

NATIONAL JUNIOR COLLEGE
Mathematics Department

General Certificate of Education Advanced Level
Higher 2

COMPUTING

Paper 2 (Lab-based)

9569/2

1 Sep 2025

3 hours

Additional Materials:

Electronic version of `candidates.txt` data file

Electronic version of `birthdays.txt` data file

Electronic version of `task4.db` database file

Electronic version of `style.css` file

Electronic version of `query_form.html` file

Electronic version of `results.html` 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 **4** marks out of 100 will be awarded for the use of common coding standards for programming style.

The number of marks is given in the brackets [] at the end of each question or part question.
The total number of marks for this paper is 100.

This document consists of 15 printed pages and 1 blank pages

Instructions to candidates:

Copy the folder from the thumb drive to the PC's desktop and rename the folder on the desktop to `<your name>`. (For example, TanKengHan). All the resource files are found in the folder and you should work on the folder in the desktop.

Your program code and output for each of Task 1 to 3 should be saved in a single `.ipynb` file.

For example, your program code and output for Task 1 should be saved as

`Task_1_<your name>.ipynb`.

You should have a total of **three** `.ipynb` files to submit at the end of the paper.

At the end of the exam, copy the working folder on your desktop to the thumb drive.

1 Name your Jupyter Notebook as `Task1_<your name>.ipynb`.

The task is to use Object-Oriented Programming (OOP) to implement two data structures that store their data using a fixed-size array:

- A queue data structure follows the First-In-First-Out (FIFO) principle, meaning that elements are removed in the same order in which they were added.
- A stack data structure follows the Last-In-First-Out (LIFO) principle, meaning that the most recently added element is the first one to be removed.

For each sub-task, 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

The class `Queue` contains the following attributes:

- `size`: the maximum number of items that can be inserted into the queue.
- `data`: a one-dimensional array of objects.
- `front`: the index in the `data` array that points to the first item in the queue.
- `back`: the index in the `data` array that points to the last item in the queue.
- `in_use`: the number of items currently in the queue.

The class `Queue` has the following methods:

- `constructor`: takes an integer, `n`, as a parameter and initialises the array data with `n` number of `None` values. It also initialises the attributes `front`, `back`, and `in_use` to appropriate values for an empty queue
- `is_empty()` : returns `True` if the queue is empty, `False` otherwise.
- `is_full()` : returns `True` if the queue is full, `False` otherwise.
- `enqueue(item)` : takes an `item` as a parameter and inserts it into the queue. If the queue is full, prints an error message “Queue full”.
- `dequeue()` : removes the first item in the queue and returns the removed item. If the queue is empty, prints an error message “Queue empty” and return `None`.
- `__repr__()` , returns a string representation of the items in the queue in the format:

[<item_1>, <item_2>, ... <item_n>], where <item_1> is the data item at the front of the queue and <item_n> is at the back of the queue. Only items in the queue should be displayed.

Write Python code to implement the class `Queue` as a **circular queue**, based on the attributes and methods described above.

[15]

Task 1.2

Write Python code to test the functionality of the class `Queue` as follows:

- create an object instance of the class `Queue` with a size of **three**.
- insert **four** items into the queue using the `enqueue` method and print the contents of the queue.
- remove **two** items from the queue using the `dequeue` method and print the items that was removed.
- insert **one** more item to the queue and then print the contents of the queue. [1]

Task 1.3

In this task, you are to implement a **stack** data structure using **two queue objects**. This means that the two stack operations, push and pop, must be implemented **using only the operations provided by queues**. The stack should behave exactly as a normal stack would (LIFO), but internally, items must be stored and accessed using the two queues.

The class `Stack` is to be implemented with the following attributes:

- `size`: the maximum number of items that can be inserted into the stack.
- `queue_1`, `queue_2`: instances of the `Queue` class as implemented in Task 1.1
- `in_use`: the number of items currently in the stack.

The class `Stack` has the following methods:

- `constructor`: initialises the attributes of the class `Stack`.
- `is_empty()`: returns `True` if the stack is empty, `False` otherwise.
- `is_full()`: returns `True` if the stack is full, `False` otherwise.
- `push(item)`: takes an `item` as a parameter and inserts it into the stack. If the stack is full, prints an error message "Stack full".
- `pop()`: removes the item at the top of the stack and returns it. If the stack is empty, prints an error message "Stack empty" and return `None`.

- `__repr__()` : returns a string representation of the items in the stack in the format:

`[<item_1>, <item_2>, ... <item_n>]`, where `<item_1>` is at the top of the stack and `<item_n>` is at the bottom. Only items in the stack should be displayed.

