Chhetri Academy

chhetriacademy.com

GCSE COMPUTER SCIENCE

6.4

Edexcel 1CP2 | Specification point 6.4.4

The need for and be able to write programs that implement authentication (ID and password, lookup)

Untiered

Answers and examiner comments

1	2023	Untiered
question groups	available papers	tier

Using this booklet

1. Answer numbers match the companion question booklet.
2. Follow the original question number on marking tables.
3. Examiner comments are included where available.

Name _____ **Date** _____

Some entries use full source marking material in an appendix. Any unavailable mark schemes are identified beside the relevant answer.

Question number	MP	Appx. Line	Answer	Additional guidance	Mark
6			Award marks as shown.		**(15)**
	6.1		Two string inputs taken (1)		
	6.2		Check for blank input for name/password using a relational operator (1)	• `len (name) == 0` • `len (password) == 0` • `name == ""` • `password == ""`	
	6.3		Table is traversed using a loop (1)	• `for (…)` • `while (…)` • Allow membership	
	6.4		Use of a logical operator to form a compound test / use of loops to keep invalid input out (1)	• Allow logical operator anywhere in program • `(len (name) == 0) or (len (password) == 0)` • `(not foundName) and (index < len (userTable)` • `while (name == "")…`	
	6.5		Individual fields of each record accessed (1)	• `userTable[index][0] == name` • `userTable[index][1] == password)` • Allow membership	
	6.6		Mechanism for distinguishing between states (1)	• Selection, which may be nested • Three states are: name not found; name found no password match; full match	
			Levels-based mark scheme to a maximum of 9, from:	Considerations for levels-based mark scheme:	
	6.7 6.8 6.9		Solution design (3)	• Solution decomposed into component parts • Stops loop when name and password match found • Stops loop when name match found, but not password • Minimum number of passes through the data (i.e. visits each record only once)	
	6.10 6.11 6.12		Good programming practice (3)	• Meaningful variable names • Layout with white space improves readability • Commenting is sufficient to completely follow the logic, without being excessive	

	6.13 6.14 6.15		Functionality (3)	• User messages are informative and fit for purpose • Robust, does not crash with syntax or runtime errors • Fully meets requirements, including working with any length of array	

Test data:

User name	Password	Output	Note
LLemon8	BeigeDresser	Welcome	Valid user and password found in the array
LLemon8	GreyOttoman	Incorrect password	The user name is found. The password belongs to a different user.
llemon8	BeigeDresser	User not found	The user name has lowercase letters so doesn't match.
OOrange99	WhiteNights	User not found	The user name is not there. The password belongs to a different user.
BJones33	GoldBed	User not found	The user does not exist. The password is not in the table.
<empty>	GreenCouch	Invalid input	User name cannot be blank.
LLemon8	<empty>	Invalid input	Password cannot be blank.

Solution design (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to decompose the problem. • Some of the component parts of the problem can be seen in the solution, although this will not be complete. • Some parts of the logic are clear and appropriate to the problem. • The use of variables and data structures, appropriate to the problem, is limited. • The choice of programming constructs, appropriate to the problem, is limited.	• There has been some attempt to decompose the problem. • Most of the component parts of the problem can be seen in the solution. • Most parts of the logic are clear and appropriate to the problem. • The use of variables and data structures is mostly appropriate. • The choice of programming constructs is mostly appropriate to the problem.	• The problem has been decomposed clearly into component parts. • The component parts of the problem can be seen clearly in the solution. • The logic is clear and appropriate to the problem. • The choice of variables and data structures is appropriate to the problem. • The choice of programming constructs is accurate and appropriate to the problem.	**3**

Good programming practices (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to lay out the code into identifiable sections to aid readability. • Some use of meaningful variable names. • Limited or excessive commenting. • Parts of the code are clear, with limited use of appropriate spacing and indentation.	• There has been some attempt to lay out the code to aid readability, although sections may still be mixed. • Uses mostly meaningful variable names. • Some use of appropriate commenting, although may be excessive. • Code is mostly clear, with some use of appropriate white space to aid readability.	• Layout of code is effective in separating sections, e.g. putting all variables together, putting all subprograms together as appropriate. • Meaningful variable names and subprogram interfaces are used where appropriate. • Effective commenting is used to explain logic of code blocks. • Code is clear, with good use of white space to aid readability.	**3**

