ANGLO-CHINESE JUNIOR COLLEGE
JC2 PRELIMINARY EXAMINATION

Higher 2

COMPUTING 9569/02

Paper 2 (Lab-based) **18 August 2020**

3 hours

Additional Materials: Electronic version of `TASK2TOUR.txt` data file
Electronic version of `TASK2NOTOUR.txt` data file
Electronic version of `INVENTORY.txt` data file
Electronic version of `INVENTORY_SERIAL.txt` data file
Electronic version of `SALES.txt` data file
Electronic version of `TECH_SUPPORT.txt` data file
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

Note that up to **6** marks out of 100 will be awarded for the use of common coding standards for programming style.

The number of marks is given in brackets [ ] at the end of each question or part question.
The total number of marks for this paper is 100.

Instruction to candidates:

Your program code and output for each of Task 1 and 2 should be downloaded in a single .ipynb file. For example, your program code and output for Task 1 should be downloaded as
`TASK1_<your name>_<centre number>_<index number>.ipynb`

1 A programmer is writing a program to implement a role-playing computer game using Object-Oriented Programming (OOP).

The players have to collect food items. A food item has the following attributes:

- `name  : STRING`
- `value : INTEGER`

and the following methods:

- `get_name()`
- `get_value()`

A player takes on the role of a person. A person has the following attributes:

- `name     : STRING`
- `health   : INTEGER` which is initialised at a value of `100`
- `strength : INTEGER` which is initialised at a value of `100`

and the following methods:

- `get_name()`
- `get_health()`
- `get_strength()`
- `eat(food)` adds the value of the food to the strength. The code should display the player's new strength.
- `attack(opponent)`

For the `attack` method, `opponent` is another person.

- A random integer `r` between `1` and `10` (inclusive) is generated.
- If the player's `strength` is less than `r`, then the player does not have enough strength to attack and there is no change to `opponent`'s `health`.
- If the player's `strength` is at least `r`, then the attack is successful and `opponent`'s `health` is decreased by `r`.
  - If `opponent`'s `health` is now negative, then `opponent` has been defeated.
- The player's `strength` is decreased by `r`.

There are two subclasses of the `Person` class – `Healer` and the `Warrior`.

`Healer` has one additional method:

- `heal(patient)`

For the `heal` method, `patient` is another person.

- A random integer `r` between `1` and 10 (inclusive) is generated.
- If the player's `strength` is less than `r`, then the player does not have enough strength to heal and there is no change to `patient`'s `health`.
- If the player's `strength` is at least `r`, then the healing is successful and `patient`'s `health` is increased by `r`, up to a maximum of `100`.

`Warrior`'s `attack` method is twice as effective, meaning that if the player has enough strength to attack, `opponent`'s `health` is decreased by `2*r`, while the player's `strength` is decreased by `r`.

Task 1.1

Write program code to define the class `Food`. [3]

Task 1.2

Write program code to define the class `Person`.

The code should display appropriate messages about the outcome of `attack`, including the new value of `opponent`'s `health`. [10]

Task 1.3

Use appropriate inheritance to write program code to define the class `Healer`.

The code should display appropriate messages about the outcome of `heal`, including the new value of `patient`'s `health`. [4]

Task 1.4

Use appropriate inheritance and polymorphism to write program code to define the class `Warrior`. [2]

Test your code with the following steps in order:

- Create a `Food` item with `name 'Cheese'` and `value 10`.
- Create a `Warrior` with `name 'Sam'`.
- Create a `Healer` with `name 'Alex'`.
- Create a `Person` with `name 'Jan'`.
- `'Jan'` attacks `'Sam'`.
- `'Sam'` attacks `'Jan'`.
- `'Alex'` heals `'Jan'`.
- `'Sam'` eats `'Cheese'`.

