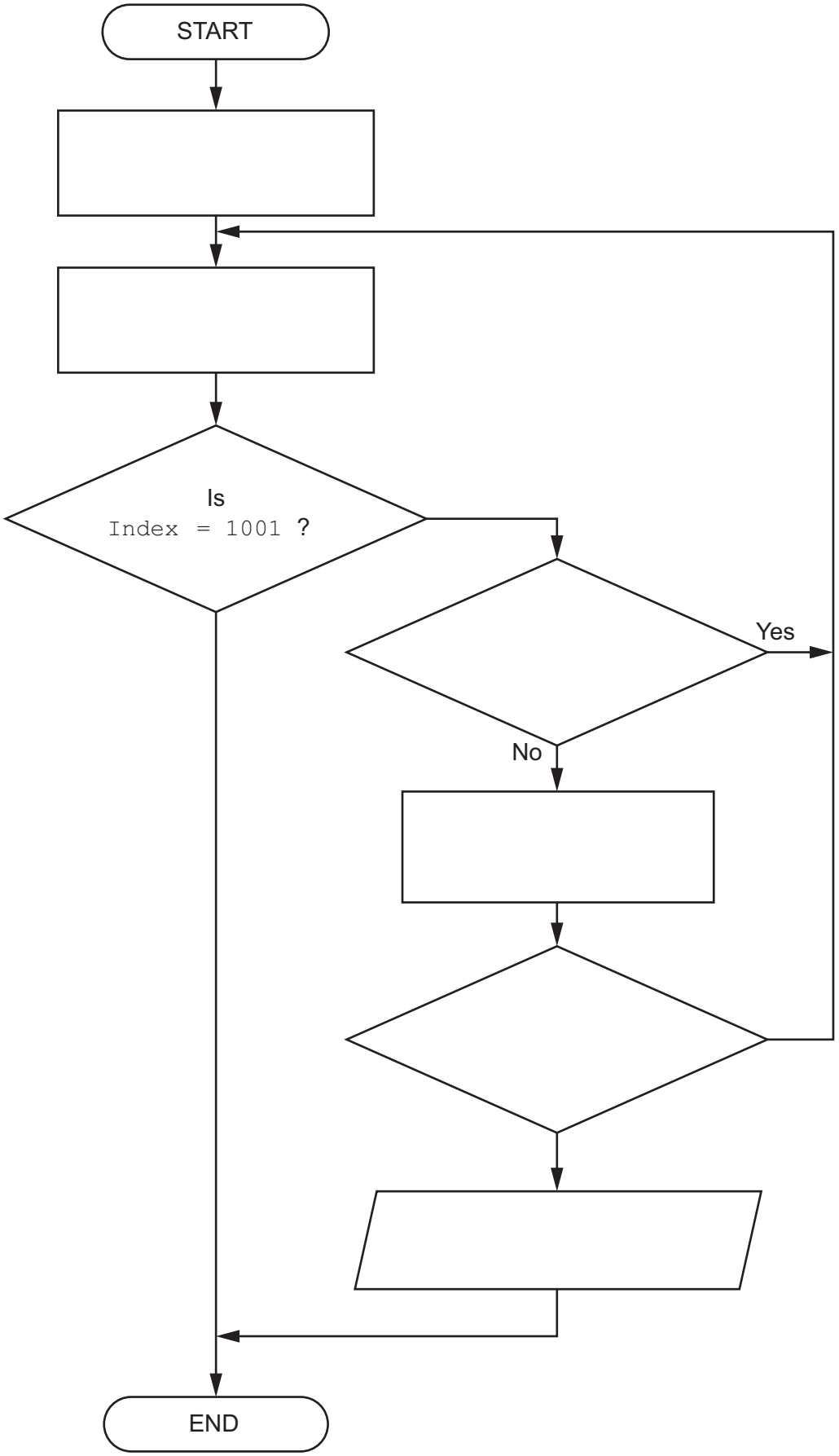
A module `BatchCheck()` will:

- iterate through a batch of 1000 component records
- call module `InRange()` to check each individual component record
- keep a count of the number of components that fail
- output a suitable warning message and immediately stop if the number of failed components exceeds 5.

