Data Structures and Algorithms
Spring 2020
Assignment # 05
CLO 02

Total Marks: 10 Marks	Class: BCE-4A
Submission Deadline: Thursday, May 14, 2020	

Complex Engineering Problem

Read this document very carefully and try to digest it.

Maximum allowed persons in a group = 02.!! ☺

Start early; it is not a one-day assignment.!!

Collaboration Policy:

You are not allowed to share code. So do not give your code and do not take code from anyone. You might end up in trouble, and always remember that, "**Honesty is the best Policy**".

Submission Policy:

Submit a **.rar file** which be having all your**.cpp** and **.h files** along-with with the input and the output text files. Another Readme file containing your group name, group members name and any other info must be accompanied with the C++ files. It is recommended that you should also make a repository of your project on GitHub, which may get you some extra credit!

Deadline will not be extended, and I expect you to fully utilize your time right from the first day and please ask in case of any ambiguities. There are 02 persons allowed in a group so that you may complete this assignment within due date. So, use all the possible means and try to use agile software development practice of **Pair Programming**.

[PS: Please take this assignment seriously, as it constitutes **10 absolute marks** in this course grading. So, try to make the best out of it.]

Coding Requirements (these carry credit):

1. You must use modular programming style i.e., multiple files for this assignment
2. Write meaningful comments. Do not litter you code with comments only when they are needed.
3. Correct working project gives 60% credit; and rest of the 40% credit will be from the viva session.

The Problem Statement:

BAHRIA UNIVERSITY, BCE students, although awesome, are major procrastinators!!! You give them two weeks for an assignment, and, inevitably, they wait till two or three days before the due date to start working on it…and that is for the average BCE students. But then there are those that just really love being penalized and wait until the actual due date to begin coding. Invariably, and as with all programming students (even those that start early), these students run into obstacles in their understanding of the assignment, the requirements, the way to code, the syntax, and they end up having a plethora of questions.

So, what happens: The Computing lab, where several (imaginary) TAs have been sitting idle, checking Facebook feeds and ticktock for a week and a half, finally gets busy. The Monday before due date, a lot of students show up. Tuesday comes, and the number of students wanting help more than doubles. Wednesday...? Well, Wednesday is just stupid! So many students come, wanting help, and unfortunately, not all receive the help they require because of the overwhelmingly large number of students.

***The Problem*:** with such a congestion of students, the TAs have no way of knowing who was first, who should come next, etc. And to get help, you basically need to be an extrovert, willing to chase down one of the TAs and drag them to your desk for help. What if you are a shy student? Fugetaboutit; good chance you may not receive help.

The solution to this problem is rather trivial: any student who has a question MUST stand in **LINE** to ask a question from the TA. To be fair and to prevent one student from monopolizing a TA's time, if a student has several questions, they can only ask one question at a time. After they ask their first question, if they have subsequent questions, they must go to the back of the line and again wait for their turn. For the purposes of this assignment, we assume that each question takes exactly 5 minutes to answer, which we will refer to as one Q/A session between a student and TA.

Additionally, for the purpose of streamlining the lab, facilitating the procurement of help, and eliminating extraneous problems related to different compilers, coding bugs and suspicious errors, etc., the BAHRIA UNIVERSITY BCE Dept. TA Lab (aka the "the Cave") comes equipped with a **Laptop Dispensing Minion** (LDM) (seriously..! 😊). The students will issue the laptops from the LDM and will use them for their work until the closing hours when they will have to deposit them back to LDM.
These state-of-the-art (well at least for 20020) laptops are sporting Microsoft Windows 10, several software along with Microsoft Visual Studio 2019 and the current version of DEV-C++ and a fully functional and updated Kaspersky's Antivirus.

Before a student can stand in the TA line and get help from a TA, each student must first stand in the **laptop line** to take/issue a laptop from the LDM.
When a student has finished asking all of their questions, or when the TAs are no longer in the lab (because they have left for the day), the student must then enter the **laptop line**

to return the laptop to the LDM, BEFORE they can exit the lab.
We assume that the process of **issuing** a laptop takes one minute, and the process of **returning** a laptop also takes one minute. We also assume (and rightly so!) that all BAHRIA UNIVERSITY BCE students are honest and would never, ever, ever take one of these rock-star laptops with them. Therefore, for the purpose of this simulation, all students who take a laptop do indeed return the laptops.

Your Assignment is to write a simulation that models the lab specifications for the three days (Monday, Tuesday, and Wednesday) before an assignment is due. Your three-day simulation will run *p* times, where *p* is the number of programs in the semester.

