HWA CHONG INSTITUTION
C2 PRELIMINARY EXAMINATION 2021

COMPUTING
Higher 2

23 AUG 2021 **Paper 2 (9569 / 02)** **1400 -- 1700 hrs**

Additional Materials:

Electronic version of `INTEGERS.txt` data file

Electronic version of `LOG.txt` data file

Electronic version of `PATIENTS.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 brackets [ ] at the end of each question or part question.

The total number of marks for this paper is **100**.

---

1. Name your Jupyter Notebook as

   `Task1_<your name>_<centre number>_<index number>.ipynb`

   For each of the sub-tasks, 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

In[2]:  # Task 1.2
        Program Code

In[3]:  # Task 1.3
        Program Code

        Output:
```

Task 1.1

The file `INTEGERS.txt` stores 100 integers. Write a program to read the integers, arrange them in ascending order using quick sort, and write the sorted integers to a file called

`SORTED_<your name>_<centre number>_<index number>.txt` [15]

Task 1.2

Write a function `BinarySearch(list_of_integers, target)` that

- takes a list of ascending integers, `list_of_integers` and an integer `target`
- performs a binary search
- prints out if `target` is found in `list_of_integers`
- returns the number of comparisons during the binary search [8]

Task 1.3

Write a program to read the list of integers from

`SORTED_<your name>_<centre number>_<index number>.txt`

obtained in Task 1.1. Generate 50 random integers between 1 and 200 (inclusive) and perform a binary search for each of these random integers in this sorted list. Output the average number of comparisons of these 50 binary searches. [2]

Save your Jupiter Notebook for Task 1.

2. FlexiMSG provides messaging services. Information of the messages are logged into a file. The log records contain the phone numbers or IP address of the sender, the date which the service is being accessed, the status indicating whether the message has been sent and the type of application used. There are two different formats used:

   ```
   <IP address> <DD/MMM/YYYY> <Status> <App>
   ```
   or
   ```
   <Phone number> <DD/MMM/YYYY> <Status>
   ```

   Below is the log records in the file, `LOG.txt`:

   ```
   54.36.149.41 22/Jan/2021 200 WA
   188.226.164.216 22/Jan/2021 0 FB
   92783423 22/Jan/2021 200
   188.226.164.216 23/Jan/2021 0 FB
   88188293 23/Jan/2021 0
   ```

   Task 2.1

   Write the SQL code to create database `ServiceLog.db` with the single table, `Log`.

   The table will have the following fields of the given SQLite types:

   - `LogID` - primary key, an auto-incremented integer
   - `Sender` - the client internet address or phone number, text
   - `AccessDate` - the access date, text
   - `Status` - the status, integer
   - `AppType` - application type, text

   Save your SQL code as
   `Task2_1_<your name>_<center number>_<index number>.sql` [2]

Task 2.2

FlexiMSG wants to use Python programming language and object-oriented programming to update the information in the log file into the database.

Create the class `ServiceRecord` that will store the following:

- `Sender` - stored as a string
- `AccessDate` - stored as a string
- `Status` - stored as integer, `0` or `200`
- `AppType` - stored as string value `'WA'` or `'FB'`

The class has the following methods:

- `isSuccess()`- returns a Boolean value to indicate whether the message has been sent.
  - returns `True` if the `Status` is 200, otherwise returns `False`

- `getAppType()`- returns a string value to indicate the type of messaging application.
  - returns the value of `AppType`

Write program code to read in the information from `LOG.txt`, creating an instance of the `ServiceRecord` class for each record and insert the information into `ServiceLog.db` database.

Save your program code as
`Task2_2_<your name>_<center number>_<index number>.py` [8]

Task 2.3

FlexiMSG wants to publish the database content on a web page.

Create class `AppServiceRecord` which inherits from `ServiceRecord`, such that:

- `getAppType()`- returns the following values based on the value of `AppType`
