HWA CHONG INSTITUTION
C2 PRELIMINARY EXAMINATION 2020

COMPUTING
Higher 2

26 AUG 2020 **Paper 2 (9569 / 02)** **1300 -- 1600 hrs**

Additional Materials:

Electronic version of `BUNS.TXT` data file

Electronic version of `CAKES.TXT` data file

Electronic version of `COUNTRY1.TXT` data file

Electronic version of `COUNTRY2.TXT` data file

Electronic version of `COVID19.TXT` data file

Electronic version of `LOAVES.TXT` data file

Electronic version of `QUEUE.TXT` data file

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

The maximum mark for this paper is **100**.

The number of marks is given in brackets [ ] at the end of each question or part question.

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.

This document consists of **11** printed pages.

1. The file `COVID19.TXT` stores the cumulative total number of confirmed covid-19 cases in Singapore from 14 April 2020 till 15 May 2020.

   A sample record is:

   ```
   1704,5050
   ```

   This means that as of 17 April 2020, a cumulative total of 5050 confirmed covid-19 cases were reported.

   The task is to determine the dates with the highest and lowest number of the confirmed new covid-19 cases in Singapore and the longest ascending streak during the period.

   Note: You cannot use the built-in `sort()`, `min()` and `max()` functions.

   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:

   Task 1.1

   Write program code to:

   - read the data from the text file
   - calculate the number of new covid-19 cases per day (from 15 April 2020 to 15 May 2020)
   - output the dates with the highest and lowest numbers of new covid-19 cases, excluding the first day on 14 April 2020.
   - If more than one dates exists with the highest/lowest number of new covid-19 cases, all dates are reported.

   Sample output:

   ```
   Highest # cases (1426) is on 20 April 2020
   Lowest # cases (447) is on 15 April 2020, 2 May 2020
   ```

   [9]

Task 1.2

Write program code to:

- Determine the ascending streak of new covid-19 cases
- output the longest ascending streak of new cases across the period from 15 April 2020 to 15 May 2020 inclusive.

Date	Cummulative Total Cases	New Cases	
1404	3252		
1504	3699	447	2-day ascending streak
1604	4427	728	
1704	5050	623	2-day ascending streak
1804	5992	942	
1904	6588	596	
.	.	.	
.	.	.	
.	.	.	
0305	18205	657	
0405	18778	573	
0505	19410	632	3-day ascending streak
0605	20198	788	
0705	20939	741	
.	.	.	
.	.	.	
1205	24671	884	
1305	25346	675	
1405	26098	752	3-day ascending streak
1505	26891	793	

Sample output:

```
Longest ascending streak is 3 days
```

[6]

Download your program code and output for Task 1 as
`TASK1_<your_name>_<centre number>_<index number>.ipynb`

2. The file `COUNTRY1.TXT` stores a list of country names.

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
```

In [2]:

```
# Task 2.2
Program Code
```

output:

In [3]:

```
# Task 2.3
Program Code
```

output:

In [4]:

```
# Task 2.4
Program Code
```

Task 2.1

Using the country name, a hash address is calculated from a hashing function as follows:

- the ASCII code is calculated for each character (in lowercase) within the country name
- the total of ASCII values is calculated
- the total is divided by 30 and the remainder is the hash address for this country

For example, the hash address for `Brazil` is 14

Write program code for the hashing function using the following specification

```
FUNCTION HashKey (Country: STRING): INTEGER
```

This function has a single parameter `Country` and returns the hash address as an integer.

[3]

Task 2.2

The hash table is implemented as a list with 30 elements. Elements are written to and read from the hash table using the above hash function with the country name as the input parameter.

Write program code which does the following:

- Read all country names from `COUNTRY1.TXT`
- Use the function `HashKey` to calculate the hash address for each country
- Store each country name in the hash table

You must ensure that when a collision occurs, your program design will deal with this situation by searching sequentially from the calculated hash address for an empty location and storing the country name at this empty location. The program will generate an error message if the hash table is full. [7]

Task 2.3

Write program code for a procedure `SearchCountry` using the following specification

