Programming Assignment #2

Lecturer: Prof. Seung-Hwan Baek
Teaching Assistants: Su-Hyun Shin, Eun-Sue Choi, Ju-hyung Choi, Yu-Jin Jeon

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

Due date: 11:59PM April 4, 2024

Evaluation policy:

- Late submission penalty
  - 11:59PM April 4 ~ 11:59PM April 5
    - Late submission penalty (30%) will be applied to the total score
  - After 11:59PM April 5
    - 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 'Replit' or 'CLion' and check your program before the submission.
- All characters in submit.txt should be in uppercase letters, e.g., 'TURE','FALSE',(except for [Task1], [Task2])
- 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:

- pa2.cpp, tree.cpp, tree.h, heap.cpp, heap.h (Do not change the filename!)

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

0. Basic Instruction

a. Please refer to the attached file named “DataStructure_PA_instructions.pdf”.

1. Quiz (2 pts)

1.1. For such binary’s in-order and post-order traversal, which has the correct answer for preorder traversal?

Inorder : 20, 30, 35, 40, 45, 50, 55, 60, 70
Postorder : 20, 35, 30, 45, 40, 55, 70, 60, 50

(1) Preorder : 50, 40, 60, 30, 45, 55, 70, 20, 35
(2) Preorder : 50, 40, 30, 20, 35, 45, 60, 55, 70.
(3) Preorder : 50, 40, 20, 30, 35, 45, 60, 55, 70.
(4) None of the aboves

1.2. What is the time complexity of **insertion** into Min-heap?

(1) O(1)
(2) O(log n)
(3) O(n)
(4) O(2^n)

- Example execution
  - If you choose “(1) Preorder and Postorder traversal of T” for 1-1., print your answer as shown below

```
>> ./pa2.exe 1 1
[Task 1]
1
```

- If you choose “(1) O(1)” for 1-2., print your answer as shown below

```
>> ./pa2.exe 1 2
[Task 1]
1
```

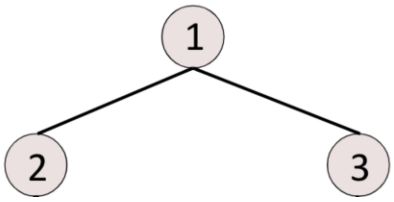
pre-2. Construct Binary Tree

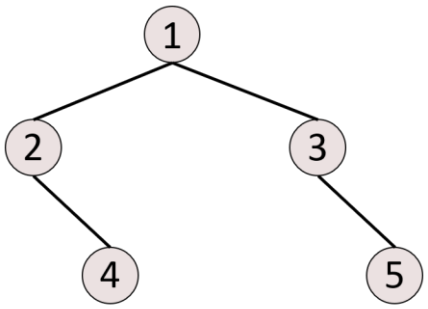
Note: pre-2 is not a problem that will be evaluated, but this is a short pre-requisite to solve problems 2,3, and 4.
Don't worry. We are providing utility functions to help you.

a. For problems 2, 3, and 4, you would need to implement member functions of `BinaryTree` class. To construct a `BinaryTree` class instance from an input, we use the string with bracket representation as input. The recursive definition of the bracket representation is as follows.

`Tree = Root(LeftChild)(RightChild).`

Below are some examples.
The left tree is represented as `1(2)(3)`, and the right tree is `1(2()(4))(3()(5))`





b. To implement "a", we provide a function to construct `BinaryTree` class from the bracket representation, which is `BinaryTree::buildFromString` function. It creates a pointer-based `BinaryTree` class instance from the given string. It would be helpful to read the implementation details of `BinaryTree::buildFromString`

c. To sum up, you will need to use `BinaryTree` class for problems 2, 3 and 4. Please try to understand the code for `BinaryTree` class.

2. Parents and Children in a Binary Tree (3 pts)

a. Implement a function that takes a binary tree in the form of a string with bracket representation, and outputs each parent node along with the number of children it has.

b. Input & Output

Input:

- String with bracket representation of binary tree.
- It is assumed that the bracket representation input received is in the correct format.
- Duplicated node names are not considered in the input.

