



VICTORIA JUNIOR COLLEGE
JC 2 PRELIMINARY EXAMINATION
Higher 2

COMPUTING

9569/02

Paper 2 (Lab-based)

28 August 2025

3 hours

Additional Materials:

- Electronic version of `ARTEFACTS.txt` file
- Electronic version of `level_order.py` file
- Electronic version of `Route.csv` file
- Electronic version of `Service.csv` file
- Electronic version of `Stop.csv` 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.

This document consists of **9** printed pages, **3** blank pages, and **1** insert.

Instruction to candidates:

Your program code and output for each of Task 1 to 4 should be saved in a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as:

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

A museum maintains information about its artefacts in a text file `ARTEFACTS.txt` in the following format:

```
[(artefactID, rating), ...]
```

where:

- `artefactID` is a single uppercase letter followed by 4 digits (e.g., A1234, B5678)
- `rating` is an integer between 1 and 100 inclusive representing the historical significance

Example of `ARTEFACTS.txt` contents:

```
[(A1234, 85), (B1234, 85), (C1234, 85), ...]
```

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:

```
ln [1]: 

#Task 1.1
Program code


```

Output:

Task 1.1

Write a function `task1_1(artefacts_list)` that:

- takes a list of tuples `artefacts_list`
- validates that each `artefactID` starts with an uppercase letter followed by a 4-digit number between 1000 and 9999 inclusive, outputting an appropriate error message if data is invalid
- validates that each `rating` is between 1 and 100 inclusive, outputting an appropriate error message if data is invalid
- returns a list of tuples containing valid `(artefactID, rating)` pairs. [3]

Test your function with the artefacts list in `ARTEFACTS.txt` and output:

- the total number of valid artefacts read
- the first 5 artefacts in the list. [1]

Task 1.2

Write a function `task1_2(artefacts_list)` that:

- takes a list of artefact tuples `artefacts_list` as a parameter
- implements a quicksort algorithm to sort the artefacts by rating in descending order
- uses the first element as the pivot
- returns the sorted list.

[5]

Test your function with the artefacts list in `ARTEFACTS.txt`.

[1]

Task 1.3

Write a procedure `task1_3(artefacts_list, target_rating)` that:

- takes a list of artefact tuples `artefacts_list` and the rating `target_rating` as parameters
- uses your function from **Task 1.2** to sort the artefacts by rating in descending order
- implements a recursive binary search to find all the artefacts with the `target_rating`
- returns the tuple `(artefactID, rating)` if found or `None` if not found.

[7]

Test your function with:

- `target_rating = 85`
- `target_rating = 100`

Output an appropriate message for each test case.

[2]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as:

TASK2_<your name>_<class>_<index number>.ipynb

A program is required to manage an ordered linked list of student records, where each record contains a student's ID (integer) and name (string). The linked list must be maintained in ascending order based on the student's ID. To optimise memory usage, a free space list will be implemented to reuse deleted nodes.

The class `Node` contains three attributes:

- `student_id` the integer ID of the student, initialised to `-1`
- `student_name` the string name of the student, initialised to empty string
- `next_pointer` points to the index of the next node in the array, initialised to `-1`.

The class `Node` has the following methods:

- `set_id()` that assigns a value to `student_id`
- `set_name()` that assigns a value to `student_name`
- `get_id()` that returns `student_id`
- `get_name()` that returns `student_name`

The class `LinkedList` contains the three attributes:

- `nodes_array` a list of `Node` objects, with size 10
- `start_pointer` points to the index of the first node in the linked list, initialised to `-1`
- `next_free_pointer` points to index of the first available node in the free space list, initialised to 0.