Functionality (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	**Functionality (when the code is run)** • The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements. • Program outputs are of limited accuracy and/or provide limited information. • Program responds predictably to some of the anticipated input. • Solution is not robust and may crash on anticipated or provided input.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that meets most of the stated requirements. • Program outputs are mostly accurate and informative. • Program responds predictably to most of the anticipated input. • Solution may not be robust within the constraints of the problem.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that fully meets the given requirements. • Program outputs are accurate, informative, and suitable for the user. • Program responds predictably to anticipated input. • Solution is robust within the constraints of the problem.	**3**



Examiners' Report
Principal Examiner Feedback

Summer 2023

Pearson Edexcel GCSE
In Computer Science (1CP2) Paper 2

1

Edexcel and BTEC Qualifications

Edexcel and BTEC qualifications are awarded by Pearson, the UK's largest awarding body. We provide a wide range of qualifications including academic, vocational, occupational and specific programmes for employers. For further information visit our qualifications websites at www.edexcel.com or www.btec.co.uk. Alternatively, you can get in touch with us using the details on our contact us page at www.edexcel.com/contactus.

Pearson: helping people progress, everywhere

Pearson aspires to be the world's leading learning company. Our aim is to help everyone progress in their lives through education. We believe in every kind of learning, for all kinds of people, wherever they are in the world. We've been involved in education for over 150 years, and by working across 70 countries, in 100 languages, we have built an international reputation for our commitment to high standards and raising achievement through innovation in education. Find out more about how we can help you and your students at: www.pearson.com/uk

Summer 2023
Publications Code 1CP2_02_2306_ER
All the material in this publication is copyright
© Pearson Education Ltd 2023

Introduction

This is the second examination of the Edexcel GCSE Computer Science (9-1), with the paper two onscreen exam. The programming language required is Python 3.

Students are supplied with a question paper, a programming language subset document, and a code file for each question. Students are required to amend the code files and save their work, using a different file name.

Centres compress the code file responses for each student. The compressed files are uploaded to Edexcel for external assessment, via the Learner Work Transfer platform.

Centre submissions

The ICE document for this series set out the format in which students' completed code files were to be submitted. The majority of centres were able to follow the instructions accurately, ensuring that a single zipped file of the COMPLETED_CODE folder was provided for each student. The submissions were correctly identified with the centre and student number.

General

Range of marks

A full range of marks was awarded for Paper 2. Examiners did see some submissions which achieved the full 75 marks available.

Attempting all questions

In common with 2022, there were a number of scripts where students did not attempt Q05 and Q06, thereby missing an opportunity to access some marks. There are partial marks that could be awarded in each question. Students are reminded to attempt all the questions on the paper.

Readability

It is not necessary to comment every line of code in a solution. In common with 2022, examiners saw some responses where the number of comments exceeded the number of code lines. Comments are to help understand the logic, so should be placed, more helpfully, at the start of blocks of code. Excessive commenting makes the response difficult to read.

White space also can help with readability, but there is no requirement to double space code. Use white space between blocks of logic. Single spacing is appropriate for code.

Execute and test the code

Marks are awarded in some questions, regardless if the code interprets and executes. However, in others, marks are awarded for interpretation and functionality. Students should always attempt to execute the code. The IDE will highlight syntax errors in the code editor or identify them with a runtime error during execution. Students can fix syntax and indentation errors this way.

In Q03, where students chose correct lines of code, the code should be executed with the test data given in the question paper. Execution would quickly identify that some incorrect lines were chosen.

Q01 – Name the part

This is a new format of question. Students open a code file, but this code file has a section of comments at the bottom. The comments are constructed to look like a table. In the first column is a statement. The students inspect the provided code to find the identifier or line number(s) of the code that corresponds to the statement in the first column. They then complete the second column with the required information.

The majority of students submitted very good responses.

Some students struggled to identify initialisation (mp1.3) and selection (mp1.4). Others confused repetition (mp1.5) and iteration (mp1.6).

Remember, however, that students can look up definitions in the PLS during the exam.

Q01 Example 1

```
# -----------------------------------------------------------------
# Constants
# -----------------------------------------------------------------
DISCOUNT_5 = 0.05           # 5% discount
DISCOUNT_10 = 0.10          # 10% discount

# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
theTemperatures = [27.7, 29.8, 33.0, 31.6, 28.4, 28.0, 27.9]
aCost = 0.0
total = 0.0

# -----------------------------------------------------------------
# Main program
# -----------------------------------------------------------------
for temp in theTemperatures:        # Display temperatures
    print (temp)

# Add up costs until the user enters zero
aCost = float (input ("Enter a cost: "))
while (aCost != 0.0):
    total = total + aCost
    aCost = float (input ("Enter a cost : "))

# Calculate discount based on the total of numbers entered by the user
if (total > 100.00):
    total = (1 - DISCOUNT_10) * total       # Get 10% discount
else:
    total = (1 - DISCOUNT_5) * total        # Get 5% discount

print (total)
# -----------------------------------------------------------------
# =====> Write your answers here in the right-hand column
# Left                                          # Right
# -----------------------------------------------------------------
# Name of a constant used in the program        #DISCOUNT_5
# Name of an array used in the program          #theTemperatures
# Line number of an initialisation of a
#      variable with a real number              #22
# Line numbers for a selection construct        #27
# Line number(s) for a repetition construct     #22
# Line number(s) for an iteration construct     #27
# Line number for an instruction that outputs
#      information to the screen                #32
```

