GradeMax

* 0000800000001 * DFD

Cambridge Assessment International Education

Cambridge International AS & A Level

CANDIDATE NAME

CENTRE NUMBER

CANDIDATE NUMBER

* 0 1 5 2 5 4 9 6 6 1 *

COMPUTER SCIENCE **9618/23**

Paper 2 Fundamental Problem-solving and Programming Skills **October/November 2025**

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 (CJ) 343243/3
© UCLES 2025

[Turn over

GradeMax

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

1 A user requires a program that implements complex data encryption to a recognised international standard. A programmer has started the design of the program.

(a) The programmer decides to use a library routine to provide part of the solution. One reason for this is that library routines may perform functions that the programmer is unable to program themselves.

State **three** other benefits of using library routines in the development of the program.

1 .....

.....

2 .....

.....

3 .....

.....

[3]

(b) The programmer has identified a different part of the algorithm that would be appropriate to implement as a subroutine (a procedure or a function); no suitable library routine exists for this part.

(i) State **two** reasons why the programmer may decide to use a subroutine.

1 .....

.....

2 .....

.....

[2]

(ii) A function header in pseudocode is defined as:

```
FUNCTION Pass2(Count : INTEGER) RETURNS BOOLEAN
```

Complete the table by describing the terms used in the function header.

The first row has been completed.

Term	Description
`Pass2`	the name of the function
`Count`	
`BOOLEAN`	

[2]

DO NOT WRITE IN THIS MARGIN

GradeMax

(c) Variables in the program have example values:

Variable	Example value
DoB	23/6/2011
Multiplier	2.5
AddressLine[1]	"35 Lincoln Avenue"

Complete the table by evaluating each expression using the example values:

Expression	Evaluates to
LENGTH(NUM_TO_STR(Multiplier))	
MONTH(DoB) > 4	
15 + STR_TO_NUM(MID(AddressLine[1], 2, 1))	

[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

2 **(a)** A program contains a global 1D array of type `STRING` containing 65 elements.

An existing text file is used to store the data in the array. Only non-blank elements (those that do **not** contain an empty string) are written to the text file.

An algorithm will:
- write the first element of the array as a new line in the text file
- continue until all elements have been written, each to a new line of the text file.

Stepwise refinement is applied to the algorithm.

Describe up to **six** steps for this algorithm that could be used to produce pseudocode.

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

Step 1 .....

.....

Step 2 .....

.....

Step 3 .....

.....

Step 4 .....

.....

Step 5 .....

.....

Step 6 .....

.....

[6]

(b) Iteration 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 how it is used.

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

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

3 A program uses a stack to hold up to 30 integer values. The stack is implemented using a global integer variable and a global 1D array.

The array is declared in pseudocode:

```
DECLARE ThisStack : ARRAY[1:30] OF INTEGER
```

Stack design notes:

- The global variable `SP` acts as a stack pointer. `SP` contains the array index of the last value pushed onto the stack.
- If the stack is empty, then `SP` is assigned the value zero.
- The first item added to the stack will be stored in `ThisStack[1]`
- `SP` is incremented each time an item is added to the stack.

A function `Pop()` is written to remove an item from `ThisStack`. The function returns an item of type `PopData` which is defined in pseudocode:

```
TYPE PopData
   DECLARE Data   : INTEGER
   DECLARE Exists : BOOLEAN
ENDTYPE
```

The value removed from the stack is assigned to `Data` and `Exists` is set to `TRUE`. If it is **not** possible to remove a value from the stack, then `Exists` is set to `FALSE`

Complete the pseudocode for `Pop()`

```
FUNCTION Pop() RETURNS PopData

   DECLARE ThisPop : .....

   IF ..... THEN

      PopData.Exists ← ..... // Stack is empty

   ELSE

      PopData.Data ← .....

      PopData.Exists ← .....

      SP ← .....

   ENDIF

   RETURN ThisPop

ENDFUNCTION
```

[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

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

4 A program contains a global 1D array `Data` containing 20 elements of type `INTEGER`

A global string `NumString` represents a sequence of three-digit numbers, separated by commas. For example:

```
"101,456,219,754,328"
```

The string contains at least **four** three-digit numbers. The total number of three-digit numbers in the string is unknown.

A procedure `Store()` will:

- extract one three-digit number at a time from `NumString`
- convert each of the three-digit numbers extracted to an integer and assign this to the next array element, starting from index 1
- end when all three-digit numbers have been stored, or when the array is full.

Complete the pseudocode for `Store()`

All local variables used must be declared.

```
PROCEDURE Store()
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
ENDPROCEDURE
```

[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

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

5 A global 1D array `Num` of integers contains four elements. A program assigns values to these elements as shown:

```
Num[1] ← 1
Num[2] ← 2
Num[3] ← 5
Num[4] ← 3
```

A procedure `Process()` manipulates the values in the array.