The class `LinkedList` has the following methods:

- a constructor that
 - initialise `nodes_array` as a list of 10 `Node` objects.
 - link all nodes into the free space list by setting `next_pointer` of each node to the next index (e.g., node 0 points to 1, node 1 points to 2, etc.), with the last node's `next_pointer` set to `-1`.
 - initialise `start_pointer` to `-1` and `next_free_pointer` to 0.
- `insert_record(student_id, student_name)` to insert a record into the ordered linked list.
- `delete_record(student_id)` to delete a record from the linked list and return the node to the free space list.
- `display_records()` to output all records in the linked list in order. The output should look like the following:
 Student ID: <insert student_id>, Name: <insert student_name>

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:

```
ln [1]: 

#Task 2.1
Program code


```

Output:

Task 2.1

Write program code to declare the class `Node`, its constructor and its methods. [4]

Task 2.2

Write program code to declare the class `LinkedList` and its constructor.. [4]

Task 2.3

Write the `display_records()` method for the `LinkedList` class. The method should output all records in the linked list in ascending order of `student_id`. The output for each record should be in the following format:

```
Student ID: <insert student_id>, Name: <insert student_name>
Student ID: <insert student_id>, Name: <insert student_name>
```

 [2]
Task 2.4

Write the `insert_record(student_id, student_name)` method for the `LinkedList` class. The method should:

- Check if the free space list has a next available node
- If there is a next available node in the free space list,
 - store the new record in this node, remove it from the free space list and insert it into the ordered linked list at the correct position
 - update `next_free_pointer` and `start_pointer` to appropriate values
- If there is no available node in the free space list, output an appropriate message. [6]

Test your program by inserting the following records in order and then, calling `display_records()`:

- (101, "Alice")
- (103, "Bob")
- (102, "Charlie") [3]

Task 2.5

Write the `delete_record(student_id)` method for the `LinkedList` class. The method should:

- Search for the record with the given `student_id`
- Remove the node from the linked list and insert it to the start of the free space list
- Update `next_free_pointer` and `start_pointer` to appropriate values
- Output "Record deleted" if successful, or "Record not found" otherwise. [6]

Test your program by:

- Deleting the record with `student_id = 102`
- Deleting the record with `student_id = 104`

Show the output of `display_records()`. [1]

Save your Jupyter Notebook for Task 2.

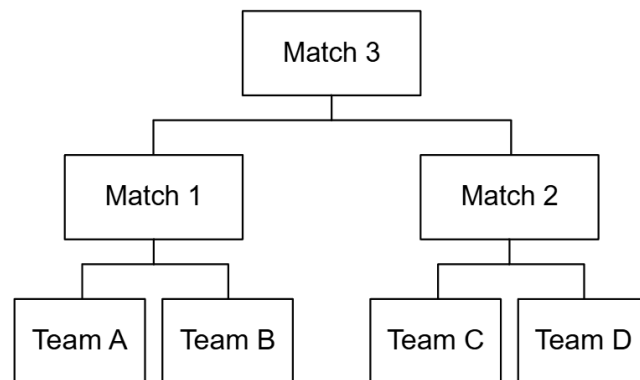
3 Name your Jupyter Notebook as:

TASK3_<your name>_<class>_<index number>.ipynb

Write a program to represent the matches in a sports tournament as a binary tree where:

- each leaf node represents a participating team
- each internal node represents a match between two winners from lower rounds
- the root node represents the final match (champion).

The number of teams must be an exponential growth with base 2, i.e. 2^n where n is a positive integer, so that every round has equal matches until the final match. An example of a binary tree used to represent matches in the sports tournament involving 4 participating teams is as follows:



The program uses object-oriented programming to implement the binary tree and its nodes.

The class `Node` contains three attributes:

- `team`: name of the team or match label stored at the node
- `left_ptr`: pointer to the left child of the node, `None` if there is no left child
- `right_ptr`: pointer to the right child of the node, `None` if there is no right child

The class `Node` has a constructor that initialises the three attributes.

The class `TournamentTree` is a binary tree that represents the matches in the tournament and contains one attribute:

- `root`: pointer to the root node of the tree, `None` if the tree is empty.