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

* 0019655333807 *

Complete the program flowchart to represent the algorithm for module `BatchCheck()`.



[5]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

4 A procedure `TwoParts()` will input a sequence of real values, one at a time.

The procedure will:
- process the sequence in two parts
- form a first total by adding the values until the first zero
- form a second total by adding the values after the first zero until the second zero
- output the average of the two totals, together with a suitable message.

Values input in the first part are totalled using global variable `TotalA` and those input in the second part are totalled using global variable `TotalB`.

(a) Write pseudocode for the procedure `TwoParts()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

* 0019655333809 *

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

(b) The value zero denotes the split between the two parts of the sequence.

The requirement changes and now there may be up to 20 parts.

(i) Identify a suitable data structure that could be used to store the different total values.

..... [2]

(ii) Describe **three** benefits of using the data structure given in part **(b)(i)**.

1 .....

.....

2 .....

.....

3 .....

.....

[3]

© UCLES 2024

GradeMax

5 A program is being designed in pseudocode.

The program contains the following declaration:

```
DECLARE Data : ARRAY[1:1000] OF STRING
```

A procedure `ArrayInitialise()` is written to initialise the values in the array:

```
PROCEDURE ArrayInitialise(Label : STRING)
   DECLARE Index : INTEGER
   Index ← 1
   WHILE Index <= 1000
      CASE OF (Index MOD 2)
         0 : Data[Index] ← FormatA(Label)
             Index ← Index + 1
         1 : Data[Index] ← FormatB(Label)
             Index ← Index + 1
      ENDCASE
   ENDWHILE
ENDPROCEDURE
```

Functions `FormatA()` and `FormatB()` apply fixed format case changes to the parameter string.

(a) The design of the procedure does **not** use the most appropriate loop construct.

Suggest a **more appropriate** construct that could be used **and** explain your choice.

Construct .....

Explanation .....

.....

[2]

(b) The algorithm calls one of the functions `FormatA()` and `FormatB()` each time within the loop.

Explain why this is **not** efficient **and** suggest a more efficient solution.

.....

.....

.....

.....

.....

.....

.....

..... [4]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

* 0019655333811 *

BLANK PAGE

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

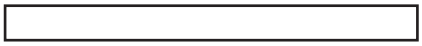
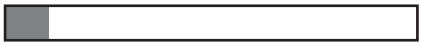
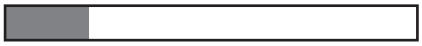
GradeMax

6 A program displays a progress bar to inform the user of the progress of tasks that take a significant time to complete, such as those involving file transfer operations.

Task progress is divided into 11 steps. Each step represents the amount of progress as a percentage. An image is associated with each step and each image is stored in a different file.

Different progress bar images may be selected. For a given image, files all have the same filename root, with a different suffix.

The table illustrates the process for using the image with filename root `BargraphA`

Step	Percentage progress	Image filename	Image
1	< 10	`BargraphA-1.bmp`	
2	>= 10 and < 20	`BargraphA-2.bmp`	
3	>= 20 and < 30	`BargraphA-3.bmp`	
⋮	⋮	⋮	⋮
9	>= 80 and < 90	`BargraphA-9.bmp`	
10	>= 90 and < 100	`BargraphA-10.bmp`	
11	100	`BargraphA-11.bmp`	

A procedure `Progress()` will:

- be called with two parameters:
  - an integer representing the percentage progress (0 to 100 inclusive)
  - a string representing the image filename root
- generate the full image filename
- call a procedure `Display()` using the full image filename as the parameter.

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

* 0019655333913 *

(a) Write pseudocode for procedure `Progress()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

(b) The definition of procedure `Progress()` is provided here for reference:

> A procedure `Progress()` will:
> - be called with two parameters:
> - an integer representing the percentage progress (0 to 100 inclusive)
> - a string representing the image filename root
> - generate the full image filename
> - call a procedure `Display()` using the full image filename as the parameter.

`Progress()` will be rewritten and a new module `Progress2()` produced with these requirements:

- an additional parameter of type integer will specify the total number of steps
- the image filename will be returned (procedure `Display()` will **not** be called from within `Progress2()`).

(i) Write pseudocode for the new module header.

.....

.....

..... [2]

(ii) State **one** benefit of increasing the number of steps.

