**Programming Assignment #2**

Lecturer: Prof. Jaesik Park

Teaching Assistants: Suhyeon Jeong, Seunguk Do, Minjae Jeong, Sangjun Lee

|  |
| --- |
| *\*\*\*\* PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT \*\*\*\**  Due date: 11:59PM April 7, 2023  Evaluation policy:   * Late submission penalty   + 11:59PM April 7 ~ 11:59PM April 8     - Late submission penalty (30%) will be applied to the total score   + After 11:59PM April 8     - 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 * Please check if your program generates “submit.txt” same as “answer.txt”, after running ./pa2.exe 0 * 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   Any questions?   * Please use PLMS - Q&A board |

1. Basic instruction
   1. Please refer to the attached file named DataStructure\_PA2\_instructions.pdf

Quiz (2 pts)

* 1. Let T is a general k-ary tree, and T’ is a binary tree converted from T. Which of the following traversal visits the nodes in the same order as **the *postorder* traversal** of T?
  2. Preorder traversal of T’
  3. Inorder traversal of T’
  4. Postorder traversal of T’
  5. None of the aboves

1.2. **How many** following combinations of traversal sequence can **always** identify the binary tree uniquely?

|  |
| --- |
| 1. inorder 2. inorder & postorder 3. postorder & preorder & level-order 4. level-order & inorder |

1. 1
2. 2
3. 3
4. 4

* 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) 1” for 1-2., print your answer as shown below

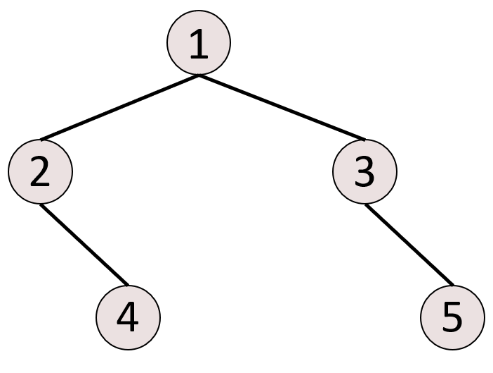
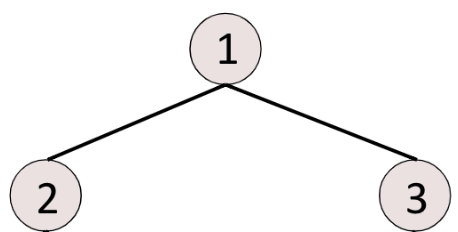
|  |
| --- |
| >> ./pa2.exe 1 2  [Task 1]  1 |

1. 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.*
2. 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))



1. 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.
2. 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.
3. Traverse Binary Tree (3 pts)
4. Implement BinaryTree::preOrder, BinaryTree::postOrder and BinaryTree:inOrder functions that traverse a subtree of a binary tree with a given traverse mode.
5. Input & Output

Input:

* A string with bracket representation for a binary tree. All node values ​​in the tree are unique.
* A string representing traverse mode. Either “preorder”, “postorder” or “inorder”.
* An integer that indicates a node in the binary tree. The node is the root of the subtree which will be traversed.

Output:

* A sequence of node values acquired from the tree traversal. The values are separated with white spaces.
* Put “error” in the answer string, if the input node is not in the binary tree

1. Example input & output

|  |  |
| --- | --- |
| Input | Output |
| “1(2)(3)” “preorder” 1 | 1 2 3 |
| “1(2)(3)” “preorder” 4 | error |
| “1(2()(4))(3()(5))” “postorder” 1 | 4 2 5 3 1 |
| “1(2()(4))(3()(5))” “postorder” 3 | 5 3 |
| “4(2(3)(1))(6(5))” “preorder” 4 | 4 2 3 1 6 5 |
| “4(2(3)(1))(6(5))” “preorder” 2 | 2 3 1 |
| “4(2(3)(1))(6(5))” “inorder” 4 | 3 2 1 4 5 6 |
| “4(2(3)(1))(6(5))” “inorder” 2 | 3 2 1 |

1. Example execution

|  |
| --- |
| >> ./pa2.exe 2 “4(2(3)(1))(6(5))” “inorder” 2  [Task 2]  3 2 1 |

1. Depth of Binary Tree (2 pts)
2. Implement BinaryTree::getDepth function that can calculate the depth of a specific node in a given binary tree. Note that depth of the root node is zero.
3. Input & Output

Input:

