NATIONAL JUNIOR COLLEGE
Mathematics Department

General Certificate of Education Advanced Level
Higher 2

COMPUTING 9569/02

Paper 2 Lab-based **19 August 2020**

3 hours

Additional Materials: Electronic version of `WORDS.TXT` data file
Electronic version of `RESTAURANTS.JSON` data file
Electronic version of `PATIENTS.CSV` data file
Electronic version of `LOCATIONS.CSV` 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 **2** marks out of 100 will be awarded for the use of common coding standards for programming style.

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

This document consists of **14** printed pages and 2 blank pages.

Instructions to candidates:

- Copy the folder from the thumb drive to the PC's desktop and rename the folder on the desktop to `<your name>_<NRIC number>`.(For example, TanKengHan_T0123456A).
- All the resource files are found in the folder and you should work on the folder in the desktop.
- Your program code and output for each of Task 1 to 3 should be saved in a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as `Task_1_<your name>_<NRIC number>.ipynb`.
- You should have a total of **three** `.ipynb` files to submit at the end of the paper.
- The answer files for Task 4 should be saved in a **single folder** `Task 4_<your name>_<NRIC number>`
- At the end of the exam, copy the working folder on your desktop to the thumb drive.

---

1 Hangman is a word guessing game where a random word is selected and a player is given a number of attempts to guess the word by entering a letter that he/she thinks is part of the word. The player wins if he/she is able to enter all the letters that makes up the word within the number of attempts he/she is given and loses if he/she cannot enter all the letters to form the word.
When the game starts, the letters in the word will be hidden and the player will start to enter letters that he/she thinks is part of the word. If a letter in the word is entered correctly, the letter and its corresponding positing in the word will be revealed. An example of the game play is shown below (a dot "." is used to indicate the position of a letter in the word that has not been guessed correctly):

Player loses:

```
Choose the length of word to guess:[1 to 11]4
You have 8 attempts. Good Luck!
....
[8]Guess letter:h
...h
[7]Guess letter:e
...h
[6]Guess letter:c
..ch
[5]Guess letter:r
..ch
[4]Guess letter:b
..ch
[3]Guess letter:a
..ch
[2]Guess letter:n
.nch
[1]Guess letter:t
.nch
LOSE..The word is inch
```

Player wins:

```
Choose the length of word to guess:[1 to 11]4
You have 8 attempts. Good Luck!
....
[8]Guess letter:a
....
[7]Guess letter:e
.e..
[6]Guess letter:b
.e..
[5]Guess letter:l
.el.
[4]Guess letter:p
.elp
[3]Guess letter:h
help
WIN
```

For each sub-task, add a comment statement, at the beginning of the code using the hash symbol "#", to indicate the sub-task the program code belongs to, for example:

```
In [1]: #Task 1.1
        Program code

        Output:
```

The file provided is `WORDS.TXT`

Task 1.1

The file `WORDS.TXT` contains a list of words commonly found in an English dictionary. It contains words of varying length. You are to write a Python function named **`getSecret()`** to return the secret word for the player to guess. The function must include the following functionality:

(a) Read the contents of the file `WORDS.TXT` into a suitable data structure to facilitate the lookup of a word based on the length of the word.

(b) Determine the shortest word and the longest word in the file.

(c) Prompt the user to enter the length of the word to be guessed. The prompt must indicate the range of word length that the user can choose from based on the shortest and longest word found in Task 1.1(b). See example in the game play above.
You **do not need** to write code to handle invalid input.

(d) Look for a word from the data structure created in Task 1.1(a) that have the word length given by the user in Task 1.1(c). If there is more than one word with the word length, randomly choose one. For example, if the length of word chosen is 4. The code should randomly select one of the 4-letter word from the data structure created in Task 1.1(a).

(e) Return the word from Task 1.1(d) for the player to guess. [6]

Task 1.2

Write the code for a player to guess the secret word returned by the `getSecret()` function defined in Task 1.1. The number of guesses that a player can make is 2 times the length of the secret word to be guessed. For example, if the length of the secret word is 4, the number of attempts is 8.

The input prompt and output should be similar to the two game play examples shown above. There should be a number indicating the number of attempts left.

You **do not need** to write code to handle invalid input. [7]

Task 1.3
Modify the Python code in Task 1.2 to reveal some of the letters in the secret word to be guessed before the player starts guessing the rest of the letters in the word.

