Pearson Edexcel International Advanced Level

Sample assessment material

Time 3 hours

Paper reference **WCP02/01**

Computer Science

International Advanced Subsidiary Level
Unit 2: Practical Programming and Problem-solving

You must have:
- a computer workstation with appropriate programming language code editing software and tools, including an IDE that you are familiar with that shows line numbers
- a 'STUDENT CODING' folder containing code and data files
- digital copy of the Programming Language Subset (PLS) document.

Instructions

- Answer **all** questions on your computer.
- Save the new or amended code in the 'COMPLETED CODING' folder using the name given in the question.
- Do **not** overwrite the original code and data files provided to you.
- You must **not** use the internet at any time during the examination.

Information

- The total mark for this paper is 80.
- The marks for **each** question are shown in brackets
  – use this as a guide as to how much time to spend on each question.
- The 'STUDENT CODING' folder in your user area includes all the code and data files you need.

Advice

- Read each question carefully before you start to answer it.
- Save your work regularly.
- Check your answers if you have time at the end.
- Try to answer every question.

S85795A

1/1/1/1/1/1/1

S 8 5 7 9 5 A

Pearson

Answer ALL questions.

Suggested time: 20 minutes

1 A program is intended to implement and test a stack.

The program runs three logical tests. The output from the tests helps to debug the errors.

Open file **Q01.py**

Amend the code to:

- fix the syntax error on original line 10
```
my_stack = [None, None, None, None None, None, None,
None, None, None]
```
- fix the syntax error on original line 33
```
def is_empty()
```
- fix the syntax error on original line 45
```
status =! True
```
- fix the syntax error on original line 94
```
push ("Banana)
```
- fix the TypeError on original line 20
```
for index in range(STACK_SIZE * 0.5)
```
- fix the TypeError on original line 29 by using type conversion
```
out_string = out_string + item + " "
```
- fix the UnboundLocalError on original line 62
```
top_stack
```
- fix the NameError on original line 91
```
clean_Stack()
```
- fix the TypeError on original line 95
```
show_stack("Apple")
```
- fix the logic error on original line 23 that reports the wrong index for the top of the stack
```
top_stack = -9
```
- fix the logic error on original line 65 that pops None, in error, when the stack is empty
```
if(is_empty()):
```
- fix the logic error on original line 54 that skips stack spaces when pushing
```
top_stack = top_stack + 2
```

Execute the code to test its functionality.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q01FINISHED.py**

(Total for Question 1 = 12 marks)

Suggested time: 20 minutes

2 A palindrome is a string that reads the same forwards and backwards.

A program is intended to determine if a string is a palindrome.

The program loops until the user enters an empty string.

The algorithm in the program must convert uppercase letters to lowercase letters and remove punctuation and symbols before processing.

The test data is for a fully-functional program.

Test data

Input	Output
civic	Original: civic Prepared: civic is a palindrome
CiVic	Original: CiVic Prepared: civic is a palindrome
No melon, no lemon	Original: No melon, no lemon Prepared: nomelonnolemon is a palindrome
Coffee	Original: Coffee Prepared: coffee is not a palindrome
34$543	Original: 34$543 Prepared: 34543 is a palindrome

The program does not work.

Some lines are commented out.

Some lines are not in the correct location.

All lines are correctly indented.

Open file **Q02.py**

Amend the code to produce the correct output.

You will need to:

- rearrange some lines
- choose the correct line of code. Make a choice by removing the # at the beginning of the lines you choose to execute.

Execute the code to test its functionality.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q02FINISHED.py**

(Total for Question 2 = 15 marks)

Suggested time: 40 minutes

3 A company sells products to customers. The products are supplied by makers. Makers specialise in different crafts. Specialisms include woodworking and 3D printing.

