Milestone 5 – Cache Manager + Binary Search Tree, Due May 7, 2025

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

- binary_search_tree.h // header file containing binary search tree class structure
- cache_manager.h // header file containing cache_manager class
- dll_node.h // header file containing dll_node 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
- tree_node.h // header file containing tree node structure
- dll_node.cpp // node constructor
- hash_node.cpp // hash node constructor
- milestone5.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 */
- tree_node.cpp // tree node constructor
- generatedOutputFile.txt // output format for processing test cases (partial)
- milestone5.json // json file containing test cases and its transactions
- milestone5_config.json // json configuration (properties) file

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

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

    1. addToTree - Add a key to the tree, and point to FIFO node
    2. removeNode - Remove a specific node from the BST
    3. getHeightOfTree - Get the height of the tree
    4. getNumberOfTreeNodes - Get the total number of nodes in the tree
    5. contains - Check if a key is in the BST
    6. getRoot - Getter for the root node of the tree
    7. isEmpty - Check if a key is in the BST
    8. clear – Removes tree

9. printNodeFromTree – Prints the data of a specific node
10. printInOrder- print the binary search tree in an in-order traversal
11. printReverseOrder – Performs a reverse traversal of the tree and prints the nodes
12. printPreOrder– print the binary search tree in a Pre-order traversal
13. printPostOrder - print the binary search tree in a Post-order traversal
14. printDepthFirst - print the binary search tree in a depth-first-order traversal
15. printBreadthFirst- print the binary search tree in a breadth-first-order traversal
16. printRange - traverse and print out the cache information given a low and high value
17. deleteTree - Deletes the tree starting from the specified node
18. getHeight - Helper function to calculate the height of a node
19. printInOrderHelper - Helper function for recursive in-order traversal
20. printReverseOrderHelper - Helper function for recursive reverse in-order traversal
21. printPreOrderHelper - Helper function for recursive pre-order traversal
22. printPostOrderHelper - Helper function for recursive post-order traversal
23. printRangeHelper - Helper function to print out the cache information given a low and high value

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

24. getTable - Return the hash table
25. getList - return the FIFO list
26. getBst - return the BST
27. getSize - return the number of items in the CacheManager
28. isEmpty - Check if the CacheManager is empty
29. add - Adds a new node to the CacheManager
30. remove – Remove node with key value
31. clear - Remove all entries from the CacheManager
32. getItem - retrieve item from the CacheManager
33. getMaxCacheSize – retrieve max size of cache from the CacheManager
34. contains - determine if a key value is in the cache
35. printCache - print out the cache information
36. printRange - traverse and print out the cache information given a low and high value
37. sort – print out the cache information in sorted order

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

38. getSize - return number of entries in the list

39. isEmpty - Check if the list is empty
40. insertAtHead - Adds a new node at the beginning of the list
41. insertAtTail - Adds a new node at the end of the list
42. remove - remove a node with a specific value from the list
43. removeHeaderNode – Removes header node
44. removeTailNode – Removes tail node
45. moveNodeToHead – Moves a specific node to the front
46. moveNodeToTail – Moves a specific node to the end
47. clear - Clear the list (delete all nodes)
48. printList - print the doubly linked list list from head to tail to console and output file
49. reversePrintList - print the doubly linked list list from tail to head to console and output file

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

50. getTable - Return the hash table
51. getSize - Return the size of the hash table
52. calculateHashCode – Perform hashing function
53. isEmpty - Check if the HashTable is empty
54. getNumberOfItems - Return number of items in the hash table
55. add - Adds a new node to the hash table
56. remove – Remove node with key value
57. clear - Remove all entries from the table
58. getItem - Returns pointer to the HashNode
59. contains - Check if a node with key exists in the table
60. printTable- print out the contents of hash table

The total number of points for this milestone is 145, 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 (4)
- Each method must have a method header containing the name of the method, description what the method does, parameters, and the return value (30)
- Program compiles with all of the provided files (1)
- The following methods run without errors:

1. addToTree - Add a key to the tree, and point to FIFO node (1)
2. removeNode - Remove a specific node from the BST (1)
3. getHeightOfTree - Get the height of the tree (1)
4. getNumberOfTreeNodes - Get the total number of nodes in the tree (1)
5. contains - Check if a key is in the BST (1)

6. getRoot - Getter for the root node of the tree (1)
7. isEmpty - Check if a key is in the BST (1)
8. clear – Removes tree (1)
9. printNodeFromTree – Prints the data of a specific node (1)
10. printInOrder- print the binary search tree in an in-order traversal (1)
11. printReverseOrder – Performs a reverse traversal of the tree and prints the nodes (1)
12. printPreOrder– print the binary search tree in a Pre-order traversal (1)
13. printPostOrder - print the binary search tree in a Post-order traversal (1)
14. printDepthFirst - print the binary search tree in a depth-first-order traversal (1)
15. printBreadthFirst- print the binary search tree in a breadth-first-order traversal (1)
16. printRange - traverse and print out the cache information given a low and high value (1)
17. deleteTree - Deletes the tree starting from the specified node (1)
18. getHeight - Helper function to calculate the height of a node (1)
19. printInOrderHelper - Helper function for recursive in-order traversal (1)
20. printReverseOrderHelper - Helper function for recursive reverse in-order traversal (1)
21. printPreOrderHelper - Helper function for recursive pre-order traversal (1)
22. printPostOrderHelper - Helper function for recursive post-order traversal (1)
23. printRangeHelper - Helper function to print out the cache information given a low and high value (1)
24. getTable - Return the hash table (1)
25. getList - return the FIFO list (1)
26. getBst - return the BST (1)
27. getSize - return the number of items in the CacheManager (1)
28. isEmpty - Check if the CacheManager is empty (1)
29. add - Adds a new node to the CacheManager (1)
30. remove – Remove node with key value (1)
31. clear - Remove all entries from the CacheManager (1)
32. getItem - retrieve item from the CacheManager (1)
33. getMaxCacheSize – retrieve max size of cache from the CacheManager (1)
34. contains - determine if a key value is in the cache (1)
35. printCache - print out the cache information (1)
36. printRange - traverse and print out the cache information given a low and high value (1)
37. sort – print out the cache information in sorted order (1)
38. getSize - return number of entries in the list (1)
39. isEmpty - Check if the list is empty (1)
40. insertAtHead - Adds a new node at the beginning of the list (1)
41. insertAtTail - Adds a new node at the end of the list (1)
42. remove - remove a node with a specific value from the list (1)

43. removeHeaderNode – Removes header node (1)
44. removeTailNode – Removes tail node (1)
45. moveNodeToHead – Moves a specific node to the front (1)
46. moveNodeToTail – Moves a specific node to the end (1)
47. clear - Clear the list (delete all nodes) (1)
48. printList - print the doubly linked list list from head to tail to console and output file (1)
49. reversePrintList - print the doubly linked list list from tail to head to console and output file (1)
50. getTable - Return the hash table (1)
51. getSize - Return the size of the hash table (1)
52. calculateHashCode – Perform hashing function (1)
53. isEmpty - Check if the HashTable is empty (1)
54. getNumberOfItems - Return number of items in the hash table (1)
55. add - Adds a new node to the hash table (1)
56. remove – Remove node with key value (1)
57. clear - Remove all entries from the table (1)
58. getItem - Returns pointer to the HashNode (1)
59. contains - Check if a node with key exists in the table (1)
60. printTable- print out the contents of 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/k5_YuyqD