GradeMax

* 0000800000001 *





Cambridge International AS & A Level

CANDIDATE NAME

CENTRE NUMBER

CANDIDATE NUMBER

9655906422 3

COMPUTER SCIENCE **9618/21**

Paper 2 Fundamental Problem-solving and Programming Skills **May/June 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 **20** pages. Any blank pages are indicated.

DC (WW/CT) 342882/4
© UCLES 2025

[Turn over

GradeMax

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

1 A program is being developed to help the manager of a shop control the stock.

(a) An identifier table has been used during the design stage.

Complete the identifier table:

Example value	Explanation	Variable name	Data type
`"Fruit"`	a category of stock that is sold in the shop		
`20/02/2025`	when an item was sold		
`12.67`	the cost of an item		
`TRUE`	to indicate if an item is in stock		

[4]

(b) A module `Sales()` is part of the stock control program.

The table contains pseudocode extracts from the module `Sales()`

Each extract may include all or part of:

- assignment
- selection
- iteration (repetition).

Complete the table by placing one or more ticks (✓) in each row:

Pseudocode extract	Assignment	Selection	Iteration
`Result ← CalculateTotal()`			
`WHILE IsClosed`			

`REPEAT`
`  INPUT Value`
`UNTIL Sales[4] > Value`			

`IF Sales[Current] <= 150 THEN`
`  Discount ← TRUE`
`ENDIF`			

`CASE OF Option`			

[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

(c) Decomposition has been used to design the program to help the shop manager control the stock.

Describe decomposition.

.....

.....

.....

.....

.....

..... [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

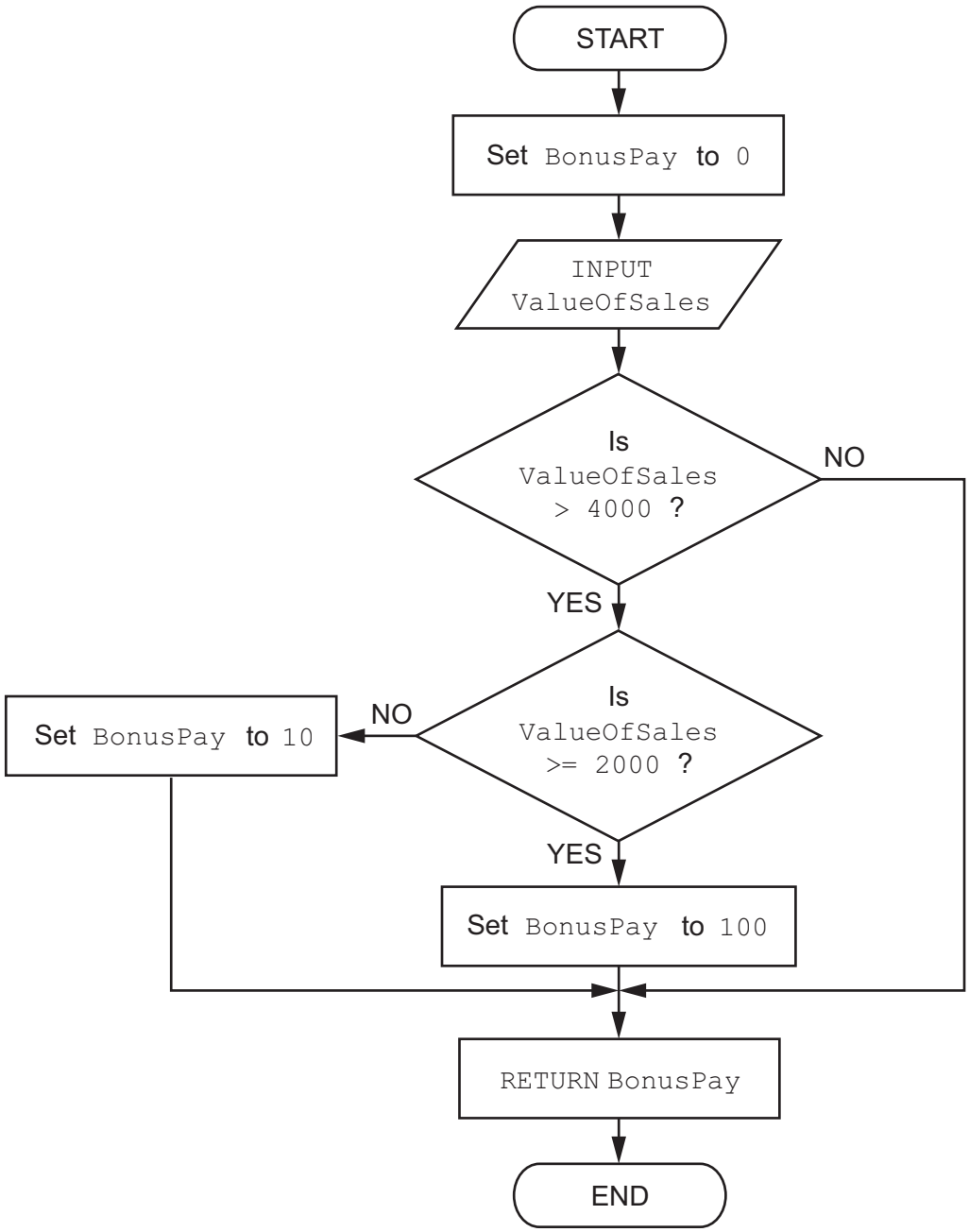
2 A program is being developed to calculate the pay of employees working for a company.

