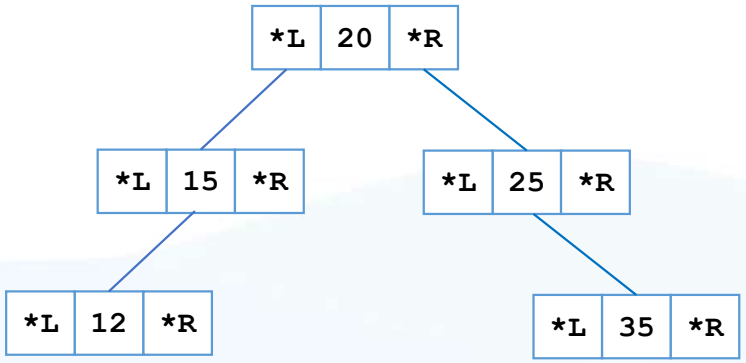
PROGRAM STUDI
TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS DIAN NUSWANTORO

Mata Kuliah
Algoritma dan
Struktur Data

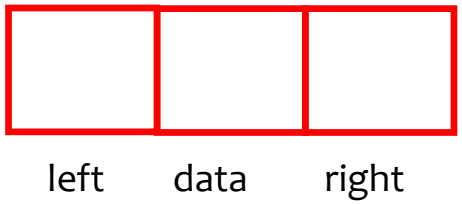


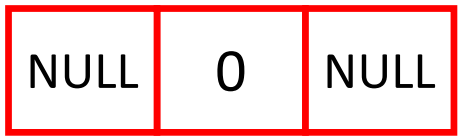
Tree

Tim Pengampu Mata Kuliah Algoritma dan Struktur Data

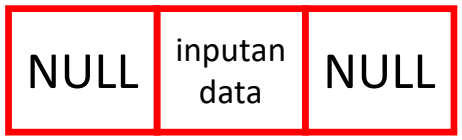
Class Node

```
class Node{
    public:
    int data;
    Node *left;
    Node *right;

    Node(){
        data = 0;
        left = right = NULL;
    }
    Node(int data){
        this->data = data;
        left = right = NULL;
    }
};
```





Konstruktor default



Konstruktor inputan
Data tergantung inputan

pointer right berisi alamat Node selanjutnya
pointer left berisi alamat Node sebelumnya

Class Tree

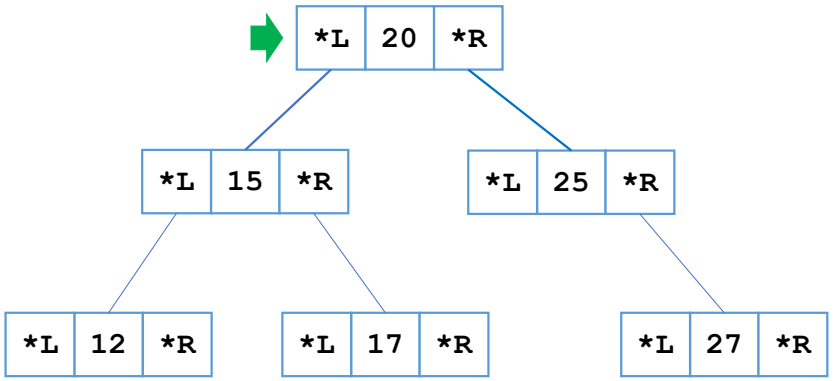
```
class Tree{
    public:
    Node *root;

    Node *insertBinaryRoot(Node *root, int nilai);
    void preOrder(Node *root);
    void inOrder(Node *root);
    void postOrder(Node *root);
    int heightNode(Node *root);
    void printCurrentLevel(Node *root, int level);
    void levelOrder(Node *root);
};
```

main.cpp insert manual

```
int main()
{
    /* representasi tree
         20
        /  \
       15  25
      / \    \
     12 17   27  */
    //insert manual
    Node *root = new Node(20);
    root->left = new Node(15);
    root->left->left = new Node(12);
    root->left->right = new Node(17);
    root->right = new Node(25);
    root->right->left = new Node(27);
    Tree pohon;
    cout << "Pre Order: " << endl;
    pohon.preOrder(root); //20 15 12 17 25 27
    cout << "\nIn Order: " << endl;
    pohon.inOrder(root); //12 15 17 20 25 27
    cout << "\nPost Order: " << endl;
    pohon.postOrder(root); //12 17 15 27 25 20

    return 0;
}
```

preOrder Traversal

```
void Tree::preOrder(Node * root){
    if(root != NULL){
        cout << root->data << " ";
        preOrder(root->left);
        preOrder(root->right);
    }
}
```



