Data Structures **Name:** _____
Fall 2016, Prof. Bolton **Net ID:** _____
Exam 2

This exam contains 5 pages (including this cover page) and 2 questions. Total of points is 5.

Conditions: You are permitted writing utensils and the exam. No other items are permitted, e.g. no notes, no text, ...

Note: The pages are double-sided. **PLEASE WRITE your name** on all pages.

I, _____, understand the above statements and agree to follow these terms, and upon my honor, I swear that the answers provided are of my design and of my effort alone. I have not received nor viewed answers from any source but myself.
< *sign* > _____

---

1. (4 points) For project 3, you implemented a hash table to represent a dictionary of words. You are tasked similarly here. Create a hashtable structure (with a universal hashing scheme – see part B) to store a dictionary of strings contained in an input dictionaryFile. The size and contents of the inputFile are unknown, but you may assume that each string is less than or equal to 30 characters long.

    A. (1 pts) Structurally describe your implementation. Please include all classes and a brief description of all member **variables**. Feel free to simply create a UML class diagram(s) with *very very brief* notes about each member variable.

B. (10 pts) You decide to use the random vector, universal hashing scheme (the variant of the random matrix method) to reduce the probability of collisions. It is impractical to create a static hash function that would work for all possible inputs, thus you implement a method that creates a universal hash dynamically (at run-time). This has the added benefit of allowing for re-draws or hash recreation (re-hashing) if a previously created hash function is poor. Using pseudocode or c-like code, implement the member method **void hashtable::createUniversalHash(...)** which intializes / creates a universal hash function (likely by initializing some member variable(s)).

C. (10 pts) Using pseudocode or c-like code, implement the hash function **int hashtable::hash(string s)** which maps a string to its hash bucket.

D. (1 pts) Collision resolution is a concern for hashtable implementations. Describe an **efficient** collision resolution scheme for this particular scenario. **Be concise yet precise.** *Note: There are many options for collision resolution and general design schemes. Justify your design decisions related to collision resolution. Support this justification with all pertinent time and space implications related to the (1) the construction of the dictionary and (2) the use of the dictionary.*

2. (1 point) Consider a scenario where a set of n entities (each with a specific priority) are vying for use of m similar shared resources, which we will call gadgets. Assume n is much greater than m. The entities can access the gadget one-at-a-time. As such, m queues are created for each of the m gadgets. Thus entities vying for access to a gadget are placed into the queue (associated with that gadget) if the gadget is in use, or if the queue is not empty. Once a gadget becomes available (and the queue is not empty), access to the shared resource is granted to the entity with the highest priority in the queue. Within this scenario, assume that the gadgets are created and destroyed very frequently; and thus, when a gadget is destroyed, the corresponding queue is merged with another gadget's queue. Given this scenario answer the following implementation questions. **Your implementation should provide for an efficient solution to this scenario.**

    A. Discuss the structural design of GadgetPriorityQueue, including all member variables (including a very brief description of each) and allocation details (no need to discuss member methods). Feel free to use a UML-class-like diagram.

B. Using pseudocode or c-like code, implement the method void GadgetPriorityQueue:: mergeAndDestroy(GadgetPriorityQueue &gpq), that merges gpq with the calling GadgetPriorityQueue and appropriately deallocates gpq (if needed).

C. There are many implementation options and design schemes for queues in general and priority queues. Justify your design decisions related to your GadgetPriorityQueue implementation. Support this justification with all pertinent time and space implications related to method GadgetPriorityQueue::mergeAndDestroy(GadgetPriorityQueue &gpq), in particular.