Nama: Angel Levyne

Kelas: D3IF-47-02

NIM: 607062330045

Hasil Program

```
Post order : 11 10 4 7 8 5 23
Pre order : 23 10 11 5 4 8 7
In order : 10 11 23 4 5 7 8
```

Penjelasan Coding

Class Main

```java
import java.util.Scanner;

/**
 * keseluruhan program diambil dari Java Program to Implement Binary Tre
 * https://www.sanfoundry.com/java-program-implement-binary-tree/
 * dengan sedikit perubahan
 */
public class Main {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        /* Creating object of BT */
        BinaryTree2 bt = new BinaryTree2();
        /* Perform tree operations */
        System.out.println("Binary Tree Test\n");
        int ch;
        do {
            System.out.println("\nBinary Tree Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. search");
            System.out.println("3. count nodes");
            System.out.println("4. check empty");
            int choice = scan.nextInt();
            scan.nextLine();
            switch (choice) {
                case 1:
```

```
                    System.out.println("Enter integer element to insert");
                    bt.insert(scan.nextLine());
                    break;
                case 2:
                    System.out.println("Enter integer element to search");
                    System.out.println("Search result : " + bt.search(
                            scan.nextLine()));
                    break;
                case 3:
                    System.out.println("Nodes = " + bt.countNodes());
                    break;
                case 4:
                    System.out.println("Empty status = " + bt.isEmpty());
                    break;
                default:
                    System.out.println("Wrong Entry \n ");
                    break;
            }
            /* Display tree */
            System.out.print("\nPost order : ");
            bt.postorder();
            System.out.print("\nPre order : ");
            bt.preorder();
            System.out.print("\nIn order : ");
            bt.inorder();
            System.out.println("\n\nDo you want to continue (Type y or n)\n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y' || ch == 'y');
    }
}
```

- **Node**: Setiap "kotak" dalam pohon memiliki dua bagian: data yang menyimpan nilai, dan dua anak yang mengarah ke node lainnya.
- **Insert**: Ketika kita menambahkan data baru ke dalam tree, kita memutuskan apakah data tersebut harus ditempatkan di kiri atau kanan node yang sudah ada
- **Traversal**: Ini adalah cara kita "menemui" setiap node dalam tree. Ada 3 cara berbeda: preorder, inorder, dan postorder.
- **Search**: Saat ingin mencari data tertentu dalam tree, kita membandingkan data tersebut dengan setiap node yang ditemui. Terus bergerak ke kiri atau kanan sampai menemukan data yang dicari atau sampai tidak ada node lagi yang bisa kita cek.

- Count Nodes: Menghitung berapa banyak node yang ada dalam tree. Mulai dari "atas" dan terus ke bawah, menghitung setiap node yang ditemui.
- Empty Check: Sebelum mulai menambahkan atau mencari data dalam tree, kita harus memeriksa apakah tree tersebut kosong atau tidak. Artinya, lihat apakah ada node dalamnya atau tidak.

Class BTNode2

```
/* Class BTNode */
class BTNode2<E> {
    BTNode2 left, right;
    E data;

    /* Constructor */
    public BTNode2() {
        left = null;
        right = null;
        data = null;
    }

    /* Constructor */
    public BTNode2(E item) {
        left = null;
        right = null;
        data = item;
    }

    /* Function to set left node */
    public void setLeft(BTNode2 n) {
        left = n;
    }

    /* Function to set right node */
    public void setRight(BTNode2 n) {
        right = n;
    }

    /* Function to get left node */
    public BTNode2 getLeft() {
        return left;
    }

    /* Function to get right node */
```

```
    public BTNode2 getRight() {
        return right;
    }

    /* Function to set data to node */
    public void setData(E d) {
        data = d;
    }

    /* Function to get data from node */
    public E getData() {
        return data;
    }

}
```

- **Class BTNode2**: Class Ini adalah "blueprint" untuk membuat setiap node dalam tree biner kita. Setiap node ini memiliki dua bagian penting: data (nilai yang disimpan di dalamnya) dan referensi ke anak kiri dan kanannya.
- **Variabel**: Di dalam kelas ini, kita memiliki tiga variabel. left dan right digunakan untuk menyimpan referensi ke anak kiri dan kanan, sedangkan data digunakan untuk menyimpan nilai yang sesuai dengan node tersebut.
- **Constructor**: Ada dua jenis konstruktor di sini. Pertama, constructor kosong yang tidak menerima argumen, yang menginisialisasi left, right, dan data menjadi null. Kedua, konstruktor lainnya menerima argumen item dan menginisialisasi data dengan nilai tersebut.
- Fungsi-fungsi: Ada beberapa fungsi di sini untuk mengelola node.
  - **setLeft()** dan **setRight():** Mengatur referensi ke anak kiri dan kanan.
  - **getLeft()** dan **getRight():** Mendapatkan referensi ke anak kiri dan kanan.
  - **setData():** Mengatur nilai data pada node.
  - **getData():** Mendapatkan nilai data dari node.