A function `CalculateBonus()` calculates bonus pay based on the value of sales.

Bonus pay is calculated as shown in the table.

Value of sales (in dollars)	Bonus pay (in dollars)
below 2000	0
between 2000 and 4000 inclusive	10
above 4000	100

A flowchart for the function `CalculateBonus()` has been designed.



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) The flowchart contains logic errors.

One logic error is that `BonusPay` will **not** be set to `10` although the `ValueOfSales` input is between `2000` and `4000` inclusive.

(i) Explain why this error occurs.

.....

..... [1]

(ii) Explain how this error could be corrected.

.....

..... [1]

(b) There are different ways to reduce the risk of errors when developing the new program, such as the use of constants.

(i) State a value that could be replaced by a constant in the function `CalculateBonus()`

..... [1]

(ii) Explain how the use of constants helps to reduce the risk of programming errors.

.....

.....

.....

..... [2]

(iii) One other way that can reduce the risk of errors when writing the program is the use of library routines.

Explain how library routines can reduce the risk of programming errors.

.....

..... [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

(c) All the logic errors have been corrected, and the program has been coded.

Identify and describe **two** other types of error that the program could contain.

Type of error .....

Description .....

.....

.....

Type of error .....

Description .....

.....

.....

[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

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 student has been asked to create a simple guessing game program. This program will generate a random integer value between 1 and 100. It will then repeatedly prompt the user to input an integer value until they input the randomly generated value.

The student has written a structured English description:

step 1 – randomly generate an integer value between 1 and 100 inclusive
step 2 – prompt the user to input an integer value
step 3 – output an appropriate message if the value input was too high; then repeat from **step 2**
step 4 – output an appropriate message if the value input was too low; then repeat from **step 2**
step 5 – output an appropriate message if the value input was the same value that was randomly generated; then end the program.

Write a pseudocode algorithm from this structured English description.

Assume no input validation is needed.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [8]

GradeMax

4 Study the algorithm:

```
DECLARE Chars : ARRAY[1:4] OF CHAR
DECLARE I, J : INTEGER
DECLARE Key : CHAR
I ← 2
Chars[1] ← 'D'
Chars[2] ← 'T'
Chars[3] ← 'H'
Chars[4] ← 'R'
WHILE I <= 4                              //Outer loop
   Key ← Chars[I]
   J ← I - 1
   WHILE J >= 0 AND Chars[J] > Key        //Inner loop
      Chars[J + 1] ← Chars[J]
      J ← J - 1
   ENDWHILE
   Chars[J + 1] ← Key
   I ← I + 1
ENDWHILE
```

(a) The outer loop structure used in the algorithm is **not** the most appropriate one to use.

State the type of loop structure that would be the most appropriate to use and justify why it is the most appropriate.

Loop structure .....

Justification .....

.....

[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

(b) Complete the trace table by dry running the algorithm.

The first row has been completed.

I	Key	J	Chars[J]	Chars			
				[1]	[2]	[3]	[4]
2				'D'	'T'	'H'	'R'

[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

5 Stacks and queues are both abstract data types.

A stack uses a top-of-stack pointer to indicate the location of the last item added to the stack.

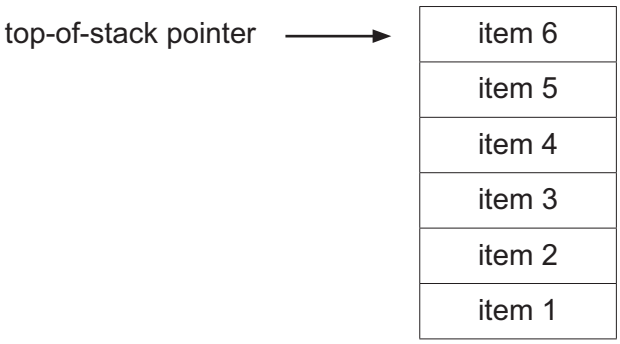
A queue uses two pointers:

- a front pointer to indicate the location of the next item to be removed from the queue
- a rear pointer to indicate the location of the next item to be added to the queue.

