**INSTRUCTIONS -** **PRACTICAL EXAM – CSD203**

**PLEASE READ BEFORE STARTING YOUR EXAM**

**=====================================================================================**

**Software Requirements**

* PyCharm, Notepad, Notepad++, Command Prompt, WinRAR / WinZip (or compress to ZIP file of Windows), Windows Explorer (File Explorer) on Windows 7 and above.

**Instructions**

* Step 1: Students download the given materials from PEA Client.
* Step 2: Students read questions and prepare answers in the given template.
* Step 3: Submit a solution for each question: The result is one folder for the question (Q1), which contains the **.py** source file. Example: **Folder “Q1” only contains Q1.py**
* **Importance:** 
  + *Solutions will be marked by Automated Marking software.*
  + *The use of tools other than those allowed in the above section whether intentionally or unintentionally, is considered a violation of the exam rules, and the mark is 0*
  + ***Do not: change*** *the names of the folders, files, and struct (format) of .py files specified in the exam. If you change it, the marking software can not find the execute file (.py) or the program output to mark, thus the score will be 0*
  + ***Do not: edit*** *the given statements. If you change, the marking software can not mark and the score will be 0.*
* **Notes 01:** 
  + *You can create new functions if you think they are necessary.*
  + *Carefully read the instructions in each question to complete the practical exam.*
* **Notes 02:** 
  + *The* ***input and expected output below*** *are only used* ***to test your codes****.*
  + *The input and expected output in the* ***real testcases*** *(for marking) are* ***different****.*

**=====================================================================================**

**Questions: (10 marks)**

The given file Q1.py already contains statements to implement a program for managing a Hotel. The structure of the main classes is as follows:

* Class **Room**: contains information about a Room object, including:
  + **code** (001, 002, 003, … 101, 102, 103, …)
  + **status:** is 0 (if the Room is empty), or 1 (if there is/are customer(s) in the Room)
  + **size (must be > 0)**: the maximum number of people renting/using this Room
  + **price (must be > 0)**
* Class **Node**: includes **info** (a Room object) and **next** (a pointer for linking).
* Class **dataList**: is **a singly linked list** that manages regular Nodes with complete information. This class plays the role of managing all Room in the Hotel.
* Class **requestQueue**: is **a queue** structure (implemented as a linked list), where the component **info** of the Nodes **contains only size** (the number of people renting the Room) and **price** (the expected price, or the maximum money the the customer(s) can pay). Note: **size>0 and price>0**.
* Class **Hotel**: is the main class of the program, containing a dataList used to manage the Room objects and a requestQueue corresponding to a queue of order.

Students are required to carefully read the provided code segments to fully understand the relationships between the classes and the functions within each class. The specific task of the test is to execute the following requirements:

1. f1(): 2.5 marks – Load data

To complete the requirement f1(), students need to fulfill two specific tasks: implement the function **addLast()** in the dataList class and the function **enQueue()** (similar to addLast) in the requestQueue class.

The input and expected output used to test your code are as follows:

INPUT:

m = 5 (generate 5 elements, then call addLast() many times to add them to the dataList)

n = 5 (generate 5 elements, then call enQueue() many times to add them to the requestQueue)

Input a question: 1 (choose f1() to run)

OUTPUT:

Data List:

001, 0, 10, 200

003, 0, 3, 70

004, 0, 4, 100

005, 0, 3, 70

=========

Request Queue:

1, 100

12, 500

4, 50

4, 400

=========

1. f2(): 2.5 marks – Serve the first request in the requestQueue

To complete the requirement f2(), students need to perform three specific tasks: (1) implement the **deQueue()** function of the requestQueue structure, (2) use the returned result of deQueue() to perform the **rent()** function, then (3) ues these two functions to serve the first request in the body of f2() function.

