Name:_________________________________ **Class:**____________

JURONG PIONEER JUNIOR COLLEGE

JC2 Preliminary Examination 2023

COMPUTING
Higher 2

9569/02

30 August 2023

Paper 2 (Practical)

3 hours

Additional materials:
Cover Page
Electronic version of `Task1_3.txt` data file
Electronic version of `Task2_2.txt` data file
Electronic version of `Task4_2_1.txt` data file
Electronic version of `Task4_2_2.txt` data file
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** the 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 task.

The total number of marks for this paper is 100.

This document consists of **8** printed pages.

[Turn over

Instructions to candidates:
Your program code and output for each of Task 1 to 4 should be downloaded in a single`.ipynb` file. For example, your program code and output for Task 1 should be downloaded as `TASK1_<your class>_<your name>.ipynb`.

1 Name your **Jupyter Notebook** as:
`TASK1_<your class>_<your name>.ipynb`

You are working as a data analyst for a research laboratory, where scientists conduct experiments and generate large datasets. To ensure accurate analysis, you need to develop a Scientific Data Analysis program that efficiently handles datasets, performs linear search to find outliers, removes them from the data, and then utilizes Quicksort to sort the remaining data for further analysis.

Task 1.1

Implement a function `linear_search_outliers(Data, Maximum)` that takes a list of **unique** numerical data and a numerical maximum value as input.

The function should perform a **linear search** to identify outliers **larger** than `Maximum` in `Data` and return a list containing the indices of the outliers. [5]

Task 1.2

Copy and paste the code in `Task1_2.txt` to initialise a list of numerical data in your main program.

Thereafter, use the indices from `linear_search_outliers` to remove the outliers larger than **90.0** from the list. You may use a not-in-place algorithm. [4]

Task 1.3

Implement a **recursive** function `quicksort(Data)` that takes a list of numerical data as input, arranges the data in ascending order and return the sorted list. There are no duplicate values in the input data set. [6]

Task 1.4

In the main program, use `quicksort` to sort the list of numerical data without outliers from Task 1.2, and display the numerical data in the sorted list. [2]

Save your Jupyter Notebook for **Task 1**.

2 Name your **Jupyter Notebook** as:
`TASK2_<your class>_<your name>.ipynb`

You are developing a Library Management System for a college library to efficiently manage book borrowing processes. The system will use a hash table with linear probing to handle hash collisions, and the ISBN (International Standard Book Number) will be encoded to generate the hash table index using a hash function.

The hash table array can store up to **53** book record objects and collisions are handled by linear probing.

Task 2.1

Implement `hash_function(ISBN)` that takes the ISBN as a digit-string input. It calculates the integer sum of its ASCII values, perform modulo 53 to the sum and returns the remainder that will be used as the hash value.
For example, `hash_function(“0205080057”)` returns 30. [4]

Task 2.2

Implement the following `Book_Record` class:

`Book_Record`

`ISBN: STRING`
`Title: STRING`
`Author: STRING`
`Due_Date: STRING`

`Constructor(ISBN, Title, Author, Due_Date)`
`Get_ISBN(): STRING`
`Get_Title(): STRING`
`Get_Author(): STRING`
`Get_Due_Date(): STRING`
`Set_Due_Date(new_due_date)`
`to_string(): STRING`

The `to_string()` method returns a string containing the values of the four attributes separated by a comma and a space. [5]

Task 2.3

Write a Python program to:

- create a hash table array `hta`,
- reads book records from a text file `Task2_3.txt`, where each line in the file contains book information in the format: `ISBN,Title,Author,Due_Date` and create `Book_Record` objects,
- use `hash_function` to generate the hash value for each `Book_Record` object, and use the value to insert the `Book_Record` object into the `hta`, and
- use **linear probing** to handle collision.

[6]

Task 2.4

Implement a function `search_book_record(hta)` that allows the library staff to find a book's information by entering its ISBN. The function should prompt the user to input the ISBN of the book to be searched, retrieve the `Book_Record` object using **hash table search** and return its information from `to_string()` method, or “`Book not on loan`” if not found. [4]

Task 2.5

Write the program to search and display the information for the following ISBNs: [2]

```
0205080057
1234567890
```

