Programming Assignment #3

Lecturer: Prof. Seung-Hwan Baek
Teaching Assistants: Suhyun Shin, Eun-Sue Choi, Juhyung Choi, YuJin Jeon

***** PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT *****

Due date: 11:59 PM May 9, 2024

Evaluation policy:

- Late submission penalty
  - 11:59 PM May 9 ~ 11:59 PM May 10
    - Late submission penalty (30%) will be applied to the total score
  - After 11:59 PM May 10
    - 100% penalty is applied for that submission

- Your code will be automatically tested using an evaluation program
  - Each problem has the maximum score
  - A score will be assigned based on the behavior of the program

- We won't accept any submission via email - it will be ignored

- Do not modify auxiliary files.
  - Such as: utils.h/cpp, evaluate.cpp

- Compile your file(s) using C++ compiler on 'Replit' or 'CLion' and check your program before the submission.

- All characters in submit.txt should be in uppercase letters except task 4, e.g., 'TURE','FALSE',(except for [Task 1], [Task 2], … [Task 6])

- Please do not use the containers in C++ standard template library (STL)
  - Such as:
    - #include <queue>
    - #include <vector>
    - #include <stack>

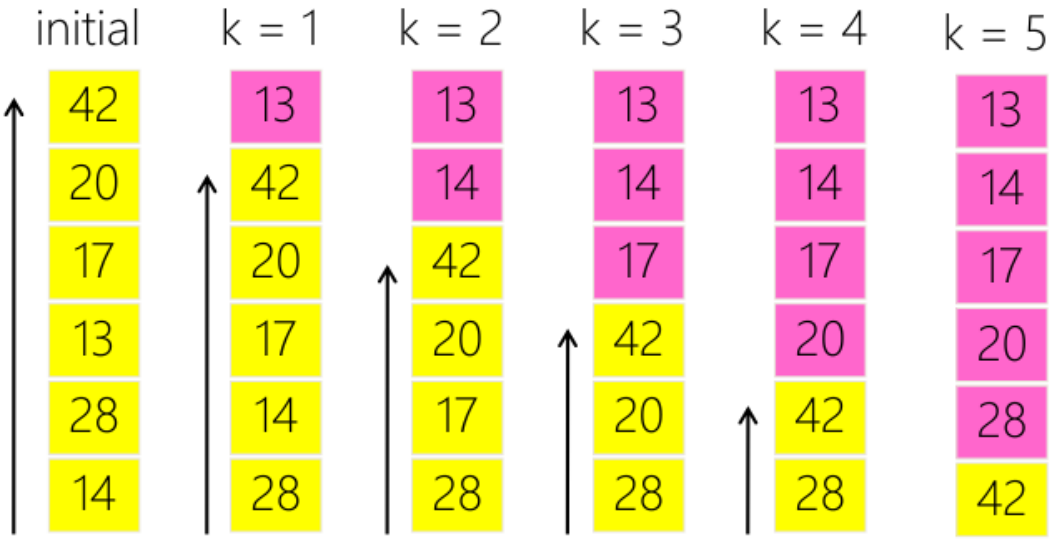
- Any submission using the containers in STL will be disregarded

File(s) you need to submit:

- Files you need to submit. (Do not change the filename.)
  - pa3.cpp
  - sort.cpp and sort.h
  - tree.cpp and tree.h
  - bst.cpp and bst.h
  - avl.cpp and avl.h
  - open_hash_function.cpp and open_hash_function.h
  - open_hash_table.cpp and open_hash_table.h
  - closed_hash_function.cpp and closed_hash_function.h
  - closed_hash_table.cpp and closed_hash_table.h

Any questions? Please use PLMS - Q&A board.

1. Bubble Sort (2 pts)



a. Implement a function that sorts a given array using the **Bubble Sort** algorithm. (k: iteration, A[ ]: array) On path k, the k-th lowest key rises to k-th position. Iteratively swap the adjacent items if the below item has a lower key value. If there were no swap in the current iteration, the array is sorted.
You can modify sort.cpp and sort.h files for this problem.

b. Input & Output
Input: A sequence of commands
   - ('insertion',integer): insert integer into the array (there will be no duplicated integers).
   - ('bubbleSort',NULL): sort the array using the Bubble Sort algorithm.