.....

..... [1]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

* 0019655333915 *

BLANK PAGE

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

7 Seven program modules form part of a program. A description of the relationship between the modules is summarised below. Any return values are stated in the description.

Module name	Description
Mod-A	calls Mod-B followed by Mod-C
Mod-B	• called with parameters Par1 and Par2 • calls either Mod-D or Mod-E, determined when the program runs • returns a Boolean value
Mod-C	• called with parameters Par1 and Par3 • Par3 is passed by reference • repeatedly calls Mod-F followed by Mod-G
Mod-D	called with parameter Par2
Mod-E	• called with parameter Par3 • returns an integer value
Mod-F	called with parameter Par3
Mod-G	• called with parameter Par3 • Par3 is passed by reference

Parameters in the table are as follows:
- Par1 and Par3 are of type string.
- Par2 is of type integer.

(a) (i) Identify the modules that would be implemented as functions.

..... [1]

(ii) Modules Mod-F and Mod-G are both called with Par3 as a parameter.
In the case of Mod-F, the parameter is passed by value.
In the case of Mod-G, the parameter is passed by reference.

Explain the effect of the **two** different ways of passing the parameter Par3.

.....

.....

.....

..... [2]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

* 0019655333917 *

(b) Draw a structure chart to show the relationship between the seven modules and the parameters passed between them.

[6]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

8 A teacher is designing a program to process pseudocode projects written by her students.

The program analyses a student project and extracts information about each module that is defined (each procedure or function). This information is stored in a global 2D array `ModInfo` of type string.

A module header is the first line of a module definition and starts with either of the keywords `PROCEDURE` or `FUNCTION`.

An example of part of the array is given below. Row 10 of the array shows that a procedure header occurs on line 27 and row 11 shows that a function header occurs on line 35. `"P"` represents a procedure and `"F"` represents a function:

	x = 1	**x = 2**	**x = 3**
`ModInfo[10, x]`	`"27"`	`"P"`	`"MyProc(Z : CHAR)"`
`ModInfo[11, x]`	`"35"`	`"F"`	`"MyFun(Y : CHAR) RETURNS BOOLEAN"`

The string stored in column 3 is called the module description. This is the module header **without** the keyword.

A valid module header will:
- be at least 13 characters long
- start with the keyword `PROCEDURE` or `FUNCTION`. The keyword may appear in either upper or lower case (or a mix of both) and **must** be followed by a space character.

The teacher has defined the first program module as follows:

Module	Description

`Header()`	• called with a parameter of type string representing a line of pseudocode
	• if the line is a valid procedure header, returns a string:
	`"P<Module description>"`
	• if the line is a valid function header, returns a string:
	`"F<Module description>"`
	• otherwise, returns an empty string
	
	For example, given the string:
	
	`"FUNCTION Zap(X : INTEGER) RETURNS CHAR"`
	
	`Header()` returns the string:
	
	`"FZap(X : INTEGER) RETURNS CHAR"`

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

* 0019655333919 *

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

(a) Write pseudocode for module `Header()`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

GradeMax

(b) A new module is required:

Module	Description
`FindModules()`	• called with a parameter of type string representing a student project file name • uses module `Header()` to check each line of the project • assigns values to the `ModInfo` array for each module declaration in the student project

As a reminder, the previous example of part of the array is repeated below:

	x = 1	**x = 2**	**x = 3**
`ModInfo[10, x]`	`"27"`	`"P"`	`"MyProc(Z : CHAR)"`
`ModInfo[11, x]`	`"35"`	`"F"`	`"MyFun(Y : CHAR) RETURNS BOOLEAN"`

Write pseudocode for module `FindModules()`.

Assume that the array contains enough rows for the number of modules in each project.

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

* 0019655333921 *

.....

.....

.....

.....

.....

.....

.....

..... [8]

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

BLANK PAGE

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

GradeMax

* 0019655333923 *

BLANK PAGE

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

© UCLES 2024

GradeMax

BLANK PAGE

DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN
DO NOT WRITE IN THIS MARGIN

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 Assessment International Education Copyright Acknowledgements Booklet. This is produced for each series of examinations and is freely available to download at www.cambridgeinternational.org after the live examination series.

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

© UCLES 2024