The number of letters to reveal is 0.2 times the number of letters in the secret word (round up to the nearest integer). The position of the letter/s to reveal is chosen randomly. For example, if the secret word is "help" , the program will reveal one letter from the secret word $[0.2 \times 4 = 1(round\ up)]$ as in the following modified game play:

```
Choose the length of word to guess:[1 to 11]4
You have 8 attempts. Good Luck!
.e..
[8]Guess letter:a
.e..
[7]Guess letter:l
.el.
[6]Guess letter:h
hel.
[5]Guess letter:p
help
WIN
```

[4]

Convert the code in Task 1.2/1.3 to be played over the network using a client/server architecture. The code for the server component should generate the secret word for the client component to guess. The client component should allow the player to guess the word by entering the letters.

Task 1.4
Write the code for the server component. You may copy the code from the previous tasks and modify them. You have to write the code to randomly select a valid word length for the player to guess instead of prompting the player for the word length to guess. [5]

Task 1.5
Write the code for the client component for the player to make the guesses. [3]

Save your Jupyter Notebook file as
`Task_1_<your name>_<NRIC number>.ipynb.`

2 A rating system for restaurants and other entertainment outlets is to be developed that allows users to view the ratings and access the details of various different entertainment outlets.

The raw data for restaurants' ratings is consolidated and stored in a JSON formatted file named `RESTAURANTS.JSON`.

The data from the file is to be processed and stored as a **collection** in a MongoDB database.

For each sub-task, add a comment statement, at the beginning of the code using the hash symbol "#", to indicate the sub-task the program code belongs to, for example:

```
In [1]: #Task 2.1
        Program code

        Output:
```

The file provided is `RESTAURANTS.JSON`

Task 2.1
The file `RESTAURANTS.JSON` contains duplicate data. Restaurants having the same name are considered to be duplicates. There are some restaurants that do not have a numerical rating value as they are not yet rated.

Write Python code to process the file:
- Load the JSON file into a data structure for processing.
- Remove the duplicated data.
- Restaurants that do not have a numerical rating value yet will have their ratings set to 0.
- The processed data structure should be named **`restaurants`**. [3]

Task 2.2
Write the Python code for a sort function named **`mergeSort`** that uses the merge sort algorithm.
The function should take an input argument `restaurants` obtained in Task 2.1 and sort the restaurants in descending order of their numerical rating values. That means a restaurant with a higher rating should be ordered before a restaurant with a lower rating. The function will return a sorted list of restaurants. [4]

Task 2.3
Write the Python code to:
- Sort the restaurants using the `mergeSort` function written in Task 2.2.
- Print the first 10 restaurants in the sorted list.

[1]

Task 2.4
Write Python code to
- create a MongoDB database named **`Entertainment`**.
- Insert the sorted list of restaurants into a MongoDB collection in the database. [2]

Save your Jupyter Notebook file as
`Task_2_<your name>_<NRIC number>.ipynb`.

3 A Visitor Management System for a hospital is to be developed to manage the visitors visiting patients in the hospital. A vital component of the system is the queuing system that is used to control and manage the number of visitors that are allowed to visit a patient at any one time.

The queuing system is to be developed using an OOP approach with the following requirements:

- A 1-D array is to be used to model the beds in the hospital. The size of the array is the number of beds in the hospital. Each element in the array will contain a Bed object.
- Each bed in the hospital is uniquely identified by its floor, ward and bed number. The floor number, ward number and bed number start at 1. The number of floors, number of wards per floor and the number of beds per ward is given during the initialisation of the Hospital object. Every floor has the same number of wards and every ward has the same number of beds.
- Each Bed object has a Queue object that is use to manage the visitors waiting to visit the patient in the hospital. The Queue object is to be implemented as a **dynamically-sized** list of Visitor objects. A Visitor object models the visitor waiting to visit a patient.
- The **highest** floor in the hospital is reserved for intensive care patients and all the beds have a restricted visiting hour from 1700 to 1900 inclusive, every day. **All the rest** of the beds have visiting hours between 1200 to 2000 inclusive.
- All visitors making a visit will need to provide his/her name, contact number the patient's floor, ward and bed number he/she is visiting, the date and time of the visitation and get his/her temperature taken. If the visit is within the valid visitation hours, the visitor will be placed in the queue for the room, otherwise the visit will be rejected.
- The system will automatically inform the visitor via a SMS (short message service) message when the visitor is allowed to visit the patient and the visitor's record will be dequeued from the system. (NOTE: You **do not need** to implement this feature)

