**HWA CHONG INSTITUTION**
**C2 PRELIMINARY EXAMINATION 2023**

**COMPUTING**
**Higher 2**

**23 AUG 2023** **Paper 2 (9569 / 02)** **1400 -- 1700 hrs**

---

**Additional Materials:**

Electronic version of `MAZE.txt` data file

Electronic version of `PERSON.txt` data file

Electronic version of `CHESS.csv` data file

Electronic version of `DONUT.txt` data file

Electronic version of `MEMBER.txt` data file

Electronic version of `SALE.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 4.3 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`

A programmer creates a remote-controlled robot and wants to find out how many steps it takes to exit a maze.

The maze is represented by a 6 by 6 square grid. Each position in the grid is represented by a pair of coordinates. The top left square display has x = 0 and y = 0.

The robot moves left, right, up or down according to a direction entered. The following are valid inputs:

| Input character | Action |
|---|---|
| 'U' | Robot moves up |
| 'D' | Robot moves down |
| 'L' | Robot moves left |
| 'R' | Robot moves right |
| '' (empty string) | Continue with previous move.<br>If no previous move, do nothing |

When a direction is entered, the robot moves one position in that direction. After the robot moves, the position it was previously on is replaced by a 'X'. The robot cannot move to the same spot twice. If the direction would place the robot on a wall or a position previously stepped on, the robot does not move. The maze is displayed after each move.

The robot is denoted by 'T', the walls '#' and empty space '.'.

```
# # # # T #
# . # . . #
# . . . # #
# # . # . #
# . . . . #
# # # # # #
```

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**

Using the maze given in `MAZE.txt`, write program code to:

- read the maze from the text file and store it in a suitable array structure
- randomize the exit along the last row of the maze
- update the exit square on the grid with a '.'
- display the maze when the robot is in its initial position at x = 4 and y = 0.

[6]

Test the program and show the output.

**Task 1.2**

Add to your program code to:

- take in and validate a direction
- calculate a new position
- check if this position is an empty space ('.')
- update the grid so that the previous position of 'T' is replaced with a 'X' and the robot is located in its new position
- display the maze
- continue this until the robot is moved to the exit of the maze
- when robot is at the exit, the number of steps taken is displayed.

[14]

Test run the program.

Below shows part of a sample run.

```
# # # # T #
# . # . . #
# . . . # #
# # . # . #
# . . . . #
# # # # . #

Enter direction ('U','D','L','R',''): D
# # # # X #
# . # . T #
# . . . # #
# # . # . #
# . . . . #
# # # # . #

Enter direction ('U','D','L','R',''): R
Can't go there!

# # # # X #
# . # . T #
# . . . # #
# # . # . #
# . . . . #
# # # # . #

Enter direction ('U','D','L','R',''): L
# # # # X #
# . # T X #
# . . . # #
# # . # . #
# . . . . #
# # # # . #

      .
      .
      .
      .
      .
      .
      .

Enter direction ('U','D','L','R',''): R
# # # # X #
# . # X X #
# . X X # #
# # X # . #
# . X T . #
# # # # . #
```

```
Enter direction ('U','D','L','R',''):
# # # # X #
# . # X X #
# . X X # #
# # X # . #
# . X X T #
# # # # . #

Enter direction ('U','D','L','R',''): D
# # # # X #
# . # X X #
# . X X # #
# # X # . #
# . X X X #
# # # # T #

The robot takes 9 moves to exit the maze.
```

Save your Jupyter Notebook for Task 1.

**2.** Name your Jupyter Notebook as:

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

This task is to perform sorting algorithms on `Person` objects held in a 1-dimensional array.

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 class `Person` contains two properties:

- `name` - stored as a string
- `age` - stored as an integer

Write program code to declare the class `Person` and its constructor and `print()` method to output the `name` and `age` of a `Person` object.

[2]

**Task 2.2**

Write a function `task2_2(filename)` that:

- takes a string `filename` which represents the name of a text file
- reads in the contents of the text file
- returns the content as a list of `Person` objects.