* A string with bracket representation for a binary tree. All node values ​​in the tree are unique.
* An integer that indicates a node in the binary tree

Output:

* Depth of the specific node in a given binary tree
* -1 if the input node is not in the binary tree

1. Example input & output

|  |  |
| --- | --- |
| Input | Output |
| “1(2)(3)” 2 | 1 |
| “1(2(3(4)))(5)” 4 | 3 |
| “1(2(3(4)))(5)” 6 | -1 |

1. Example execution

|  |
| --- |
| >> ./pa2.exe 3 “1(2(3(4)))(5)” 4  [Task 3]  3 |

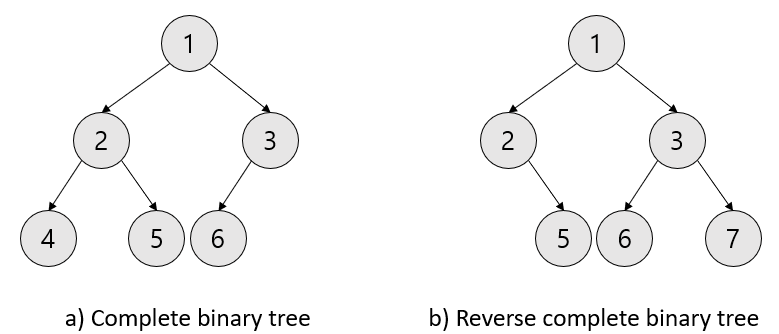
1. Reverse Completeness of Binary Tree (3 pts)
2. Suppose a *reverse complete binary tree* is a binary tree in which every level, except possibly the last, is completely filled, and all nodes in the last level are as far *right* as possible. Figure 1 shows an example. Implement BinaryTree::isReverseComplete function that can check whether a given binary tree is a reverse complete binary tree or not.

Figure 1: An example of a complete binary tree and a reverse complete binary tree.

1. Input & Output

Input:

* String with bracket representation

Output:

* String “True” if the given binary tree is reverse complete, “False” otherwise

1. Example input & output

|  |  |
| --- | --- |
| Input | Output |
| “1(2)(3)” | True |
| “1(2(4)(5))(3(6)())” | False |
| “1(2()(5))(3(6)(7))” | True |
| “1(2(4)(5))(3()(7))” | False |

1. Example execution

|  |
| --- |
| >> ./pa2.exe 4 “1(2()(5))(3(6)(7))”  [Task 4]  True |

1. Max-heap Insertion (2 pts)

*Note: For solving problems 5 and 6, the similar utility functions provided in PA2 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.*

1. Implement a function that **inserts** a new element to a binary max-heap using an array. Your heap should maintain the max-heap property even after the insertion. The limit of length of string to insert is 100. The heap and input string only contains lowercase letters. The alphabet in front has higher priority in the heap structure.
2. Input & Output

Input: A sequence of commands

* (‘insert’,string): insert string into the current max heap

Output:

* Letters in a heap in a node number order, in a string separated with the white space
* Do not consider the exceptional cases such as overflow, underflow or empty heap. We will not use the test cases for those scenarios.

1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [(‘insert’,“apple”)] | a e p p l |
| [(‘insert’,“iloveprogramming”)] | a e i g g m m i o r l p o r n v |
| [(‘insert’,“hello”)] | e h l l o |

1. Example execution

|  |
| --- |
| >> ./pa2.exe 5 “[(‘insert’,“cab”)]”  [Task 5]  a c b |

1. Max-heap Deletion (3 pts)
2. Implement a function that **deletes** the maximum value from the binary max-heap. Your heap should maintain the max heap property even after the deletion.
3. Input & Output

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

* (‘insert’,string): insert string into the current max heap
* (‘delMax’,NULL): delete highest priority letter from current binary max heap and rearrange heap to maintain the max heap property.

Output:

* Letters in a heap in a node number order, in a string separated with the white space
* Do not consider the exceptional cases such as overflow, underflow or empty heap. We will not use the test cases for those scenarios.

1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [(‘insert’,“apple”),(‘delMax’,NULL)] | e l p p |
| [(‘insert’,“lion”), (‘delMax’,NULL)] | l n o |
| [(‘insert’,“hello”),(‘delMax’,NULL),(‘insert’,“man”)] | a l h o m l n |

1. Example execution

|  |
| --- |
| >> ./pa2.exe 6 “[(‘insert’,“apple”),(‘delMax’,NULL)]”  [Task 6]  e l p p |