```
PROCEDURE SearchCountry (Country: STRING, HashTable: LIST)
```

This procedure has two parameters, `Country` and `HashTable`, and does the following:

- Calculate the hash address for the country
- Locate the country name in the hash table
- Report whether or not this country name was found. If found, also output the address of the hash table where the country was found.

You must ensure that your program can deal with collision when searching. [6]

Devise a set of **three** test cases with the country to be used. [3]

Task 2.4

The file `COUNTRY2.TXT` stores a list of countries with their corresponding numbers of confirmed cases and death cases of COVID19 pandemic on May 16, 2020.

Each record has the following format:

```
<country>,<no. of confirmed cases>,<no. of death case>
```

A sample record is

```
Brazil,233142,15633
```

Death rate for each country is calculated as the number of death cases / the number of confirmed cases. This rate is output as a percentage to 1 decimal place. For the example above, the death rate of Brazil is 15633/233142 = 6.7%.

Write program code which does the following:

- Read the data from `COUNTRY2.TXT`
- Implement **bubble sort** to arrange the countries from highest to lowest death rate
- Generate a text file `RATE.TXT` to display the list of countries and their death rate. Each record has the format `<country>,<death rate>`.

If two countries have the same death rate, their order in the list does not matter. [9]

Download your program code and output for Task 2 as

`TASK2_<your_name>_<centre number>_<index number>.ipynb`

3. A bakery currently keeps records on paper of all its products it sells in the shop. It decided to trial a database to manage its products. It is expected that the database should be normalized.

The following information of the bakery is recorded:

- `ProductCode` – Unique code of the item
- `Name` – Name of the item
- `Type` – The type of product – `'Cake', 'Loaf', 'Bun'`
- `Location` – The location at which the product is made – `'North', 'South', 'East', 'West'` Kitchen
- `Price` – The selling price of the product

For cakes, the following additional information is recorded:

- `ServingSize` – the estimated number of servings per cake
- `Shape` – Shape of the cake – `'Square', 'Circle', 'Roll'`

For loaves, the following additional information is recorded:

- `Weight` – weight of loaf in gram

For buns, the following additional information is recorded:

- `PiecesPerPackage` – number of pieces per package

The information is to be stored in four different tables:

```
Product
Cake
Loaf
Bun
```

Task 3.1

Create a SQL file named `Task3_1_<your_name>_<centre number>_<index number>.sql` to show the SQL code to create the database `bakery.db` with the four tables. The table, `Product`, must use the `ProductCode` as its **primary key**. The other tables must refer to the `ProductCode` as a **foreign key**.

Save your SQL code as

`Task3_1_<your_name>_<centre number>_<index number>.sql` [4]

Task 3.2

The files `CAKES.TXT`, `LOAVES.TXT` and `BUNS.TXT` contain information about the bakery's cakes, loaves and buns respectively for insertion into the bakery database. Each row in the three files is a comma-separated list of information about a single product.

For `CAKES.TXT`, the information about each cake is given in the following order:

```
ProductCode, Name, Location, Price, ServingSize, Shape
```

For `LOAVES.TXT`, the information about each loaf is given in the following order:

```
ProductCode, Name, Location, Price, Weight
```

For `BUNS.TXT`, the information about each bun is given in the following order:

```
ProductCode, Name, Location, Price, PiecesPerPackage
```

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

Save your program code as

`Task3_2_<your_name>_<centre number>_<index number>.py` [6]

Task 3.3

Write SQL code to show the `ProductCode`, `Name`, `Location`, `Price` and the `ServingSize` of each cake with `'Circle'` Shape. Run the query.

Save this code as

`Task3_3_<your_name>_<centre number>_<index number>.sql` [4]

Task 3.4

The bakery wants to filter the products 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 details of the product based on the `Location` in ascending order of `Price`. The table will display the following columns: `Name, Type, Price.`

Save your Python program as

`Task3_4_<your_name>_<centre number>_<index number>.py`

with any additional files/sub-folders as needed in a folder named

`Task3_4_<your_name>_<centre number>_<index number>`

Run the web application. Save the output of the program when `'North'` is entered as the `Location` as

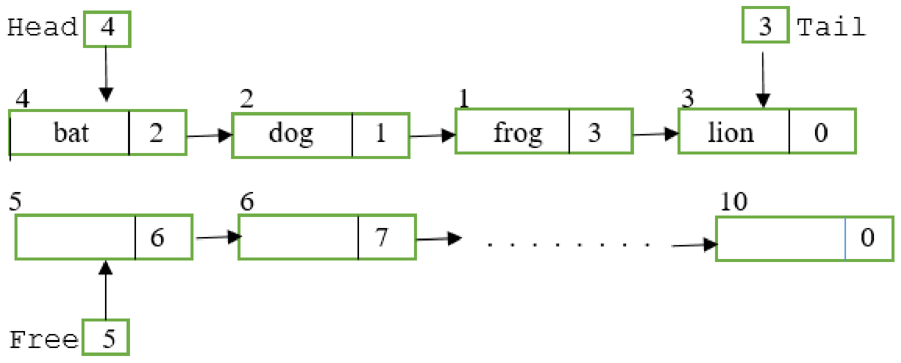
`Task3_4_<your_name>_<centre number>_<index number>.html` [10]

4. A programmer is writing a program to manipulate different data structures using Object-Oriented Programming.

The superclass, `LinkedStructure`, will store the following data:

- A linear linked list of data items held in an **array** of size 10. Array index starts at 1.
- Head pointer, pointing to the first element in the linked list
- Tail pointer, pointing to the last element in the linked list
- Free pointer, pointing to the first element in the free list

The diagram shows the linked structure after four items have been added and the unused nodes are linked together.



This superclass has the following methods:

- `Initialise()` method sets up an empty linked list. Should link all nodes to form the free list. Initialise values for head pointer, tail pointer and free pointer
- `Add(item)` appends the parameter into its correct **alphabetical** order in the linked list.
- `Remove(item)` removes the parameter from the ordered linked list
- `Display()` displays the data items in the linked list in alphabetical order
- `PrintStructure()` displays the current state of all pointers and the array contents in index order
- `IsEmpty()` tests for empty linked list
- `IsFull()` tests for no unused nodes

The superclass is used to implement a linear queue.

The subclass `Queue` has the following methods:

- `Add(item)` appends the parameter to the queue and overrides the `LinkedStructure` add method .
- `Remove()` returns and removes the next item in the queue
- `Display()` method should display the queue contents in order (e.g. the earliest added item first) and should override the `LinkedStructure` display method.

Each method updates its appropriate pointers, and produces suitable errors (or returns different values) to indicate if the actions are not possible, e.g. if the structure is empty.

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
```
output:

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

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

Task 4.1

Write program code for the superclass `LinkedStructure`. [20]

Task 4.2

Write program code to:

- create a `LinkedStructure` object
- add the following three data items in the order shown to the ordered linked list
  `Japan, Singapore, China`
- output all pointers and array contents using the `PrintStructure()` method after adding the items
- output the current contents of the linked list using the `Display()` method
- remove two data items `China, Japan` in that order from the linked list
- output all pointers and array contents using the `PrintStructure()` method after the removal of the items.

[5]

Task 4.3

Write program code for the subclass `Queue`.

Use appropriate inheritance and polymorphism in your designs. [5]

Task 4.4

The file `QUEUE.TXT` stores data to test your program.

Write program code to:

- create a new queue and add the data in the file `QUEUE.TXT` to the queue
- output the current contents of the queue
- remove and output two items from the queue
- output all pointers and the array contents of the queue after the removal of the items.

All outputs should have appropriate messages to indicate what they are showing. [3]

Download your program code and output for Task 4 as
`TASK4_<your_name>_<centre number>_<index number>.ipynb`