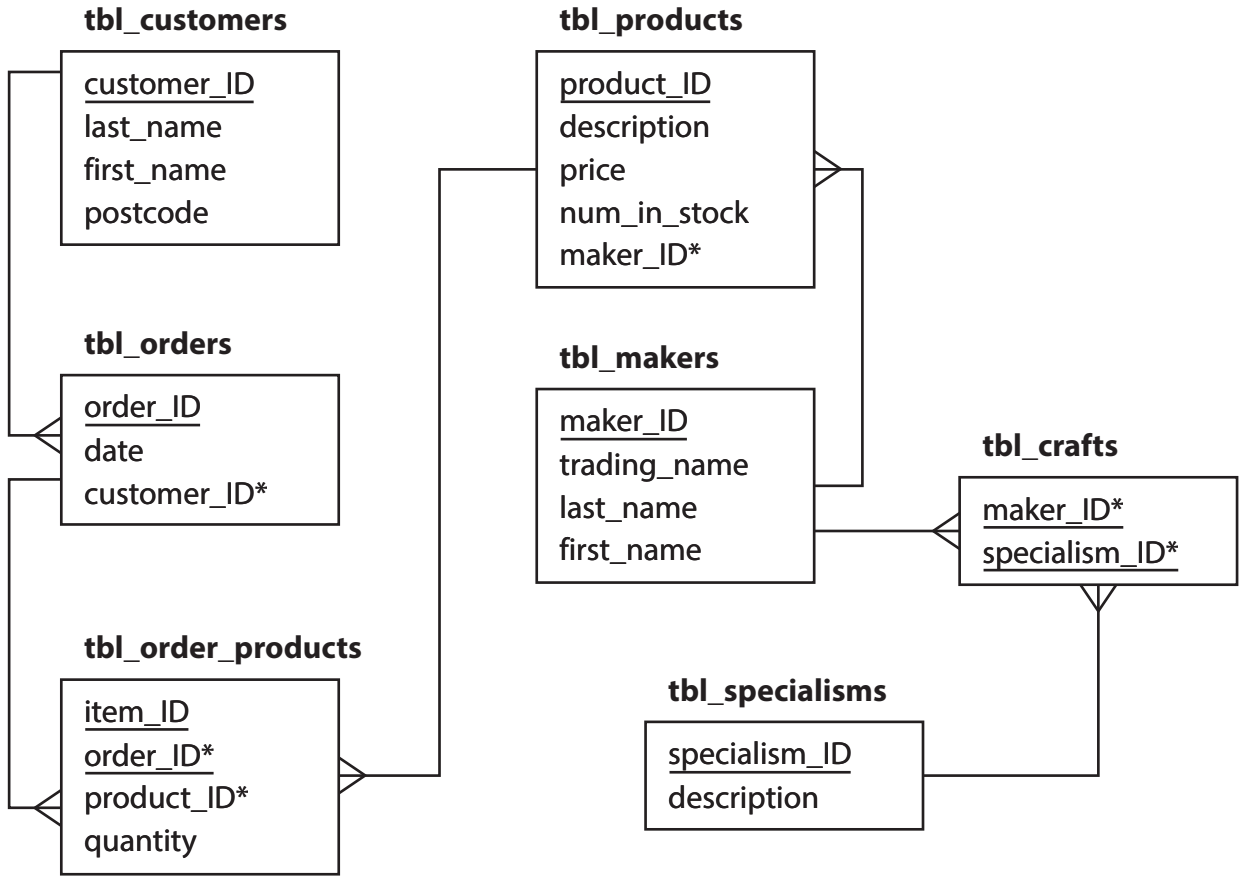
The entity-relationship diagram is for the existing CustomCrafts database.

Table names are emboldened.

Primary keys are underlined.

Foreign keys are indicated by *

Entity-relationship



The CustomCrafts database is supplied in the file CustomCrafts.db

An employee table is defined by:

tbl_employees (employee_ID, address_1, address_2, postcode)

- employee_ID is an integer
- address_1, address_2, and postcode are text.

The output is from a fully-functional program.

Output

```
There are 12 different products.
Product 9 has sold 5 times.
The customers who have postcodes that start with 608087 are:
(327675, 'Evans', 'Erin', '6080873745')
(534980, 'Adams', 'Aspen', '6080878902')
(571526, 'Franks', 'Faye', '6080872932')
The trading names of the makers who specialise in Woodworking are:
('Natural Materials',)
('Delicate Designs',)
```

Open file **Q03.py**

The code has not implemented any error handling because no error handling is required.

Amend the code to:

- determine how many different products are held in the database
- determine the quantity of product ID 9 that has been sold
- determine all the customers who have postcodes that start with 608087
- add the employee table to the database
- determine the trading names of the makers who specialise in woodworking.