This example was awarded five marks. It demonstrates good understanding of the use of constants and arrays. It has confused iteration and selection.

Q01 Example 2

```
# -----------------------------------------------------------------
# Constants
# -----------------------------------------------------------------
DISCOUNT_5 = 0.05           # 5% discount
DISCOUNT_10 = 0.10          # 10% discount

# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
theTemperatures = [27.7, 29.8, 33.0, 31.6, 28.4, 28.0, 27.9]
aCost = 0.0
total = 0.0

# -----------------------------------------------------------------
# Main program
# -----------------------------------------------------------------
for temp in theTemperatures:        # Display temperatures
    print (temp)

# Add up costs until the user enters zero
aCost = float (input ("Enter a cost: "))
while (aCost != 0.0):
    total = total + aCost
    aCost = float (input ("Enter a cost : "))

# Calculate discount based on the total of numbers entered by the user
if (total > 100.00):
    total = (1 - DISCOUNT_10) * total       # Get 10% discount
else:
    total = (1 - DISCOUNT_5) * total        # Get 5% discount

print (total)
# -----------------------------------------------------------------
# =====> Write your answers here in the right-hand column
# Left                                          # Right
# -----------------------------------------------------------------
# Name of a constant used in the program        #the values of the temperature
# Name of an array used in the program          #theTemperatures
# Line number of an initialisation of a
#      variable with a real number              #17
# Line numbers for a selection construct        #27 and 29
# Line number(s) for a repetition construct     #22 and 17
# Line number(s) for an iteration construct     #lines 4 and 5
# Line number for an instruction that outputs
#      information to the screen                #32
```

This example was awarded four marks. This response demonstrates an understanding of initialisation and selection.

5

Q02 – Fix the errors

Solutions required students to fix syntax errors, complete lines of code, fix runtime errors and logic errors to produce the required functionality.

The majority of students submitted good responses.

The most frequently lost marks were the corrections of the logic errors (mp2.4, mp2.6).

Examiners did also see instances where the use of white space (mp2.8) was not awarded. The question paper clearly states use of white space as a requirement. Students should make sure to read the question paper as well as the instructions in the code files.

Q02 Example 1

```
# -----------------------------------------------------------------
# Import libraries
# -----------------------------------------------------------------
import random
# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
exerciseTable = ["squats", "planks", "pushups", "lunges", "burpees"]

index = 0
name = ""
numExercises = 0
# -----------------------------------------------------------------
# Main program
# -----------------------------------------------------------------
print ("Here is the exercise table")

for exercise in exerciseTable:
    print (exercise)
numExercises = int (input ("How many exercises do you need (1-5)? "))

for count in range (numExercises + 1):
    index = random.randint (0, 4)
    name = exerciseTable[index - 1]
    print (name)
```

This example was awarded six marks. It demonstrates a good example of fixing both syntax and runtime errors. However, it has not fixed the logic errors.

Q02 Example 2

```
1  # -----------------------------------------------------------------
2  # Import libraries
3  # -----------------------------------------------------------------
4  import random
5  # -----------------------------------------------------------------
6  # Global variables
7  # -----------------------------------------------------------------
8  exerciseTable = ["squats", "planks", "pushups", "lunges", "burpees"]
9  index = 0
10 name = ""
11 numExercises = 0
12 # -----------------------------------------------------------------
13 # Main program
14 # -----------------------------------------------------------------
15 print ("Here is the exercise table")
16
17 for exercise in exerciseTable:
18     print (exercise)
19
20 numExercises = int (input ("How many exercises do you need (1-5)? "))
21
22 for count in range (numExercises):
23     index = random.randint(1,5)
24     name = exerciseTable[index - 1]
25     print(name)
```

This example was awarded seven marks. Although the call to the random function is not awarded, line 24 has been amended to make the code functional. As a result, that correction was awarded as follow through.

7

Q03 – Choose the lines

Solutions required completion of the given code lines or selecting the correct line of code from four options.