Write Python code to implement the `Stack` class. In this task,

- you are NOT ALLOWED to add any new attributes.
- No additional storage space is allowed (e.g. creating a new list) to solve the problem.
- You are NOT ALLOWED to access the attributes of the `Queue` class directly. Only the methods provided by the `Queue` class are allowed to be used.

[12]

Task 1.4

Write Python code to test the functionality of the class `Stack` as follows:

- create an object instance of the class `Stack` with a size of **three**.
- insert **four** items into the stack using the `push()` method and print the contents of the stack.
- remove **one** item from the stack using the `pop()` method and print the item that was removed.
- insert **one** more item into the stack and then print the contents of the stack. [1]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as `Task2_<your name>.ipynb`

A military special forces unit is organised into squads, each consisting of members with **unique** roles. Every year, new candidates apply to join the unit. A selection test is conducted to select suitable candidates. The file `candidates.txt` contains the scores of the selection test for these candidates.

Each line in the file has three fields in this order: **name**, **role**, and **score**. The fields are separated by a comma (,). Each candidate has a pre-assigned role based on their previous experience and training.

Your task is to write a program to select new candidates to form new squads in the special forces unit as well as to determine those that are not selected. You are allowed to use Python built-in functions in this task.

For each sub-task, 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

Write a Python function, `read_data(filename)` to read in the data from a text file and returns a list of tuples. Each tuple represents a candidate record in the file. The function needs to:

- take the name of a file `filename` as input.
- reads the contents of the file. Each line of the file has three fields: name, role, and score, separated by ','.
- returns a Python list of tuples, each tuple should be in the form
`(name, role, score)`

[3]

Task 2.2

Write a Python function, `get_roles(candidates)`, where `candidates` is the list of tuples obtained in Task 2.1, and returns a Python **dictionary** containing each unique role as keys and a list of candidates who belonged to that role. Your code should dynamically determine the unique roles present in the `candidates` list and must not create the dictionary by hardcoding specific role names like 'Sniper' or 'Commander'.

Example: calling `get_roles(candidates)` should return

```
{'Sniper': [('Chadwick Griffin', 'Sniper', 95),
            ('Joella Wessner', 'Sniper', 93),
            :
            ('Luanne Lett', 'Sniper', 61)],
 'Commander': [('Marcella Daigneault', 'Commander', 95),
               :
               ('Ione Wolfe', 'Commander', 90)],
 :
 '2IC': [('Maryam Brosius', '2IC', 92),
         :
         ('Sharie Schall', '2IC', 58)]
}
```

[3]

Task 2.3

Using the data file `candidates.txt` and the functions `read_data()` and `get_roles()` implemented in Task 2.1 and Task 2.2 respectively, write Python code to :

- count the number of candidates in each unique role.
- determine the lowest and highest score obtained by the candidates in each unique role.
- output the statistics in the following format:

Roles	Count	Low Score	High Score
Sniper	8	61	95
:	:	:	:
:	:	:	:
2IC	7	58	92

[3]

The selection criteria for the special forces unit to form new squads is as follows:

- A squad must contain members with unique roles, meaning no duplicate roles are allowed within the squad.
- The number of members in a squad is determined by the number of unique roles determined in Task 2.2, meaning if there are seven unique roles identified, all the squads formed must have **exactly** seven members. You cannot hardcode the number of members in the squad, this number must be dynamically determined.
- When selecting a candidate for a particular role in the squad, the candidate with **the highest score** in that role will be chosen first.
- A selected candidate can only be a member of one squad.

Task 2.4

Write a Python function, `form_squads(roles)` to form new squads based on the selection criteria described above. The function will:

- have an input, `roles`, a dictionary object obtained by calling `get_roles()` implemented in Task 2.2.
- returns a list of squads. Each squad is a list of candidates.

Example: calling `form_squads(roles)` should return

```
[('Chadwick Griffin', 'Sniper', 95),
 ('Marcella Daigneault', 'Commander', 95),
 ('Marg Thorington', 'Medic', 84),
 ('Terrence Shannon', 'Breacher', 94),
 ('Shaunda Sieg', 'Navigator', 84),
 ('Reiko Stack', 'Communication', 83),
 ('Maryam Brosius', '2IC', 92)
],
[('Joella Wessner', 'Sniper', 93),
 ('Fredricka Gormley', 'Commander', 92),
 :
 ('So Atkins', 'Communication', 75),
 ('Russell Gillison', '2IC', 91)
],
:
[('Jalisa Stoudemire', 'Sniper', 76),
 ('Johnna Lecuyer', 'Commander', 48),
 :
 ('Marguerita Mciver', 'Communication', 48),
 ('Sharie Schall', '2IC', 58)
]
]
```

[6]

Task 2.5

Using the data file `candidates.txt` and the functions `read_data()`, `get_roles()` and `form_squads()` implemented in Task 2.1, Task 2.2 and Task 2.4 respectively, write Python code to output the new squads formed as follows:

- each squad is assigned a unique integer identifier.
- the average score of the squad's members.
- list of squad members with their names and roles.

