DUNMAN HIGH SCHOOL
Preliminary Examination
Year 6

COMPUTING

9569 / 02

Paper 2 (Lab-based)

31 August 2020

3 hours

Additional Materials:

Electronic version of `TOKEN.txt` data file
Electonic version of `LASTNAMES.txt` data file
Electronic version of `DEMOAPP.txt` data file
Electronic version of `PATIENTS.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.

For Tasks 1, 2 and 4, save your work using the naming convention `TASK`*i*`_<your name>_<centre number>_<index number>.ipynb`, where *i* is the task number.

For Task 3, use meaningful filenames and extensions or as specified in the question and save all related files in a `task3` subfolder.

Task 1

On 28 June 2020, nearly 10,000 TraceTogether tokens were distributed to vulnerable seniors. The TraceTogether token supplements the TraceTogether mobile app by extending contact tracing to groups in the community who do not have smart phones and those whose phones do not work well with the TraceTogether app.

The TraceTogether token is designed to capture proximity data based on Bluetooth signals. Every five minutes, it scans to detect other TraceTogether users on the token or the app. The more 'hits' between two TraceTogether users, the more likely they are in close proximity for an extended period of time. Proximity can be estimated by the strength of the Bluetooth signal. The closer users are to one another, the stronger the signal and vice versa.

There are only four types of data contained in the token:

- user's randomised ID
- randomised ID of any other user in proximity
- Bluetooth signal measured using RSSI*
- timestamp of the encounter.

*Received Signal Strength Indicator (RSSI) is a measure of the power level at the receiver. A more negative number indicates a device is further away. For example, a value of -20 to -30 indicates a device is close while a value of -120 indicates it is near the limit of detection.

It is important to note that these IDs do not refer to NRIC number, but randomised and anonymised IDs linked to a personal identifier like a mobile number. Also, no data is extracted unless a user has tested positive for COVID-19. From there, MOH has a special software key that can unlock the device and reveal the data for use in contact tracing.

A senior is tested positive for COVID-19 and MOH needs to perform contact tracing. With the user's permission, data from its TraceTogether token is retrieved and extracted to the file `TOKEN.txt` and has the following structure:

```
UserRandomisedID,OtherRandomisedID,RSSI,Timestamp
```

Timestamp is in the format YYYY-MM-DD HH:MM:SS.

Task 1.1

Prolonged exposure is currently defined as being in close contact for at least 15 minutes within a single session. For simplicity, you may assume close contact as a Bluetooth signal strength of greater than or equal to -30. Generate the list of close contacts' randomised IDs which MOH needs to perform contact tracing. [10]

Task 1.2

Another 3 seniors with randomised ID 75348257, 45174591 and 02548147 have also been tested as COVD-19 positive. Write a Boolean function `is_close_contact(rid1, rid2)` to determine if they are close contacts of 57345286. If yes, your function should also return the date(s) they are under prolonged exposure with each other, the start and end times as well as the total time in hours and minutes they are in close contact. [10]

Task 2

Write a program that sorts a list of student last names, but the sort only uses the first two letters of the name. Nothing else in the name is used for sorting. However, if two names have the same first two letters, they should stay in the same order as in the input (this is known as a 'stable sort'). Sorting is case sensitive based on ASCII order (with uppercase letters sorting before lowercase letters i.e. A<B<…<Z<a<b<…<z).

Input

The input file `LASTNAMES.txt` consists of a sequence of up to 500 test cases. Each case starts with a line containing an integer 1 ≤ n ≤ 200. After this follow n last names made up of only letters (a–z, lowercase or uppercase), one name per line. Names have between 2 and 20 letters. Input ends when n = 0.

Output

For each case, print the last names in sort-of-sorted order, one per line. Print a blank line between cases.

Sample Input

```
3
Mozart
Beethoven
Bach
5
Hilbert
Godel
Poincare
Ramanujan
Pochhammmer
0
```

Sample Output

```
Bach
Beethoven
Mozart

Godel
Hilbert
Poincare
Pochhammmer
Ramanujan
```

You should make use of an appropriate data structure and one or more suitable sorting algorithms from the syllabus. Indicate as comments your choice of how you have adapted them or your case for designing and implementing your own. [13]

Task 3

A school needs a web application for teachers to use during English lessons to record and view students' marks for weekly in-class presentations.

The teacher should be required to login and should only be able to input and view any information for students of the classes they teach.

The maximum score per presentation is 100. Each English lesson is always taught by the same teacher, each English lesson would only have one teacher, and each class would only have one English lesson per day.

