June 9th, 2020

Exercises for Programming Practice 2

Today's topic is Trees.

In Exercise 9 to 11, the classes "TreeNode.java" and "TestData.java" are used. These can be obtained from Week 6 of Resources page, "Programming Practice 2" course, manaba+R.

> Note:
> ✓ Do not create "module-info.java", when you create a Java Project.
> ✓ Do not set Package name in the window "New Java Class".
> ✓ Do not use the title of exercise for "contents of the program".
> Think about "contents of the program" yourself.

The deadline for submitting the programs is 18:00 on June 11th, 2020.

The binary tree as shown Fig. 1 is made in "TestData.java" in Fig. 2 using "TreeNode.java" in Fig. 3.

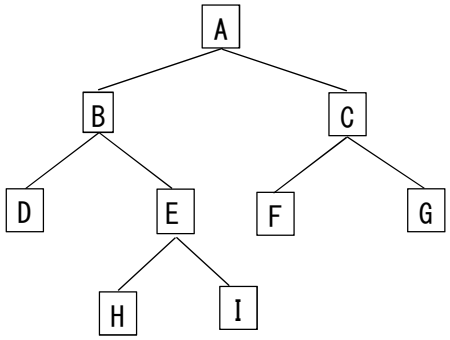

Fig. 1 Binary tree used in Exercise 9 to 11.

```
public class TreeNode {
    String data;
    TreeNode left;
    TreeNode right;

    public TreeNode(String data_, TreeNode left_, TreeNode right_){
        data = data_;
        left = left_;
        right = right_;
    }
}
```

Fig. 2 TreeNode.java

```
public class TestData {
    TreeNode tree;

    public TestData() {
        tree = new TreeNode("A", null, null);
        tree.left = new TreeNode("B", new TreeNode("D", null, null), null);
        tree.left.right = new TreeNode("E",
                                       new TreeNode("H", null, null),
                                       new TreeNode("I", null, null));
        tree.right = new TreeNode("C",
                                  new TreeNode("F", null, null),
                                  new TreeNode("G", null, null));
    }

    public TreeNode getData() {
        return tree;
    }
}
```

Fig. 3 TestNode.java

Exercise 9 (file name “Exercise9.java”)

Create a program named “Exercise9” to display the value of the element of visited node in a preorder traversal in which a node is visited before its descendants. A simple algorithm of preorder traversal is shown in Fig. 4.

```
preorder(v)
    if v is not null
        visit(v)
        for each child w of v
            preorder(w)
```

Fig. 4 A simple algorithm of preorder traversal

The outline of “Exercise9.java” is shown in Fig. 5 and the expected output is shown in Fig. 6.

```
/* comments */
public class Exercise9 {
    TestData testData;
    TreeNode tree;

    public static void main(String[] args) {
        Exercise9 ex9 = new Exercise9();
        ex9.execution();
    }

    public void execution() {
        testData = new TestData();
        tree = testData.getData();
        preorder(tree);
    }

    public void preorder(TreeNode tree) {
        // Complete this part
        }
    }

    public void visit(TreeNode tree) {
        System.out.print(" " + tree.data);
    }
}
```

Fig. 5 The outline of “Exercise9.java”.

```
A B D E H I C F G
```

Fig. 6 the expected output of “Exercise9.java”.

Exercise 10 (file name “Exercise10.java”)

Create a program named “Exercise10” to display the value of the element of visited node in a postorder traversal in which a node is visited after its descendants. A simple algorithm of preorder traversal is shown in Fig. 7.

```
postorder(v)
    if v is not null
        for each child w of v
            postorder(w)
        visit(v)
```

Fig. 7 A simple algorithm of postorder traversal

The outline of “Exercise10.java” is shown in Fig. 8 and the expected output is shown in Fig. 9.

```
/* comments */
public class Exercise10 {
    TestData testData;
    TreeNode tree;

    public static void main(String[] args) {
        Exercise10 ex10 = new Exercise10();
        ex10.execution();
    }

    public void execution() {
        testData = new TestData();
        tree = testData.getData();
        postorder(tree);
    }

    public void postorder(TreeNode tree) {
        // Complete this part
        }
    }

    public void visit(TreeNode tree) {
        System.out.print(" " + tree.data);
    }
}
```

Fig. 8 The outline of “Exercise10.java”.

```
D H I E B F G C A
```

Fig. 9 the expected output of “Exercise9.java”.

Exercise 11 (file name “Exercise11.java”)

Create a program named “Exercise11” to display the value of the element of visited node in an inorder traversal in which a node is visited after its left subtree and before its right subtree. A simple algorithm of inorder traversal is shown in Fig. 10.

```
inorder(v)
    if v has a left child
        inorder(left(v))
    visit(v)
    if v has a right child
        inorder(right(v))
```

Fig. 10 A simple algorithm of inorder traversal

The outline of “Exercise11.java” is shown in Fig. 11 and the expected output is shown in Fig. 12.

```
/* comments */
public class Exercise11 {
    TestData testData;
    TreeNode tree;

    public static void main(String[] args) {
        Exercise11 ex11 = new Exercise11();
        ex11.execution();
    }

    public void execution() {
        testData = new TestData();
        tree = testData.getData();
        inorder(tree);
    }

    public void inorder(TreeNode tree) {
        // Complete this part
    }

    public void visit(TreeNode tree) {
        System.out.print(" " + tree.data);
    }
}
```

Fig. 11 The outline of “Exercise10.java”.

```
D B H E I A F C G
```

Fig. 12 the expected output of “Exercise9.java”.