The following class diagrams are to be used in your implementation, you may **include** any **additional classes** and **add** any **helper methods** or **attributes** that you deemed necessary :

Visitor
-name: STRING -contactNumber: STRING -temperature: REAL +timeStamp: **DateTime***
+constructor(STRING, STRING, INTEGER) +getName(): STRING +getContact(): STRING +getTemperature(): REAL -__str__(): STRING

Bed
+floor: INTEGER +ward: INTEGER +bedNo:INTEGER +occupiedBy: Patient +queue: Queue +visitHourStart: INTEGER +visitHourEnd: INTEGER
+constructor(INTEGER, INTEGER, INTEGER, INTEGER, INTEGER) -__str__():STRING

***DateTime :** You may implement the DateTime data type as a custom class, use the Python datetime class or implement it as a native Python data type like string, tuple or list. The data type must be able to represent the day, month, year, hour and minute.

Queue
-buffer: LIST of Visitor
+constructor() +enqueue(Visitor) +dequeue():Visitor +peekMaxTemp(): REAL -__str__(): STRING

Patient
-name: STRING -NRIC: STRING
+constructor(STRING. STRING) +getName():STRING +getNRIC():STRING -__str__():STRING

Hospital
+beds: ARRAY of Bed +noFloors: INTEGER +noWards: INTEGER +noBeds: INTEGER
+constructor(INTEGER, INTEGER, INTEGER) +visit(Visitor, INTEGER, INTEGER, INTEGER**, DateTime***) : BOOLEAN -hash(INTEGER, INTEGER, INTEGER): INTEGER +occupy(Patient, INTEGER, INTEGER, INTEGER) +showOccupancy()

The following are the descriptions of the various attributes and methods of the above-mentioned classes. All methods must be implemented according to the description. If the run time complexity is not given, your algorithm must use the most efficient run time:

Attributes/Methods	**Description**
Visitor.constructor (STRING, STRING, REAL)	The arguments for the constructor are name, contact number and temperature taken of the visitor.
Visitor.__str__(): STRING	Returns the string representation of the Visitor object. The format is as follows: <name>:<temperature> where <name> and <temperature> are the name and temperature attributes of the Visitor object.
Patient.constructor (STRING, STRING)	The arguments for the constructor are name and NRIC.
Patient.__str__(): STRING	Returns the name attribute of the Patient object.
Bed.occupiedBy	The Patient object occupying the bed. None, if the bed is not occupied.
Bed.queue	The Queue object used to keep track of pending visitors.
Bed.visitHourStart	The start of the visiting hours. Represented in 24 hour format.(0 to 24). Visiting hours start at the hour.
Bed.visitHourEnd	The end of the visiting hours. Represented in 24 hour format.(0 to 24). Visiting hours end at the hour.
Bed.constructor (INTEGER, INTEGER, INTEGER, INTEGER, INTEGER)	The arguments for the constructor are, floor, ward, bed number, start of visiting hours, end of visiting hours.
Bed.__str__(): STRING	Returns the string representation of the Bed object. The format is as follows: <floor>-<ward>-<bedNo>:<name> where <floor>, <ward> and <bedNo> are the floor, ward and bedNo attributes of the Bed object and <name> is the name attribute of the Patient object if the bed is occupied or None if the bed is unoccupied.
Queue.enqueue(Visitor)	Adds the Visitor object to the end of the queue. Run time complexity must be **O(1) or O(n)**.
Queue.dequeue(): Visitor	Removes and returns the Visitor object from the front of the queue. Returns None if queue is empty. Run time complexity must be **O(1)**.
Queue.peekMaxTemp(): INTEGER	Returns the Visitor object with the highest temperature that is currently in the queue. Returns None if queue is empty. This operation **must** be implemented in **O(1)** time complexity.

