H

NATIONAL JUNIOR COLLEGE
Mathematics Department

General Certificate of Education Advanced Level
Higher 2

COMPUTING 9569/02

Paper 2 Lab-based

15 September 2020

3 hours

Additional Materials:	Electronic version of `PARTICIPANTS.TXT` data file
	Electronic version of `PATIENTS.CSV` data file
	Electronic version of `SCHOOL.TXT` data file
	Electronic version of `STAFF.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 5 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 **13** printed pages and 3 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 In a software development hackathon competition, participants are being assigned different roles. The participants information is stored in a file, `PARTICIPANTS.TXT` There are three fields on each line which indicates name, role and gender. The fields are separated by ';'

```
Rufus Schuck;Coder;F
Ione Wolfe;Manager;F
Hillary Curl;Tester;M
…
```

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 provied is `PARTICIPANTS.TXT`

Task 1.1
Implement the Python function `read_data(filename)` which takes `filename` as a string and returns a 2-dimension Python list that follows the format as shown in the example below.

```
>>> read_data("PARTICIPANTS.TXT")
[['Rufus Schuck', 'Coder', 'F'],
['Ione Wolfe', 'Manager', 'F'],
['Hillary Curl', 'Tester', 'M'],…]
```

Take note that the list shown above is just an example.

This list shown above is known as the `participants` list and it contains a list of participants' records. [2]

Task 1.2
Implement the Python function

```
gender_count(participants)
```

which takes a list `participants,` obtained in **Task 1.1** and returns the number of male and female participants.
The following shows an example of how the function Is used.

```
>>> participants = read_data("PARTICIPANTS.TXT")
>>> number_males, number_females = gender_count(participants)
>>> print(number_males, number_females)
17 33
```

[3]

Task 1.3
Implement the Python function

```
role_statistics(participants)
```

which takes a list `participants` obtained in **Task 1.1** as input and outputs the number of students for each role in the following format. You are not allowed to hardcode the roles, roles must be determined at run-time.

The following shows an example of how the function Is used.

```
>>> participants = read_data("PARTICIPANTS.TXT")
>>> role_statistics(participants)
Role        Number
Coder       7
Manager     13
Maker       11
Designer    11
Tester      7
Snakeeater  1
```

[4]

Task 1.4
Implement the Python function

```
form_group(participants)
```

which takes a list `participants` obtained in **Task 1.1** as input and returns a list consisting of 5 participants' records. This list of participants forms a group and must consist of **one coder, one maker, one tester, one manager and one designer**. The participants picked for each role must be random. The five participants that forms the group must have their corresponding records removed from the `participants` list.

If there is not sufficient roles or participants to form a group, return an empty list.

The following shows an example of how the function Is used.

```
>>> participants = read_data("PARTICIPANTS.TXT")
>>> form_random_group(participants)
[['Shaunda Sieg', 'Coder', 'F'],
 ['Darlena Crimi', 'Manager', 'F'],
 ['Phyliss Rolen', 'Maker', 'M'],
 ['Russell Gillison', 'Designer', 'F'],
 ['Kathlene Collar', 'Tester', 'M']]
```

Note:

```
'Shaunda Sieg'is a Coder
'Darlena Crimi'is a Manager
'Phyliss Rolen'is a Maker
''Russell Gillison' is a Designer
'Kathlene Collar' is a Tester
```

[10]

Task 1.5

Using your solutions in the previous tasks, write Python code to form as many groups from the `participants` list obtained in **Task 1.1** and output the groups to a file `GROUPS.TXT`.

The file should also include a Group Number for each group formed. An example of how the file should look like is as follows:

```
Group 1
Toney Mcnab,Coder,M
Luanne Lett,Manager,F
See Borne,Maker,F
Laverna Halpern,Designer,F
Chadwick Griffin,Tester,M
Group 2
Fredricka Gormley,Coder,F
Ashely Faye,Manager,M
Phyliss Rolen,Maker,M
Carolann Kintner,Designer,M
Reiko Stack,Tester,F
Group 3
Rufus Schuck,Coder,F
Odis Levalley,Manager,M
Teodoro Negrin,Maker,F
Russell Gillison,Designer,F
Apryl Soileau,Tester,F
Group 4
Angle Linck,Coder,F
Johnna Lecuyer,Manager,F
Tashia Bowen,Maker,F
Grazyna Kitzman,Designer,M
Kathlene Collar,Tester,M
```

[5]

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

2 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:
```

Task 2.1
Implement a recursive function in Python code to compute the summation of a geometric series:

$$S = \sum_{n=1}^{n=m} ar^n = ar + ar^2 + ar^3 + \cdots + ar^m$$

a, r and m are **integers** entered from standard input in its individual line.

The final program should print out

- All items, separated in whitespace, in the series.
- Summation of the series

For example, the inputs for a, r and m are as follows:

```
1
2
10
```

Your final program should print

```
2 4 8 16 32 64 128 256 512 1024
2046
```

[9]

Task 2.2

The n-th Fibonacci number, in a Fibonacci sequence is defined as :

$$F_n = F_{n-1} + F_{n-2}$$

with seed values $F_0 = 0$ and $F_1 = 1$

It can be implemented using a recursive Python function as follows:

```
def Fibonacci(n):
    if n<0:
        print("Incorrect input")
    # First Fibonacci number is 0
    elif n==0:
        return 0
    # Second Fibonacci number is 1
    elif n==1:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)
```

Convert the recursive Python function into an iterative version. [6]

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

3 A Visitor Management System for a hospital is to be developed to manage visitors visiting patients in the hospital. A vital component of the system is the queuing system 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 class.
- Each Bed object has a Queue object that is use to manage the visitors waiting to visit the patient in the hospital room. The maximum queue size is **10**.
- 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. If the visit is within the valid visitation hours, the visitor will be placed in the queue for the room.
- 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
+constructor(STRING, STRING, INTEGER) +getName(): STRING +getContact(): STRING -__str__(): STRING

Queue
-buffer: ARRAY of Visitor
+constructor() +enqueue(Visitor):BOOLEAN +dequeue():Visitor -__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

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) : 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, if the run time complexity is not given, your algorithm must use the most efficient run time. Note that not all methods/attributes are described here.

Attributes/Methods	**Description**
Visitor.constructor (STRING, STRING)	The arguments for the constructor are name and contact number of the visitor.
Visitor.__str__(): STRING	Returns the name and contactNumber attribute of the Visitor object. The format is as follows: `Name:contactNumber`
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): BOOLEAN	Adds the Visitor object to the end of the queue. Returns True if success else False.
Queue.dequeue(): Visitor	Removes and returns the Visitor object from the front of the queue. Returns None if queue is empty.
Queue.__str__():STRING	Returns the string representation of the Queue object. The format is as follows: `[<Visitor obj>,<Visitor obj>]` where `<Visitor obj>` is the string representation of a Visitor object.
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.

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_object><queue_object>` where `<index>` is the index of the beds array `<bed_object>` is the string representation of the Bed object `<queue_object>` 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 output is shown below:

```
In [1]: Lynna
        Carine
        Abbott:955-505-11
        Norby:955-535-82
```

[2]

Task 3.3
Write program code for the Queue class. [10]

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 and add into the Queue.
- Print the Queue.

A sample of the output is shown below:

```
In     [Abbott:955-505-11,Norby:955-535-82]
 [1]:  [Norby:955-535-82]
       [Norby:955-535-82,Alvinia:955-561-84]
```

[2]

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.

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 three visitors to visit a patient (eg Lynna).
- Display the beds in the hospital that are being occupied by patients. 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 A school directory stores information about schools and staff teaching in the schools. It allows users to query for staff information working in a particular school and department.

Information about a school includes:

- SchoolCode: a unique 4-digit number to identify the school.
- Name: name of the school.
- Address: address of the school

For staff working in schools, the following information is recorded:

- SchoolCode : schoolcode of the school the staff is working at
- Name: name of the staff
- Department: the department in the school the staff belongs to
- Contact: telephone number of the staff

It is assume that no staff working in the same schook will have the same name.

The information described above is currently stored in two files:

- `SCHOOL.TXT`
- `STAFF.TXT`

Task 4.1
Create an SQL file called `TASK4_1_<your name>_<NRIC number>.sql`.
Write the SQL code to create the the **two tables** in the database named `schools.db`. You are to create the necessary primary, foreign keys and constraints in your SQL code.

Excute the sql code in DB Browser to create the database and tables. [3]

Task 4.2
The files `SCHOOL.TXT` and `STAFF.TXT` contain information currently stored in the school directory system. Write Python code to migrate them to the database tables created in Task 1.1. The database tables should only contain the data in the two files.

Save your Python code as
`TASK4_2_<your name>_<NRIC number>.py` [3]

Task 4.3
Write Python code to generate a web form that allows a user to search for staff/s using two fields:

i. Name of School
ii. Department

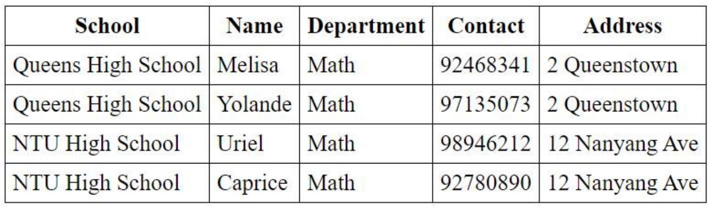
a) The web form should be the default page when you access the web application and should have appropiate formating and layout.
b) The search field for Name of School should allow for partial match. Example, the search for a school named "High" should return
   - NTU High School
   - Queens High School

   The search field for Department should be an exact match. [3]

Task 4.4
Write Python code to:

a) return a HTML page thate contains a table showing the results of the query in **Task 4.3**
b) The information to be returned are:
   - Name of school
   - Name of staff
   - Department
   - Contact
   - Address of school

c) The format of the results should look like:



School	Name	Department	Contact	Address
Queens High School	Melisa	Math	92468341	2 Queenstown
Queens High School	Yolande	Math	97135073	2 Queenstown
NTU High School	Uriel	Math	98946212	12 Nanyang Ave
NTU High School	Caprice	Math	92780890	12 Nanyang Ave

Save the results of the query as `TASK4_4_<your name>_<NRIC number>.html` [5]
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

BLANK PAGE