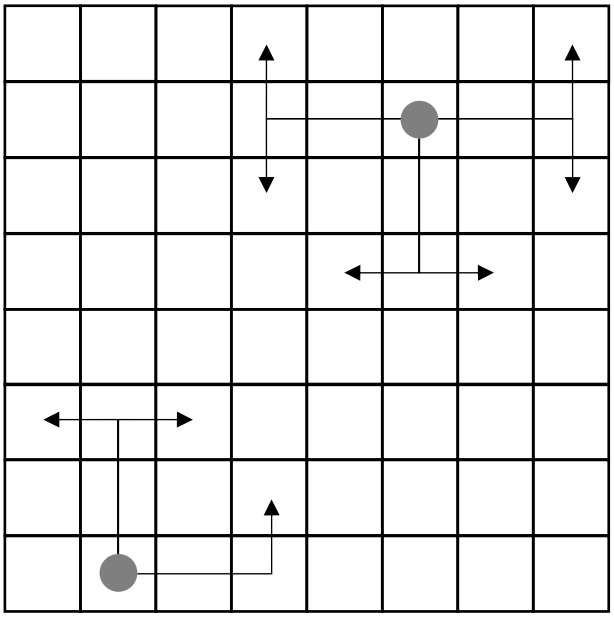
Download your program code and output for Task 1 as
`TASK1_<your name>_<centre number>_<index number>.ipynb` [3]

2 An `n × n` chessboard consists of an `n × n` grid of small squares. For this task, the squares are numbered using a coordinate system as a tuple of two integers. The first integer is the column number, starting from `1` at the left, and the second integer is the row number, starting from `1` at the bottom.

A chess knight is a piece that occupies a square, and can then move to another square according to one of the following rules:

- moving two squares horizontally and then one square vertically in either direction, or
- moving two squares vertically and then one square horizontally in either direction.

See the diagram below for examples.



The knight at `(2,1)` can move to `(1,3)`, `(3,3)` or `(4,2)`.
The knight at `(6,7)` can move to `(4,6)`, `(4,8)`, `(5,5)`, `(7,5)`, `(8,6)` or `(8,8)`.

Only the starting and ending squares of the knight's move are counted as being visited by the knight, and not the squares that the knight passes over while moving.

A knight's tour is a sequence of moves that a chess knight makes on a chessboard, so that it visits every square of the chessboard exactly once. It does not need to return to its starting square.

Task 2.1

For a given value of `n` and a list of squares, write program code to determine if the list is a knight's tour of an `n × n` chessboard.

Test your code using the values `n=7` and the list of squares given in the files:

- `TASK2TOUR.txt`, which is a valid tour;
- `TASK2NOTOUR.txt`, which is not a valid tour. [10]

An algorithm to generate a knight's tour needs to keep track of the squares already visited, so that the knight does not visit them a second time during the tour.

Task 2.2

Suppose the knight is currently on square `square`, and a list of squares already visited by the knight is given in `lis`.

Write a function `available(square,lis)` that returns a list of squares which the knight can visit on its next move from `square`, and are not currently in `lis`. These are the squares available to the knight as it tries to complete the tour.

The output should be given in lexicographic order, that is, with the column numbers in ascending order, and, among the squares with the same column number, with the row numbers in ascending order. [6]

Task 2.3

The knight starts at `(1,1)`, the bottom left square, of an 8 × 8 chessboard.

From each square, the knight moves to the available square from which it has the smallest number of available squares after that. If there is a tie, the square which comes first in lexicographic order is chosen.

Write program code to generate the knight's tour as a list of squares in the order they are visited.

Download your program code and output for Task 2 as
`TASK2_<your name>_<centre number>_<index number>.ipynb` [9]

3 A text file `INVENTORY.txt` contains the inventory data for a certain electronics store. Each line in the file contains tab-delimited data that shows the product name, product type, purchase price, selling price and quantity available.

Each line is in the format

```
Name\tType\tPurchase_Price\tSelling_Price\tQuantity
```

where `\t` represents the tab character.

Task 3.1

Write program code to:

- Read the inventory data from the text file;
- Find the average selling price of products belonging to the `Laptop` product type and display this value;
- Count the number of products in each product type and store it in appropriate data structure called `TypeCount`;
- Display the product type with the greatest number of products. If there is a tie, display all of the product types with the greatest number of products.

Download your program code and output for Task 3.1 as
`TASK3_1_<your name>_<centre number>_<index number>.ipynb` [6]

Task 3.2

The profit margin of each product can be calculated by the following equation:

Profit margin = selling price – purchase price.