Queue.__str__():STRING	Returns the string representation of the Queue object. The format is as follows: `[<Visitor obj>,<Visitor obj>] <maxTempVisitor obj>` where `<Visitor obj>` is the string representation of a Visitor object. `<maxTempVisitor obj>` is the string representation of the Visitor object with the highest temperature, None if queue is empty.
Hospital.constructor (INTEGER, INTEGER, INTEGER)	The arguments for the constructor are number of floors, number of wards in each floor, and the number of beds in each ward. This method will create and initialise the beds array with the correct number of Bed objects initialised with no occupancy and their respective valid visiting hours.
Hospital.visit(Visitor, INTEGER, INTEGER, INTEGER, DateTime*): BOOLEAN	The arguments are the Visitor object, the floor, ward and bed number of the patient to visit and the timestamp(day, month, year, hour and minute) of the visit. The method must perform the following validation - if the bed is unoccupied, return False - if the visit is within the valid visitation hours for that bed, the visitor is put in a queue for the Bed object and True is returned, else return False.
Hospital.hash (INTEGER, INTEGER, INTEGER): INTEGER	Arguments are floor, ward and bed number. Maps the floor, ward and bed number to an index of the beds array where the Bed object is located. Note that the hash function should not produce any collision.
Hospital.occupy(Patient, INTEGER, INTEGER, INTEGER)	Arguments are patient, floor, ward and bed number. Assigns the Patient object to the bed object in the beds array. Use the Hospital.hash function to map the floor, ward and bed number to the correct index of the beds array.
Hospital.showOccupancy ()	Display the indexes and the Bed objects in the beds array for beds that are **currently occupied** by a patient. The format for each line is as follows: `<index>-><bed obj><queue obj>` where `<index>` is the index of the beds array `<bed obj>` is the string representation of the Bed object `<queue obj>` is the string representation of the queue attribute of the Bed object.

For each sub-task, add a comment statement, at the beginning of the code using the hash symbol "#", to indicate the sub-task the program code belongs to, for example:

```
In [1]: #Task 3.1
        Program code

        Output:
```

The file provided is `PATIENTS.CSV`

Task 3.1
Write program code for the Patient class and Visitor class. [6]

Task 3.2
Write program code to test your implementation of the Patient and Visitor classes by:

- creating two instances of the Patient class
- creating two instances of the Visitor class
- printing the 4 objects

A sample of the test cases and output is shown below:

```
In [1]: #Task 3.2
        v1 = Visitor("Abbott","955-505-11",37.5)
        v2 = Visitor("Norby","955-535-82",35.6 )
        p1 = Patient("Lynna","S1849733F")
        p2 = Patient("Carine","T6884733D")
        print(p1,p2,v1,v2)

        Lynna Carine Abbott:37.5 Norby:35.6
```

[2]

Task 3.3
Write program code for the Queue class. [14]

Task 3.4
Write program code to test your implementation of the Queue class.

- Create an instance of the Queue object.
- Use the two Visitor objects created in Task 3.2 to add into the queue
- Print the Queue object
- Dequeue one item from the Queue object
- Print the Queue object
- Create another instance of Visitor object with a temperature of 37.8 and add into the Queue.
- Print the Queue.

A sample of the test cases and output is shown below:

```
In [1]: #Task 3.4
        q=Queue()
        q.enqueue(v1) #v1 from Task 3.2
        q.enqueue(v2) #v2 from Task 3.2
        print(q)
        q.dequeue()
        print(q)
        v3 = Visitor("Alvinia","955-561-84",37.8)
        q.enqueue(v3)
        print(q)

        [Abbott:37.5,Norby:35.6] Abbott:37.5
        [Norby:35.6] Norby:35.6
        [Norby:35.6,Alvinia:37.8] Alvinia:37.8
```

[4]

Task 3.5
Write program code for the Bed class. [3]

Task 3.6
Write program code for the Hospital class. [14]

Task 3.7
The file PATIENTS.CSV contains the data for creating and populating a Hospital object.
The first line of the file contains the following information about the hospital:
`<number of floors>,<number of wards>,<number of rooms>`
where
`<number of floors>` is the number of floors in the hospital.
`<number of wards>` is the number of wards in each floor of the hospital.
`<number of beds>` is the number of beds in each ward of the hospital.
The subsequent lines contain the following patients' information:
`<name>,<NRIC>,<floor number>,<ward number>,<bed number>`
where
`<name>` is the name of the patient.
`<NRIC>` is NRIC number of the patient.
`<floor number>` is the floor number where the patient is staying.
`<ward number>` is the ward number where the patient is staying.
`<bed number>` is the bed number where the patient is staying.

Write code to

- Read the contents of the file PATIENTS.TXT.
- Create an instance of a Hospital object based on the information in the file.
- Populate the beds array of the hospital object with the patients' information in the file
- Display the beds in the hospital that are being occupied by patients using the `Hospital.showOccupancy()` method.