Task 2.6

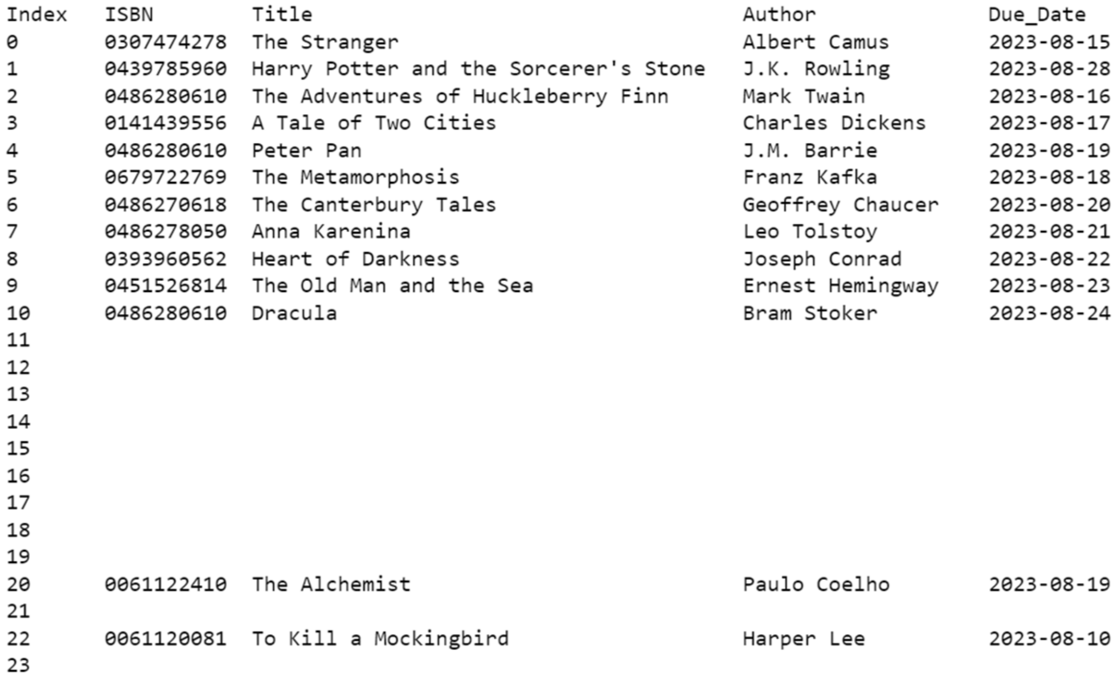
Implement a procedure `update_book_record(hta)` that allows the library staff to update a book's due date by entering its ISBN. The procedure should prompt the user to input the ISBN of the book to be updated, retrieve the `Book_Record` object from the hash table and update its `Due_date`. You may assume the ISBN entered will exist in the hash table. [4]

Task 2.7

Write the program to:

- update the due date of the book with ISBN `0679760806` to `2023-09-01`
- display the updated hash table with index and book information in neat columns.

The following is a sample partial output:



```
Index   ISBN        Title                                   Author              Due_Date
0       0307474278  The Stranger                            Albert Camus        2023-08-15
1       0439785960  Harry Potter and the Sorcerer's Stone   J.K. Rowling        2023-08-28
2       0486280610  The Adventures of Huckleberry Finn      Mark Twain          2023-08-16
3       0141439556  A Tale of Two Cities                    Charles Dickens     2023-08-17
4       0486280610  Peter Pan                               J.M. Barrie         2023-08-19
5       0679722769  The Metamorphosis                       Franz Kafka         2023-08-18
6       0486270618  The Canterbury Tales                    Geoffrey Chaucer    2023-08-20
7       0486278050  Anna Karenina                           Leo Tolstoy         2023-08-21
8       0393960562  Heart of Darkness                       Joseph Conrad       2023-08-22
9       0451526814  The Old Man and the Sea                 Ernest Hemingway    2023-08-23
10      0486280610  Dracula                                 Bram Stoker         2023-08-24
11
12
13
14
15
16
17
18
19
20      0061122410  The Alchemist                           Paulo Coelho        2023-08-19
21
22      0061120081  To Kill a Mockingbird                   Harper Lee          2023-08-10
23
```

[2]

Save your Jupyter Notebook for **Task 2**.

3 Name your **Jupyter Notebook** as:
`TASK3_<your class>_<your name>.ipynb`

The linked list is implemented as a collection of nodes in object-oriented programming.

The `Node` class contains two properties:

- `data` is the data in the node
- `next` points to the next node

A stack, used to store string values, is implemented using a linked list.

The `Stack` class contains one property:

- `top` is a pointer to the node at the top of the stack.