<u>You will need to implement this as follows:</u>

1. <u>Students:</u>

You will need to make a student **struct** as follows:

```
typedef struct DataStructuresStudent {

      string firstName;
      string lastName;
      int enterTime;
      int numQuestions;
      int numAnswered;
      int laptopSerialNum;
} student;
```

- **firstName** and **lastName** should be self-explanatory
- **enterTime** is the time, in minutes after 12:00 PM that a student enters the lab. For example, if the value here was 0, that means the student enters the lab when it opens, at 12:00 PM. If the value here were 141, that means that this student enters the lab at 141 minutes after 12:00 PM, or 2:21 PM
- **numQuestions** is a single positive integer representing the number of questions that this student wants to ask
- **numAnswered** is the number of questions that were answered. Depending on lab congestion, there is a chance that not all questions will get answered. As such, now we will track many questions each student has had answered, and we will use this when printing individual exit detail as well as the overall lab statistics
- **laptopSerialNum** is the serial number of the laptop this student borrows while in the lab

You will read the list of students attending office hours from a file. These students will be stored in an array of pointers, where each pointer points to a **<u>`struct student`</u>** in memory. So, at some point of the simulation, if there are 40 students that are read into the file, you will make an array, 40 cells long, of pointers to **<u>`struct student`</u>**. Then, for each student you read in, you allocate space dynamically for a **<u>`struct student`</u>** and store their information into that **<u>`struct`</u>**. Students will be in chronological order in the file – i.e. the first student in the file will be first to enter the lab. More than one student can enter the lab at any given minute. However, when this happens, the order they appear in the file is the order in which they will be added to the Laptop Line.

2. Laptop Waiting Line (Laptop Queue):

As mentioned, as soon as a student enters the lab, they must stand in line to check out a laptop. You will implement this line as a **queue**. The choice is yours as to whether this will be an array-based queue or a linked list-based queue or even an STL Queue. Either way, you will be using pointers since each student is saved in a struct **`student`**, which is referenced by way of a pointer.

3. Stacks of Laptops:

The LDM stores all laptops in a stack and identifies them by their serial number. Therefore, you will implement this as a **stack**. The choice of an array-based stack or linked-list based stack is up to you. The input file will have a list of laptops at the beginning which will be placed into the **stack**. This is constant over the whole simulation – i.e.: no new laptops are added after the beginning. However, the order in which they are stored in the stack will surely change, just as the order of trays in a cafeteria change. Some students will use the laptops longer than others, resulting in the laptops being placed back on the stack at variable times. Note: the laptop stack will NOT reset to its original state after each simulated day. Rather, the order of the laptops, at the end of any given day, will be the order for the start of the next day.

4. TA Waiting Line (TA Queue):

The TA waiting line will also be implemented as a **queue**; again, the choice is yours as to whether this will be an array based queue or a linked list based queue. If you choose the latter, in addition to the head/front pointer, which points to the front of the list, it does make sense to maintain a pointer to the end of the list, thereby allowing for an O(1) enqueue operation. Otherwise, all enqueues result in traversing the entire list, an O(n) operation. This also applies to the Laptop waiting queue.

5. The TAs:

The TAs should be stored, alphabetically*, in an array of structs, where each cell of the array represents one TA, whose information is stored in the following struct:

```
typedef struct teachingAssistant {
      string name;     // only use first names for the TAs
      int startTimes[3];
      int endTimes[3];
      student studentWithTA;
      int minute;
} TA;
```

- **name** is just the first name of the given TA
- **startTimes** is an array of start times for each of the 3 days of the simulation. The start times are stored as several minutes after 12:00 noon.

- **endTimes** is an array of end times for each of the 3 days of the simulation. The end times are guaranteed to be later than the start times for each day, unless they are both 0, indicating that this TA has no office hours on a given day. Note that the start time may be 0, indicating that their office hours start at noon, and the end time may be something else. Only if both numbers are 0 does it mean that they have no hours that day.
- **studentWithTA** is a pointer of type struct **DataStructures** student, showing which specific student the TA is helping. When a student is dequeued from the TA waiting line, what we are dequeuing is simply a pointer (an address) of that student in memory.
  This pointer is then saved into this member of the appropriate **TA's struct**, showing that the TA is working with this student. This allows us to maintain the integrity of the "student", in memory, therefore allowing us to the enqueue that student back into the TA waiting queue (if they have more questions) or into the Laptop queue if they are finished with questions.
- **minute**s is an integer indicating the number of minutes a TA has spent with their current student. This allows you to make sure that the student only spends 5 minutes with the TA

When any TA is done helping a student and it is past their quitting time, they should leave the lab. At this point, a check should be done to see if there are any TAs still on duty. If not, all students currently waiting in the laptop line to pick up a laptop will need to leave, and any students in the TA line will need to move into the laptop line to return their laptops, no matter how many questions they still had. The day's simulation should continue until all laptops have been returned to the LDM.