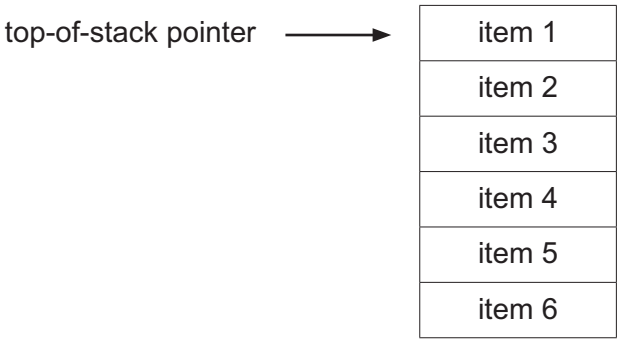
A queue can be used to reverse the items stored on a stack.

For example, if a stack contains six items:

Initial state of the stack:



Final state of the stack when the items have been reversed:



Describe how the queue could be used to reverse the items that are currently stored on the stack.

Your description must include how the pointers are used in both the stack and queue.

Assume:

- The stack initially contains an unknown number of items.
- The queue can store all the items currently stored on the stack.
- The queue is initially empty.

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [7]

GradeMax

6 A program monitors the speed of vehicles as they move around a large building site.

Each vehicle contains a sensor which reads an integer value that represents the speed of the vehicle. The value is expected to be in the range 0 to 60 inclusive.

The sensors cannot read values less than 0.

A program module has been written to validate the values read by the sensors.

(a) A test plan is needed to fully test the module.

Complete the table. The first line has been completed for you.

Assume the sensors generate only integer values.

Type of test data	Test data value	Expected outcome
normal	36	data item is accepted

[4]

(b) Each sensor used in the system has a unique sensor ID number in the range 1 to 50.

The program that is used to monitor the speed of each vehicle reads each sensor value every second and stores this value along with the sensor's unique ID into a global 2D array `Reading`

The global array `Reading` has been declared as follows:

```
DECLARE Reading : ARRAY[1:2000, 1:2] OF INTEGER
```

The array contains 4000 elements organised as 2000 rows and 2 columns.

Column 1 contains the sensor value, and column 2 contains the sensor ID.

When 2000 sensor readings have been taken, the array is full and the system stops taking any more sensor readings until the array has been processed.

A procedure `Sort()` is needed to sort the array into ascending order of sensor value using an efficient bubble sort algorithm.

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

Write **efficient** pseudocode for the procedure `Sort()`

[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

7 A program is being developed to implement a customer loyalty scheme for a coffee shop.

The programmer has decided that the following data items need to be stored for each customer:

Data item	Description
customer ID	a unique six-digit string
points	an integer value that is increased by one for every cup of coffee the customer orders

When a customer visits the shop and orders coffee, the scheme operates as follows:

- The total number of points is increased by the number of coffees ordered.
- If just one cup of coffee is ordered and the number of points goes above 10, then:
  - the cup of coffee they have just ordered is given to them free of charge
  - the number of points is reduced by 11.
- If the order is for multiple coffees and the number of points goes above 10, then:
  - they get one coffee free of charge for every 11 points
  - the number of points is reduced by 11 for each free coffee.

For example, the:

- customer currently has 9 points
- customer orders 16 cups of coffee
- total number of points now becomes 25
- customer gets 2 free coffees and now has 3 points left.

The programmer has defined a program module that is called every time a customer places an order:

Module	Description

`CustomerOrder()`	• called with two integer parameters:
	  ◦ the number of coffees ordered
	  ◦ the current number of points
	• output a suitable message giving the number of free coffees
	• return a value for the new points total.

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 pseudocode for module `CustomerOrder()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [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) A text file `Loyalty.txt` will be used to store the data items for the loyalty scheme. The data items for each customer will be stored on a separate line of the text file where each data item is separated by a comma:

```
<CustomerID>,<Points>
```

The contents of the text file `Loyalty.txt` will always be stored in ascending order by customer ID.

When the data items are read from or written to the text file `Loyalty.txt`, they may need to be converted to the appropriate data type.

Each customer has a unique customer ID starting at `"100001"` with this value increasing by one each time a new customer joins the loyalty scheme.

When a customer joins the loyalty scheme, they are assigned the next customer ID and value of points is set to 0.

For example, if the loyalty scheme has 204 customers and a new customer joins the loyalty scheme, the following line is added to the text file `Loyalty.txt`:

```
"100205,0"
```

You can assume that the number of customers in the loyalty scheme will never be more than 9000.

The programmer has defined a program module as follows:

Module	Description
`AddNewCustomers()`	• called with an integer parameter representing the number of new customers to be added to the loyalty scheme • adds a new line, containing the required information, to the text file `Loyalty.txt` for each customer added to the loyalty scheme • outputs each new customer ID added to the loyalty scheme

(i) Write pseudocode for module `AddNewCustomers()`

Assume that there is at least **one** customer already in the loyalty scheme.

.....

.....

.....

.....

.....

.....

.....

.....

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

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [8]

GradeMax

(ii) The requirements for `AddNewCustomers()` are changed. There may be no existing customers in the loyalty scheme.

Explain the changes that will need to be made to the module `AddNewCustomers()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [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

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