Once, all the selections are made, students can execute their code to find and amend any lines where the wrong option has been chosen.

The majority of students submitted good responses.

The most frequently missed marks were those associated with the naming of the input and output files. Another challenge was the use of constants to open and close the files.

Q03 Example 1

```
# -----------------------------------------------------------------
# Constants
# -----------------------------------------------------------------
SPECIFIED_MATERIAL = "Copper"

# =====> Add the correct input file name
INPUT_FILE = open("Screws.txt")

# =====> Add the correct extension to this file name
OUTPUT_FILE = ("Bricks","w")

# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
# =====> Complete the line with the correct variable name for the array of bricks
brickTable= ["Rustic", "Heather",
             "Staffordshire", "Tudor", "Hampton",
             "Norman", "Northcote",
             "Tuscan", "Regency",
             "Concrete Common",
             "Old English",
             "Hadrian Gold"]

inFile = ""
outFile = ""
foundCount = 0

# =====> Choose the correct value to initialise the variable
#total = 0.0
#total - ""
total = 0
#total = True

# =====> Choose the correct value to initialise the variable
outLine = False
#outLine = ""
#outLine = 0.0
#outLine = 0

```

```
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------

# Process the screws

# =====> Choose the correct line to open the file
#inFile = open ("Screws", "r")
#inFile = open ("Screws", "a")
#inFile = open (INPUT_FILE, "a")
inFile = open (INPUT_FILE, "r")

for line in inFile:
    # =====> Choose the correct line to locate the
    #        substring in the line
    #if (line.find (SPECIFIED_MATERIAL) == -1):
    #if (line.find (SPECIFIED_MATERIAL) != -1):
    #if (line.find (SPECIFIED_MATERIAL) == False):
    if (line.find (SPECIFIED_MATERIAL) == True):
        foundCount = foundCount + 1
# =====> Complete the line to increment total
    total = total + 1

# =====> Choose the correct line to close the file
#inFile.close ()
Screws.close ()
#INPUT_FILE.close ()
#outFile.close ()

# =====> Choose the correct line to display the output
#print ("Total screws: ", foundCount, "SPECIFIED_MATERIAL", total)
#print ("Total screws: ", total)
#print ("Total screws: " + str (foundCount) + "Copper" + str (total))
print ("Total screws: " + str (total) + " " + SPECIFIED_MATERIAL + " screws: " +
str (foundCount))

# Process the bricks

# =====> Choose the correct line to open the bricks file
#outFile = open (OUTPUT_FILE, "r")
outFile = open (OUTPUT_FILE, "w")
#outFile = open ("Bricks", "r")
#outFile = open (OUTPUT_FILE, "a")

for brick in brickTable:
    # =====> Choose the correct line to convert the case
    brick = brick.upper ()
    #brick = brick.isalpha ()
    #brick = brick.format ("{}")
    #brick = brick.isupper ()

    # =====> Choose the correct line to complete the output line
    #outLine = brick
    #outLine = brick + "\r"
    outLine = brick + "\n"
    #outLine = brick + ","

    # =====> Choose the correct line to write the line to the file
    #outFile.writelines (brickTable)
    #outFile.write (outLine)
    #outFile.writelines (brick)
    outFile.write (brick)

outFile.close ()

# =====> Choose the correct line to display the output
print ("Wrote", len (brickTable), "brick names to file")
#print ("Wrote", total, "brick names to file")
#print ("Wrote {:^5.2f} brick names to file".format (len (brickTable)))
#print ("Wrote {:^5.2f} brick names to file".format (total))
```

This response was awarded ten marks. It demonstrates an understanding of data types and the manipulation of strings. It highlights the common errors around file handling.

9

Q03 Example 2

```
# -----------------------------------------------------------------
# Constants
# -----------------------------------------------------------------
SPECIFIED_MATERIAL = "Copper"

# =====> Add the correct input file name
INPUT_FILE = open ("Screws.txt", r)

# =====> Add the correct extension to this file name
OUTPUT_FILE = "Bricks"

# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
# =====> Complete the line with the correct variable name for the array of bricks
bricks = ["Rustic", "Heather",
                "Staffordshire", "Tudor", "Hampton",
                "Norman", "Northcote",
                "Tuscan", "Regency",
                "Concrete Common",
                "Old English",
                "Hadrian Gold"]

inFile = ""
outFile = ""
foundCount = 0

# =====> Choose the correct value to initialise the variable
#total = 0.0
#total - ""
total = 0
#total = True

# =====> Choose the correct value to initialise the variable
#outLine = False
#outLine = ""
#outLine = 0.0
outLine = 0