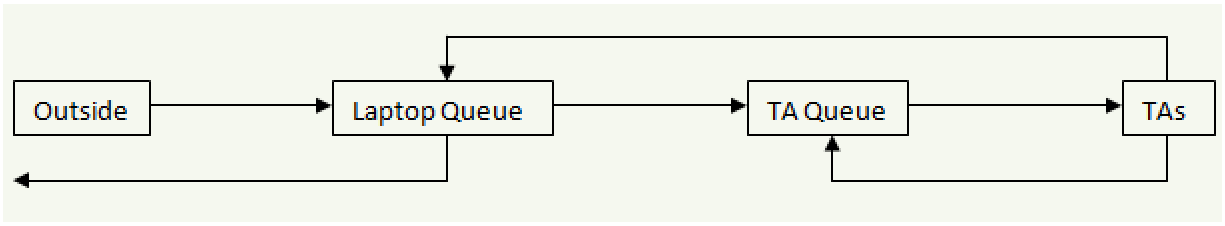
*TAs are already alphabetized in the input file. You do not need to check for this doing your program if you read in and store the TAs properly. Then when the TAs take a student (from the TA line) to work with that student, the TAs do this according to their alphabetical ordering (for example Ali first, then Basit, and then Saba). But again, if you read them in correctly, since they are already alphabetized in the input file, you should be fine.

More Details:

For the purposes of the simulation, the lab will open at noon, and "close" no later than 8:00 PM.
Thus, you do not need to distinguish between AM and PM, as all the hours will be PM. When time is displayed, it must be displayed in standard time formatting, e.g.: "12:23 PM", "1:30 PM", "6:59 PM", etc.

If at any point, there are no TAs on duty, you may assume that the lab hours for the day are over, and any students still waiting in line are out of luck. TAs will remain on duty at least if their office hours for each day, and possibly up to 4 minutes longer if they are helping a student. But once their office hours are over, they may not help any new students. If they were the last TA on duty at that point, all remaining students should return their laptops, the lab should close immediately afterwards, and you should print out the appropriate information (see output). Students who did not get any help that day may or may not be in the line again the next day, but you do not need to keep track of this.

The following chart will help you keep track of how to organize the students into the various lines. The "outside" box refers to the array of students that you read in at the beginning. You will refer to this array to see when students enter the lab, which of course, equates to entering the Laptop waiting queue. The Laptop Queue contains students picking up or dropping off laptops. The TA Queue contains students who have a laptop and are waiting for a TA, and the TAs section represents students who are currently being helped by TAs.



And now the nuts and bolts:

- The lab does not open until the first TA arrives. You can assume that students know the TAs' schedule and do not arrive to the lab until the lab is open (the TA is present).

- If a student is in the outside line and the current time is equal to their arrival time, they should immediately be placed into the laptop queue. As already mentioned, multiple students CAN have the same arrival time. But the order that you place them into the laptop queue is based on the order that they are in the file.

- Each minute, one person may interact with the LDM (Laptop Dispensing Minion), and either pick up or return a laptop. (Assume that this takes a full minute, so nobody else can talk with the LDM until the next minute.)

- If a student enters the lab at 12:20 PM, for example, they will also get in the laptop line at 12:20 PM. The borrowing takes 1 minute, so at 12:21 PM, the borrowing transaction is complete, and the student can enter the TA line.

- At every minute interval, if a TA is not working with a student, they will take a student from the front of the TA line.

- Let us say a student begins working with a TA at 3:19 PM. So, at 3:24 PM, they will either be enqueued back into the TA line (if they have more questions), or they will be enqueued into the laptop line to return the laptop (if they do not have more questions). ALSO, at 3:24 PM, that TA can immediately begin working with the next student (if one is available).

- So, there should be no delay from when a TA finishes with one student and takes the next.

- If student A enters the lab at the same time that student B finishes his Q&A session and enters the laptop line (to return a laptop), student B will enter the laptop line BEFORE student A enters the laptop line (this just makes the coding less messy). So, you can simply process the laptop line last.

- If student A issues a laptop and enters the TA line at the same time that student B finishes a Q&A session and re-enters the TA line (for another question), student A should get in the TA line first, since they haven't had the chance to even ask one question.