Call the function `task2_2` with the file `PERSON.txt` and print `Person` objects using the following statements:

```
list_of_person = task2_2('PERSON.txt')
for person in list_of_person:
    person.print()
```

[4]

**Task 2.3**

One method of sorting is the insertion sort.

Write a function `task2_3(list_of_person, key, order)` that:

- accepts three parameters:
  - `list_of_person` contains a list of `Person` objects
  - `key` should be one of the values:
    - `name` – list to be sorted by name
    - `age` – list to be sorted by age
  - `order` should be one of the values:
    - `asc` – list to be sorted by `key` in ascending order
    - `desc` – list to be sorted by `key` in descending order
- sorts `list_of_person` by `key` in `order` using **insertion sort**.

Call the function `task2_3` with the contents of the file `PERSON.txt` and print the sorted `Person` objects using the following statements:

```
list_of_person = task2_2('PERSON.txt')
task2_3(list_of_person, 'name', 'asc')
for person in list_of_person:
    person.print()
```

[8]

**Task 2.4**

Another method of sorting is the quick sort.

Write a function `task2_4(list_of_person, key, order)` that:

- accepts three parameters:
  - `list_of_person` contains a list of `Person` objects
  - `key` should be one of the values:
    - `name` – list to be sorted by name
    - `age` – list to be sorted by age
  - `order` should be one of the values :
    - `asc` – list to be sorted by `key` in ascending order
    - `desc` – list to be sorted by `key` in descending order
- sorts `list_of_person` by `key` in `order` using **quick sort**.

Call the function `task2_4` with the contents of the file `PERSON.txt` and print the sorted `Person` objects using the following statements:

```
list_of_person = task2_2('PERSON.txt')
task2_4(list_of_person, 'age', 'desc')
for person in list_of_person:
    person.print()
```

[8]

**Task 2.5**

Write a function `task2_5(list_of_person, method, key, order)` that:

- accepts four parameters:
  - `list_of_person` contains a list of `Person` objects
  - `method` should be one of the values:
    - `insertion sort` – sort the list using insertion sort
    - `quick sort` – sort the list using quick sort
  - `key` should be one of the values:
    - `name` – list to be sorted by name
    - `age` – list to be sorted by age
  - `order` should be one of the values:
    - `asc` – list to be sorted by `key` in ascending order
    - `desc` – list to be sorted by `key` in descending order
- sorts `list_of_person` by `key` in `order` using `method`.

Call the function `task2_5` with the contents of the file `PERSON.txt` and print the sorted `Person` objects using the following statements:

```
list_of_person = task2_2('PERSON.txt')
task2_5(list_of_person, 'quick sort', 'name', 'desc')
for person in list_of_person:
    person.print()
```

[2]

Save your Jupyter Notebook for Task 2.

**3.** Name your Jupyter Notebook as:

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

A chess club wants to keep a record of players who registered for a team chess competition. The record is implemented using Object-Oriented Programming (OOP).