Output:

- String with parent node and number of children for each parent node.
- If the input is in the form of 'A(B(D)())(C)', the output should be 'A2B1D0C0', indicating the number of children each node has, following a preorder traversal of the binary tree.

c. Example input & output

Input	Output
"A(B(D)())(C)"	A2B1D0C0
"A(B)(C()(D))"	A2B0C1D0
"A(B(C()(D()(E)))(F))()"	A1B2C1D1E0F0

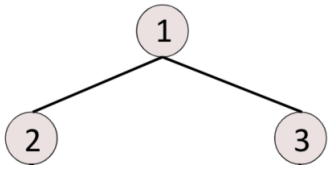
d. Example execution

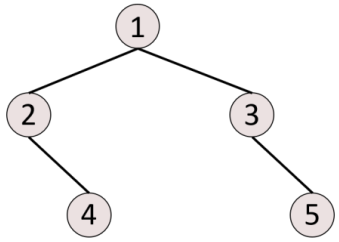
```
>> ./pa2.exe 2 "A(B)(C()(D))"
[Task 2]
A2B0C1D0
```

pre-3. Height, Level and Level Order

Note: pre-3 is not a problem that will be evaluated, but this is a short pre-requisite to solve problems 3.
Don't worry. We are providing information to help you.

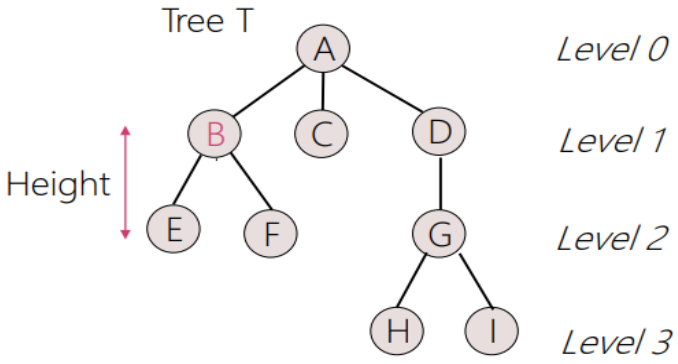
a. For problems 3, you would need to implement level order traverse. Level Order traversal of a binary tree involves visiting all the nodes of the tree level by level. Starting at the root, you visit all nodes at level 1, then all nodes at level 2, and so on.
b. Below are some examples.
Level order of left tree : 1 2 3, right tree : 1 2 3 4 5





1) Here is a simple step-by-step breakdown of how level order traversal works:
   c. Start at root node
   d. Move to the next level and visit all the nodes at that level from left to right
   e. Repeat step 2 for each subsequent level until you reach the last level of the tree

2) Here are the definition of height and level. We define root level as 0.



3. Height, Level and Level Order of Binary Tree (2 pts)

a. Implement `BinaryTree::findHeightOfNode`, `BinaryTree::findLevelOfNode` and `BinaryTree::levelOrder` function that can traverse a binary tree with given level order traverse mode and finds the level and height of a specific node.
You can additionally implement other functions to facilitate the traversal of a binary tree (HINT : `BinaryTree::_currentLevel` used in level order traverse mode.)
** We only consider 1 to 9 integer inputs for binary tree nodes.

b. Input & Output
Input:
- String with bracket representation and String representing specific node.

Output:
- A sequence of node values acquired from the ① level order traverse mode and the ② height and ③ level of a specific node. The value is separated with a white space for traverse mode, and height level with enter. Check the example below.
- You should print out -1 for each height and level for nodes that are not in the binary tree.

c. Example input & output

Input	Output

`"1(2)(3)" "2"`	`1 2 3`
	`0`
	`1`

`"1(2()(4))(3()(5))" "1"`	`1 2 3 4 5`
	`2`
	`0`

`"4(2(3)(1))(6(5))" "8"`	`4 2 6 3 1 5`
	`-1`
	`-1`

d. Example execution

```
>> ./pa2.exe 3 "4(2(3)(1))(6(5))" "6"
[Task 3]
4 2 6 3 1 5
1
1
```

4. Leaf count & Width of Binary Tree (3 pts)

a. Implement `BinaryTree::leafCount` & `BinaryTree::getWidth` function. `BinaryTree::leafCount` calculates the number of leaves of the given binary tree. `BinaryTree::getWidth` calculates width, the largest number of nodes at any single level. Maximum depth of the tree is limited to 100.

b. Input & Output

Input:
- String with bracket representation
- Specify what you want to check. Either “leaf” or “width”

Output:
- The number of the leaves of the given binary tree for “leaf” command.
- Width of the the given binary tree for “width” instruction.

