Programming Assignment #2

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

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

Due date: 11:59PM October 18, 2023

Evaluation policy:

- Late submission penalty
  - 11:59PM October 18 ~ 11:59PM October 19
    - Late submission penalty (30%) will be applied to the total score
  - After 11:59PM October 19
    - 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
- 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!)

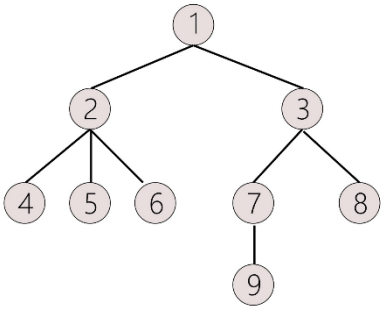
Any questions? Please use PLMS - Q&A board.

0. Basic instruction

a. Please refer to the attached file named "DataStructure_PA_instructions.pdf".

1. Quiz (2 pts)

1.1. For such tree, which has the correct pair for preorder traversal and post order traversal of the tree?



(1) Preorder : 1 2 4 5 6 3 7 9 8
Postorder : 4 5 6 2 9 7 8 3 1
(2) Preorder : 4 5 6 2 9 7 8 3 1
Postorder : 1 2 4 5 6 3 7 9 8
(3) Preorder : 1 2 4 5 6 3 7 9 8
Postorder : 4 5 6 2 1 9 7 3 8
(4) None of the aboves

