Binary tree is a structure that consists of nodes. Each node has 2 children: left and right. Each child can be a nullptr or another node. Each node can contain some value.

In binary tree a leaf is a node without children. Every node of the tree has exactly one parent. The only exception to this rule is a root node which has no parent.

A full tree is a binary tree in which the length of the paths between each leaf and a root is the same. Examples: an empty tree, a tree with only 1 node, a tree with exactly 3 nodes in configuration: root + left child + right child. Full binary trees have these numbers of nodes: 0, 1, 3, 7, 15, 31, and so on.

Trees are used to store some values and to traverse through the nodes to find, collect, and do ather fings on these values. You can traverse or visit nodes in a tree in many ways. The most common are: pre-order, in-order and post-order.
Pre-order travelsal is a travelsal over all nodes that for each node we visit (or rather perform some action) first on the node, then on the left child of the node, then on the right child of the node.
In-order travelsal is a travelsal over all nodes that for each node we visit (or rather perform some action) first on the left child of the node, then on the node, then on the right child of the node.
Post-order travelsal is a travelsal over all nodes that for each node we visit (or rather perform some action) first on the left child of the node, then on the right child of the node, and then on the node.

This time you only get this short description and the main.cxx file. The rest of the files are empty. Your task is to use them and fill them with your code.

PART 1 (1 point):
Create a structure BinaryTreeNode with all public:
- pointer to the left child,
- pointer to the right child,
- integer value of the node,
- constructor that takes: an integer value for the node, optional (default: nullptr) pointer to the left node child, optional (default: nullptr) pointer to the right child,
- destructor, which destroys children.

PART 2 (1 point):
Add print() method to the BinaryTreeNode to print to the standard output the value of the node. Do not print values of node’s children.

PART 3 (1 point):
Add printAll() method to the BinaryTreeNode() to print to the standard output the value of the node and the values of the children, and children of the children and so on. All of the values in the connected nodes should be printed. Each value only once. The order of printed values should be pre-order.

PART 4 (1 point):
Implement BinaryTree class. This class has private root, which is a pointer to BinaryTreeNode.
Implement:
- constructor which takes as a parameter a pointer to BinaryTreeNode,
- destructor, which deletes root node,
- print method which prints all of the nodes in the tree in pre-order,
This class should also be able to represent an empty tree (without any nodes).

PART 5 (1 point):
Implement BinaryTreeNodeWithInOrder class as a subclass of the BinaryTreeNode.

Implement:
- constructor,
- destructor,
- printAll() method, that should override the method from BinaryTreeNode. This method should reimplement the printing logic so that all the values are printed in in-order. Make printAll() virtual in the base class. What about the destructors in both classes?

PART 6 (1 points):
Implement an additional constructor in BinaryTree class. It takes an array with values and the length of this array. The values from the array are used to construct a full binary tree.

In the constructor check the number of values passed in parameter. If the number of values is different than the number that allows to build a full tree, then throw an exception. For this part initializes a root node with nullptr. Modify main.cxx file to catch exception and print the message from the exception to the standard output.

You will extend this logic in the next part.

PART 7 (2 points):
Extend the constructor from PART 6. Assume that the order of the values in the parameter is in-order. If the values pass the test from PART 6, then create the tree form nodes of BinaryTreeNodeWithPostOrder class. There are many ways to write this constructor. You may need to add also an extra constructor to BinaryTreeNodeWithPostOrder class. Hint: for the root the value is stored in the last element of the array, the first half of the values should be used to construct the left child (and all its children and descendants), the right half (without last element – used for value of the root) should be used to construct the right child (and all its children and descendants).