The program must:

- output messages that are fit for purpose
- call the supplied subprogram, `show_result`, to display results that have multiple items.

Use the library and variables provided.

Execute the code to test its functionality.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q03FINISHED.py**

(Total for Question 3 = 15 marks)

Suggested time: 40 minutes

4 A program is required to sort lists of integers into ascending order.

There are two lists in the program code file. The `little_list` is filled with integers. The `big_list` is empty and must be filled with random integers.

An in-place insertion sort algorithm is required to do the sorting.

The table is an example of using the in-place insertion sort algorithm to sort the five integers in `little_list`.

Table

`little_list`					**Description**
44	22	11	33	55	Original
44		11	33	55	22 is copied into a target variable
	44	11	33	55	44 shifts right
22	44	11	33	55	22 is copied into the gap
22	44		33	55	11 is copied into a target variable
	22	44	33	55	44 shifts right; 22 shifts right
11	22	44	33	55	11 is copied into the gap
11	22	44		55	33 is copied into a target variable
11	22		44	55	44 shifts right
11	22	33	44	55	33 is copied into the gap
11	22	33	44		55 is copied into a target variable
11	22	33	44		No shifts required
11	22	33	44	55	55 is copied into the gap
11	22	33	44	55	End of list reached

Open file **Q04.py**

Amend the code to:

- import a required library
- complete the insertion sort procedure that takes a list as a parameter and sorts the list in place. No return value is required.
- manipulate `little_list` to:
  - display the unsorted list (no formatting required)
  - sort the list by calling the insertion sort procedure
  - display the sorted list (no formatting required)
- complete the `display_list_matrix()` procedure to display the list in five rows of 20 integers each
- manipulate `big_list` to:
  - fill the list with 100 random integers between 0 and 100
  - display the unsorted list by calling `display_list_matrix()`
  - sort the list using the insertion sort procedure
  - display the sorted list by calling `display_list_matrix()`.

Use the constant and variables provided.

Execute the code to test its functionality.

Do **not** use a built-in or library sorting subprogram.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q04FINISHED.py**

(Total for Question 4 = 18 marks)

Suggested time: 60 minutes

5 An air quality monitoring system has a network of sensors placed across a city.

Sensors take readings of the amount of nitrogen dioxide (NO_2) in the air.

Sensors take one reading every 10 minutes, 24 hours a day, seven days a week.

Readings are stored as $\mu g/m^3$ (micrograms per cubic metre of air).

Readings are reported to one decimal place.

The minimum valid sensor reading is 10.0

The maximum valid sensor reading is 51.9

A sensor is faulty when it reports the first invalid reading.

Engineers will use a program to identify faulty sensors before the sensors fail completely.

Description of the dataset

A sample of the full dataset is provided in the program code file.

Each row of the dataset consists of:

- sensor ID (integer)
- latitude (float)
- longitude (float)
- location (string)
- timestamp (integer, UTC format)
- reading (float).

The output is from a fully-functional program.

The output shows the first time a sensor gives an invalid reading.

Output

```
There are 4 faulty sensors

ID           Location                 Date/Time          Reading
-------------------------------------------------------------------
16       Hyde Park Street       2024-11-05 02:50:00      -33.7
18          Abbey Wood          2024-11-01 00:10:00       55.7
47        Greenwich Park        2024-11-05 06:30:00        0.0
125         Green Park          2024-11-03 04:50:00      122.8
```

Formatting requirements describes the output layout.

Formatting requirements

ID	Location	Date/Time	Reading
Left aligned	Centre aligned	Centre aligned	Right aligned
5 characters	22 characters	20 characters	7 characters

Open file **Q05.py**

Write the code to:

- use the dataset to produce the output shown in output
- format the output to meet the requirements shown in formatting requirements
- display the output to the screen
- write the output to a text file.

The program must:

- work for any number of sensors in the dataset
- include at least one user-defined subprogram that takes parameters.

Use best practice techniques to:

- design effective and efficient code
- write clean code
- ensure the solution functions as intended.

Use the constants and variables provided.

Execute the code to test its functionality.

Do **not** change the intended functionality of the provided code.

Do **not** add any additional functionality.

Save your amended code as **Q05FINISHED.py**

(Total for Question 5 = 20 marks)

TOTAL FOR PAPER = 80 MARKS