TEMASEK JUNIOR COLLEGE

2023 JC2 March Common Test

Higher 2

COMPUTING 9569/02

Paper 2 (Lab-based) **22nd March 2023**

2 Hours

Additional Materials: Removable storage device
Electronic version of `plaintext.txt` data file
Electronic version of `students.txt` data file
Electronic version of `MeetingRoom.csv` data file
Electronic version of `Member.csv` data file
Electronic version of `Booking.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 **3** marks out of 66 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 66.

[Turn over

Instruction to candidates:

Your program code and output for each of Task 1, 2, and 3 should be saved in a single `.ipynb` file using Jupyter Notebook. For example, your program code and output for Task 1 should be saved as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

Make sure that each of your `.ipynb` files shows the required output in Jupyter Notebook.

1 Name your Jupyter Notebook as:

`TASK1_<your name>_<centre number>_<index number>.ipynb`

The task is to implement a Vigenère cypher encryption algorithm.

Vigenère cypher is a method of encrypting alphabetic plaintext using multiple substitution alphabets. A Vigenère table or square that is used to encrypt each character as shown below:

Plaintext

Key	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

For example, if a character in the plaintext is M, and the key is E, then the ciphertext of that character is Q, which is the intersection between column M and row E.

Another way of interpreting it is that we can take the position of plaintext M as 12, the position of key E as 4, therefore the position of the ciphertext is 12 + 4 = 16, which is Q. (Assuming the position of A is 0 and the position of Z is 25)

When the ciphertext goes beyond the letter Z, it returns to A.

A keyword that is more than one letter is used to encrypt the plaintext. The keyword will be written repeatedly under the plaintext to form the key, and then the plaintext can be encrypted accordingly.

For example, if the plaintext is “COMPUTING” and the keyword is “GREAT”, we may follow the steps below to encrypt the plaintext:

Plaintext	C	O	M	P	U	T	I	N	G
Position	2	14	12	15	20	19	8	13	6

Key	G	R	E	A	T	G	R	E	A
Position	6	17	4	0	19	6	17	4	0

Ciphertext	I	F	Q	P	N	Z	Z	R	G
Position	8	5	16	15	13	25	25	17	6

For this task, we may assume that we only use uppercase English letters (A – Z) to perform Vigenère cypher.

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:
```

Task 1.1

Write a function `substitution(char, key)` that:

- takes in an uppercase letter of the plaintext, `char`, and a character of the key which is also an uppercase letter, `key`
- returns the encrypted letter. [5]

Task 1.2

Write program code to:

- read in 2 uppercase letters from the user, which is a letter for the plaintext and a letter for the key
- call your function from **Task 1.1** with these letters
- output its encrypted letter. [2]

Test your program **two times**, with the following test data:

- `char = 'A', key = 'P'`
- `char = 'Y', key = 'C'` [2]

Task 1.3

Write a function `vigenere_cipher(plaintext, keyword)` that:

- takes in a string `plaintext`, which consists of uppercase letters
- takes in a string `keyword`, which consists of uppercase letters
- uses Vigenère cypher to encrypt `plaintext` using `keyword`
- uses the function from **Task 1.1** to encrypt each letter
- return the encrypted string, ciphertext. [4]

Test your function using the following data:

- `plaintext = 'HELLOWORLD', key = 'PYTHON'` [1]

Task 1.4

The text file `plaintext.txt` contains a message that needs to be encrypted using Vigenère cypher and then stored in a text file named `ciphertext.txt`

Write program code to:

- read the data from the text file `plaintext.txt`
- remove all non-alphabetical characters from the data and convert all the alphabets to uppercase letters
- read in a keyword from the user, ensuring that the user input is a valid keyword consisting of at least 2 alphabetical characters
- use your function from **Task 1.3** to encrypt the plaintext
- store the encrypted message, ciphertext in the text file `ciphertext.txt` [5]

Test your program with `plaintext.txt`

Show the contents of `ciphertext.txt` after you have run the program. [1]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as:

`TASK2_<your name>_<centre number>_<index number>.ipynb`

A programmer is writing a class, `LinkedList`, to represent a linked list of students. A linked list is a collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next.

Another class, `Person`, is written to represent a student as a node in the linked list.

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:
```

