MINISTRY OF EDUCATION, SINGAPORE
in collaboration with
CAMBRIDGE ASSESSMENT INTERNATIONAL EDUCATION
General Certificate of Education Advanced Level
Higher 2

COMPUTING

9569/02

Paper 2 (Lab-based)

October/November 2021

3 hours

Additional Materials: Electronic version of `TEN.txt` data file
Electronic version of `HUNDRED.txt` data file
Electronic version of `THOUSAND.txt` data file
Electronic version of `Task2_timing.py` file
Electronic version of `Task3data.txt` data file
Electronic version of `Task4.db` 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 numbers 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.

Singapore Examinations and Assessment Board

Cambridge Assessment International Education

Instruction to candidates:

Your program code and output for each of Task 1, 2 and 3 should be downloaded in a single `.ipynb` file. For example, your program code and output for Task 1 should be downloaded as

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

1 The IMEI number is a unique value used to identify mobile devices, such as phones and tablets. The IMEI number has 15 digits: 14 digits with an extra check digit added to the right-hand side.

The check digit is calculated using the following algorithm, on the left-most 14 digits of an IMEI number:

1. starting from the right, the first digit is location number 1
2. double all digits in the odd numbered positions
3. sum all the digits, including both the unchanged digits (i.e. those in the even numbered positions) as well as those doubled (e.g. 16 contributes 1 + 6)
4. the check digit is the value between 0 and 9 that must be added to the sum to make the result exactly divisible by 10.

For example, given the 14 digits 14576567654934:

Step 1: 1 4 5 7 6 5 6 7 6 5 4 9 3 4
Step 2: 1 8 5 14 6 10 6 14 6 10 4 18 3 8
Step 3: 1 + (8) + 5 + (1 + 4) + 6 + (1 + 0) + 6 + (1 + 4) + 6 + (1 + 0) + 4 + (1 + 8) + 3 + (8)
Sum = 68
Step 4: Check digit = 2

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

Write a function `task1_1(input_value)` that returns an integer.

The function should:

- validate that the parameter `input_value` is either an integer, or a string containing a valid integer
- check that it is 14 digits in length
- return −1 if the value received is invalid for any reason
- calculate the check digit for the given first 14 digits of the IMEI number
- return the calculated check digit. [6]

Task 1.2

Create **four** tests that should fully test your function. Ensure that it validates the inputs accurately and returns the correct expected result.

Each of the four tests will be a pair of data items: the input value and its expected result.

Your four input values should be:

- a string containing just a valid integer
- a valid integer
- a string containing characters that are not numbers
- a value of the incorrect length.

Test your function with your four input values by calling it using the following statement:

```
print(task1_1(input_value) == expected)
```

The four statements should all print `True`.

For example:

```
print(task1_1("14576567654934") == 2)
```

Do not use the example provided but create your own. [4]

Task 1.3

A full 15-digit IMEI number can be validated by removing the check digit, calculating the check digit from the remaining 14 digits and comparing it to the removed digit.

Write a function `task1_3(input_value)` that returns a Boolean. The function should:

- validate that the parameter `input_value` is either an integer, or a string containing a valid integer
- check that it is 15 digits in length
- return `False` if the value received is invalid for any reason
- remove the digit on the right-hand side
- use your function from Task 1.1 to calculate the check digit from the remaining 14 digits
- return a Boolean representing the comparison between the calculated check digit and the original removed digit. [4]

Test your function with four input values (these may be based on those from Task 1.2 or otherwise), two should be correct IMEI numbers, two should be in error.

If a value of `True` is expected, use the following statement:

```
print(task1_3(input_value))
```

If a value of `False` is expected to be returned by the comparison, this can be changed to still output `True` by using the `not` keyword, for example:

```
print(not task1_3("12345r"))
```

The four statements should all print `True`. [4]

Save your Jupyter notebook for Task 1.

2 For this question you are provided with three text files, each contains a valid list of positive integers, one per line:

- `TEN.txt` has 10 lines
- `HUNDRED.txt` has 100 lines
- `THOUSAND.txt` has 1000 lines.

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

Write a function `task2_1(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 integers. [3]

Call your function `task2_1` with the file `TEN.txt`, printing the returned list and its length, using the following statements:

```
result = task2_1('TEN.txt')
print(result)
print(len(result))
```
[2]

Task 2.2

One method of sorting is the insertion sort.

Write a function `task2_2(list_of_integers)` that:

- takes a list of integers
- implements an insertion sort algorithm
- returns the sorted list of integers. [5]

Call your function `task2_2` with the contents of the file `TEN.txt`, printing the returned list, for example, using the following statement:

```
print(task2_2(task2_1('TEN.txt')))
```
[1]

Task 2.3

Another method of sorting is the quicksort.

Write a function `task2_3(list_of_integers)` that:

- takes a list of integers
- implements a quicksort algorithm
- returns the sorted list of integers. [7]

Call your function `task2_3` with the contents of the file `TEN.txt`, printing the returned list, for example, using the following statement:

```
print(task2_3(task2_1('TEN.txt')))
```
[1]

Task 2.4

The `timeit` library is built into Python and can be used to time simple function calls. Example code is shown in `Task2_timing.py`. (The sample code assumes that it has access to the `task2_2` function.)

