H2

MINISTRY OF EDUCATION, SINGAPORE
in collaboration with
CAMBRIDGE INTERNATIONAL EDUCATION
General Certificate of Education Advanced Level

COMPUTING **9569/02**

Paper 2 (Lab-based) **For examination from 2027**

SPECIMEN PAPER **3 hours**

0123456789

You will need: `formgroups.txt`
`hashdata.txt`
`students.txt`
`weather.txt`
Reference Guide

INSTRUCTIONS

- Answer **all** questions.
- You must do all the tasks using an authorised computer. 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.
- Save your work using the file name given in the question.
- Save each task as you complete it.
- You may use an approved calculator.
- You can use built-in functions, where appropriate, unless stated otherwise.

INFORMATION

- The total mark for this paper is 100.
- The number of marks for each question or part question is shown in brackets [ ].

This document has **12** pages.

Singapore Examinations and Assessment Board

CAMBRIDGE International Education

Note that up to 6 marks out of 100 will be awarded for the use of common coding standards for programming style.

Instructions to candidates

Your program code and output for each of Task 1 to 4 should be saved in a single `.ipynb` file using JupyterLab. 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 files shows the required output in JupyterLab.

Task 1

Save your file as:

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

A hash table is used to store data records. The hash table is created as a list with 1000 indices for records to be stored.

Each data record has three values:

- an integer key between 1 and 9999 inclusive
- an integer data value
- a second integer data value.

The records are currently stored in the text file `hashdata.txt`

The hash calculation is the remainder after dividing the key by 1000.

If there is a collision (two keys generating the same index) the algorithm appends the record at the same index, creating a nested list of records.

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

The class `Record` contains three attributes:

- `key` – the integer key that is used to generate the hash value
- `data_1` – the first integer data item
- `data_2` – the second integer data item.

The class `Record` has a constructor method that takes a `key, data_1` item and `data_2` item as parameters. Each parameter is assigned to the appropriate attribute.

Write program code to declare the class `Record`

[3]

Task 1.2

The class `HashTable` contains one attribute:

- `the_hash_table` – a list with space for 1000 records.

The class `HashTable` has the following methods:

- a constructor that initialises the list to store 1000 values of `-1`
- `insert_data()` takes a record as a parameter, calculates the hash value from the record's key and stores the record in the appropriate index in the list. If there is a collision, the record is appended to the same index, creating a nested list.

Write program code to declare the class `HashTable` and its methods. [7]

The class `HashTable` has an additional method `get_data()` that:

- takes an integer key as a parameter
- calculates the hash value for the record
- searches the list at the index for a matching record
- returns the record if it is found
- returns `False` if the record is **not** found.

Write program code for `get_data()` [5]

Task 1.3

The text file `hashdata.txt` contains key and data for each record.
Each row contains three values:

- the key
- data item 1
- data item 2.

The format of each data entry in the file is:

```
<key>,<data_1>,<data_2>
```

The first 3 lines from the text file are as follows:

```
8222,23,2039
5170,19,2187
1925,97,2901
```

Write program code to:

- create a new instance of `HashTable`
- open the file `hashdata.txt`
- read each line of data and separate into the three values
- create an instance of `Record` for each line of data
- call `insert_data()` for the hash table with each record.

[6]

Task 1.4

Write program code to:

- call `get_data()` with the key 8222
- output 'No record' if the return value shows the record was **not** found
- output the key and both data values separated with a space if the return value shows the record was found.

[3]

Run your program. [1]

Save your JupyterLab notebook for Task 1.

Task 2

Save your file as:

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

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

A circular queue data structure can store up to 10 positive integers.

The queue is declared as a 1-dimensional array. Each element in the array is initialised with the integer −1.

The queue has two pointers:

- `headpointer` that stores the index of the first element in the queue where items are dequeued. This is initialised to `-1` as a temporary placeholder.
- `tailpointer` that stores the index of the last element in the queue where items are enqueued. This is initialised to `-1` as a temporary placeholder.

The variable `items_in_queue` stores the number of integers currently stored in the queue. This is initialised to `0`

Write program code to declare and initialise the queue array, the two pointers and `items_in_queue` [1]

Task 2.2

The function `enqueue()`:

- takes an integer parameter
- returns `False` if the queue is full
- stores the parameter in the queue if the queue is **not** full, updates the appropriate pointer(s) and `items_in_queue` and then returns `True`

Write program code for `enqueue()` [4]

Task 2.3

The function `store_data()`:

- prompts the user to enter 12 positive integers
- takes each integer as input until it is a positive integer
- calls `enqueue()` with each valid input
- outputs an appropriate message to identify the value was stored in the queue or if the queue is full.

