NANYANG JUNIOR COLLEGE
JC2 MID YEAR COMMON TEST

Higher 2

COMPUTING 9569/02

Paper 2 (Lab-based) **1 July 2020**

3 hours

Additional Materials: Electronic version of `Message.txt` data file
Electronic version of `MONITORS.txt` data file
Electronic version of `PRINTERS.txt` data file
Electronic version of `LAPTOPS.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 to 4 should be saved in a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as `TASK1_<your name>.ipynb`

1 The task is to implement a columnar transposition cipher to encode a message.

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
```

Output:

```
In [2]: #Task 1.2
        Program code
```

Output:

```
In [3]: #Task 1.3
        Program code
```

Output:

In a columnar transposition, a message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the length of the rows and the permutation of the columns are usually defined by a keyword. For example, the keyword ZEBRAS is of length 6 (so the rows are of length 6), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be "6 3 2 4 1 5".

In a regular columnar transposition cipher, any spare spaces are filled with nulls. Finally, the message is read off in columns, in the order specified by the keyword. For example, suppose we use the keyword ZEBRAS and the message WE ARE DISCOVERED. FLEE AT ONCE. In a regular columnar transposition, we write this into the grid as follows:

```
6 3 2 4 1 5
W E A R E D
I S C O V E
R E D F L E
E A T O N C
E Q K J E U
```

providing five nulls (QKJEU), these letters can be randomly selected as they just fill out the incomplete columns and are not part of the message. The ciphertext is then read off as:

```
EVLNE ACDTK ESEAQ ROFOJ DEECU WIREE
```

Task 1.1

Write a program to:

- prompt the user to enter a cipher key (string) that is not more than 10 characters.
- validate that each character is a single letter in the ranges A - Z or a - z.
- if invalid, display a suitable error message and allow the user to input the key again.
- call `GenerateKeyOrder(CipherKey)`.

This user defined function will:

- calculate the column permutation using the order of the letters (1 - 26) in the cipher key.
- return this order in a string with a space between each number. [6]

Test your program with the following test data:

```
Zebras
Output: 6 3 2 4 1 5
```

[2]

Task 1.2

Extend your program code to:

- read a text to be encoded from `Message.txt` ignoring spaces and punctuation marks.
- compute the number of rows and columns needed to construct the transposition table.
- use a suitable data structure to create the table.
- read the message into the table and assign `'!'` as a null character if needed.
- call `GenerateKeyOrder(CipherKey)` using `Zebras` as the key.
- generate the encoded message by reading each column according to the column permutation of the key.
- display the message with a space to separate words in different columns.

Your screen display should look like this:

```
The encoded message is:
EVLN! ACDT! ESEA! ROFO! DEEC! WIREE
```

[12]

Task 1.3

Write a Decode(EncodedMessage, Key) function that will take your encoded message, decode it and display the decoded message. To decode a message:

- calculate the number of rows by dividing the message length (ignore spaces) by the key length.
- construct the transposition table.
- read in the message into the table.
- call `GenerateKeyOrder(Key)` to get the permutation order.
- display the decoded message by reading the columns following the permutation order ignoring null characters.

Your screen display should look like this:

```
The decoded message is:
WEAREDISCOVEREDFLEEATONCE
```

[6]

Save your program code and output for Task 1 as
`TASK1_<your name>.ipynb`

2 Merge sort is an efficient sorting algorithm which falls under divide and conquer paradigm and produces a stable sort. It operates by dividing a large array into two smaller subarrays and then recursively sorting the subarrays.

In the recursive approach, the problem is broken down into smaller, simple subproblems in **top-down** manner until the solution becomes trivial.

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 2.1
        Program code
```

```
        Output:
```

```
In [2]: #Task 2.2
        Program code
```

```
        Output:
```

Task 2.1

Write program code to:

- create an array of 50 random integers between 0 - 99.
- display these numbers in rows of 10 integers.
- implement the function MergeSort(SortArray) that takes in an array of unsorted integers and sort them in ascending order.
- call MergeSort() function to sort your array of 50 random integers.
- display the sorted result in rows of 10 integers.

[12]

Bottom-up mergesort (non-recursive) consists of a sequence of passes over the whole array, doing merges on subarrays of size sz, starting with sz equal to 1 and doubling sz on each pass. The final subarray will be of size sz if the array size is an even multiple of sz, otherwise it is less than sz.

		a[i]														
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
sz = 1																	
merge(a, 0, 0, 1)		E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 2, 2, 3)		E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 4, 5)		E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 6, 6, 7)		E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 8, 8, 9)		E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
merge(a, 10, 10, 11)		E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 12, 12, 13)		E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 14, 14, 15)		E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
sz = 2																	
merge(a, 0, 1, 3)		E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
merge(a, 4, 5, 7)		E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
merge(a, 8, 9, 11)		E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L
merge(a, 12, 13, 15)		E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
sz = 4																	
merge(a, 0, 3, 7)		E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, 8, 11, 15)		E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
sz = 8																	
merge(a, 0, 7, 15)		A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Trace of merge results for bottom-up mergesort

Task 2.2

Write program code to:

- implement a non-recursive iterative MergesortLoop(Array, SortOrder) function to sort an array of 50 integers.
- allow the user to specify whether to sort in ascending (`'A'`) or descending order (`'D'`).
- display the sorted result in rows of 10 integers.