The procedure is written in pseudocode:

```
PROCEDURE Process(Start : INTEGER)
   DECLARE CaseVar, Index, Count : INTEGER

   Index ← Start
   Count ← 0

   WHILE Count <= 20
      CaseVar ← Num[Index]
      CASE OF CaseVar
         1 : Num[Index] ← Num[Index] + Index //clause one
             Index ← Index + 1
             Count ← Count + 1

         2 : Num[Index] ← Num[Index] + Index //clause two
             Index ← Index + 2
             Count ← Count + 2

         3 : Num[Index] ← Num[Index] * 2     //clause three
             Index ← Index + 3
             Count ← Count - 1

         4 : Count ← Count + 4               //clause four
         OTHERWISE : Count ← 20
      ENDCASE

      Index ← (Index MOD 4) + 1

   ENDWHILE

ENDPROCEDURE
```

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

(a) Complete the trace table by dry running the procedure when it is called in the statement:
`CALL Process(1)`

Index	CaseVar	Count	Num[1]	Num[2]	Num[3]	Num[4]

[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

(b) As a reminder, the `CASE` structure in the pseudocode is:

```
CASE OF CaseVar
   1 : Num[Index] ← Num[Index] + Index //clause one
       Index ← Index + 1
       Count ← Count + 1

   2 : Num[Index] ← Num[Index] + Index //clause two
       Index ← Index + 2
       Count ← Count + 2

   3 : Num[Index] ← Num[Index] * 2     //clause three
       Index ← Index + 3
       Count ← Count - 1

   4 : Count ← Count + 4               //clause four
   OTHERWISE : Count ← 20
ENDCASE
```

The `CASE` structure could be optimised by combining two existing clauses.

(i) Identify the `CaseVar` values for these **two** existing clauses.

..... [1]

(ii) Write pseudocode for a single clause to replace the **two** clauses identified in **(b) (i)**.

.....

.....

..... [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

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 Students are learning about a simple check digit method for data validation. In this method, a single check digit is appended to the end of an original number to give a new number.

The students are studying a method which:

- calculates the sum of all the digits in the original number
- uses integer division to calculate the remainder when the sum is divided by 10
- uses the remainder as the check digit
- appends the check digit to the original number, creating the new number.

For example:

original number	4162
sum of all digits	4 + 1 + 6 + 2 = 13
remainder when the sum is divided by 10 using integer division	3
new number	41623

The original number is always at least three digits in length.

When the new number is input, the check digit is used to validate the new number.

(a) A function `Generate()` is written to take an original number as a parameter and to return the check digit.

Outline a test plan that could be used to fully test function `Generate()`

Assume that the parameter is valid.

.....

.....

.....

..... [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

(b) A function `CheckNumber()` will take an integer value and return the Boolean value `TRUE` if the check digit is correct, otherwise return `FALSE`

Write pseudocode for the function `CheckNumber()`

Assume that the parameter is valid.

.....

.....

.....

