Milestone 3 – Cache Manager, Due March 19, 2025

The following files will be provided to you, for completion of your milestone:

- cache_manager.h // header file containing cache manager class
- dll_node.h // header file containing doubly linked list structure
- doubly_linked_list.h // header file containing doubly linked list class
- hash_node.h // header file containing hash node structure
- hash_table.h // header file containing hash table class
- json.hpp // header file for processing json files
- dll_node.cpp // dll node file constructor
- hash_node.cpp // hash node file constructor
- testCase1OutputFile.txt // generated output file format (partial results)
- milestone3.json // json file containing test cases and its transactions
- milestone3_config.json // json configuration (properties) file
- milestone3.cpp /* cpp file containing main, which does the following:
    - Reads configuration file (json format) to:
        - retrieve inputFile (test case file (json format)
        - retrieve outputFile (text file containing generated output)
        - retrieve errorLogFile (text file containing error messages)
    - process inputFile test cases
    - write output to outputFile */

Write a FIFO list and basic hash table implementation, which uses the files listed above, and includes the following in separate cpp files:

- cache_manager.cpp – implementation file that contains the following methods:

    1. getTable - return the hash table
    2. getList - return the FIFO list
    3. isEmpty - Check if the HashTable is empty
    4. getSize - return the number of items in the CacheManager
    5. add - adds a new node to the CacheManager and to doublyLinkedList
    6. remove – remove node with key value
    7. clear - remove all entries from the CacheManager
    8. getItem - retrieve item from the CacheManager, and moves node to the head of doublyLinkedList
    9. getMaxCacheSize - retrieve max size of cache from the CacheManager
    10. contains - check if a node with key exists in the table, and moves node to the head of doublyLinkedList, if true
    11. printCache - print out the cache information

- doubly_linked_list.cpp – implementation file that contains the following methods:

    12. getSize - Get the size of the list
    13. isEmpty - Check if the list is empty
    14. insertAtHead - Adds a new node at the beginning of the list
    15. insertAtTail - Adds a new node at the end of the list
    16. remove - Searches for a node with a specific value and deletes it from the list
    17. removeHeaderNode – Removes header node
    18. removeTailNode – Removes tail node
    19. moveNodeToHead – Moves a specific node to the front
    20. moveNodeToTail – Moves a specific node to the end
    21. clear - Clear the list (delete all nodes)
    22. printList - print the doubly linked list from head to tail to console and output file
    23. reversePrintList- print the doubly linked list from tail to head to console and output file

- hash_table.cpp – implementation file that contains the following methods:

    24. getSize - return the size of the hash table
    25. calculateHashCode – perform hashing function
    26. isEmpty - Check if the HashTable is empty
    27. getNumberOfItems - return number of items in the hash table
    28. add - adds a new node to the hash table
    29. remove – remove node with key value
    30. clear - remove all entries from the table
    31. getItem - returns pointer to the HashNode
    32. contains - check if a node with key exists in the table
    33. printTable - print out contents of hash table to console and output file
    34. getTable - return the hash table

The total number of points for this milestone is 88, which will be based upon the following:

- Each submitted/modified file must have student's name (-10% of total milestone points if missing)
- Each submitted/modified file must include description of changes made to a program, and its change date (3)
- Program compiles with all of the provided files (1)
- The following methods run without errors:

    1. getTable - return the hash table (1)
    2. getList - return the FIFO list (1)

3. isEmpty - Check if the HashTable is empty (1)
4. getSize - return the number of items in the CacheManager (1)
5. add - adds a new node to the CacheManager and to doublyLinkedList (1)
6. remove – remove node with key value (1)
7. clear - remove all entries from the CacheManager (1)
8. getItem - retrieve item from the CacheManager, and moves node to the head of doublyLinkedList (1)
9. getMaxCacheSize - retrieve max size of cache from the CacheManager (1)
10. contains - check if a node with key exists in the table, and moves node to the head of doublyLinkedList, if true (1)
11. printCache - print out the cache information (1)
12. getSize - Get the size of the list (1)
13. isEmpty - Check if the list is empty (1)
14. insertAtHead - Adds a new node at the beginning of the list (1)
15. insertAtTail - Adds a new node at the end of the list (1)
16. remove - Searches for a node with a specific value and deletes it from the list (1)
17. removeHeaderNode – Removes header node (1)
18. removeTailNode – Removes tail node (1)
19. moveNodeToHead – Moves a specific node to the front (1)
20. moveNodeToTail – Moves a specific node to the end (1)
21. clear - Clear the list (delete all nodes) (1)
22. printList - print the doubly linked list from head to tail to console and output file (1)
23. reversePrintList- print the doubly linked list from tail to head to console and output file (1)
24. getSize - return the size of the hash table (1)
25. calculateHashCode – perform hashing function (1)
26. isEmpty - Check if the HashTable is empty (1)
27. getNumberOfItems - return number of items in the hash table (1)
28. add - adds a new node to the hash table (1)
29. remove – remove node with key value (1)
30. clear - remove all entries from the table (1)
31. getItem - returns pointer to the HashNode (1)
32. contains - check if a node with key exists in the table (1)
33. printTable - print out contents of hash table to console and output file (1)
34. getTable - return the hash table (1)

- The following test cases are processed, and produce expected output (10 per test case; 50 total)
- Extra Credit – use industry standard test program and/or extract test cases, in separate json test file

Please accept this GitHub Assignment: https://classroom.github.com/a/nAmE70Q0