Computer Science 1 - Program 4
KnightsRegistrar (Stacks & Queues)
Assigned: 2/29/12
Due: 3/21/12 (Wednesday) by 11:55 pm (WebCourses time)

Objective
Practice using stacks and queues.

The Problem
The unfortunate effects of the University-wide budget cuts are seemingly limitless, with practically every department hurting in some way. Due to lack of funding, the online class registration portal is no longer available; we've turned back the clock about twenty years! In order to register for classes, you must be physically present at the Registrar's Office in Millican Hall. (If it makes you feel any better, all UCF students are in the same boat!) Upon arriving to the Registrar's office (which runs from 12:00 PM to 5:00 PM, 7 days a week), you notice a massive line of students thinking the same thing as you, **"You have GOT to be kidding me!"** Sorry buddy: welcome to the current economy! So, with really no choice left, you must get into this crazy line in order to register. Fortunately the process is streamlined and relatively straightforward, although possibly WAAAY time consuming!

As you get closer to the front of this seemingly endless line, you notice that you won't actually register via another human. Rather, once a student reaches the front of the line, they briefly interact with a Laptop Dispensing Minion (LDM), who issues them a laptop (from a **stack** of laptops) that gives them access to the registration portal via UCF's internal intranet. You're thinking: "Budget cuts forced the online registration portal to be shut down; yet, UCF can somehow afford all these laptops to register with!" You are set at ease, however, upon grabbing yours. Reminding you of the Star Wars intro, "A long time ago in a galaxy far, far away...", these massive bricks, err, um laptops, are most likely from the '90s, sporting some ancient version of Microsoft Windows along with a barely working internet browser. Amazingly, however, they do they work, albeit very slowly. To add insult to injury, however, there are only five laptops! So not only must students wait in this excessively long laptop line, but even once at the front, they may have to wait for one of these rockstar laptops to become available.

Once a student is issued a laptop, they will then spend exactly five minutes registering for classes. For the purposes of this simulation, once the five minutes are up, we will assume that they student has finished their registration. The student must then get into a separate laptop return line, which is, thankfully, a lot quicker than the laptop waiting line (cuz there are only five laptops). Once students are at the front of the return line, they briefly interact with a Laptop Returning Minion (LRM), who confirms the registration is correct, prints out a confirmation page for the student, and places (pushes) the laptop back on the stack of laptops. All registrations are saved in an alphabetical linked-list for the purpose of printing the UCF Daily Registration Report. As in real life, some students will make mistakes with their registration (the input file will detail which students will make mistakes and which ones will not). Upon reaching the front of the laptop return line and interacting with the LRM, and if this student made mistakes, the LRM will inform the student that there are mistakes with the registration. This student will then spend another five minutes correcting the mistakes, and they will then re-enter the laptop return line. It is guaranteed that the five-minute redo will fix the mistakes.

Your Assignment is to write a simulation that models the aforementioned Registration over *n* number of days, where the simulation runs over each minute of every day, from 12 PM to 5 PM.

Implementation

You must use the following structs, as shown. You may, at your own discretion, add struct members as you see fit. The "next" member of the student struct assumes the use of a linked-list queue.

```
typdef struct ucfClass {
      char ID[10];      // class ID, ex: "COP-3502"
      char days[6];     // a combination of "MTWRF" (days)
      char time[20];    // class time, ex: "10:30 AM - 12:00 PM"
} class;

typedef struct ucfStudent {
      char lastName[21];      // student last name
      char firstName[21];     // student first name
      int ID;                 // student ID
      int laptopID;           // serial number of the laptop the student picks up
      int errorCode;          // flag to determine if they will make mistakes
      int numClasses;         // number of classes the student will register
      class *classes;         // array of said classes (2 be dynamically allocated)
      int enterTime;          // time student arrived, in minutes, after 12:00 PM
      int timeLeft;           // countdown timer to measure the 5 min. reg. process
      int timeRegistered;     // Time student finished reg. and left Registrar
      struct ucfStudent *next;  // pointer to next student in queue
} student;
```

Outside Line:

For a given day, you will first read in, from a file, all the students that will ultimately arrive during that day. Students will be in chronological order in the file – i.e. the first student in the file will be first to enter the Registrar's office. You must malloc space for each student, save their appropriate (read-in) information into their student node, and then you will enqueue them into, what we refer to as, the "outside line." What is the "outside line"? The outside line (which is clearly a queue) will contain all of the students who are expected to come during that particular day. As time moves forward, you will remove (dequeue) students from the outside line if the **currentTime** (your daily looping variable over each minute) equals the "arrival" time of the student who is at the front of the outside line. Once a student "arrives" (is dequeued from the outside line), they are immediately enqueued into the laptop waiting line. Note: it is possible that multiple students will "arrive" at the Registrar's office at the same time. Meaning, multiple students can possibly have the same arrival time within the "outside line". Students that arrive at the same time are placed in the laptop line in the order that they arrive (based on the chronological order of the input file). The choice is yours as to whether this "outside line" will be an array based queue or a linked list based 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.