The rent() action consists of two steps:

* + First, **search** the best Room to rent. If found, proceed to step 2. The best Room is:
    - An empty Room
    - Size >= the size in the request
    - Price <= the price in the price
    - If there are more than one Room satisfying these three conditions below, you should choose the Room with lowest price.
    - If there are more than one Room satisfying these four conditions below, you should choose the first Room. For example: if Room 003, 005, and 105 satisfy four conditions below, the answer is Room 003.
  + If the best Room is found, perform the renting action: **update** the status in the dataList from 0 to 1.

The input and expected output used to test your code are as follows:

INPUT:

m = 5 (generate 5 elements, then call addLast() many times to add them to the dataList)

n = 5 (generate 5 elements, then call enQueue() many times to add them to the requestQueue)

Input a question: 2 (choose f2() to run)

OUTPUT:

Before

Data List:

001, 0, 10, 200

003, 0, 3, 70

004, 0, 4, 100

005, 0, 3, 70

=========

Request Queue:

1, 100

12, 500

4, 50

4, 400

=========

After

Data List:

001, 0, 10, 200

003, 1, 3, 70

004, 0, 4, 100

005, 0, 3, 70

=========

Request Queue:

12, 500

4, 50

4, 400

=========

**Explaination**:

* + The request (1,100) will be removed from the requestQueue, and then this request will be processed.
  + The request is for renting a room for one person (size=1), with the available money being 100 (price=100).
  + The system finds two suitable rooms (empty (status=0), priced <100, with the lowest price being 70), which are 003 and 005.
  + The system then choose room 003 because it appears before room 005 in the dataList. Finally, the status of this room will be updated to status=1 to indicate that it is now used.

1. f3(): 2.5 marks – Serve all requests in the requestQueue

You should perform the pair of operations **deQueue()** and **rent()** for **all elements** in the **requestQueue**. The input and expected output used to test your code are as follows:

INPUT:

m = 5 (generate 5 elements, then call addLast() many times to add them to the dataList)

n = 5 (generate 5 elements, then call enQueue() many times to add them to the requestQueue)

Input a question: 3 (choose f3() to run)

OUTPUT:

Before

Data List:

001, 0, 10, 200

003, 0, 3, 70

004, 0, 4, 100

005, 0, 3, 70

=========

Request Queue:

1, 100

12, 500

4, 50

4, 400

=========

After

Data List:

001, 0, 10, 200

**003, 1, 3, 70**

**004, 1, 4, 100**

005, 0, 3, 70

=========

Request Queue:

Empty

=========

**Explaination**:

* + Similar to the f2() function, the request (1,100) will be served in room 003.
  + The request (12,500) will not find a suitable Room due to the large number of people (size=12), which exceeds the size of all available rooms.
  + The request (4,50) also fails to find a room because it carries too little money (price=50), specifically less than the price of the cheapest room (price=70).
  + The request (4,400) has two suitable rooms: 001 (price=200) and 004 (price=100), so room 004 is chosen because it has the lower price.
  + As a result, rooms 003 and 004 are rented, and the requestQueue is empty.

1. f4(): 2.5 marks – Count the available Rooms after serve all requests in the requestQueue

You should: (1) perform the pair of operations deQueue() and rent() for all elements in the requestQueue, then (2) count the available Rooms with status = 0.

The input and expected output used to test your code are as follows:

INPUT:

m = 5 (generate 5 elements, then call addLast() many times to add them to the dataList)

n = 5 (generate 5 elements, then call enQueue() many times to add them to the requestQueue)

Input a question: 4 (choose f4() to run)

OUTPUT:

Before

Data List:

001, 0, 10, 200

003, 0, 3, 70

004, 0, 4, 100

005, 0, 3, 70

=========

Request Queue:

1, 100

12, 500

4, 50

4, 400

=========

After

Data List:

001, 0, 10, 200

003, 1, 3, 70

004, 1, 4, 100

005, 0, 3, 70

=========

Request Queue:

Empty

=========

Available Room(s): 2

**Explaination**:

Similar to the function f3(), rooms 003 and 004 are rented. At that point, the Hotel has two available rooms: 001 and 005.

--- End ---