Assignment #3
CS 163 Data Structures

Submit your assignment to the D2L Dropbox (sign on via d2l.pdx.edu)

****Assignments in CS163 consist of written homework and programming****
****All parts are required to get a grade on this homework ****

1) **Evaluate Data Structures.** We learned about linear linked lists in CS162, but now we have been working with combinations of arrays and linked lists in our solutions. As part of this think about the positive and negative aspects of combining data structures (such as with the flexible array or hash table with chaining). Your answers should include evaluating the data structures in terms of standard functions (insert, remove, retrieve, display) and run time efficiency and memory usage.

2) **Ethics.** *Background:* The idea of table abstractions is to allow clients to work by the value of the data without concern about the order or how the data is stored. *Question:* As a designer of a table ADT, what is our ethical responsibility to provide fast retrieval for all possible search keys? In your answer consider the run time and memory impact to this answer. This must represent *your thoughts about ethical concerns that may arise.*

3) **Terminology.** In your own words, define the following terms and discuss how they might be used:
   a) Source Code Control
   b) Makefile
   c) Truth Table
   d) Namespace
   e) GDB
   f) Operand
   g) grep

Programming Project

Programming –Goals: The goal of the third program is to create a hash table using chaining per Topic#6. Hash tables are very useful in situations where an individual wants to quickly find their data by the "value" or "search key". You could think of them as similar to an array, except the client program uses a "key" instead of an "index" to get to the data. The key is then mapped through the hash function which turns it into an index!

Programming – Problem Statement: In program #2 we started building a game program. The first step was to create our list of Avatars with their corresponding weapons queues. In Program #3 we will build the commands or functionality of the game. Here is the minimal set of commands to support:

1. Movement: A player (e.g., their Avatar) can move in different ways (forward, backward, jumping).
2. Weaponry or Tools: They can use their weapon (fire, reload, discard) or tool.
3. Special Powers: There may be magic where a player can get special powers.
4. Modal Settings: Can the player sleep? Is the player in creative (building) or living (adventure) mode?

The specifics of these types of commands depend on what game you are thinking of. Your client will load up the commands which will be stored in a hash table. Then, begin a test program by simulating the use of the commands. Some commands (e.g., using a weapon) may need access to the weapons queue. *If your stack and queue were not fully functional, then use a simple CLL for the weapons (without the queue). That is acceptable.*

Other commands may need access to a data structure to store the Avatar's location. Right now pick a simple data structure for that (an array or a LLL). When we get to Program #5 we will build a map using "graph" that will keep track of their location!

Data Structures: Write a C++ program that implements and uses a **table abstract data type using a hash table (with chaining)** to store, search, display, and remove game commands. Create a hash function based on the command (e.g., fire weapon). The hash table must at a minimum keep track of the following:

(1) Name of command

(2) Functionality expected for this command

(3) What is its state? (not relevant for all commands) On or Off

(4) What rules are defined for it?

(5) Others?

What does retrieve need to do? It should take the command and supply back to the calling routine information about the state and rules of the command. Retrieve, since it is an ADT operation, should not correspond with the user (i.e., it should not prompt, echo, input, or output data).

Evaluate the performance of storing and retrieving items from this table. Monitor the number of collisions that occur for a given set of data that you select. Make sure your hash table is small enough that collisions actually do occur, even without vast quantities of data so that you can get an understanding of how chaining works. And, make sure your hash table's size is a prime number. Try different table sizes, and evaluate the performance (i.e., the length of the chains!). **Your design writeup must discuss what you have discovered. You need to test this with large quantities of data to get an understanding of how chaining is a solution for collision resolution!**

Things you should know...as part of your program:

1) Do not use statically allocated arrays in your classes or structures. All memory must be dynamically allocated and kept to a minimum!
2) All data members in a class must be private
3) Never perform input operations from your class in CS163
4) Global variables are not allowed in CS163
5) **Do not use the String class! (use arrays of characters instead and the cstring library!)**
6) Use modular design, separating the .h files from the .cpp files. Remember, .h files should contain the class header and any necessary prototypes. The .cpp files should contain function definitions. You must have at least 1 .h file and 2 .cpp files. **Never "#include" .cpp files!**
7) Use the iostream library for all I/O; do not use stdio.h.
8) Make sure to define a constructor and destructor for your class. Your destructor must deallocate all dynamically allocated memory.
9) Remember that 20% of each program's grade is based on a written discussion of the design. *Take a look at the style sheet which gives instruction on the topics that your write-up needs to cover.*