  - `WA` - returns `'WHATSAPP'`
  - `FB` - returns `'FACEBOOK MESSENGER'`

- `getSuccess()`- returns the following values based on the returned value of `isSuccess()`
  - `True` - returns `'SUCCESS'`
  - `False` - returns `'FAILED'`

Create class `SmsServiceRecord` which inherits from `ServiceRecord`, such that:

- `getAppType()`- always returns '`SHORT MESSAGE SERVICE`'
- `getSuccess()`- returns the following values based on the returned value of `isSuccess()`
  - `True` - returns '`SUCCESS`'
  - `False` - returns '`FAILED`'

Save your program code to
`Task2_3_<your name>_<center number>_<index number>.py` [4]

Task 2.4

Write a Python program and the necessary files to create a web application that enables the list of log records to be displayed.

For each record, the web page should include the:

- `Sender`
- `AccessDate`
- `AppType` (either `WHATSAPP`, `FACEBOOK MESSENGER` or `SHORT MESSAGE SERVICE`)
- `Status` (`SUCCESS` or `FAILED`)

Save your program code as
`Task2_4_<your name>_<center number>_<index number>.py`
with any additional files/sub-folders as needed in a folder named
`Task2_4_<your name>_<center number>_<index number>.` [9]

Run the web application and save the output of the program as
`Task2_4_<your name>_<center number>_<index number>.html` [1]

3. Name your Jupyter Notebook as

   `TASK3_<your name>_<centre number>_<index number>.ipynb`

   The task is to implement a priority queue using Object-Oriented Programming.

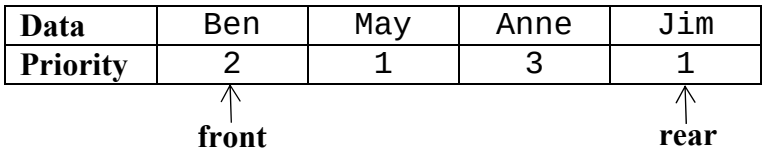
   A priority queue is an extension of queue with the following properties.

   - Every element has a priority associated with it. Smaller integer value has a higher priority.
   - An element with high priority leaves the queue before an element with low priority.
   - If two elements have the same priority, they are served according to their order in the queue, i.e. the earlier element will be served before the later element (FIFO).

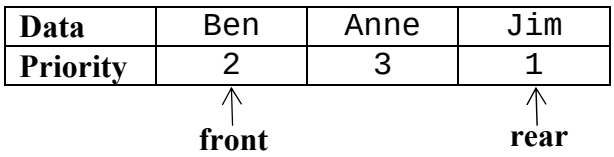
   For example, the emergency room in a hospital assigns patients with priority numbers. The patient with the highest priority is treated first, regardless of the order of arrival.

   An example of operations on a priority queue is shown below:

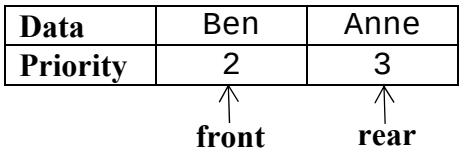
   Initial state of priority queue:



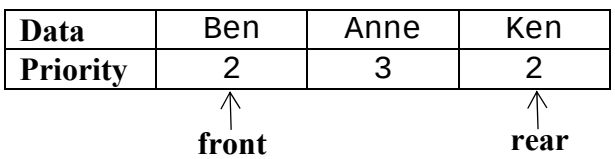
   Remove from priority queue: `May` is removed



   Remove from priority queue: `Jim` is removed



   Insert `'Ken'` **with priority** `2`



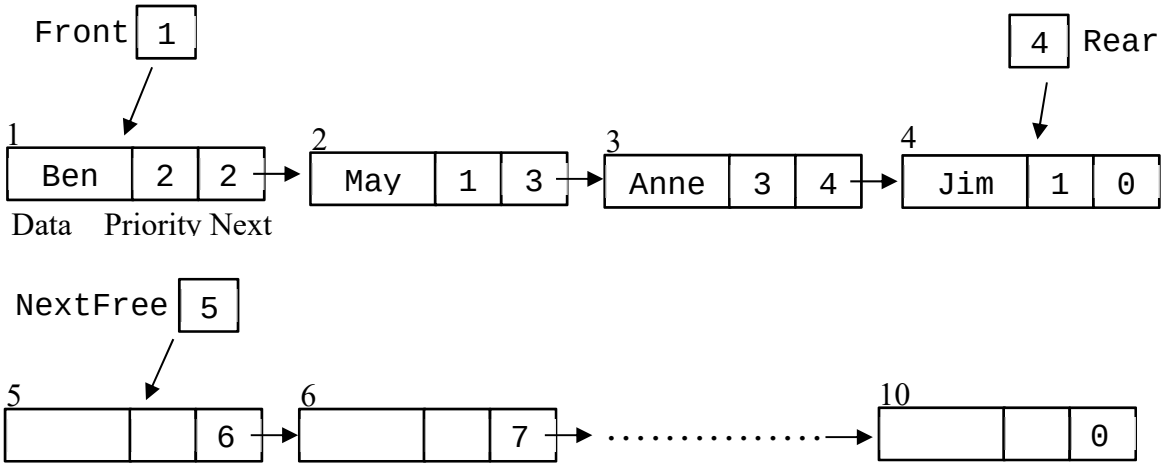
A **priority queue** abstract data type (ADT) is to be implemented as a linked list using object-oriented programming. Two classes `Node` and `PQueue` have been identified.

Class: `Node`		
Identifier	**Data Type**	**Description**
`Data`	`STRING`	The node data
`Priority`	`INTEGER`	Indicates priority of node. Smaller value has higher priority
`Next`	`INTEGER`	Pointer to next node in queue.

Class: `PQueue`		
Identifier	**Data Type**	**Description**
`ThisPQueue`	`ARRAY[1:10] Of Node`	The priority queue data.
`Front`	`INTEGER`	Index for front node of queue.
`Rear`	`INTEGER`	Index for rear node of queue.
`NextFree`	`INTEGER`	Index for the next unused node.
`Initialise()`	`PROCEDURE`	• Set pointers to indicate all nodes are unused and linked. • Initialise values for `Front`, `Rear` and `NextFree`.
`PQInsert(NewItem, Priority)`	`PROCEDURE`	• Assign `NewItem` and `Priority` passed as parameters to a node. • Insert the node to the rear of the priority queue.
`PQDelete()`	`FUNCTION`	• Remove a node of highest priority from the priority queue. • Return the `Data` attribute of the node that is removed.
`DisplayPQueue()`	`PROCEDURE`	• Display the values of `Front`, `Rear`, `NextFree` and the contents of `ThisPQueue` array in index order.

The diagram shows the linked list with:

- the data items 'Ben', 'May', 'Anne' and 'Jim' (inserted in that order) in the priority queue
- the unused nodes linked together



For each of the sub-tasks, 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
```

```
In[2]:  # Task 3.2
        Program Code
```