Write program code for `store_data()` [4]

Task 2.4

Write program code to call `store_data()` [1]

Test your program with the following inputs in the order given:

1
2
−1
3
−100
4
5
6
7
8
9
10
11
12

[2]

Task 2.5

The function `dequeue()`:

- returns `-1` if the queue is empty
- returns the next element in the queue if the queue is **not** empty and updates the appropriate pointer(s) and `items_in_queue`

Write program code for `dequeue()` [3]

Task 2.6

Write program code to call dequeue() five times, add together each positive integer that is returned, and output this total in an appropriate message.

Output an appropriate message if any of the function calls indicate the queue is empty. [4]

Test your program. [1]

Save your JupyterLab notebook for Task 2.

Task 3

Save your file as:

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

The task is to implement a nearest neighbours algorithm to predict whether it will rain in a specific place during the month of June.

The text file `weather.txt` contains a set of data values. Each set contains three values:

- the longitude where the measurement was taken
- the latitude where the measurement was taken
- the amount of rainfall in the month of June.

The format of each data entry in the file is:

`<longitude>,<latitude>,<rainfall>`

The first 3 lines from the file are as follows:

```
51.5072,0.1276,0.41
-35.2835,149.1281,2.2
-8.8368,13.2343,0
```

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

Write program code to:

- import the `numpy` library and rename it as `np`
- create two arrays `training` and `results`, using `numpy` to store the data from the text file
- store each floating-point longitude and latitude from `weather.txt` in `training`
- check the value of the amount of rainfall for each longitude and latitude pair, and:
  - store `1` in `results` if the rainfall value is greater than or equal to `1`
  - store `0` in `results` if the rainfall value is less than `1`
- output the contents of each array.

[8]

Run your program. [1]

Task 3.2

Write program code to:

- import the `KNeighborsClassifier` library
- create a new `KNeighborsClassifier` with `n_neighbors` set to `5`
- fit the classifier using `training` and `results`

[5]

Task 3.3

Write program code to output the mean accuracy of the `KNeighborsClassifier` for `training` and `results` using `score()` [2]

Run your program. [1]

Task 3.4

Write program code to predict whether it will rain for a location with the longitude 50.351 and latitude 0.15845.

Output:

- the nearest neighbour
- the probability estimate for the data.

[3]

Run your program. [1]

Save your JupyterLab notebook for Task 3.

Task 4

Save your file as:

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

A website allows a user to search for the quantity of students in a form group.

A database stores data about students and form groups.

Two tables in the database are shown:

`Student (StudentID, GivenName, FamilyName, FormGroupID)`

`Form (FormGroupID, TutorID, Year)`

A primary key is indicated by underlining one or more attributes. Foreign keys are indicated with a dashed underline.

The database is being created and tested using sample data provided in the following text files: `students.txt` and `formgroups.txt`

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

         Output:
```

Task 4.1

Write a Python program that uses SQL to create a database with two tables: `Student` and `Form`

Define the primary and foreign keys for each table. [5]

Task 4.2

Write a Python program that uses SQL to insert the data from each of the two text files into the appropriate fields in each database table.

Run your program. [4]

Task 4.3

A CSS file defines the design of the web pages.

The table shows the formatting rules to be used.

HTML tag	formatting rules
h1	red font centre-aligned text font family Arial font size 50px bold font
h2	blue font left-aligned text font family Arial font size 30px bold font
p	black font justified text font family Arial font size 14px

Create the CSS file that defines the rules in the table. [4]

Task 4.4

The `index.html` web page uses the CSS file defined in Task 4.3 contains:

- the h1 heading “Students and form groups”
- the h2 heading “Find form groups”
- the text “Enter form group ID” with a text box and a submit button below it.

Create the `index.html` web page. [4]

Task 4.5

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

- accesses the form group entered in the text box on `index.html` when the submit button is pressed
- uses SQL to search the database you created in Task 4.1 to retrieve the quantity of students in the form group from `index.html`, the year group of the form group and the form tutor ID
- formats the retrieved data into a sentence that follows the structure:
  `There are <quantity> students in form <form group ID>. The form is in year group <year group> with the form tutor <form tutor ID>`
- returns a HTML document that displays the created sentence.

Save your program as:

`TASK_4_5_<your name>_<centre number>_<index number>.ipynb`

Place any additional files/subfolders in a folder named:

`TASK_4_5_<your name>_<centre number>_<index number>` [10]

Run the web application.

Save the web page output (HTML only) as:

`TASK_4_5_<your name>_<centre number>_<index number>.html` [1]

Save your JupyterLab notebook for Task 4.