Laptop Waiting Line:

As mentioned, as soon as a student enters the Registrar's office, 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. Either way, you will be using pointers since each student is saved in a `struct student`, which is referenced by way of a pointer.

Stacks of Laptops:

The laptops are to be stored in a stack and are identified by their unique serial number. Therefore, you will implement this as a **stack**. 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 (because of mistakes during registration), 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. The choice is yours as to whether this will be an array based stack or a linked-list based stack.

Laptop Return Line:
The laptop return 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.

Completed Registrations:
Successful registrations will be saved into an ordered linked-list, which will be ordered alphabetically (by last name and then by first name). Although students may have the same last name or the same first name, it is guaranteed that all students have a unique first and last name combination. All registrations in this linked list must be erased on a daily basis at the time of printing the daily summary. We'll just assume they get saved into a master UCF database (beyond the scope of this assignment).

And now the nuts and bolts:

- When a student arrives at the Registrar's office, that student immediately enters the laptop check-out line (during the same minute).
- Once at the front of the laptop check-out line, you can only leave this line once a laptop is available. So even if you are at the front of the line, if no laptops are available, you will not leave the line. Let's say a laptop becomes available at 12:20 PM, then, also at 12:20 PM, you will leave the laptop line and work with the LDM to check out a laptop, which then takes one minute, as described below, resulting in your getting the laptop at 12:21 PM.
- The "brief interaction" with the LDM takes one minute; meaning, it takes one minute to get a laptop. So if you arrive at 4:30 PM and find an empty laptop line, and even if laptops are available, you will not actually get your laptop until 4:31 PM.
- As such, a maximum of one laptop is dispensed every minute. So even if three students arrive at 1:12 PM, they will all enter the laptop line at 1:12 PM, and if all five laptops are available, the first student will get a laptop at 1:13 PM, the second at 1:14 PM, and the third at 1:15 PM.
- If a student gets a laptop at 4:31 PM, they will "finish" registering and will enter the laptop return line at 4:36 PM.
- The "brief interaction" with the LRM also takes one minute; meaning, it takes one minute to confirm successful registration and return the laptop. So if you are at the front of the laptop return line at 4:40 PM (with no mistakes on the registration), you will return the laptop and leave the Registrar's office at 4:41 PM.
- On the flipside, if you are at the front of the laptop return line at 4:40 PM and you have mistakes, you still have this "brief interaction" with the LRM who informs you of said mistakes. You then start your second (and final) five minute registration session at 4:41 PM.
- At a given minute, you could end up with a student taking a laptop from the stack and a student, who is finished, placing a laptop on the stack. So we need a rule: always process the student returning the laptop before processing the student taking a laptop.
- Also, at a given minute, two students could be trying to enter the laptop return line at the same time. This could occur when a student, who had a mistake and had to spend another five minutes, finishes their second five minutes at the same time that another student finishes their original five minutes. So we need a rule: if more than one student is trying to enter the laptop return line at a given minute, always enqueue, first, the student who has the earliest enterTime (the student who has been

at the office the longest).

- Students can arrive at the Registrar's office up until 4:59 PM. You are guaranteed that no student arrive after this time.
- Although the Registrar's office officially closes at 5:00 PM, the office does stay open to finish the registrations of all students who arrived by 4:59 PM. So even if 100 students arrive right at 4:59 PM, this is no problem. The Registrar's office will stay open until all students have been processed. However, it is guaranteed that this will never take beyond 11:59 PM.

Input File Specifications

You will read in input from a file, "KnightsRegistrar.in". Have this AUTOMATED. Do not ask the user to enter "KnightsRegistrar.in". You should read in this automatically (this will expedite the grading process). The file will contain an integer *j* followed by *j* random, unique laptop serial numbers. Next will be an integer *n*, followed by *n* number of days. For each day there will be an integer *k*, followed by *k* number of students, where each student's information will be on a single line as follows:

```
ENTERTIME LASTNAME FIRSTNAME ID NUMCLASSES ERRORCODE
```

`ENTERTIME` is the time in minutes from 12:00 PM. `LASTNAME` is the last name of the student arriving. `FIRSTNAME` is the first name of the student arriving. `ID` is the student's ID number. `NUMCLASSES` is the number of classes the student will register. `ERRORCODE` is the number used to determine if the student has will make an error while registering. 1 signifies an error; 0, otherwise.