The output should look like this:

```
3->1-1-4:Rozamond []None
29->3-2-2:Lynna []None
31->3-2-4:Bartolomeo []None
32->3-3-1:Hewett []None
33->3-3-2:Carine []None
42->4-2-3:Maye []None
45->4-3-2:Anastasie []None
52->5-2-1:Crystie []None
62->6-1-3:Marven []None
66->6-2-3:Carry []None
```

- Create two Visitor objects.
- Using the `Hospital.visit()` method to simulate the two visitors to visit a patient (eg Lynna).
- Display the beds in the hospital that are being occupied by patients suing the `Hospital.showOccupancy()` method . The output should be similar to this:

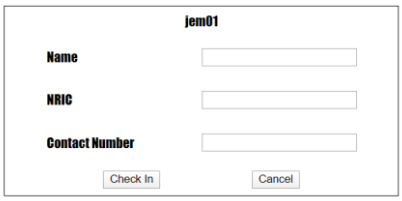
```
3->1-1-4:Rozamond []None
29->3-2-2:Lynna [Abbott:37.5,Norby:35.6] Abbott:37.5
31->3-2-4:Bartolomeo []None
32->3-3-1:Hewett []None
33->3-3-2:Carine []None
42->4-2-3:Maye []None
45->4-3-2:Anastasie []None
52->5-2-1:Crystie []None
62->6-1-3:Marven []None
66->6-2-3:Carry []None
```

[5]

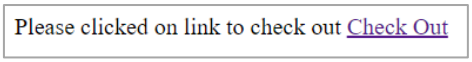
Save your Jupyter Notebook file as
`Task3_<your name>_<NRIC number>.ipynb.`

4 In order to facilitate contact tracing, a web application is to be developed to allow visitors at different locations to check in when they entered the premises and check out when they leave the premises.
An example of the check in and out workflow is as follow:

(a) Visitor uses a mobile phone to scan a QR code encoded with the url, example http://localhost:5000/jem01.
(b) The following form appears on the visitor 's mobile web browser



(c) Visitor fills up form and click Check In
(d) The web app will returned a web page containing a link for the user to check out when he/she leaves the premise as follows:



(e) After visitor clicked on the Check Out link. The following message will be displayed on the mobile web browser:

A database containing 2 tables with their data attributes is described as follows:

`Location( LocationID:STRING, Name:STRING, Address:STRING, URL:STRING)`

`Visitor (NRIC:STRING, LocationID*:STRING, Name:STRING, Contact:STRING, Date:STRING, TimeIn:STRING, TimeOut:STRING)`

Underline attribute : Primary Key
* Foreign Key

Task 4.1
Use the relevant tools or code to:

- Create a database named **`EntryDB`**.
- Create the 2 tables described above.
- Populate the `Location` table by importing the data from the file `LOCATIONS.CSV`
- If you are writing code, name your code file/s, `TASK4_1.py/TASK4_1.sql` [2]

At the entrance of each location, a QR code is encoded with the URL of the location that is stored in the `Location` table in Task 4.1.

When a visitor entered the premise, he/she will scan the QR code and the web browser will be directed to the URL for that location. The format of the URL is as follows: `http://<server_dns_name>/<location_id>`.
where
`<server_dns_name>` is the dns name of the web server.
`<location_id>` is the value of the LocationID attribute stored in the Location table.

A check-in form will be rendered on the web browser for the visitor to entered the information required to create a record in the Visitor table. The visitor will only need to enter

- name.
- NRIC.
- contact number.

Task 4.2

- Create a jinja template named checkin.html
    - The LocationID must be auto-filled and displayed in the form.
- Write Python code to rendered the checkin.html form when the visitor checks in by scanning the QR code on his/her mobile device.
- You can assume that the QR code scanner app will automatically launch a web browser and fills in the URL on the address bar. [5]

Task 4.3

Write Python code to

- retrieve the data in the form submitted by the visitor and insert a new Visitor record in the Visitor table. In addition, the following attributes in the table are to be generated automatically:
    - Date in YYMMDD format.
    - TimeIn in HHMM 24 hour.
      (YY: Year, MM:Month, DD:Day, HH:Hour, MM:Minute)
- Return a web page containing a link for the visitor to check-out as described in the example workflow. [5]

Task 4.4

When the visitor clicked on the Check Out link as described in the example workflow, write Python code to:

- Update the TimeOut in the visitor's record to indicate the time that the visitor leaves the premise
- Return a web page with a message with the visitor's NRIC as described in the example workflow. [3]

Save your Python program as
`TASK4_4_<your name>_<NRIC number>.py`
with any additional files and sub-folders as needed in a **folder** named
`Task4_<your name>_<NRIC number>`

END OF PAPER

BLANK PAGE

BLANK PAGE