The `Stack` class contains the following methods:

- constructor to set `top` to `None`,
- `push(word)` will insert `word` to the top of the stack,
- `pop()` will remove and return the top element in the stack, and
- `to_string()` returns a string containing the elements starting from the top, separated by a comma and a space, e.g.: in the form: `apple, orange, pear`

Task 3.1

Write the `Node` class and the `Stack` class.

[10]

Task 3.2

Write program code to:

- declare a new instance of `Stack`
- store each value from the following list `lst` as a new node in the stack,
  `lst = ['plane','bus','car','train','yacht','ship']`
- print the resulting content in the stack using the `to_string()` method
- print the first three elements to be popped from the stack

[3]

Task 3.3

A queue, used to store string values, is also implemented using a linked list.

The `Queue` class contains one property:

- `head` is a pointer to the node at the head of the queue.

The `Queue` class contains the following methods:

- constructor to set `head` to `None`,
- `enqueue(word)` will insert `word` to the end of the queue,
- `dequeue()` will remove and return the first element in the queue, and
- `to_string()` returns a string containing the elements starting from the head, separated by a comma and a space.

Write the `Queue` class.

[9]

Task 3.4

Write program code to:

- declare a new instance of `Queue`
- store each value from the following list `lst` as a new node in the queue,
```
     lst = ['plane','bus','car','train','yacht','ship']
```
- print the resulting content in the queue using the `to_string()` method
- print the first three elements to be dequeued from the queue [3]

Save your Jupyter Notebook for **Task 3**.

4 Name your **Jupyter Notebook** as:
`TASK4_<your class>_<your name>.ipynb`

The upcoming Merlion Theme Park wishes to develop a web application to allow its visitors to buy park entry tickets online. The database will have two tables: a table to store data about the tickets and a table about the sales. The fields in each table are:

`Ticket:`

- `tDate`: a **unique** string date of the format YYYY-MM-DD assigned to the ticket.
- `dayOfWeek`: the day of week of the ticket's date e.g.: Monday, Tuesday, …
- `unitPrice`: the unit price of a ticket in Singapore dollars.

  Ticket is priced at $40 for weekday, and $60 for weekend, public holiday, and school holiday.
- `totQuan`: the total quantity of tickets for the date.
- `availQuan`: the quantity of tickets still available for the date.

`Sale`:

- `saleID`: a **unique** autoincrement integer ID assigned to the sale.
- `tDate`: the ticket's **unique** string date.
- `quan`: the quantity of ticket in the sale.
- `totalPrice`: the total price of the sale.

Task 4.1

Write a Python program that uses SQL code to create the database `MerlionThemePark` with the two tables given. Define the primary and foreign keys for each table. [5]

Task 4.2

The text files `Task4_2_1.txt` and `Task4_2_2.txt` 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 data in the correct place in the database. [5]

Task 4.3

Write a Python program to input a month and output the ticket information of all the dates of that month, displayed in columns with header. [5]

Test your program by running the application with the month 11. [1]

Save your Jupyter Notebook for Task 4.

Task 4.4

Write a Python program and the necessary files to create a web application that:

- has a **home page** for visitors to
  - input month into a textbox, and
  - click on submit button,

- on the next page, the **ticket page**, visitors can
  - view the ticket information (in a table with headers: Date, Day of Week, Unit Price $, and Available Quantity) according to the month entered in the previous page,
  - input month (MM), day (DD) and quantity of tickets into three textboxes, and
  - click on submit button to buy,

- assume the visitors' input for month and day are valid

- if quantity is within the quantity available, visitors can view the **sale confirmation page** containing the date of ticket, quantity, total price, and a message, "Transaction is successful".

- if the quantity exceeds the quantity available, visitors will view a **notification page** with the message, "Insufficient quantity. Transaction unsuccessfully."

For a successful sale (valid input for month, day and quantity), the `Ticket` and `Sale` tables are to be updated as follows:

- `availQuan` in `Ticket` table to be decreased accordingly, and

- a new record is created in the `Sale` table.

Save your Python program as:
`TASK_4_4_<your class>_<your name>.py`

with any additional files/ subfolders in a folder named:
`TASK_4_4_<your class>_<your name>`

Run the web application using the following inputs to obtain a successful ticket sale,
month = 11, day = 02, quantity = 10 [14]

Save the **sales confirmation page** of a successful ticket sale as:
`TASK_4_4_<your class>_<your name>.html` [1]

END OF PAPER