Task 2.1

The `Person` class represents a student as a node in the linked list and has the following attributes:

- `name` – a string containing the student's name
- `CG` – a string containing the student's civics group
- `ptr` – a pointer pointing to the next node (`Person`), it has a default value of `None`

The `Person` class has the following method:

- `display()` – outputs a string containing the information of the student in the following format:
  `Person with name <name> and CG <CG>.`

Write the `Person` class in Python. [3]

Task 2.2

Write the `LinkedList` class in Python. Use of a simple Python list is not sufficient. Include the following methods:

- `insert(name, CG)` inserts a `Person` with `name` and `CG` at the beginning (head) of the linked list
- `delete(name)` attempts to delete the `Person` with `name` from the linked list. Return `True` if the deletion is successful. If the `Person` was not present, display an appropriate message and return `False`
- `search(name)` searches for the `Person` with name from the linked list. If found, use the `display()` method from the `Person` class to display the information of the person and return `True`. If not found, return `False`
- `count()` should return the number of elements in the linked list, or zero if empty
- `display()` displays all the current contents in the linked list and uses the `display()` method from the `Person` class to display the information. [8]

Task 2.3

A Python subclass `Queue` uses `LinkedList` as its superclass.

The subclass `Queue` has the following methods:

- `enqueue(name, CG)` appends a `Person` with `name` and `CG` to the end of the queue
- `dequeue()` removes and returns the data item at the front of the queue. If the queue is empty, display an appropriate message and return `None`
- `display()` displays the current contents in the queue, using the `display()` method from the `Person` class to display the information, clearly labelling where the front and the end of the queue are.

Using appropriate inheritance and polymorphism, write program code for the subclass `Queue`. [6]

Test `Queue` by using the data in the file `students.txt`. Print the first five elements to be dequeued from the queue. [3]

Save your Jupyter Notebook for Task 2.

3 For Task 3.2 to 3.4, name your Jupyter Notebook as:

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

CoWorking is a local company that rents out meeting rooms to its members for co-working purposes. You are tasked to implement a database to aid the company in managing their room booking. The database will have three tables: a table to store data about the meeting rooms, a table about the members and a table about the bookings. The fields in each table are:

`MeetingRoom:`

- `RoomNo` – unique room number of the meeting room, for example, 101.
- `Capacity` – integer storing the number of people the meeting room can accommodate. This value should be in between 2 to 10 inclusive.
- `RatePerDay` – integer storing the rate of usage of the room per day.

`Member:`

- `MemberID` – member's unique number, for example, 70201.
- `FamilyName` – member's family name.
- `GivenName` – member's given name.
- `Membership` – member's membership type. The membership type should either be `'Elite'` or `'Normal'`.

`Booking:`

- `BookingID` – unique booking number, for example, 1.
- `MemberID` – the member's unique number.
- `RoomNo` – the unique room number.
- `DateBooked` – the date that the member booked the meeting room.
- `Released` – `TRUE` if the meeting room's booking has been released, or `FALSE` if it has not been released.

Note:

- You are required to include check condition for `Capacity`, `Membership` and `Released` fields.
- You may assume that one member will not book more than one room within the same day.

Task 3.1

Create an SQL file called `Task3_1.sql` to show the SQL code to create the database `coworking.db` with the three tables given. Define the primary and foreign keys for each table. [7]

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 3.2
         Program code
         Output:
```

Task 3.2

The data files `MeetingRoom.csv`, `Member.csv` and `Booking.csv` store the comma-separated values for each of the tables in the database.

Write a Python program to read in the data from each file and then store each item of the data in the correct place in the database. [5]

Task 3.3

Write a Python program to input a room number and return all the dates booked, family names and given names of the members who booked the room, and whether the booking has been released.

Print appropriate message if there is no booking for the room number. [5]

Test your program **two times**, using the room number 104 and 105. [2]

Task 3.4

Write a Python program to input a member's ID and the family name that they intend to update to, then change the family name of the selected member in the database. You may assume that the user will input a valid member's ID. [3]

Test your program by running the application and change the family name of the member with ID 70207 from Kacey to Caleb. [1]

Save your Jupyter Notebook for Task 3.

- End of Paper -