```

10

```
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------

# Process the screws

# =====> Choose the correct line to open the file
#inFile = open ("Screws", "r")
#inFile = open ("Screws", "a")
inFile = open (INPUT_FILE, "a")
#inFile = open (INPUT_FILE, "r")

for line in inFile:
    # =====> Choose the correct line to locate the
    #        substring in the line
    #if (line.find (SPECIFIED_MATERIAL) == -1):
    if (line.find (SPECIFIED_MATERIAL) != -1):
    #if (line.find (SPECIFIED_MATERIAL) == False):
    #if (line.find (SPECIFIED_MATERIAL) == True):
        foundCount = foundCount + 1
# =====> Complete the line to increment total
    total =

# =====> Choose the correct line to close the file
#inFile.close ()
#Screws.close ()
INPUT_FILE.close ()
#outFile.close ()

# =====> Choose the correct line to display the output
#print ("Total screws: ", foundCount, "SPECIFIED_MATERIAL", total)
#print ("Total screws: ", total)
#print ("Total screws: " + str (foundCount) + "Copper" + str (total))
print ("Total screws: " + str (total) + " " + SPECIFIED_MATERIAL + " screws: " +
str (foundCount))

# Process the bricks

# =====> Choose the correct line to open the bricks file
outFile = open (OUTPUT_FILE, "r")
#outFile = open (OUTPUT_FILE, "w")
#outFile = open ("Bricks", "r")
#outFile = open (OUTPUT_FILE, "a")

for brick in brickTable:
    # =====> Choose the correct line to convert the case
    brick = brick.upper ()
    #brick = brick.isalpha ()
    #brick = brick.format ("{}")
    #brick = brick.isupper ()

    # =====> Choose the correct line to complete the output line
    #outLine = brick
    #outLine = brick + "\r"
    outLine = brick + "\n"
    #outLine = brick + ","

    # =====> Choose the correct line to write the line to the file
    #outFile.writelines (brickTable)
    #outFile.write (outLine)
    outFile.writelines (brick)
    #outFile.write (brick)

outFile.close ()

# =====> Choose the correct line to display the output
#print ("Wrote", len (brickTable), "brick names to file")
print ("Wrote", total, "brick names to file")
#print ("Wrote {:^5.2f} brick names to file".format (len (brickTable)))
#print ("Wrote {:^5.2f} brick names to file".format (total))
```

This response was awarded six marks. It also demonstrates a good understanding of how to manipulate string data.

11

Q04 – Comment first coding

In this question students are writing lines of code from scratch. The logic to solve the problem is already designed for the student and is presented as comments in the code file. This is the first question in the paper that uses the Functionality Levels-based Mark Scheme.

Where students followed the logic set out in the comments to guide them in writing the code, very good marks were awarded.

The majority of responses correctly translated all the formulae from the question into the required program code.

The question paper states to use relational and logical operators to check for invalid inputs. The design of the comments should lead students to do just that. However, some students decided to use three while loops to force the user to keep entering numbers, for each dimension, until a valid one was encountered. This was not a requirement set out in the question paper. It also means that the requirement to use a logical operator could be missed.

In some instances, examiners saw the use of <string>.format() instead of round(area, 2). Where this was used only for display, it works. Where the output of the <string>.format() is assigned to area, the reduction in precision will affect the accuracy of the final outputs.

Q04 Example 1

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
# =====> Write your code here

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# =====> Write your code here
# Take three decimal inputs from the user
base_width = float(input("What is the width of the base of the triangle? "))
height = float(input("What is the height of the triangle? "))
length = float(input("What is the length of the triangle? "))

# Check for invalid inputs, using relational and logical operators
# Invalid input should not be processed
while base_width <= 0 or height <= 0 or length <= 0:
    # Display an error message if any input is invalid
    print("Invalid input")
    base_width = float(input("What is the width of the base of the triangle? "))
    height = float(input("What is the height of the triangle? "))
    length = float(input("What is the length of the triangle? "))
    # invalid inputs prevented from being processed by a while loop

# Process all valid inputs
# Calculate the area of the triangle
area = height * base_width * 0.5

# Display the area of the triangle, rounded to two decimal places
print("{:2.2f}".format(area))

# Calculate the volume of the prism
volume = area * length

# Display the volume of the prism using the <string>.format() function
# in eight columns with two decimal places
print("{:2.2f}".format(volume))
# In all cases, display a goodbye message just before terminating
print("Goodbye.")
```

This response was awarded 13 marks. It is a good example of using relational and logical operators, along with a while loop, to control for valid input.

Q04 Example 2