The class `TournamentTree` has the following methods:

- a constructor that initialises the `root` pointer to an appropriate value
- `create_tree(num_of_teams)` that takes the number of teams, `num_of_teams`, as a parameter and create the binary tree representing the matches.
- `level_order()` that displays the names of the matches or teams by levels of the binary tree
- `pre_order()` that displays the names of the matches or teams by performing pre-order traversal on the binary tree
- `post_order()` that displays the names of the matches or teams by performing post-order traversal on the binary tree
- `search(target_name)` that takes the name of the team or match to be searched for in the binary tree. It returns `True` if the name can be found, or `False` otherwise
- `calculate_matches(num_of_teams)` that takes the number of teams, `num_of_teams`, participating in the sports tournament and returns the number of matches needed.

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:

```
ln [1]: 

#Task 3.1
Program code


```

Output:

Task 3.1

Write program code to declare the class `Node` and its constructor. [2]

Task 3.2

Write program code to declare the class `TournamentTree` and its constructor. [2]

Task 3.3

Amend your code for **Task 3.2** to declare the `TournamentTree` method `create_tree()` and add in the code provided for `level_order()` from the file `level_order.py`.

In a new cell, write program code to

- initialise a new `TournamentTree`
- call `create_tree()` to create a tree with 8 participating teams, named Team A - H
- call `level_order()` to display the tree by level. [8]

Test your program and show the output. [1]

Task 3.4

Amend your code for **Task 3.2** to declare the `TournamentTree` methods `pre_order()` and `post_order()`.

In a new cell, write program code to

- call `pre_order()` to perform pre-order traversal
- call `post_order()` to perform post-order traversal. [4]

Test your program and show the output. [2]

Task 3.5

Amend your code for **Task 3.2** to declare the `TournamentTree` method `search()`.

In a new cell, write program code to test the `search()` method using these 2 test cases,

- Team E
- Team J [3]

Show the output. [1]

Save your Jupyter notebook for Task 3.

4 Name your Jupyter Notebook as:

TASK4_<your name>_<class>_<index number>.ipynb

A student has records about bus services in Singapore stored in comma-separated value format. They want to create a suitable database to store the data and to allow them to run searches for bus route information. The database will have three tables: a table to store data about bus stops, a table about bus services, and a table about the bus routes. The fields in each table are:

Service:

- ServiceNo – The bus service (e.g. 118A)
- Operator – The bus operator company (e.g. SBST)
- Category – The type of route

Stop:

- RoadName – The road that the bus stop is on
- Description – A short description of the bus stop location
- BusStopCode – The unique code of the bus stop (e.g.75009)

Route:

- ServiceNo – The bus service of the route
- Operator – The bus operator company
- Direction – The direction of the route in integer representation (1 or 2 only)
- StopSequence – The stop's numerical sequence (First stop = 1, second stop = 2, ...)
- BusStopCode – The unique code of the bus stop

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:

```
ln [1]: 

#Task 4.1
Program code


```

Output:

Task 4.1

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

Task 4.2

The text files `Route.csv`, `Service.csv` and `Stop.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 data in the correct place in the database. [6]

Task 4.3

Write a Python program to input a bus service number and return its routes, showing:

- the direction of the route
- the bus stop code of each stop on the route
- the road name of the bus stop
- the description of the bus stop
- in ascending order by direction and then, by stop sequence.

[7]

Save your Jupyter Notebook for Task 4.1 to 4.3.

Task 4.4

Write a Python program and the necessary files to create a web application that allows the user to input a bus stop code. The web application displays the bus service numbers that stop at that bus stop, and the corresponding bus operator. If the bus stop code is invalid, it displays an appropriate error message instead.

Save your Python program as:

`TASK_4_4_<your name>_<class>_<index number>.py`

with any additional files/ subfolders in a folder named:

`TASK_4_4_<your name>_<class>_<index number>`

[6]

Run the web application with the following inputs:

- 01111
- 01112

Save the webpage output for each input as:

`TASK_4_4_<your name>_<class>_<index number>_1.html`

`TASK_4_4_<your name>_<class>_<index number>_2.html`

[2]