- If a TA is forced to stay a bit longer than their hours because they have not finished that specific 5-minute question/answer, this is allowed. However, once a TA is past their official hours, they may not help any new students. For example, if the TA finishes at 4:00PM, a student can ask a question of this TA up through 3:59 PM. So, for example, if a student asks this TA a question at 3:57 PM, that Q/A session will finish at 4:02 PM. But once a TA is officially no longer on duty, they cannot help a new student.

- At 8PM, all students in the TA waiting line enter the Laptop line (in their same order) to return the laptops.

- When the lab closes, you must empty the Laptop Queue. There is a possibility that some of the students in the queue at this point have no laptops. In real life, they would just leave, as there is no reason to wait in a line. However, that complicates the code. So, we will assume that students can only leave the line via dequeuing, which translates to them staying in line until they are at the front of the line. However, to expedite this process, if a student does not have a laptop to return, we assume that this takes ZERO time, thus allowing you to immediately work on the first actual student that does have a laptop to return. For example, the lab closed, and we are emptying the laptop line. At time X, let us say there are 10 people in line, with the first three not having a laptop (they were waiting for one before the lab closed). At time X, those three students are dequeued. Additionally, the fourth student is dequeued and initiates the laptop return.

- You can assume that there will always be enough laptops for the number of students; meaning, this system will never experience deadlock. The number of laptops is equal to or greater than the number of students that will enter the lab on a given day.

<u>Input File Specification (input.txt)</u>

You will read in input from a file, "input.txt". The name MUST BE "input.txt". Have this AUTOMATED. Do not ask the user to enter "input.txt". You should read in this automatically. (This will expedite the grading process.)

The input file has a single positive integer, *n*, on its first line, specifying the number of laptops to be placed in the machine. On the next line, there will be *n* serial numbers, indicating each of the laptops. These are represented as positive numbers small enough to be stored in an int variable. They should be placed into the stack in the order the numbers appear. Thus, the last laptop on the stack will be the first one removed by the first student who needs one.

On the next line will be another positive integer, *t*, indicating the number of TAs that will be on duty during the simulation. On the following *t* lines, there will be a set of data indicating each TA's name, along with their office hours for each of the days of the week. These will be given in the order start, end, start, end, etc. for each of the 3 days, starting with Monday. TAs may have overlapping hours on some days, and there may be many of them on duty at once. This TA information will remain fixed over all the simulations.

The next line will be a single positive integer, *p*, representing the number of programs for the semester, which, in turn, represents the number of times we need to run this three-day simulation.

On the next lines will be *p* sets of data for each of the three-day simulations. Note, that since each of the *p* simulations is really a simulation over three days, each of the following *p* sets of data will have three days' worth of information.

The first line of each simulation will contain a single positive integer, *m*, never exceeding 300, representing the number of students who get into line for that Monday, followed by *m* lines containing information about each student (see below). The next line will contain a single positive integer, *t*, never exceeding 300, representing the number of students who get into line for that Tuesday, followed by *t* lines containing information about each student. Finally, the next line will contain a single positive integer, *w*, never exceeding 300, representing the number of students who get into line for that Wednesday, followed by *w* lines containing information about each student.

Note: for each of the three days, and for each simulation, the order that the students are found in the input file will be the order that they arrive in the lab.

<u>Information on each student</u>: The first piece of information on each of these lines will be a non- negative integer (less than or equal to 480) representing the number of minutes, after noon, the student arrives to get into line. This is followed by the first and last name of the student (all upper or lower-case letters with a length of fewer than 20 characters each). After this will be a single positive integer indicating the number of questions the student needs to have answered. All pieces of information will be separated by a space on each line.

Output Specification

<u>*NOTE*: You should generate your output to a FILE that you will call "out.txt".</u>

For each of the semester programs (each resulting in a 3-day simulation), print out a header with the following format:

```
**********
Program X:
```

Where X is the ith program of the semester. Follow this with a blank line. For each day of the three-day simulation, print the following header: `Day`:
where *Day* represents the day of the simulation (*"Monday", "Tuesday", or "Wednesday"*).

Follow this with a blank line.

The following lines will give information about a TA beginning or ending their lab hours, or a student either entering or leaving, picking up or returning a laptop, or starting or finishing their session with a TA. These lines should be printed in the order in which the actions occur. The formatting for these lines is as follows:

```
TIME:  TA_NAME has begun lab hours.
TIME:  TA_NAME has finished lab hours.
TIME:  There are no TAs on duty.  Arslan is now closed.
TIME:  STUDENT_NAME has arrived in Arslan.
TIME:  STUDENT_NAME has borrowed laptop SERIAL and moved to the TA line.
TIME:  STUDENT_NAME is getting help from TA_NAME.
TIME:  STUDENT_NAME has had one question answered and gotten back in line.
TIME:  STUDENT_NAME has no more questions and will now return the laptop.
TIME:  STUDENT_NAME has returned laptop SERIAL and went home HAPPY.
TIME:  STUDENT_NAME has returned laptop SERIAL and went home FRUSTRATED.
TIME:  STUDENT_NAME never even got a laptop and went home FRUSTRATED.
TIME:   STUDENT_NAME has had one question answered but must now return
the laptop and exit the lab.
```

where TIME is the time the customer checked in, STUDENT_NAME is the name of the student (first name followed by a space and then followed by the last name), TA_NAME is the name of the TA, and SERIAL is the serial number of a laptop.