c. Example

Input	Output
`"1(2)(3)" "leaf"`	2
`"1(2(4)(5))(3(6))" "leaf"`	3
`"1(2)(3)" "width"`	2
`"1(2(4(7))(5))(3(6))" "width"`	3

d. Example execution

```
>> ./pa2.exe 4 "1(2)(3)" "leaf"
[Task 4]
2
```

5. Priority Queue Implementation (2 pts)

Note: For solving problems 5 and 6, the similar utility functions provided in PA1 will be used to parse an input string. Therefore, you won't need to try implementing a string parser. Please read pa2.cpp, and find the lines where your code would be located.

a. You are required to implement several functions that operate on a Priority queue using a heap data structure. The Priority Queue must preserve its properties following each operation. The specific tasks are as follows:
   i. **Insert**: implement a function to insert a new element into the Priority Queue. Ensure that the queue maintains its heap property after each insertion. You will be tested with scenarios involving the insertion of fewer than 100 input values.
   ii. **Delete**: Implement a function to delete the node with the highest priority from the Priority Queue. The queue should continue to uphold its heap property after the node is deleted.

b. Input & Output
   Input: A sequence of commands
   - `('insert',integer)`: insert an `integer` into the current priority queue. If the priority queue is full with other nodes, print "ERROR" and exit the program. If a node with the specified priority already exists in the queue, print "ERROR" and do not proceed with the insertion. The integer value indicates both the input value and its own priority.
   - `('delete',NULL)`: delete node with the maximum priority value from current priority queue and rearrange the priority queue to maintain the its property. If the current priority queue is empty, print "Empty" and continue the program.
   - `('isEmpty', NULL)`: Reply whether the priority queue is currently empty. The answer is either "TRUE" or "FALSE"
   - `('getMax', NULL)`: Return a value with the maximum priority. If the priority queue is empty, print "EMPTY" and continue the program.

Output:

- Values in a priority queue in a node number order, in a string separated with the white space (automatically printed with built-in function)
- Do not consider the exceptional cases such as overflow, underflow and other input types. We will not use the test cases for those scenarios.

c. Example Input & Output

Input	Output
`[('insert',5),('insert',-3),('insert',2)]`	`5 -3 2`
`[('insert',4),('insert',-2),('insert',9), ('insert',10),('insert',15),('insert',-25)]`	`15 10 4 -2 9 -25`
`[('insert',28),( 'insert',9),( 'insert',27), ('insert',10),( 'insert',3),( 'insert',45), ('delete',NULL),( 'insert',22)]`	`28 10 27 9 3 22`
`[('getMax', NULL), ('insert', 3)]`	`EMPTY 3`
`[('isEmpty', NULL), ('insert', 3), ('isEmpty', NULL)]`	`TRUE FALSE 3`

d. Example execution

```
>> ./pa2.exe 5 "[('insert',5),('insert',-3),('insert',2)]"
[Task 5]
5 3 2
```

6. Advanced Priority Queue (3 pts)

a. You are required to implement an advanced functions that operate on a Priority queue using a heap data structure. We recommend that utilize the implementation in task 6. The Priority Queue must preserve its properties following each operation. The specific tasks are as follows:
   i. **changeMax**: Implement an operation named “changeMax” to modify the priority of a specified node to become the new maximum priority node within the Priority Queue. This operation should find a node with a given priority (e.g., 3) and change its priority to one more than the current maximum priority. If the max priority is 10, the node’s new priority would be 11. This change should promote the selected node to become the new max priority node. Ensure that after this operation, the priority queue maintains its heap property.

b. Input & Output
   Input: A sequence of commands, which is one of the following
   - `(‘changeMax’,integer)`: modify the priority of a node specified by the given integer so that it becomes the new maximum priority node. This is done by increasing the node’s priority to one more than the current maximum priority value in the Priority Queue. If there is no corresponding node, print “ERROR”.

   Output:
   - Values in a priority queue in a node number order, in a string separated with the white space (automatically printed with built-in function)
   - Do not consider the exceptional cases such as overflow, underflow and other input types. We will not use the test cases for those scenarios.

c. Example Input & Output

Input	Output
`[('insert',5),('insert',-3),('insert',2), ('changeMax', 2), ('getMax', NULL)]`	`6 6 -3 5`

`[('insert',5),('insert',-3),('insert',2), ('changeMax', 6), ('getMax', NULL)]`	`ERROR 5 -3 2`

d. Example execution

```
>> ./pa2.exe 6 "[('insert',5),('insert',-3),('insert',2),
('changeMax', 2), ('getMax', NULL)]"
[Task 6]
6 6 -3 5
```