Example:

```
Squad 1, Avg.Score(89.57)
(Chadwick Griffin,Sniper)
(Marcella Daigneault,Commander)
(Marg Thorington,Medic)
(Terrence Shannon,Breacher)
(Shaunda Sieg,Navigator)
(Reiko Stack,Communication)
(Maryam Brosius,2IC)

Squad 2, Avg.Score(85.29)
(Joella Wessner,Sniper)
(Fredricka Gormley,Commander)
(Hillary Curl,Medic)
(Lanita Sciortino,Breacher)
(Teodoro Negrin,Navigator)
(So Atkins,Communication)
(Russell Gillison,2IC)
:
:
Squad 7, Avg.Score(54.57)
(Jalisa Stoudemire,Sniper)
(Johnna Lecuyer,Commander)
(Virgilio Britt,Medic)
(Juli Barnhill,Breacher)
(Tashia Bowen,Navigator)
(Marguerita Mciver,Communication)
(Sharie Schall,2IC)
```

[3]

Task 2.6

Write Python code to determine and output the candidate/s who cannot be placed in any special forces squad and therefore was/were not selected. No output is needed if all the candidates were selected and placed in the squads.

Example:

Luanne Lett, who is a Sniper with a score of 61 is not selected [2]

Save your Jupyter Notebook for Task 2.

3 Name your Jupyter Notebook as `Task3_<your name>.ipynb`.

A program is needed to efficiently retrieve the names of people whose birthdays fall on a given date. The file `birthdays.txt` contains a list of people, with each line in the format:

`name, birthdate`

where the `birthdate` is given in the format **YYYYMMDD**.

For each sub-task, 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

Each person should be represented by a Python class named `Person` as described by the following class diagram:

Person
+ name
+ birthdate
+ <code>__repr__()</code>

Write Python code to :

- define the class `Person`.
 - `__repr__` should return a string representation of the `Person` object in the format: `<name:birthdate>`
- read the contents of the file `birthdays.txt`
- create a list of object instances of the class `Person` named `birthdays` and print the list.

[3]

Task 3.2

Write a Python function `sort_birthdays(birthdays)` that uses the **merge sort** algorithm to sort a list of `Person` objects in ascending order of their birthdates.

The function will:

- have an input, `birthdays`, which is a list of `Person` objects created in Task 3.1.
- returns a sorted list of `Person` objects in ascending order of their birthdates. [7]

Task 3.3

Using the list of `Person` objects created in Task 3.1 and the function

`sort_birthdays()` in Task 3.2, write Python code to output the **name** and **age** of each person, sorted from youngest to oldest. Age is calculated based only on their year of birth. Example:

```
Jane Davis, 15
Michael Brown, 15
David Brown, 16
:
Sarah Jones, 43
Alex Wilson, 44
Emily Wilson, 45
Emily Williams, 45
```

[3]

Task 3.4

Write a Python function `search_birthday(sorted_birthdays, date)` that uses the **binary search** algorithm to find the smallest and largest indices (inclusive) in the list of `Person` objects whose birthdate matches the given date, where

- `sorted_birthdays` is the sorted list of `Person` objects returned by the function `sort_birthdays()` in Task 3.2.
- `date` is the target to be searched, a date string in the format **YYYYMMDD**.
- returns the leftmost(smallest) and rightmost(largest) indices of the elements in the list whose birthdate matches `date`.

Example, calling `search_birthday(sorted_birthdays, "19980701")` should return `(53, 56)` indicating that the elements at indices 53, 54, 55, 56 have birthdate that matches `date`.

- returns `(-1, -1)` if there are no matches.

Note:

- there may be more than one person with the same birthday.
- no marks will be given if the solution does not use a binary search algorithm. Full marks for this sub-task will only be given if the solution has a worst-case time complexity of $O(\log N)$.

[10]

Task 3.5

Write Python code that uses the indices returned by the function

`search_birthdays` to print a list of matching `Person` objects from

`sorted_birthdays`.