1.2. What is the time complexity of **rearranging** min-heap into a max-heap?

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

- Example execution
  - If you choose “(1) Preorder 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
```

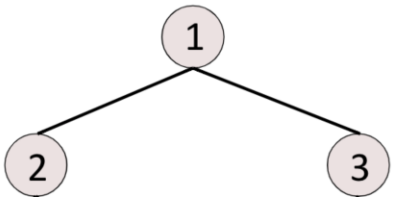
pre-2. Construct Binary Tree

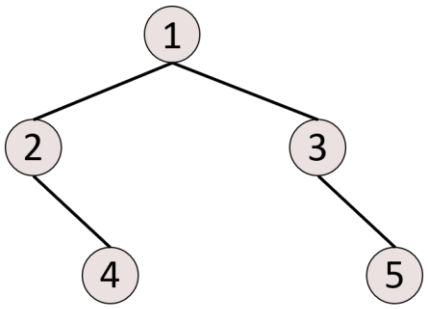
Note: pre-2 is not a problem that will be evaluated, but this is a short pre-requisite to solve problems 2,3, and 4.
Don't worry. We are providing utility functions to help you.

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. Binary Tree and General Tree (3 pts)

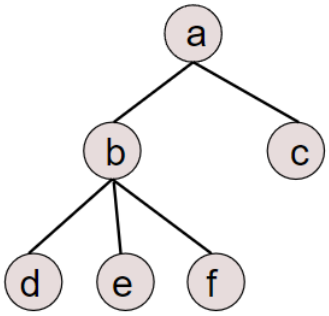
a. As mentioned in pre-2., The `BinaryTree` class takes input in forms of bracket representation. The recursive definition of the bracket representation is as follows:

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

Here, we will generalize this representation and then be able to represent not only a binary tree but also a general tree in follows form like:

`Tree = Root(1st child)(2nd Child)...(last Child).`

Below is an example.



The tree is represented as a(b(d)(e)(f))(c), and we do not consider an empty bracket due to the unknown number of children.

b. In the lecture, we learned about the Left-Child/Right-Sibling implementation of the general k-ary tree.

c. Implement a function that prints the bracket representation of a general tree which is given as input in forms of the Left-Child/Right-Sibling binary tree.

d. Input & Output
Input:
- String with bracket representation of binary tree.

Output:
- String with bracket representation of general k-ary tree.

e. Example input & output

Input	Output
"1(2(4()(5()(6)))(3))()"	1(2(4)(5)(6))(3)
"1(2(4)(3(5()(6()(7(8()(9))()))))()))()"	1(2(4))(3(5)(6)(7(8)(9)))

f. Example execution

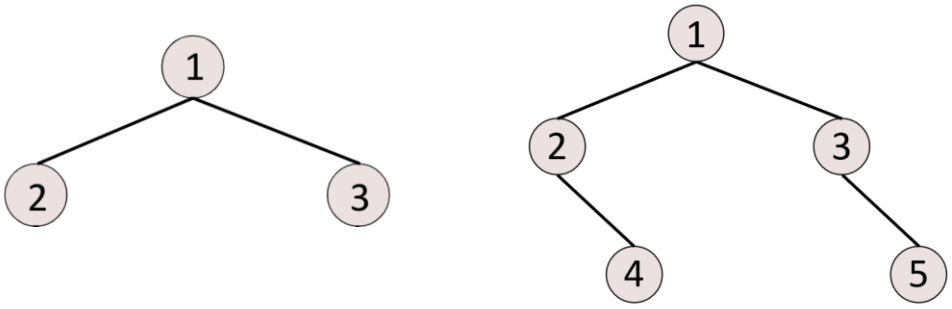
```
>> ./pa2.exe 2 "1(2(4()(5()(6)))(3))()"
[Task 2]
1(2(4)(5)(6))(3)
```

pre-3. 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.

d. For problems 3, you would need to implement four different traverse modes. 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.

e. Below are some examples.
Level order of left tree : 1 2 3, right tree : 1 2 3 4 5



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

3. Traverse Binary Tree (2 pts)

a. Implement `BinaryTree::preOrder, BinaryTree::postOrder` and `BinaryTree::inOrder, BinaryTree::levelOrder` function that can traverse a binary tree with given traverse mode.
You can additionally implement other functions to facilitate the traversal of a binary tree (HINT : `BinaryTree::_currentLevel` used in level order traverse mode.)

b. Input & Output
Input:
- String with bracket representation.
- String representing traverse mode. Either "`preorder`", "`postorder`", "`inorder`", or "levelorder"

Output:
- A sequence of node values acquired from the tree traversal. The value is separated with a white space

c. Example input & output

Input	Output
`"1(2)(3)" "preorder"`	`1 2 3`
`"1(2()(4))(3()(5))" "postorder"`	`4 2 5 3 1`
`"4(2(3)(1))(6(5))" "preorder"`	`4 2 3 1 6 5`
`"4(2(3)(1))(6(5))" "inorder"`	`3 2 1 4 5 6`
`"4(2(3)(1))(6(5))" "postorder"`	`3 1 2 5 6 4`
`"4(2(3)(1))(6(5))" "levelorder"`	`4 2 6 3 1 5`

d. Example execution

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

4. Rebuild Unique Binary Tree (3 pts)

Note: We recommend implementing the function using a recursive function

a. We know that we can identify a binary tree uniquely by two traversal sequences. Rebuild a binary from given two traversals. You can use orderIndex, inorder, preorder, postorder, inorderIdx initialized above.

b. Input & Output

Input:

- Two strings with integers divided by ',' covered by brackets. First string is inorder traversal and the second one is preorder or postorder traversal
- Type of second traversal. `"pre", "post".`

Output:

- "error" if given traversals cannot identify a binary tree.
- String of a binary tree of format introduced in pre-2.

c. Example

Input	Output
`"[2,1,3]" "[1,2,3]" "pre"`	`1(2)(3)`

`"[20,30,35,40,45,50,55,60,70]"`
`"[20,35,30,45,40,55,70,60,50]"`
`"post"`
	`50(40(30(20)(35))(45))(60(55)(70))`

`"[1,2,3]" "[3,1,2]" "post"`	`error`
`"[2,1,3]" "[1,2,2]" "pre"`	`error`

d. Example execution

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

5. Priority Queue Insertion (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. Implement a function that **inserts** a new element into a Priority Queue based on heap. Your Priority Queue should maintain its property even after the insertion. Each test case will involve inserting fewer than 100 input values.

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. The integer value indicates both the input value and its own priority.
- `('isEmpty', anything)`: Reply whether the priority queue is currently empty. The answer is either "True" or "False"
- `('getMax', anything)`: 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',4),('insert',-2),('insert',9), ('insert',10),('insert',15),('insert',-25), ('getMax', 0)]	15 15 10 4 -2 9 -25
[('getMax', 0), ('insert', 3)]	Empty 3
[('isEmpty', 0), ('insert', 3), ('isEmpty', 0)]	True False 3

d. Example execution

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

6. Priority Queue Deletion (3 pts)

a. Implement a function that **deletes** the node having maximum priority from the Priority Queue. Your priority queue should maintain its property even after the deletion.

b. Input & Output

Input: A sequence of commands, which is one of the following

- (‘insert’,integer): insert an integer into the current priority queue
- (‘removeMax’,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.

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
[('removeMax', 0), ('insert', 3)]	Empty 3
[('insert',5),('insert',-3),('insert',2), ('removeMax', 0)]	2 -3
[('insert',28),('insert',9),('insert',27), ('insert',10),('insert',3),('insert',45), ('removeMax',NULL),('insert',22)]	28 10 27 9 3 22
[('insert', 3), ('removeMax', 0), ('isEmpty', 0)]	True

d. Example execution

```
>> ./pa2.exe 6 "[('insert',5),( 'insert',-3),( 'insert',2),
('removeMax', 0)]"
[Task 6]
2 -3
```