[10]

Design 2 test cases to test your function and explain the purpose of the test data. Show the output of your test cases.

[3]

Save your program code and output for Task 2 as
`TASK2_<your name>.ipynb`

3 You are to design an object-oriented program which simulates a print queue for a printer on a local area network (LAN). The print queue consists at any time of none, one, or more print jobs.

Each user can send a print job from any of the terminals on the LAN. Each terminal on the network is identified by an integer number in the range 1 to 200.

The program you are to design will record for each print job:

- the user ID
- the terminal number from which the print request was sent
- the job type - `'C'` for colour, `'B'` for black and white
- the file size (integer in Kbytes)

In practice, there are several print queues each associated with a different printer. Each printer is identified by a short name such as `Comp_Lab2`.

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
```

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

```
In [4]: #Task 3.4
        Program code
```

```
Output:
```

Task 3.1

Design and write program code to define one or more classes and other appropriate data structures for this application. [6]

A print queue behaves as a queue data structure.

Assume, for testing purposes:

- there is a single printer on the LAN.
- the maximum print queue size for the printer is ten print jobs.

The main program will simulate:

- the sending of print jobs to the printer by different users.
  - that is, the addition of a print job to the print queue.
- the output of a job from the print queue.
  - that is, the removal of a print job from the print queue.

The program design has the following menu:

```
1. Add print job to print queue
2. Output next print job from printer
3. Display current print queue (all jobs in queue)
4. End simulation
```

The program simulates the working of the print queue as follows:

1. The empty print queue is initialized.
2. The program user selects menu options 1, 2 and 3 in any order.
3. The program user selects menu option 4.

Task 3.2

Write program code to:

- display the main menu.
- input the choice by the user.
- run the appropriate code for the choice made. [3]

Task 3.3

Write program code to initialize the print queue for `Comp_Lab2` printer.

Write program code to display the current state of the print queue. [4]

Task 3.4

Write program code to add a new print job to the print queue. The requirement will be:

- program user enters data for the new print job.
- print job is added to the print queue.

Write program code to output the next print job from the printer. This code will execute from menu option 2. Test your program by:

- adding three print jobs
- outputting the next print job. [6]

Save your program code and output for Task 3 as
`TASK3_<your name>.ipynb`

4 A large company currently keeps records on paper of all the computing equipment it owns. Every computer device has its information recorded when it is purchased.

The company decided to trial a database to manage its computing equipment records. It is expected that the database should be normalised.

When a computer device is purchased, the following information is recorded:

- `SerialNumber` - unique serial number of devices
- `Type` - type of device ('Monitor', 'Laptop' or 'Printer')
- `Model` - model of device
- `Location`- where the device is used
- `DateOfPurchase` - date of purchase
- `Writtenoff` - whether the device is still in use (`'TRUE'` means device is written off and NOT in use, `'FALSE'` means device is still in use)

For monitors, the following extra information is recorded:

- `DateCleaned` - the last date the monitor was cleaned

For laptops, the following extra information is recorded:

- `WeightKg` - the weight in kilograms

For printers, the following extra information is recorded:

- `Toner` - type of toner required
- `DateChanged` - the last date the toner cartridge was changed

The information is to be stored in four different tables:

```
Device
Monitor
Laptop
Printer
```

Task 4.1

Create an SQL file called `TASK4_1_<your name>.sql` to show the SQL code to create the database `equipment.db` with the four tables. The table, `Device`, must use `SerialNumber` as its **primary key.** The other tables must refer to the `SerialNumber` as a **foreign key**.

Save your SQL code as
`TASK4_1_<your name>.sql` [5]

Task 4.2

The files `MONITORS.txt`, `LAPTOPS.txt` and `PRINTERS.txt` contain information about the company's monitors, laptops and printers respectively for insertion into the equipment database. Each row in the three files is a comma-separated list of information about a single device.

For `MONITORS.txt`, information about each monitor is given in the following order:
`SerialNumber,Model,Location,DateofPurchase,WrittenOff,DateCleaned`

For `LAPTOPS.txt`, information about each laptop is given in the following order:
`SerialNumber,Model,Location,DateofPurchase,Writtenoff,WeightKg`

For `PRINTERS.txt`, information about each printer is given in the following order:
`SerialNumber,Model,Location,DateofPurchase,Writtenoff,Toner,DateChanged`

Write a Python program to insert all information from the three files into the `equipment` database, `equipment.db`. Run the program.

Save your program code as
`TASK4_2_<your name>.py` [5]

Task 4.3

Write SQL code to show the serial number, model, and the location of each monitor, with the date it was last cleaned. Run this query.

Save this code as
`TASK4_3_<your name>.sql` [4]

Task 4.4

The company wants to filter the devices by `Location` and display the results in a web browser. Write a Python program and the necessary files to create a web application that:

- receives a `Location` string from a HTML form, then
- creates and returns a HTML document that enables the web browser to display a table tabulating the `SerialNumber` and `Type` of devices still in use at that exact `Location`

Save your Python program as
`TASK4_4_<your name>.py` with any additional files / sub-folders as needed in a folder named `TASK4_4_<your name>`

Run the web application. Save the output of the program when "Office 51" is entered as the `Location` as `TASK4_4_<your name>.html` [10]