Write program code to:

- Calculate and display the total profit the store could make if all products are sold;
- Sort the inventory data using a Merge sort algorithm in descending order of profit margin;
- Display the sorted inventory data in the format given below.

```
Product           Product Type      Profit Margin
ThinkingPad 14    Computer          300
Bapple 8          Phone             450
```

Download your program code and output for Task 3.2 as
`TASK3_2_<your name>_<centre number>_<index number>.ipynb` [9]

Task 3.3

A store manager decided to make some changes to `INVENTORY.txt` and saved it as `INVENTORY_SERIAL.txt`. Each line in the updated file contains tab-delimited data that shows the serial number, product name, product type, purchase price, selling price and quantity available.

Each line is in the format:

```
Serial_No\tName\tType\tPurchase_Price\tSelling_Price\tQuantity
```

where `\t` represents the tab character.

Write program code to insert the data from `INVENTORY_SERIAL.txt` into a NoSQL database `OUTLETS` under the collection `GEM`.

Download your program code for Task 3.3 as
`TASK3_3_<your name>_<centre number>_<index number>.py` [5]

Task 3.4

The database administrator wants to validate that the store manager did not make any errors when he edited the text file. Write program code to check that the database conforms to the below specifications:

- `Serial_No` consists of one digit followed by two letters, followed by one digit (e.g. `1AB7`);
- `Name` consists of only letters, digits and spaces;
- `Quantity` is a positive integer.

Any document that has an error should be removed from the database. You may assume data fields not specified above are error free. Display the documents that were removed.

Download your program code for Task 3.4 as
`TASK3_4_<your name>_<centre number>_<index number>.py` [5]

4 A company keeps records of the employees working for it. The following are the information stored in the company's database:

- `Employee_name` – name of the employee
- `Employee_ID` – unique ID number allocated to each employee
- `Job_type` – type of job the employee is employed for (`'Sales'` or `'Tech_support'`)
- `Date_of_employment` – date the employee joined the company
- `Service_status` – whether the employee is still in service (`'True'` means the employee is still in the company, `'False'` means the employee has left the company)

For sales employees, the following extra information is recorded:

- `Total_sales` – the amount of sales in dollars made by the employee

For tech support employees, the following extra information is recorded:

- `Bugs_resolved` – the total number of bugs the employee has resolved

The database is expected to be normalised and stored in three different tables:

```
Employee
Sales
Tech_support
```

Task 4.1

Create an SQL file called `TASK4_1_<your name>_<centre number>_<index number>.sql` to show the SQL code to create the database `records.db` with the three tables. Primary keys and foreign keys should be defined where appropriate.

Save your SQL code as
`TASK4_1_<your name>_<centre number>_<index number>.sql` [5]

Task 4.2
The files `SALES.txt` and `TECH_SUPPORT.txt` contain information regarding the sales and tech support employees respectively. The information should be inserted into the database.

For `SALES.txt`, information is given in the following order:
`Employee_ID,Employee_name,Date_of_Employment,Service_status,Total_Sales`

For `TECH_SUPPORT.txt`, information is given in the following order:
`Employee_ID,Employee_name,Date_of_Employment,Service_status,Bugs_resolved`

Write a python program to insert all information from the two files into the `records` database, `records.db`. Run the program.

Save your program code as
`TASK4_2_<your name>_<centre number>_<index number>.py` [5]

Task 4.3

The company wants to filter the employees by `Service_status` and display the results in a web browser.

Write a Python program and the necessary files to create a web application that:

- receives a `Service_status` string from a HTML form, then
- creates and returns a HTML document that enables the web browser to display either
    - an ordered list of employees that are still in service, or
    - an ordered list of employees that are no longer in service,

  depending on the `Service_status` string entered by the user.

The list should be sorted in alphabetical order.

Save your Python program as
`TASK4_3_<your name>_<centre number>_<index number>.py`
with any additional files / sub-folders as needed in a folder named
`TASK4_3_<your name>_<centre number>_<index number>`

Run the web application. Save the output of the program when `'TRUE'` is entered as the `Service_status` as `TASK4_3_<your name>_<centre number>_<index number>.html`. [12]