```
# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
# =====> Write your code here

# ----------------------------------------------------------------------
# Main program
# ----------------------------------------------------------------------
# =====> Write your code here
# Take three decimal inputs from the user
base = float(input("please enter the length of the base of your triangle on your
triangular prism "))
height = float(input("please enter the height of your triangular prism "))
length = float(input("please enter the length of your triangular prism"))
# Check for invalid inputs, using relational and logical operators
while base <= 0:
    base = float(input("base must be less than 0, input again: "))
while height <= 0:
    height = float(input("height must be less than 0, input again: "))
while length <= 0:
    length = float(input("length must be less than 0, input again: "))

# Display an error message if any input is invalid
# Invalid input should not be processed

# Process all valid inputs
# Calculate the area of the triangle
areaTriangle = round(0.5 * (base * height), 2)
# Display the area of the triangle, rounded to two decimal places
print("the area of the triangle is: " + str(areaTriangle))
# Calculate the volume of the prism
volumePrism = areaTriangle * length
# Display the volume of the prism using the <string>.format() function
# in eight columns with two decimal places
layout = "{:^8.2f}"
print("the volume of your prism is: " + layout.format(volumePrism) + "cm^3")
# In all cases, display a goodbye message just before terminating
print("thank you for trying my program, have a nice day :)")
```

This response was also awarded 13 marks. It demonstrates accurate translation of formulae, but preserves the result of rounding.

13

Q05 – Subprograms

In this question, students are required to create a programmed solution to a problem. Students are provided with the requirements of the problem in the question paper. This is followed up in the student code file with instructions, located where the changes and additions are required.

Most responses were awarded marks for outputting messages and conversion of strings to lower case.

Many responses changed the subprogram parameters to make their names different to the global variable names. Fewer followed that through to change the names of the variables inside the program to match the new parameter names. Some responses were seen where the first two parameter names were changed, but not the third.

Q05 Example 1

```
# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
lastName = ""
firstName = ""
dob = ""
myID = ""
valid = False
# ----------------------------------------------------------------------
# Subprograms
# ----------------------------------------------------------------------
# =====> Change the names of the local variables to distinguish them
#        from the global variables with the same name
def makeID (p_lastName, p_firstName, p_dob):
    namePart = ""
    numberPart = 0

    namePart = lastName.lower() + firstName[0].lower()  # Letter part

    # =====> Correct the logic error caused by using the int() function
    #        in the number part calculation rather than using a function
    #        that returns the ASCII value of the character
    for character in dob:
        numberPart = numberPart + ord (character)

    yourID = namePart + str (numberPart)

    return (yourID)

# =====> Add a procedure, with no parameters, to display a
#        welcome message for the user
welcome_message = "Welcome User"
# ----------------------------------------------------------------------
# Main program
# ----------------------------------------------------------------------
# =====> Call the welcome procedure before taking input from the user
print (welcome_message)
# Get last name and first name from the user
lastName = input ("Enter your last name: ")
firstName = input ("Enter your first name: ")

# =====> Convert last name and first name to lowercase after they
#        are inputted by the user
lastName.lower()
firstName.lower()
# Get date of birth from the user
dob = input ("Enter your date of birth (ddmmyyyy): ")

# =====> Check that only the digits 0 to 9 appear in the date of birth
if dob.isdigit():
    valid = True
# =====> Call the makeID() function, if the date of birth is valid
if valid == True:
    myID = makeID (lastName, firstName, dob)
    print (myID)

# =====> Tell the user, if the date of birth is invalid
else:
    print("dob is invalid")
```

This response was awarded 11 marks. It demonstrates changing input parameters and case conversions.

Q05 Example 2

```
# -----------------------------------------------------------------
# Global variables
# -----------------------------------------------------------------
lastName = ""
firstName = ""
dob = ""
myID = ""

# -----------------------------------------------------------------
# Subprograms
# -----------------------------------------------------------------
# =====> Change the names of the local variables to distinguish them
#        from the global variables with the same name
def makeID (name1, name2, birth):
    namePart = ""
    numberPart = 0

    namePart = name1 + name2[0]  # Letter part

    # =====> Correct the logic error caused by using the int() function
    #        in the number part calculation rather than using a function
    #        that returns the ASCII value of the character
    for character in birth:
        numberPart = numberPart + character

    yourID = namePart + str (numberPart)

    return (yourID)

# =====> Add a procedure, with no parameters, to display a
#        welcome message for the user
print("welcome")
# -----------------------------------------------------------------
# Main program
# -----------------------------------------------------------------
# =====> Call the welcome procedure before taking input from the user