A student is "happy" if all of their questions were answered and is "frustrated" otherwise. Students only go home happy if they had at least 75% of their questions answered. Otherwise, they go home frustrated. So, if a student had 10 questions, but only 7 were answered, they would go home frustrated. But if that student had 8 questions answered, they would go home happy.

The time should be printed out in the following format: `(H)H:MM PM`

The first one (or possibly two) digits represent the hour. The hour must **not** be printed as 0 if it is between noon and 1:00, so you will need to check for this. This is followed by a colon and then the next two digits represent the minute. If the minute value is less than 10, you'll need to add a leading 0, so that it prints as 3:05, not 3:5. (It probably makes sense to have a function that takes in as input the number of minutes after noon and in turn prints out the corresponding time in this format.)

Explanation of Output:

Most of the output is straightforward. The following explanation covers the few cases that may not be completely clear

```
TIME:  STUDENT_NAME is getting help from TA_NAME.
```

- This output should occur AS SOON AS the student begins working with the TA.

```
TIME:  STUDENT_NAME never even got a laptop and went home FRUSTRATED.
```

- This output should occur if a student is in line, to GET a laptop (they came REALLY late to the lab) and the lab closes before they even get a laptop.

```
TIME:  STUDENT_NAME has had one question answered but must now return
the laptop and exit the lab.
```

- - This output occurs when a student is working with a TA, whose hours have finished AND who is the last TA on duty (meaning the lab is now closed). So, when that student finishes with the TA, if they have had at least 75% of their questions answered, they go home happy. Otherwise, you need the above output.

```
TIME:  There are no TAs on duty.  Arslan is now closed.
```

- This message should only print AFTER the last TA has finished their hours AND AFTER their respective "TA_NAME has finished lab hours." message has printed.

Output Day Summary:

Follow each day's output with one blank line. Then you will print that day's lab summary as follows:

```
Day's Lab Summary:
The TA Lab was open for H hours M minutes.
Y students visited the lab.  Out of those students, only Z left happy.
The remaining left frustrated.

Lesson Learned:  do not procrastinate!  Start programs early!
```

where "Days" represents that day of the simulation (Monday's, Tuesday's, or Wednesday's). The lab is open from when the first TA arrives until the last student leaves the lab (by returning their laptop. So if the last TA finishes their office hours at 6:30 PM, works an extra 4 minutes with a student till, 6:34 PM, and if it then takes all the students till 6:42 PM to exit the lab, you would say that the lab is open for 6 hours and 42 minutes.

Follow each day's summary with a TWO blank lines.

*****Helpful Suggestions*****

- As stated, you can use either array based or linked-list based stacks and queues.
- Considering that the array of student pointers is in chronological order based on the time that they enter the lab, it does make sense to view this array as a queue. How is that helpful? You are already implementing two queues (laptop and TA line), and as such, you will already be using the various array or linked-list based queue operations. Therefore, we can "peek" at the enter time of the student at the front of the queue. If it is their time to enter, we can simple perform a dequeue operation. Of course, you continue to dequeue for all students whose enter time as at this specific minute, as multiple students can enter at the same time.
- The structs shown in this file contain the minimum number of fields you need. Feel free to add struct members as you feel appropriate.
- **PS: You may use the Standard Template Library (STL) based implementations of the stacks, and queues etc.**

Implementation Restrictions

There must a single main file named as *mainFile.cpp*. You must use at least two queues and one stack (for the laptops). You must write functions that operate on these structures analogous to the stack and queue functions shown in class. Look closely at the sample input and output files provided with the assignment and make sure you follow the formatting exactly.

Grading Details

Your program will be graded upon the following criteria:

1) Correctness.
2) Use of stacks and queues. Since the purpose of this assignment is to teach you to use stacks and queues, you will not receive credit if you do not use them.
3) Your programming style, comments, and use of white space.

Deliverables

Your submission must include a text file which will be having particulars of your group members and any other information, along with the C++ files and input and output text files.

☺ 💻 **...Best of Luck.** 💻 ☺