Test Cases	Expected Output
<code>search_birthdays (sorted_birthdays, "19980701")</code>	<code>[<Alice Smith:19980701>, <Sarah Smith:19980701>, <Eric Banner:19980701>, <Emily Brown:19980701>]</code>
<code>search_birthdays (sorted_birthdays, "19841231")</code>	<code>[<Katie Brown:19841231>]</code>
<code>search_birthdays (sorted_birthdays, "20041231")</code>	<code>[]</code>

[2]

Save your Jupyter Notebook for Task 3.

- 4 A competition has three qualifying rounds. Competitors could score up to 100 points in each round. A SQLite relational database, `task4.db` has been created and populated with the data from the competition. The two tables in the database are defined as follows:

```
competitor (id, name)
```

```
scores (id*, round, score)
```

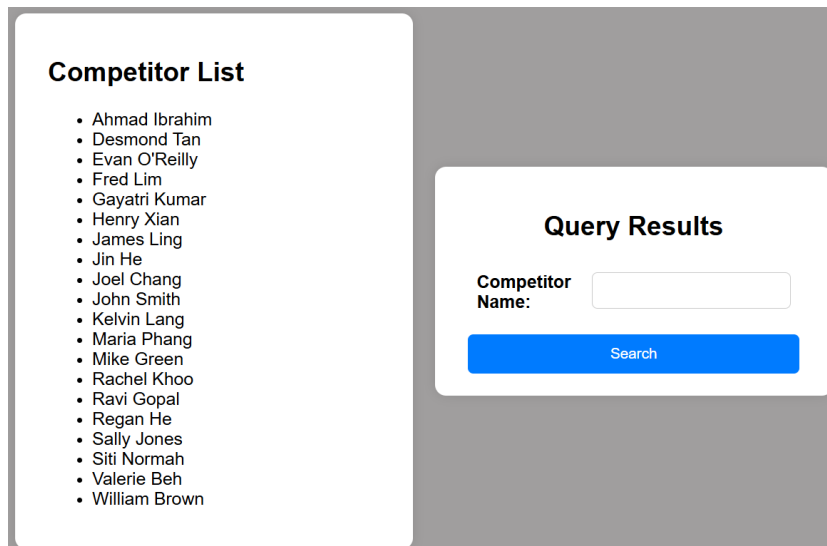
where the primary key is indicated by underling one or more attributes. Foreign key is indicated by an asterisk character `*`.

The tables have been pre-loaded with data.

Task 4.1

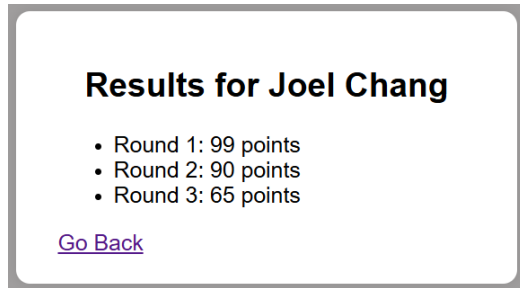
Write a Python program and the necessary files to to create a web application. The landing or home page of the web application should should

- display the names of all the competitors in alphabetical order.
- provide a user interface where users can enter and submit the name of the competitor whose results they are searching for
- the landing page should like this:



The image shows a web application interface with a light gray background. On the left, there is a white box titled 'Competitor List' containing a bulleted list of 20 names in alphabetical order: Ahmad Ibrahim, Desmond Tan, Evan O'Reilly, Fred Lim, Gayatri Kumar, Henry Xian, James Ling, Jin He, Joel Chang, John Smith, Kelvin Lang, Maria Phang, Mike Green, Rachel Khoo, Ravi Gopal, Regan He, Sally Jones, Siti Normah, Valerie Beh, and William Brown. On the right, there is a white box titled 'Query Results' containing a label 'Competitor Name:' next to a text input field, and a blue button labeled 'Search' below it.

The program should perform a query on the database and render the results as follows:



If there are no results found, display an appropriate message.

A stylesheet file named `style.css` has been created for you. You should use the styles and formatting instructions from this file in all your HTML pages. Two incomplete HTML files, `query_form.html` and `results.html`, are also provided.

Save your program code as `Task4_1_<your_name>.py` and any additional files/subfolders as needed in a folder named `Task4`.

Run the web application and save the two outputs of the program as `Task4_1a.html` and `Task4_1b.html` respectively. [17]

Task 4.2

The finalists for the competition are the top **five** competitors in round **3** of the competition. For publicity purposes, a text file needs to be created with the names and scores of the five finalists. The file should have the name and score of each finalist in the following format :

```
[{'name': 'Rachel Khoo', 'score': 91}, {...}, ... {...}]
```

Write Python code to query the database and save the results in a text file named `finalist.txt`.

Save your program code as `Task4_2_<your_name>.py` [5]

END OF PAPER