Class BinaryTree2

```
class BinaryTree2<E extends Comparable<E>> {
    private BTNode2 root;

    /* Constructor */
    public BinaryTree2() {
```

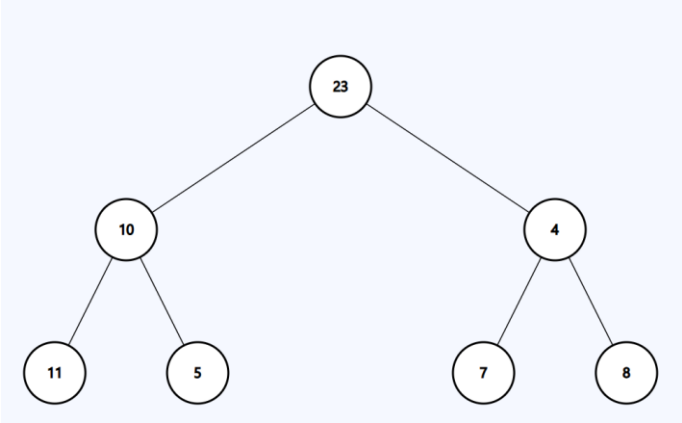
```
        root = null;
    }

    /* Function to check if tree is empty */
    public boolean isEmpty() {
        return root == null;
    }

    /* Functions to insert data */
    public void insert(E data) {
        root = insert(root, data);
    }
    private BTNode2<E> insert(BTNode2<E> node, E data) {
        if (node==null){
            return new BTNode2<>(data);
        }
        if (data.compareTo(node.getData())<0){
            node.setLeft(insert(node.getLeft(), data));
        }else if(data.compareTo(node.getData())>0){
            node.setRight(insert(node.getRight(), data));
        }
        return node;
    }

    /* Function to count number of nodes */
    public int countNodes() {
        return countNodes(root);
    }

    /* Function to count number of nodes recursively */
    private int countNodes(BTNode2 r) {
        if (r == null)
            return 0;
        else {
            int l = 1;
            l += countNodes(r.getLeft());
            l += countNodes(r.getRight());
            return l;
        }
    }

    /* Function to search for an element */
    public boolean search(E val) {
        return search(root, val);
    }
```

```java
/* Function to search for an element recursively */
private boolean search(BTNode2 r, E val) {
    if (r.getData() == val)
        return true;
    if (r.getLeft() != null)
        if (search(r.getLeft(), val))
            return true;
    if (r.getRight() != null)
        if (search(r.getRight(), val))
            return true;
    return false;
}

/* Function for inorder traversal */
public void inorder() {
    inorder(root);
}

private void inorder(BTNode2 r) {
    if (r != null) {
        inorder(r.getLeft());
        System.out.print(r.getData() + " ");
        inorder(r.getRight());
    }
}

/* Function for preorder traversal */
public void preorder() {
    preorder(root);
}

private void preorder(BTNode2 r) {
    if (r != null) {
        System.out.print(r.getData() + " ");
        preorder(r.getLeft());
        preorder(r.getRight());
    }
}

/* Function for postorder traversal */
public void postorder() {
    postorder(root);
}
```

```
    private void postorder(BTNode2 r) {
        if (r != null) {
            postorder(r.getLeft());
            postorder(r.getRight());
            System.out.print(r.getData() + " ");
        }
    }
}
```

- **Class BinaryTree2**: Ini adalah class utama yang digunakan untuk membuat dan mengelola pohon biner.
  - Ada satu variabel private, root, yang digunakan untuk menyimpan referensi ke node pertama (akar) dari tree biner.
  - Contructor yang menginisialisasi root menjadi null, ini artinya tree sedang kosong saat dibuat.
- Fungsi-fungsi:
  - **isEmpty():** Fungsi ini digunakan untuk memeriksa apakah pohon biner kosong atau tidak dengan memeriksa apakah root sama dengan null.
  - **insert(E data**): Fungsi ini digunakan untuk menyisipkan data baru ke dalam pohon biner. Ini memanggil metode insert rekursif yang akan mencari tempat yang sesuai untuk menempatkan data baru.
  - **countNodes():** Fungsi ini menghitung dan mengembalikan jumlah total node dalam pohon biner dengan memanggil metode rekursif countNodes.
  - **search(E val):** Fungsi ini digunakan untuk mencari apakah nilai tertentu ada dalam pohon biner atau tidak dengan menggunakan pencarian rekursif.
  - **inorder(), preorder(), postorder():** Ini digunakan untuk menelusuri tree biner dalam urutan tertentu: inorder, preorder, dan postorder. Ini membantu dalam menampilkan atau memproses data dalam pohon dengan cara tertentu.
- **Metode Rekursif**: Beberapa metode menggunakan pendekatan rekursif untuk melakukan operasi seperti Insert, Search, atau Count Nodes dalam tree.

Gambar dari pohon yang terbentuk sesuai dengan masukan pada program



pohon biner terbentuk seperti itu karena efisiensi dan skalabilitasnya. Bentuk pohon memungkinkan untuk pencarian yang cepat dan mudah, serta insert dan delete data yang efisien.

1. Preorder (depth first order)
   mempunyai urutan;
   a. Cetak isi simpul yang di kunjungi (root)
   b. Kunjungi Cabang Kiri
   c. Kunjungi Cabang Kanan
2. Inorder
   mempunyai urutan :
   a. Kunjungi Cabang Kiri
   b. Cetak isi simpul yang dikunjungi (Akar)
   c. Kunjungi Cabang Kanan
3. Postorder,

   mempunyai urutan :

   a. Kunjungi Cabang Kiri
   b. Kunjungi Cabang Kanan
   c. Cetak isi simpul yang dikunjungi (Akar)