Output:
   - Print every value in the array **whenever the k-th lowest key rises to k-th position**, including the initial state, separating the values with white space. Please use a built-in function to print the array.
   - We won't test array size over 20 or array size of 0.

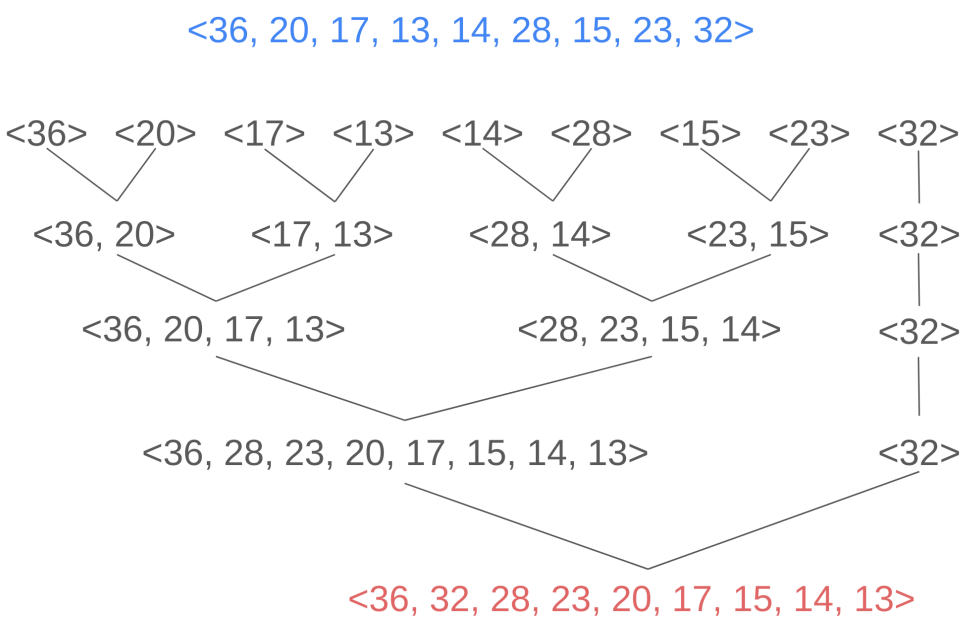
c. Example Input & Output

Input	Output
"[('insertion',42), ('insertion',20), ('insertion',17), ('insertion',13), ('insertion',28), ('insertion',14), ('bubbleSort',NULL)]"	42 20 17 13 28 14 13 42 20 17 14 28 13 14 42 20 17 28 13 14 17 42 20 28 13 14 17 20 42 28 13 14 17 20 28 42
"[('insertion',17), ('insertion',20), ('insertion',2), ('insertion',21), ('insertion',4), ('bubbleSort',NULL)]"	17 20 2 21 4 2 17 20 4 21 2 4 17 20 21
"[('insertion',7), ('insertion',6),('insertion',5), ('insertion',4),('insertion',3), ('insertion',2), ('insertion',1), ('bubbleSort',NULL)]"	7 6 5 4 3 2 1 1 7 6 5 4 3 2 1 2 7 6 5 4 3 1 2 3 7 6 5 4 1 2 3 4 7 6 5 1 2 3 4 5 7 6 1 2 3 4 5 6 7

d. Example execution

```
>> ./pa3.exe 1 "[('insertion',42), ('insertion',20),
('insertion',17), ('insertion',13), ('insertion',28),
('insertion',14), ('bubbleSort',NULL)]"
[Task 1]
42 20 17 13 28 14
13 42 20 17 14 28
13 14 42 20 17 28
13 14 17 42 20 28
13 14 17 20 42 28
13 14 17 20 28 42
```

2. Non-recursive Merge Sort (2 pts)



a. Implement a function that sorts a given array using the **Merge Sort** algorithm in descending order using **non-recursive** merge sort. Start with the sorted segments of size 1 and do pairwise merging of these sorted segments as in the upward pass.
You can modify sort.cpp and sort.h files for this problem.

b. Input & Output
Input : A sequence of commands
- ('insertion',integer): insert integer into the array.
- ('mergeSort',NULL): sort the array using the Merge Sort algorithm.