In a training session conducted for teachers, a demo version of the app would be pre-populated with demo data as shown in the following table.

Teacher's username	Teacher's password	Index number of student	Student's class	Date of presentation	Marks obtained
mr_raj	cr53aYJP	3	19S306	20200315	95
mr_james	8orjqiZc	24	19S301	20200315	60
mdm_rahayu	7iqndCjW	2	19S302	20200315	35.5
mr_james	8orjqiZc	11	19S304	20200325	60

You have been tasked to create this web application. The data file `DEMOAPP.txt` contains the demo data.

Task 3.1
Create the user interface using HTML and CSS for the application. The fields should include:

- TeacherUsername
- TeacherPassword
- StudentIndex
- Class
- PresentationDate
- Marks

The information should be presented in a tabular form. [7]

Task 3.2
Create a normalised relational database scheme for this application. Provide the SQL statements for the creation of the tables. [6]

Task 3.3
Populate the table(s) with the demo data described above. Provide SQL statements for the insertion of the records in `task3.sql`. [4]

Task 3.4
Provide the processing logic in `app.py` and the associated template file(s). [13]

Task 3.5
Using SQL or otherwise, determine and display the total marks scored by each class to the appropriate template file. [5]

Task 4
A dental clinic wishes to manage its patients' information using a NoSQL database.

There are 2 rooms in the clinic: Room 1 and Room 2.
There are 3 dentists in the clinic: Doctor 1, Doctor 2, Doctor 3.

Information about the patients is stored in the data file `PATIENTS.txt` in the following format:

```
PatientID,Name,Appointment Date,Appointment Start Time,Doctor
Assigned,Room Number,Amount Charged
```

- PatientID is an integer
- Name is made up of letters and space only
- Appointment Date is in DDMMYYYY format
- Appointment Start Time is HHMM in 24-hour format
- Doctor Assigned is either Doctor 1, Doctor 2 or Doctor 3
- Room Number is either Room 1 or Room 2
- Amount Charged is a float to 2 decimal places

Task 4.1
Write program code to convert and import the data from `PATIENTS.txt` into a MongoDB database Clinic under the collection Patient. [4]

Task 4.2
Write program code to allow the Admin Clerk to add new patients by requesting for the following information:

- New patient's Name
- Appointment Date in DDMMYYYY foormat
- Appointment Start Time in 24-hour HHMM format
- Doctor to be Assigned

The program should automatically assign a PatientID. Your program should perform the necessary validation for each field. [8]

Task 4.3
The Admin Clerk wants to know if there are issues with appointments. Write program code to output appointments clashes and double bookings. If an appointment clash occurs, reschedule the latter appointment to the nearest available appointment on the same date. If a double booking occurs, delete the latter duplicate booking. [8]

Task 4.4
Write program code for the Admin Clerk to determine the top 3 paying patients. [4]

Task 4.5
The Admin Clerk realised that there are patients with the same name. Write program code to identify patients with the same names and output their PatientIDs. [3]

Bonus Task

The postfix notation is used to represent mathematical expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix.

For example, the postfix expression 231+*5- is the equivalent of the infix expression 2*(3+1) -5.

The following algorithm evaluates postfix expressions:

1. Create a stack to store operands (data values).
2. Scan the given expression and do following for every scanned element.
    - If the element is a number, push it into the stack.
    - If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack
3. When the expression has ended, the number in the stack is the final answer.

Example:

Let the given expression be "231+*5-". Scan all elements one by one from left to right.

1. Scan '2', it is a number, so push it to stack. Stack contains '2'
2. Scan '3', again a number, push it to stack, stack now contains '23' (from bottom to top)
3. Scan '1', again a number, push it to stack, stack now contains '231'
4. Scan '+', it is an operator, pop two operands from stack, apply the + operator on the operands, we get 3 + 1 which results in 4. We push the result '4' to stack. Stack now becomes '24'
5. Scan '*', it is an operator, pop two operands from the stack, apply the * operator on the operands, we get 2 * 4 which results in 8. We push the result '8' to stack. Stack now becomes '8'
6. Scan '5', it is a number, we push it to the stack. Stack now becomes '85'
7. Scan '-', it is an operator, pop two operands from stack, apply the - operator on operands, we get 8 - 5 which results in 3. We push the result '3' to stack. Stack now becomes '3'
8. There are no more elements to scan, we return the top element from the stack (the only element left) as the final result 3.

Thus the postfix expression is evaluated in a single pass without the need for parenthesis.

Implement the evaluation of postfix expression using a stack and OOP. You should save your work as `taskb.ipynb`. You may assume that all operands are single digit numbers.

[13]