GradeMax

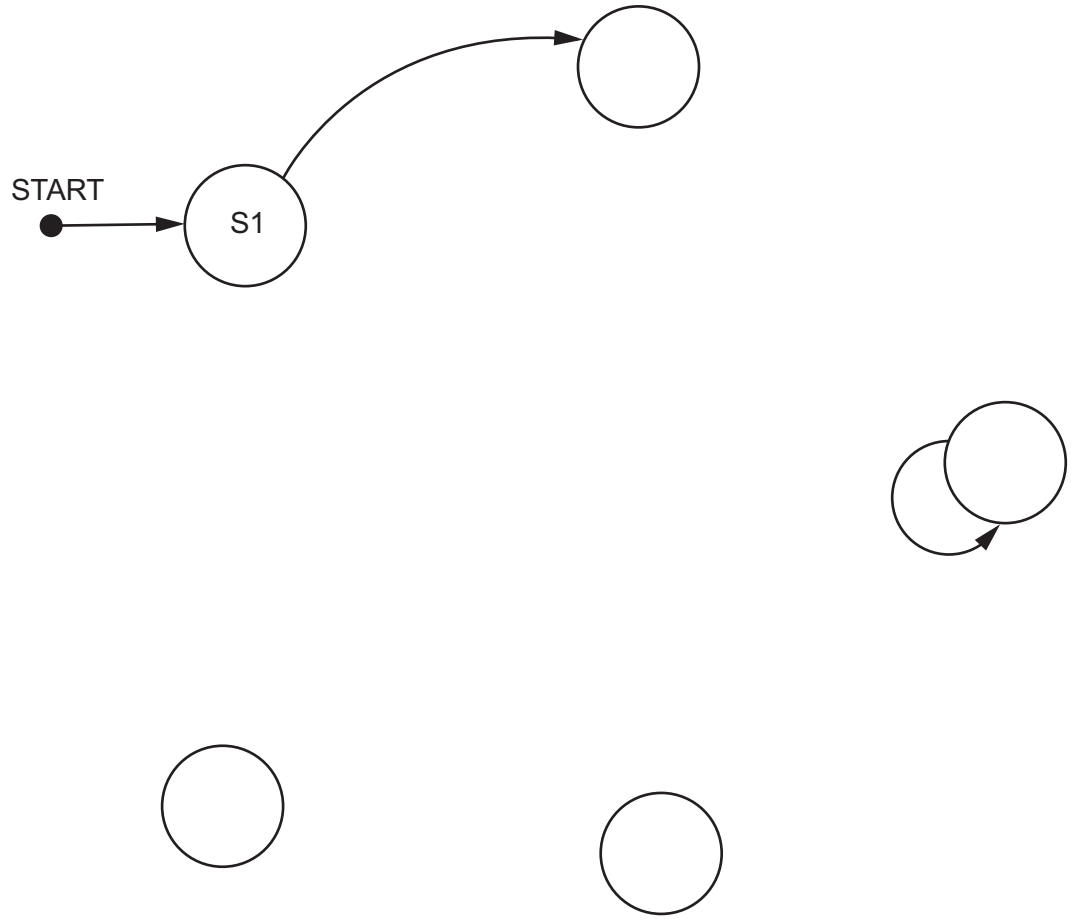
7 There are several different ways to express an algorithm during the design of a program.

(a) One part of the program contains an algorithm which is represented by a state-transition diagram.

The table shows the inputs, outputs and states for the algorithm:

Current state	Input	Output	Next state
S1	A1		S2
S2	A2	X4	S3
S3	A1	X1	S3
S3	A2	X1	S3
S3	A3	X3	S4
S3	A4		S2
S4	A3	X4	S5
S4	A4	X4	S5
S4	A1		S2

Complete the state-transition diagram to represent the information given in the table:



[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

(b) A structure chart is used to document a different part of the program, made up of five modules.

Program notes:
- module Setup calls either module Restart, or module Confirm
- module Confirm takes a string as a parameter and returns an integer
- module Modify has no parameters and returns a Boolean
- module Update takes a string as a parameter
- module Restart repeatedly calls module Update followed by module Modify
- module Restart takes a string as a parameter that is passed by reference.

Draw a structure chart to represent the relationship between the **five** modules, including all parameters and return values.

[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

8 A program is being developed to manage student book loans from a college library.

The programmer has defined a record type to define each loan.

The data items are:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book The first three characters of a `StudentID` represent a tutor ID. Each student has one tutor.
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned

The programmer has defined a global array `Loan` to store 8000 loan records.

There are more elements in the array than books in the library. Unused elements have the `StudentID` set to an empty string. These may occur anywhere in the array.

The programmer has defined a program module:

Module	Description
`LoanStatus()`	• called with two parameters of type `STRING` representing a `StudentID` and a `BookID` • outputs a message saying whether a given loan has been returned or not • outputs a warning message if a record of the given loan is **not** found

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

(a) Write efficient pseudocode for the module `LoanStatus()`

Assume that each combination of `StudentID` and `BookID` can occur only once.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [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

(b) A second module is defined:

Module	Description
`LoansPerTutor()`	• called with a parameter of type `STRING` representing a tutor ID (as a reminder, the first three characters of a `StudentID` represent a tutor ID) • returns an integer value representing the number of books currently on loan to students who have the given tutor

Reminder: unused elements have the `StudentID` set to an empty string. These may occur anywhere in the array.

Write pseudocode for the module `LoansPerTutor()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[7]

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

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

(c) It is decided to mark as unused all records for book loans that have been returned. The data for these records will first be written to a new text file for archive purposes.

(i) The archive program will automatically generate a meaningful filename each time it is run.

Outline a meaningful format to use for the filename.

.....

.....

.....

..... [2]

(ii) There is a problem that will need to be overcome before the data items can be written to a text file.

As a reminder, the data items are:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book The first three characters of a `StudentID` represent a tutor ID. Each student has one tutor.
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned

Explain the problem.

.....

..... [1]

(iii) One way of storing the data items in a text file is to store each data item on a separate line.

Identify **one** benefit and **one** drawback of this way of storing the data.

Benefit .....

.....

Drawback .....

.....

[2]

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

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

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 2025