Output
- Starting from the initial state of the array, print all values of the array at **each iteration step after performing the necessary sorting**. Print the array values at the initial state and at each iteration step where sorting is performed.
- Separate the values in the array with a space for each iteration step.
- You don't need to consider exceptional cases such as overflow or an empty array. We will not test such cases.

c. Example Input & Output

Input	Output
"[('insertion',56),('insertion',42), ('insertion',20),('insertion',17), ('insertion',13),('insertion',28), ('insertion',14),('mergeSort',NULL)]"	56 42 20 17 13 28 14 56 42 20 17 28 13 14 56 42 20 17 28 14 13 56 42 28 20 17 14 13
"[('insertion',36), ('insertion',20), ('insertion',17), ('insertion',13), ('insertion',14), ('insertion',28), ('insertion',15), ('insertion',23), ('insertion',32),('mergeSort',NULL)]"	36 20 17 13 14 28 15 23 32 36 20 17 13 28 14 23 15 32 36 20 17 13 28 23 15 14 32 36 28 23 20 17 15 14 13 32 36 32 28 23 20 17 15 14 13
"[('insertion',1), ('insertion',2),('insertion',3), ('insertion',4),('insertion',5), ('insertion',6), ('insertion',7), ('mergeSort',NULL)]"	1 2 3 4 5 6 7 2 1 4 3 6 5 7 4 3 2 1 7 6 5 7 6 5 4 3 2 1

d. Example execution

```
>> ./pa3.exe 2 "[('insertion',56),('insertion',42),
('insertion',20),('insertion',17),
('insertion',13),('insertion',28),
('insertion',14),('mergeSort',NULL)]"
[Task 2]
56 42 20 17 13 28 14
56 42 20 17 28 13 14
56 42 20 17 28 14 13
56 42 28 20 17 14 13
```

3. BST Insertion / Deletion / Summation (3pts)

a. Implement functions that **inserts** and **deletes** an element into a binary search tree (BST). Also you should find the **sum** of nodes that is larger than specific node value, and **print** the pre-order and in-order of the tree. You can modify `bst.cpp` and `bst.h` files for this problem.
   - Input of nodes are integers from 1 to 99

b. Input & output of `BinarySearchTree::insertion`
   Input:
   - (`'insertion',integer`): Key of the element to be inserted. The key has a positive integer value.

   Output:
   - Return the -1 if the key already exists in the tree, `0` otherwise. (If the key already exists, do not insert the element)

c. Input & output of `BinarySearchTree::deletion`
   Input:
   - (`'deletion',integer`): Key of the element to be deleted.

   Output:
   - Return -1 if the key does not exist in the tree, `0` otherwise. If the key does not exist, do not delete any element.

   Note that replace the smallest key in right subtree when delete the node with degree 2

d. Input & output of `BinarySearchTree::sum`
   Input:
   - (`'sum',integer`): Sum of the nodes greater than the `integer`.

   Output:
   - Return the sum of nodes that is larger than the key value.

e. task_3 prints
   i. the return for each insertion/deletion for command (‘insertion’, integer)/ (‘deletion’, integer) and
   ii. the return summation of nodes that is larger than a specific node value for command (‘sum’, integer) and

iii. the results of preorder and inorder traversal of the constructed tree for command (‘print’, NULL).
(If empty tree, don’t return preorder or inorder traveral results

f. Example Input & Output

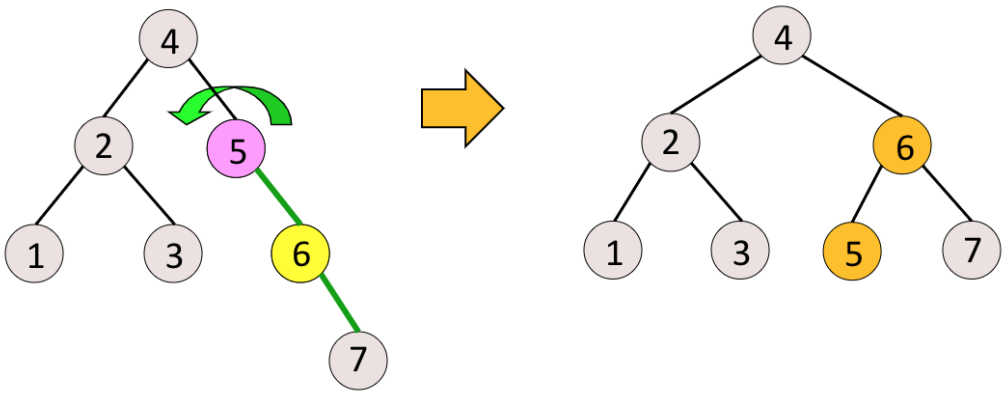
Input	Output
[('insertion',4), ('insertion',6), ('insertion',6), ('insertion',7), ('deletion',7), ('print', NULL)]	0 0 -1 0 0 4 6 4 6
[('insertion',4), ('insertion',2), ('sum', 1), ('insertion',10), ('insertion',9), ('insertion',15), ('insertion',1), ('deletion',1), ('deletion',4), ('deletion',10), ('sum', 5), ('print', NULL)]	0 0 6 0 0 0 0 0 0 24 9 2 15 2 9 15
[('deletion', 3),('insertion', 10),('deletion', 10), ('sum',2), ('print',NULL)]	-1 0 0 0

