Lab Exercise #6
Due Date: Mar 29th 2024
LRU Cache

Description:
For this lab exercise, you will implement a **LRU Cache.** The Least Recently Used (LRU) cache is a data structure that stores a fixed number of items. When the cache reaches its capacity, it evicts the least recently used item to make room for new ones. This eviction policy optimizes cache performance by removing items that are least likely to be needed again soon, thereby maximizing the utility of limited cache space.

starter.zip (Starter Code):
The following is a brief description of the starter code provided to you:

- **lru.h**
    - **class Node:** Stores the key and value member attributes of a node item in the LRU Cache
    - **class LRUCache:**
        - The LRU Cache contains the following private member variables:
            - **int maxSize:** The maximum size of the LRU Cache taken as a parameter in the constructor. If the size of the LRU Cache exceeds the maximum size, the least recently used item must be evicted from the LRU Cache
            - **list<Node> l:** A Doubly Linked List STL data structure that stores Node objects in reverse LRU order. The head of the linked list must contain the item that is most recently used and the tail must contain the item that is least recently used.
            - **unordered_map<string, list<Node>::iterator> m:** This map stores the keys of various node items and the value is the iterator pointing to the Node in the list that contains the key.
        - The LRU Cache must define the following public methods:
            - **void insertKeyValue(string key, int value):** Inserts a Node with a particular key and value into the LRU Cache. If a Node with the key already exists in the LRU Cache, the value is updated. Upon inserting/updating the Node with this key, this Node now becomes the most recently used item in the cache. Upon insertion, if the size of the LRU Cache exceeds the maximum size, the least recently used item must be evicted from the Cache (i.e. should not be stored anymore).
            - **int* getValue(string key):** If a Node item with the key exists, a pointer to the value of the key is returned. If the key does not exist, you can return nullptr. If the key exists, the corresponding Node with this key becomes the most recently used item in the cache.

        - **string mostRecentKey()**: If the LRU Cache is not empty, return the key of the Node item which is most recently used. If the LRU Cache is empty, return an empty string (“”).
- **main.cpp**
    - You can use this file to instantiate LRUCache objects and test your code.

Contract (TASK):
Your task is to implement the methods of the LRUCache class. You have been provided with non-functional definitions of these methods. **You must implement these method definitions in the given lru.h file**, by replacing the current non-functional definitions with your functional implementations. **Do not create an additional cpp file to define these functions, make sure that they are part of the header file.**

Deliverables and Submissions:

In a zip folder named using the format **<FirstName>-<LastName>-<UIN>-<LE6>.zip** you must submit the following file(s) to Canvas:
- **lru.h** (containing the functional method definitions of the LRUCache class)

Do not use angular brackets in the name of the submission folder.

Grading:
Coding (20 points)
Each method of the LRUCache will be tested using an automatic testing infrastructure. You will be provided with the testing infrastructure which you can use to evaluate your implementations.

The grading rubric is as follows:
- Basic Insertion & Retrieval: 3 points
- Updated Existing Key: 3 points
- Retrieve Value: 5 points
- Insert Additional Items (Cache Full): 5 points
- Non-Existent Keys: 1 point
- Empty Cache: 1 point
- Small Cache Size Limit: 2 points

testInfrastructure.zip (Test Methodology):
In order to familiarize you with the auto-grading environment on our end we are providing you with an automated test infrastructure that can be used to check the correctness of your code. The testing infrastructure is constructed as follows:
- **lru_test.cpp**: This cpp file defines all the test functions that have been mentioned in the grading rubric above. It also runs these tests and outputs your final score.

- **test.py:** This file compiles lru_test.cpp in order to produce the executable. It then runs this executable which will indicate your score (as well as the test cases you have passed or failed).

Note: There is a possibility that one or more of your methods causes a segmentation fault which in turn causes the test script to crash when calling that particular method. In such a case, we will not be able to calculate your score for the assignment and you will be assigned a default grade of 0. After the grades are posted, you will be given 1 week to rectify your error(s) and resubmit the assignment. You will be subject to a 25% penalty on the grade obtained from the resulting resubmission of the assignment.

How to test your program natively (run your own test cases):
You can use main.cpp provided in the starter.zip folder to instantiate LRUCache objects. You can then test the functionality of your methods by calling these methods and checking whether what they return is what you expect.
To compile your own tests in main.cpp, **you can use the makefile provided within starter.zip to compile your program.** In your terminal:

- Run **make** on the terminal (this will create an executable named main)
- Next, run **./main** on your terminal to run your native tests

How to use testInfrastructure:
To check your score using testInfrastructure, you can follow either of the two options mentioned below:

- **Option 1:** Move your lru.h file into the testInfrastructure directory. After this, run **python3 test.py**
- **Option 2:** Move your lru.h file into the testInfrastructure directory. **Ensure that the makefile in this directory is the one given to you originally in the testInfrastructure.zip folder and not in the starter.zip folder.** In your terminal:
    - Run **make** in order to compile the program (which will compile lru_test.cpp and create an executable named lru_test).
    - Next, run **./lru_test** on your terminal to actually run the tests and output your score.