# Get last name and first name from the user
lastName = input ("Enter your last name: ")
firstName = input ("Enter your first name: ")

# =====> Convert last name and first name to lowercase after they
#        are inputted by the user
lastName.lower()
firstName.lower()
# Get date of birth from the user
dob = input ("Enter your date of birth (ddmmyyyy): ")

# =====> Check that only the digits 0 to 9 appear in the date of birth
if (dob >= 1) and (dob <= 9):
    print("valid")
else:
    print("invalid")
# =====> Call the makeID() function, if the date of birth is valid
myID = makeID (lastName, firstName, dob)
print (myID)

# =====> Tell the user, if the date of birth is invalid
print("invalid")
```

This response was awarded six marks. It demonstrates following through the changing of input parameters, although the names could be more informative.

Q05 Example 3

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
lastName = ""
firstName = ""
dob = ""
myID = ""

# ------------------------------------------------------------------
# Subprograms
# ------------------------------------------------------------------
# =====> Change the names of the local variables to distinguish them
#        from the global variables with the same name
def makeID (sirName, theName, birthDate):
    namePart = ""
    numberPart = 0

    namePart = sirName + theName[0]  # Letter part

    # =====> Correct the logic error caused by using the int() function
    #        in the number part calculation rather than using a function
    #        that returns the ASCII value of the character
    for character in birthDate:
        numberPart = numberPart + ord(character)

    yourID = namePart + str (numberPart)

    return (yourID)

# =====> Add a procedure, with no parameters, to display a
#        welcome message for the user
def welcomeMessage():
    print("welcome!")
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# =====> Call the welcome procedure before taking input from the user
welcomeMessage()
# Get last name and first name from the user
lastName = input ("Enter your last name: ")
firstName = input ("Enter your first name: ")

# =====> Convert last name and first name to lowercase after they
#        are inputted by the user
lastName = lastName.lower()
firstName = firstName.lower()
# Get date of birth from the user
dob = input ("Enter your date of birth (ddmmyyyy): ")

# =====> Check that only the digits 0 to 9 appear in the date of birth
if dob.isdigit:
# =====> Call the makeID() function, if the date of birth is valid
    myID = makeID (lastName, firstName, dob)
    print (myID)
# =====> Tell the user, if the date of birth is invalid
else:
    print("invalid")
```

This response was awarded 13 marks. It demonstrates how to create and use the required procedure to welcome the user.

Q06

In this question, students are required to create a programmed solution to a problem. There is no scaffolding provided in this question.

There was a range of creative solutions which demonstrated the main requirements of the problem. Decomposition was evident by design in responses that made a reasonable attempt to answer the question.

A common approach was to iterate across the userTable with a for loop. Most of these solutions did not go on to implement an early exit from the loop when the matching username and passcode were found. That impacted the marks available in the Design LBMS.

In addition, it was common for responses to omit the validation for blank input, as required in the question.

Q06 Example 1

```
# ----------------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------------
userTable = [["LArmstrong3", "RedChair"],
             ["SBarrett7", "PurpleDesk"],
             ["EChisholm4", "YellowStool"],
             ["VDunn1", "OrangeFuton"],
             ["DElms5", "GreenCouch"],
             ["EFirsova13", "PinkMattress"],
             ["JGolland6", "GreenTable"],
             ["FHartley13", "BrownMirror"],
             ["DJohnstone12", "GoldBed"],
             ["GKirkhope8", "WhiteNights"],
             ["LLemon8", "BeigeDresser"],
             ["HMacCunn6", "GreyOttoman"],
             ["PNewland10", "BlackWardrobe"],
             ["AOldham5", "OrangeFuton"],
             ["JPook8", "YellowStool"]]

# =====> Write your code here
userName = "" # initialising the user variables
userPassword = ""

isFound = False
index = 0
# ----------------------------------------------------------------------
# Main program
# ----------------------------------------------------------------------

# =====> Write your code here
userName = input("Please enter your username: ")# prompts for and obtai
userPassword = input("Please enter your password: ")# prompts for and o


while index < len(userTable) and (not isFound): # the array is searched
    if userTable[index][0] == userName:  # checks if the username match
        if userTable[index][1] == userPassword:  # checks if the passwo
            isFound = True
            print("The username was found, and the matching password is
        else:   # if the username is matches, but the password is not
            isFound = True
            print("The username was found, however the password does no
    else:
        index = index + 1 # moves on the search to the next record if t


if not isFound: # if the username is not found in the array, that is to
    print("The username was not found.")
```

This example was awarded 12 marks. It demonstrates an effective way to use linear search on a two-dimensional array. It has an early exit when either the username and password match are found or when the username without a password match is found. However, it has not implemented the check for blank inputs.