g. Example execution

```
>> ./pa3.exe 3 "[('insertion',4), ('insertion',6),
('insertion',6), ('insertion',7), ('deletion',7), ('sum', 2)]"
[Task 3]
0
0
-1
0
0
10
```

4. Duplicate Letter Counting Problem (4 pts)

- Insert 7
  - Violation at node 2
  - Left (RR) rotation



Example of left rotation to resolve RR imbalance

a. Implement a class ‘AVLTree’ to efficiently count and report the frequency of each alphabet letter in a string using AVL tree operations. This involves handling imbalances that might occur during insertions and deletions, as well as ensuring that alphabet letters are counted accurately. As the string is received, insert each letter sequentially from the beginning to the end into the AVL tree.

b. AVL Tree Implementations
   - Modify or implement functions in the ‘AVLTree’ class that insert and delete elements (letters in this case).
   - Implement functions in the ‘AVLTree’ class that print the all node information in pre-order, in-order, and reverse-in-order manner.
   - Ensure the AVL tree can resolve imbalances (LL, LR, RL, RR) that might occur after modifications.
   - You can modify ‘avl.cpp’ and ‘avl.h’ files, and if needed, add public members to the ‘Node’ class implemented in ‘tree.h’.

c. Details of the process
- Sequentially insert each letter from the string. If a node for that letter already exists, delete the existing node, increment the count, and re-insert the node.
- During the deletion process, if a node has both children, use the replacement method where you find the minimum in the right subtree to replace the node
- If counts are equal, prioritize based on the lexicographical order of the letters. For example, (c, 1) < (a, 4) < (b, 4).

d. Input & Output:
- Input: A string consisting of at least one alphabetical letter. You don't need to consider any other input cases and uppercase and lowercase of letters are same.
- Output: (1) Frequency counts of letters, displayed from the most to the least frequent. If multiple letters have the same frequency, they should be displayed in lexicographical order.
- Output: (2) The results of the preorder and inorder traversal of the constructed AVL tree after all operations.

e. `task_4` prints
   i. The results of reverse-inorder, inorder, preorder traversal of the constructed tree.
   ii. The format of node for printing is "(letter, count)"

f. Example Input & Output

Input	Output
"aAbbbcccc"	Frequencies of alphabetic letters in the given string (c, 4) (b, 3) (a, 2) Inorder (a, 2) (b, 3) (c, 4) Preorder (b, 3) (a, 2) (c, 4)
"a"	Frequencies of alphabetic letters in

	the given string (a, 1) Inorder (a, 1) Preorder (a, 1)
"aabbbccccddDDd"	Frequencies of alphabetic letters in the given string (d, 5) (c, 4) (b, 3) (a, 2) Inorder (a, 2) (b, 3) (c, 4) (d, 5) Preorder (b, 3) (a, 2) (c, 4) (d, 5)

g. Example execution

```
>> ./pa3.exe 4 "aabbb"
[Task 4]
Frequencies of alphabetic letters in the given string
(b, 3) (a, 2)
Inorder
(a, 2) (b, 3)
Preorder
(a, 2) (b, 3)
```

5. Open hash table (1 pts)

a. Suppose there is an **open hash table** with **digit-folding-method**. This hash table uses a singly linked list as a collision handling method. This hash table is used with integer keys and hashing into a table of size M. Every component of the key is folded with a size of 1(digit) and calculates their sum as described in our Lecture Note.