Each student line is followed by `NUMCLASSES` lines with class data on each line:

```
CLASSID DAYS TIME
```

`CLASSID` is a string representing the ID of a class (ex. "COP-3502"). `DAYS` is a string of letters representing the days of the week a class occurs on (ex. "MTWRF"). `TIME` is a string representing the time a class occurs at (ex. "10:30 AM - 12:00 PM"). Note: this time (10:30 AM - 12:00 PM) will need be stored as one string. **HINT**: visit Clarification section of discussion board for advice on how to read in this "TIME" string of characters. I will post the easy method to do this.

Output File Specifications

Your program must output to a file, called **"KnightsRegistrar.out". You must follow the program specifications exactly.** You will lose points for formatting errors and spelling.

For each day, print out a header with the following format:

```
**********
Day X:
**********
```

Where X is the n^{th} day of the simulation. Follow this header with one blank line.

The following lines will give information about students entering, checking-out up a laptop, finishing registration and entering the laptop return line, making mistakes and having to redo the registration, and finally, returning the laptop upon successful completion of registration. These lines should be printed in the order in which the actions occur. The formatting for these lines is as follows:

```
TIME:  STUDENT_NAME has arrived at the Registrar and entered the laptop line.
TIME:  STUDENT_NAME has checked-out laptop # SERIAL.
TIME:  STUDENT_NAME finished registering and entered the laptop return line.
TIME:  STUDENT_NAME made an error and must redo the registration.
TIME:  STUDENT_NAME successfully registered and returned laptop # SERIAL.
```

Where `TIME` is the time an action occurred, `STUDENT_NAME` is the name of the student (first name, followed by a space, and then followed by the last name), and `SERIAL` is the serial number of the laptop the student is using.

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'll 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 12:00 PM and, in turn, prints out the corresponding time in this format.)

Follow each day's output with one blank line. Then you will print the day's statistics as follows:

```
*** Day X:  UCF Daily Registration Report ***:

The Registrar received Y registrations as follows:
```

Where `X` is the day number (of the simulation), and `Y` is total number of students that came that day. This is followed by an alphabetical (last name, and then first) printout of all registrations that occurred during the given day:

```
LASTNAME, FIRSTNAME, ID # STUDENTID
      Time Registered:  (H)H:MM PM
      Classes:
      | CLASSID  | DAYS  | TIME               |
      ...
LASTNAME, FIRSTNAME, ID # STUDENTID
      Time Registered:  (H)H:MM PM
      Classes:
      | CLASSID  | DAYS  | TIME               |
      ...
```

Where `LASTNAME`, `FIRSTNAME`, and `STUDENTID` are the respective student data. This is followed by the time that the student completed Registration (with the LRM) and left the Registrar. Finally, this will be followed by the class data (`CLASSID`, `DAYS`, and `TIME`) of the classes the student registered for, printed in the order they were originally read in from the file. The print literal used to print the formatted list of registered classes is: `"\t| %-8s | %-5s | %-19s |\n"`

Follow each day's summary with TWO blank lines.

See sample input and output files for examples.

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

- As stated, you can use either array based or linked-list based stacks and queues. Although arrays are generally easier to code, some students find array-based queue implementations confusing. My personal suggestion would be to use the linked-list based queue code that is shown on the course website (with necessary modifications, of course). I also recommend to use the ARRAY version of stacks, since our stack is simply a stack of ints (serial numbers of laptops)! So you can basically use the EXACT same array-based code.
- Read and FULLY understand this write-up BEFORE trying to code.
- As this is a simulation, the "work" will be done in the main loop that simulates over each minute of the day. During any given minute, many things get processed. You must process the Laptop Return Line, the "outside line", the Laptop Check-out Line, the LDM and LRM, and you must process the actual students who have laptops and are in the middle of registration (their 5 minutes).
- The order that you process these various elements will dictate the order of the various print statements (shown above) that you will print to your output file. So **you will need to study the sample output file** provided to try to figure out what order you need to process (code) things in.

*****WARNING*****

Your program MUST adhere to this EXACT format (spacing capitalization, use of colons, periods, punctuation, etc). The graders will use very large input files, resulting in very large output files. As such, the graders will use text comparison programs to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, as the graders will be forced to go to text editing of your program in order to give you an accurate grade.

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications listed on this write-up.
2) Your algorithmic design.
3) Correctness.
4) **Use of stacks and queues. If your program does not use stacks and queues, you will NOT get credit for the assignment. Period.**
5) The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 85-90% on it.)
6) Compatibility to CodeBlocks. (If your program does not compile in CodeBlocks, you will get a sizable deduction from your grade.)
7) Your output MUST adhere to the EXACT output format shown in the sample output file.

Restrictions

Name the file you create and turn in KnightsRegistrar.c. Although you may use other compilers, your program must compile and run using CodeBlocks. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. You should also include comments throughout your code, when appropriate.

Deliverables

A single source file named KnightsRegistrar.c turned in through WebCourses.