Output:

```
In[3]:  # Task 3.3
        Program Code
```

Output:

Task 3.1

Write program code for the classes `Node` and `PQueue`, including the `Initialise`, `PQInsert`, `PQDelete` and `DislayPQueue` methods. The code should follow the specification given. [17]

Task 3.2

The program is to be tested.

Write a main program to:

- create a `PQueue` object
- read from file `PATIENTS.txt` all the data items with its priorities into the priority queue by calling `PQInsert` method.
- output the priority queue by calling `DisplayPQueue` method. [2]

Task 3.3

Write additional code in your main program to do the following in order by calling the appropriate methods from `PQueue` class.

No.	Operation	Data	Priority
1	Remove patient	-	-
2	Remove patient	-	-
3	Add patient	`Carol`	`4`
4	Remove patient	-	-
5	Remove patient	-	-
6	Display priority queue	-	-

Save your Jupiter Notebook for Task 3. [3]

4. Name your Jupyter Notebook as

   `TASK4_<your name>_<centre number>_<index number>.ipynb`

   The task is to write program code for a Tic-Tac-Toe-Tomek game for two players.

   Tic-Tac-Toe-Tomek is a game played on a 4 x 4 square board. The board starts empty, except that a single 'T' symbol may appear in one of the 16 squares. There are two players: X and O. They take turns to make moves, with X starting. In each move a player puts her symbol in one of the empty squares. Player X's symbol is 'X', and player O's symbol is 'O'.

   After a player's move, if there is a row, column or a diagonal containing 4 of that player's symbols, or containing 3 of her symbols and the 'T' symbol, she wins and the game ends. Otherwise the game continues with the other player's move. If all of the fields are filled with symbols and nobody won, the game ends in a draw.

   Given the 4 x 4 board description containing 'X', 'O', 'T' and '.' characters (where '.' represents an empty square). The following examples show the various winning positions.

```
X X X T    X O X O    O O X X    X X X O    O X X X
. . . .    X X O O    O X X X    . . O .    X O . .
O O . .    O X T X    O X . T    . O . .    . . O .
. . . .    X X O O    O . . O    T . . .    . T . O
```

   X won | Draw | O won | O won | O won

   For each of the sub-tasks, 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 4.1
        Program Code
```

```
In[2]:  # Task 4.2
        Program Code
```

```
In[3]:  # Task 4.3
        Program Code
```

```
In[4]:  # Task 4.4
        Program Code
```

```
In[5]:  # Task 4.5
        Program Code
```

```
Output:
```

Task 4.1

Write program code to:

- initialize the data structure to represent the 4 x 4 square board, using the identifier `board`
- generate a pair of random numbers between 1 and 4
- place 'T' at that random position on the board [3]

Task 4.2

Write a function `displayBoard` that will display the game board clearly to the players. You should use the `board` as a parameter in `displayBoard`. [2]

Task 4.3

Write a function `getPlayerMove` to get players to make their move (by marking 'X' or 'O') on the board. You should include validation on player's input and check that the space is not already occupied. Use `board` as a parameter. You may include any other suitable parameters. [4]

Task 4.4

Write a function `checkWin` that checks all the conditions for winning a game and returns `True` if a player has won the game, otherwise returns `False`. Use `board` as a parameter. You may include any other suitable parameters. [5]

Task 4.5

Write a `main` function that makes use of the identifiers and functions from Task 4.1 to Task 4.4 and allows two players, X and O, to play a game of Tic-Tac-Toe-Tomek.

The `main` function should include the following:

- display the initial game board with the single `'T'` displayed in it
- start with player X to make the first move
- ensure players X and O take turns to make their move
- display the game board after every move made by a player
- check for winner
- display message on which player has won the game or whether the game ends in a draw.

[7]

Run your `main` function and produce outputs of **three** games where player X wins one game, player O wins another game, and a drawn game.

Copy and paste all outputs in a text file as

`TASK4_5_<your name>_<centre number>_<index number>.txt` [3]

Save your Jupyter Notebook for Task 4.

BLANK PAGE