M which is the size of the hash table, insertions, and deletions of integers are given as instructions. First, for each insertion, record if a collision occurred. A collision occurs when a key is inserted into a non-empty slot. After all operations are complete, print the total number of collisions that occurred and the length of the longest chain in the hash table. You don't need to consider a deletion of an unseen key and multiple insertions of the same key.

You can modify `open_hash_function.cpp`, `open_hash_table.cpp`, `open_hash_function.h` and `open_hash_table.h` files for this problem.

b. Input & Output

Input: A sequence of commands

- (`'M',integer`): the size of a hash table.
- (The first command should always be 'M')
- (`'insertion',integer`): insert `integer` into the hash table.
- (`'deletion',integer`): delete `integer` from the hash table.

Output: For each slot of the hash table, print out

- the number of collisions occurred during execution.
- the length of the longest chain in the final table.

c. Example Input & Output

Input	Output
`[('M',4), ('insertion',32615)]`	`0 1`
`[('M',4), ('insertion',32615), ('insertion',315), ('insertion',6468), ('insertion',94833)]`	`1 2`
`[('M',4), ('insertion',32615),`	`1 1`

('insertion',315), ('insertion',6468), ('insertion',94833), ('deletion',32615), ('deletion',6468)]	

d. Example execution

```
>> ./pa3.exe 5 "[('M',4), ('insertion',32615)]"
[Task 5]
0 1
```

6. Closed hash table (3 pts)

a. Implement insertion of a **closed hash table** with **rehashing** implementation. This hash table is used with integer keys and hashing into a table of size M. This hash table uses **pseudo-random probing** as a collision handling method. The index of the key k after i-th collision, $h_i(k)$, is:

$$h_i(k) = (h(k) + d_i)\ mod\ M$$

where $h(k)$ is the **digit-folding method** hash function. It works same with the hash function in task 5.

$d_1, d_2, \dots, d_{M-1}$ is a pseudo-random permutation of integers 1, … , M-1 and it is generated by **shift-register sequence**. Please refer to slide "Example: Shift-Register Sequence" in "Hashing" Lecture Note for detail.

Given M, k, d_1 for the shift-register sequence and inserted integers, print the result hash table. If the hash table is full or the collision cannot be resolved, stop the insertion, and print results of the insertions made up to that point, then print FAIL. Refer to instruction b & c for more detail.

You don't need to consider a deletion of a key, multiple insertions of the same key. You can modify `closed_hash_function.cpp`, `closed_hash_table.cpp`, `closed_hash_function.h` and `closed_hash_table.h` files for this problem.

b. Input & Output

Input: A sequence of commands

- `('M',integer)`: the size of a hash table.
  (The first command is always `'M'`)

- `('k',integer)`: a constant used for shift-register sequence.
  (The second command is always `'k'`)
- `('d',integer)`: the first probing offset for the shift-register sequence .
  (The third command is always `'d'`)
- `('insertion',integer)`: insert `integer` into the hash table.

Output:

For each slot of the hash table, print out

- the value, if the state of the slot is occupied.
- `'EMPTY'` if the state of the slot is empty.

`'FAIL'` if the hash table is full or the collision cannot be resolved.

c. Example Input & Output

Input	Output

`[('M',4), ('k',3), ('d',2),`
`('insertion',32615), ('insertion',315),`
`('insertion',6468), ('insertion',94833)]`
	`0: 6468`
	`1: 32615`
	`2: 94833`
	`3: 315`

`[('M',4), ('k',3), ('d',2), ('insert',32615),`
`('insertion',315), ('insertion',6468),`
`('insertion',94833), ('insertion',22)]`
	`0: 6468`
	`1: 32615`
	`2: 94833`
	`3: 315`
	`FAIL`

`[('M',5), ('k',3), ('d',2), ('insertion',2),`
`('insertion',7), ('insertion',11),`
`('insertion',20), ('insertion',18)]`
	`0: EMPTY`
	`1: 11`
	`2: 2`
	`3: EMPTY`
	`4: 7`
	`FAIL`

d. Example execution

```
>> ./pa3.exe 6 "[('M',4), ('k',3), ('d',2),
('insertion',32615), ('insertion',315), ('insertion',6468),
('insertion',94833)]"
[Task 6]
0: 6468
1: 32615
2: 94833
3: 315
```