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
```

Output:

**Task 3.1**

The class `Player` is created with the following attributes:

- `name`, the name of the player
- `elo`, an integer representing the elo rating of the player
- `ptr` pointer, an integer pointing to the index of the next lower elo rating `Player` in the list

The class `PlayerList` contains three properties:

- `data` array of size `n`, with each element being a `Player` object
- `head` pointer, an integer pointing to the index of the first element in the linked list
- `free` pointer, an integer pointing to the index of the first element in the free list

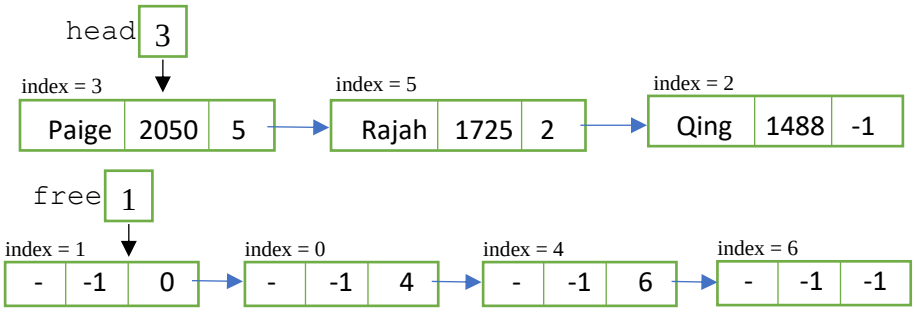
The class `PlayerList` is created with the following methods:

- a constructor to set `head` to `-1`, `free` to `0`, and creates the `data` array with `n` empty `Player` nodes indicated with `name` set to '`-`' and `elo` set to `-1`
- `size()` method which returns the number of registered players
- `register(name, elo)` method which registers a player with `name` and `elo`, outputting a suitable error message when the `data` array is full
- `withdraw(name)` method which removes `name` from `PlayerList`, displaying an error message if `name` is not found
- `display()` method which displays the value of the `head` pointer, the value of the `free` pointer, and the contents of the `data` array in array index order.

As an example, 3 players are stored in the `data` array of size 7 as follows:

`data`

| | name | elo | ptr |
|---|---|---|---|
| 0 | - | -1 | 4 |
| 1 | - | -1 | 0 |
| 2 | Qing | 1488 | -1 |
| 3 | Paige | 2050 | 5 |
| 4 | - | -1 | 6 |
| 5 | Rajah | 1725 | 2 |
| 6 | - | -1 | -1 |




Write the program code for the `Player` class and `PlayerList` class. [20]

**Task 3.2**

The players who registered for the chess team have their name and elo rating recorded in a file named `CHESS.csv`.

Write program code to:

- create a `PlayerList` object, `cteam`, that accepts registration for up to 7 players
- read `CHESS.csv` and register them into `cteam`
- use `display()` to show the list of the players
- remove the player named `Taylor` from the team
- print the size of the team
- use `display()` to show the list of the players who are still in the team

[5]

Save your Jupyter Notebook for Task 3.

**4.** Name your Jupyter Notebook as:

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

A donut store owner currently keeps paper records about its members, donuts on sale and the purchase records by members. The store owner wants to create a suitable database to store the data and to allow them to run searches for specific data. The database will have three tables: a table to store data about the donuts, a table about the members and a table about the sales. The fields in each table are:

`Donut`:

- `DonutID` – donut's unique number, for example, 5
- `DonutName` – donut's name
- `UnitPrice` – price of one donut

`Member`:

- `MemberNumber` – member's unique number, for example, 101
- `MemberName` – member's name
- `Phone` – member's contact number

`Sale`:

- `SaleID` – the purchase's unique number, for example, 1030
- `MemberNumber` – the member's unique number
- `DonutID` – the donut's unique number
- `Date` – the date that the member purchased the donut, for example, `'20230720'`
- `Quantity` – the number of donuts purchased

For each of the sub-tasks 4.1 to 4.3, 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
```

Output:

**Task 4.1**

Write a Python program that uses SQL code to create the database `STORE` with the three tables given. Define the primary and foreign keys for each table. [4]

**Task 4.2**

The text files `DONUT.txt`, `MEMBER.txt`, and `SALE.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. [3]

**Task 4.3**

Write a Python program to input a member's number and display

- the member's name,
- a table tabulating the donut names, dates and quantity of all the sales from this member

Test your program by running the application with the member number 104. [6]

Save your Jupyter Notebook.

**Task 4.4**

The store owner wants to filter the purchases by `Date` and display the results in a web browser.

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

- receives a `Date` string from an HTML form,
- returns an HTML document that enables the web browser to display a table tabulating the names and the total quantity of each donut sold on that date, in descending order of the total quantity.

Save your Python program as

`Task4_4_<your name>_<centre number>_<index number>.py`

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

`Task4_4_<your name>_<centre number>_<index number>`

Run the web application with the date entered as '`20230721`'. Save the output as `Task4_4_<your name>_<centre number>_<index number>.html`

[12]