Using the `timeit` module, or other evidence, and the three text files provided with this question, compare and contrast, including mention of orders of growth, the time complexity of the insertion sort and quicksort algorithms. [5]

Save your Jupyter notebook for Task 2.

3 A programmer is writing a class, `LinkedList`, to represent a linked list of unique integers. A linked list is a collection of data elements, whose order is not given by their physical placement in memory. Instead, each element points to the next.

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 the `LinkedList` class in Python. Use of a simple Python list is not sufficient. Include the following methods:

- `insert(integer_value)` inserts the `integer_value` at the beginning (head) of the list
- `delete(integer_value)` attempts to delete `integer_value` from the list; if the item was not present, return `None`
- `search(integer_value)` returns a Boolean value: `True` if `integer_value` is in the list, `False` if not in the list
- `count()` should return the number of elements in the list, or zero if empty
- `to_String()` should return a string containing a suitably formatted list with the elements separated by a comma and a space, with square brackets at either end, e.g. in the form:

```
[11, 2, 7, 4]
```

[8]

Test `LinkedList` by using the data in the file `Task3data.txt`. Use the `to_String()` method to print the resulting contents of the list. [3]

Task 3.2

Write a Python subclass `SortedLinkedList` using `LinkedList` as its superclass.

The insert method in the `SortedLinkedList` subclass should ensure that the elements are stored in ascending order. [5]

Test `SortedLinkedList` by using the data in the file `Task3data.txt`. Use the `to_String()` method to print the resulting contents of the list.

Print the result of searching the `SortedLinkedList` for the value 94. [2]

Task 3.3

Write a Python subclass `Queue` using `LinkedList` as its superclass.

Additional `enqueue` and `dequeue` methods are to be defined on the `Queue` class:

- `enqueue(integer_value)` will insert `integer_value` to the end of the queue
- `dequeue()` will return the first element in the queue. If the queue is empty, return `None`. [6]

Test `Queue` by using the data in the file `Task3data.txt`. Print the first five elements to be dequeued from the list. [1]

Save your Jupyter notebook for Task 3.

4 A competition has three qualifying rounds. Competitors could score up to 100 points in each round. Each competitor has a unique ID number. The competitor's ID, name, and the scores for each round are held in a database, `Task4.db`, provided with this question.

There are two tables:

- `competitor(id, name)`
- `scores(id, round, score)`

Not every competitor competed in all three rounds, but they did all compete in round 1.

Task 4.1

Write a Python program and the necessary files to create a web application. The web application offers the following menu options:

Round 1 Scores
Round 2 Scores
Round 3 Scores
Mean Scores
Qualifiers

Save your program code as

`Task4_1_<your name>_<centre number>_<index_number>.py`

With any additional files/subfolders as needed in a folder named

`Task4_1_<your name>_<centre number>_<index_number>`

Run the web application and save the output of the program as

`Task4_1_<your name>_<centre number>_<index_number>.html` [3]

Task 4.2

Write an SQL query that, for a given round number, shows:

- competitor names and their scores for that particular round
- only those who had a score for that round
- the scores sorted in descending order.

The results of the query should be shown on a web page in a table that:

- lists the name of each competitor and their score
- has the results shown in descending order of the competitor's score.

The resulting web page for each round should be accessed from the corresponding menu option from Task 4.1.

Save all your SQL code as

`Task4_2_<your name>_<centre number>_<index_number>.sql`

With any additional files/subfolders as needed in a folder named

`Task4_2_<your name>_<centre number>_<index_number>` [6]

Run the web application and save the output of the program as

`Task4_2_<your name>_<centre number>_<index_number>_1.html`

`Task4_2_<your name>_<centre number>_<index_number>_2.html`

`Task4_2_<your name>_<centre number>_<index_number>_3.html` [2]

Task 4.3

Write a SQL query that shows:

- competitor name
- the mean score for each competitor, based on the number of rounds in which they competed.

The query's results should be shown on a web page in a table that:

- lists the name of each competitor and their mean score (to 2 decimal places)
- has the results shown in ascending alphabetical order of the competitor's name.

The resulting web page should be accessed from the corresponding menu option from Task 4.1.

Save your SQL code as

`Task4_3_<your name>_<centre number>_<index_number>.sql`

With any additional files/subfolders as needed in a folder named

`Task4_3_<your name>_<centre number>_<index_number>` [6]

Run the web application and save the output of the program as

`Task4_3_<your name>_<centre number>_<index_number>.html` [2]

Task 4.4

In order to qualify for the final of the competition, competitors need to score over 250, in total, in the first three rounds.

Write a SQL query that shows:

- competitor's name
- their total score
- whether that competitor has qualified for the final.

The results of the query should be shown on a web page in a table that:

- lists the names of the competitors, their scores and whether they qualified
- has the results shown in descending order of the competitor's score.

The resulting web page should be accessed from the corresponding menu option from Task 4.1.

Save your SQL code as

`Task4_4_<your name>_<centre number>_<index_number>.sql`

With any additional files/subfolders as needed in a folder named

`Task4_4_<your name>_<centre number>_<index_number>` [6]

Run the web application and save the output of the program as

`Task4_4_<your name>_<centre number>_<index_number>.html` [2]

BLANK PAGE

BLANK PAGE

BLANK PAGE