Q06 Example 2

```
# ----------------------------------------------------------------
# Global variables
# ----------------------------------------------------------------
userTable = [["LArmstrong3", "RedChair"],
             ["SBarrett7", "PurpleDesk"],
             ["EChisholm4", "YellowStool"],
             ["VDunn1", "OrangeFuton"],
             ["DElms5", "GreenCouch"],
             ["EFirsova13", "PinkMattress"],
             ["JGolland6", "GreenTable"],
             ["FHartley13", "BrownMirror"],
             ["DJohnstone12", "GoldBed"],
             ["GKirkhope8", "WhiteNights"],
             ["LLemon8", "BeigeDresser"],
             ["HMacCunn6", "GreyOttoman"],
             ["PNewland10", "BlackWardrobe"],
             ["AOldham5", "OrangeFuton"],
             ["JPook8", "YellowStool"]]

# =====> Write your code here

# Initialisation and declaration
Username = ""
Password = ""
curName = ""
curPass = ""
output = ""
accountNum = 0

#
```

```
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# =====> Write your code here

# INPUTS
Username = input ("Enter your username:  ")
while Username == "":   # Authentication to ensure the inputted
username is not blank
    Username = input ("Enter your username:  ")

Password = input ("Enter your password:  ")
while Password == "":   # Authentication to ensure the inputted
password is not blank
    Password = input ("Enter your password:  ")

# PROCESSES
for accountNum in range(len(userTable)):   # Iterating though the
number of arrays in the 2D array 'userTable'

    curName = userTable[accountNum][0]
    curPass = userTable[accountNum][1]

    if (curName == Username) and (curPass == Password):  # Checking
    if the password and username match the ones the user entered to
    see if they are in the list
        output = "Welcome " + Username + ". You may access the
        database."
        break

    elif (curName == Username):  # Checking if the username is
    correct but the password isn't
        output = "Your password is incorrect. Try again"
        break

    elif accountNum == (len(userTable)-1):  # checking to see if the
    end of the list has been reached without finding the user's
    details (so they aren't in the list)
        output = "The username '" + Username + "' does not appear to
        be in our database."

# OUTPUTS
print(output)  # displaying the correct message
```

This response was awarded 14 marks. It demonstrates another way to use a linear search on a two-dimensional array. The solution, in this case, uses a for loop. In order to meet the efficiency requirements of an early exit, a break statement has been incorporated. This does work to exit the loop early. However, the use of a break could make the code more challenging to debug. A while loop, with a Boolean flag, leads to a code block with only one exit test. This response also demonstrates one method for checking that the inputs are not blank. This code contains excessive comments.

Q06 Example 3

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
userTable = [["LArmstrong3", "RedChair"],
             ["SBarrett7", "PurpleDesk"],
             ["EChisholm4", "YellowStool"],
             ["VDunn1", "OrangeFuton"],
             ["DElms5", "GreenCouch"],
             ["EFirsova13", "PinkMattress"],
             ["JGolland6", "GreenTable"],
             ["FHartley13", "BrownMirror"],
             ["DJohnstone12", "GoldBed"],
             ["GKirkhope8", "WhiteNights"],
             ["LLemon8", "BeigeDresser"],
             ["HMacCunn6", "GreyOttoman"],
             ["PNewland10", "BlackWardrobe"],
             ["AOldham5", "OrangeFuton"],
             ["JPook8", "YellowStool"]]

# =====> Write your code here
userName = str(input("enter your username"))
user_password = str(input("enter your password"))
found = False
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# =====> Write your code here
if (userName == " ")and(user_password == " "):
    print("username and password cant be blank")
else:
    while found == False:
        for item in userTable :
            if (userName==item[0]) and (user_password == item[1]):
                found = True
                print("both the username and password are valid")

        if (userName==item[0]) and (user_password != item[1]):
            print("your username was found but the password is invalid")
        elif (userName!=item[0]) and (user_password != item[1]):
            print("username wasn't found try again")

```

This response was awarded nine marks. It demonstrates another method for validating input. It also uses a for loop to access the first dimension in userTable and then uses an additional index to access the second dimension.

Summary

Students should:

- Attempt every question in the paper
- Follow the instructions in the paper and do not rewrite the supplied code
- Remove all the syntax errors from code so that it will translate
- Execute and test code with the data supplied in the question
- Consider the design of the overall solution, not just the single lines of code
- Use effective, but not excessive, commenting and white space to make the program logic clear