**Project 3 Guidelines**

**Introduction**

Imagine a situation where you are receiving packets of information about students at irregular intervals. Each packet contains information about a single student: name, student ID, gpa, and major. As data arrives, it should be stored in a queue until it can be processed. Data is "processed" by popping it out of the queue, and then storing it into a hash table so that it can be easily searched in the future.

We will simulate incoming data packets with a text file that contains packet information and the packet arrival time. The code automatically inserts a time delay to simulate the packet's arrival time. While you are reading data into the queue, a separate thread will be simultaneously running. This thread will be constantly checking the queue. If the queue contains data, then you will pop one student packet from the queue, hash it, and store it in the hash table.

Finally, you will demonstrate that your code works by looking up some students by their student ID.

**The Details**

Here are the specific steps you will need to implement:

[1] create a class named "student" that stores the fields for each student  
[2] create a queue (hint: super easy in python)  
[3] push each student's object into the queue  
[4] implement a hashing function that hashes the student ID number  
[5] pop students off the queue and place them into the hash table appropriately  
[6] implement a function that will look up students in the hash table and print out their information

**The Hints**

Items 1-3 are meant to be simple

Item 4 is the trickiest part - you will have to determine a good hash function for the student ID numbers that will reasonably  distribute the data within the hash table. As a hint: each 9-digit student ID number was constructed using different bits of information about each student. Some of that information, such as year of birth, will be common to many students. Therefore many students will share certain common digits in their student ID numbers, and those would make for bad hash inputs because they wouldn't distribute uniformly.

Items 5-6 are medium difficulty

Please don't re-invent the wheel. I've written a lot of code to try to make this as straightforward as possible. For example, I handle reading the data from file. No need for you to do that.

Feel free to make a shortened version of the data file while you are testing and debugging your code.