Contoh preOrder Traversal sudah ada pada file Pertemuan 14-15 – Tree.pdf
Slide 25-46

inOrder Traversal #1

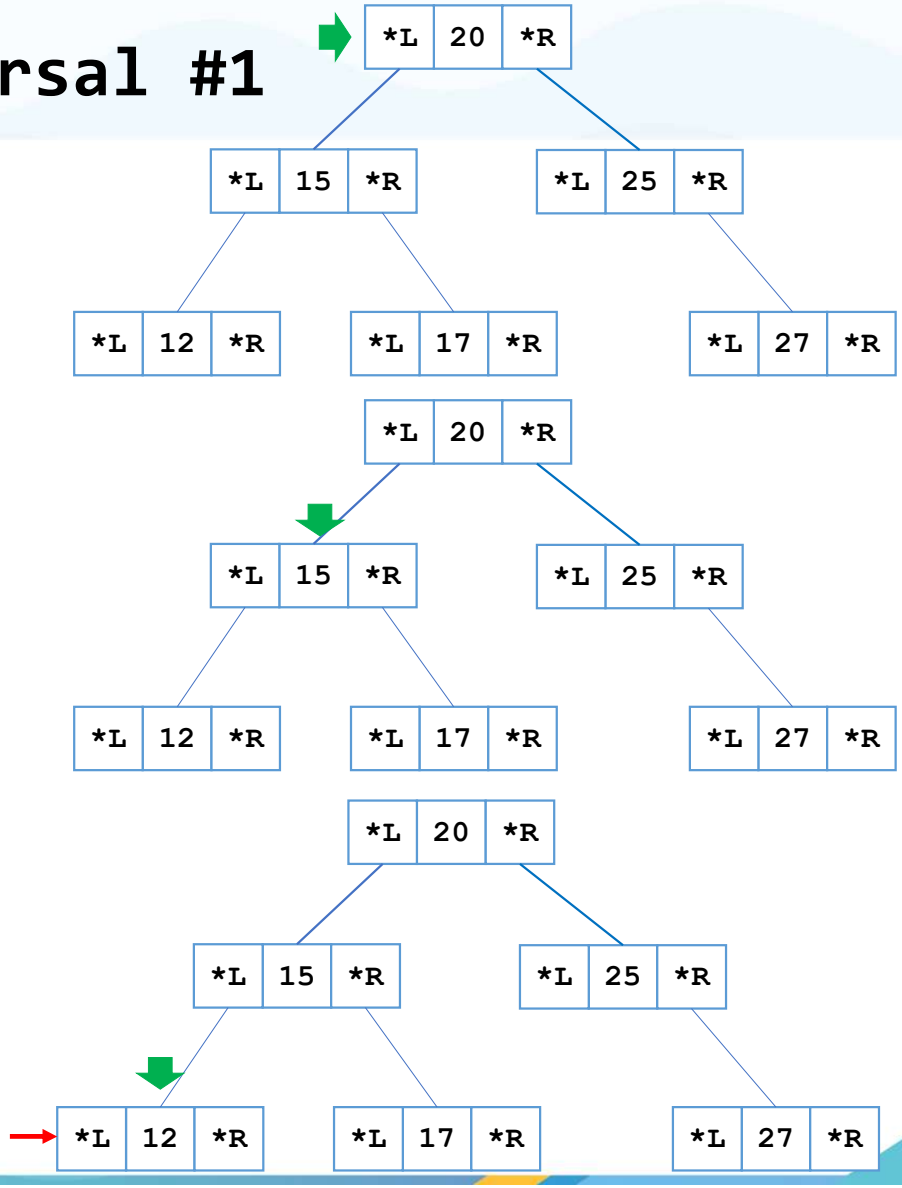
```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

Cetak = 12

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
```



inOrder Traversal #2

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

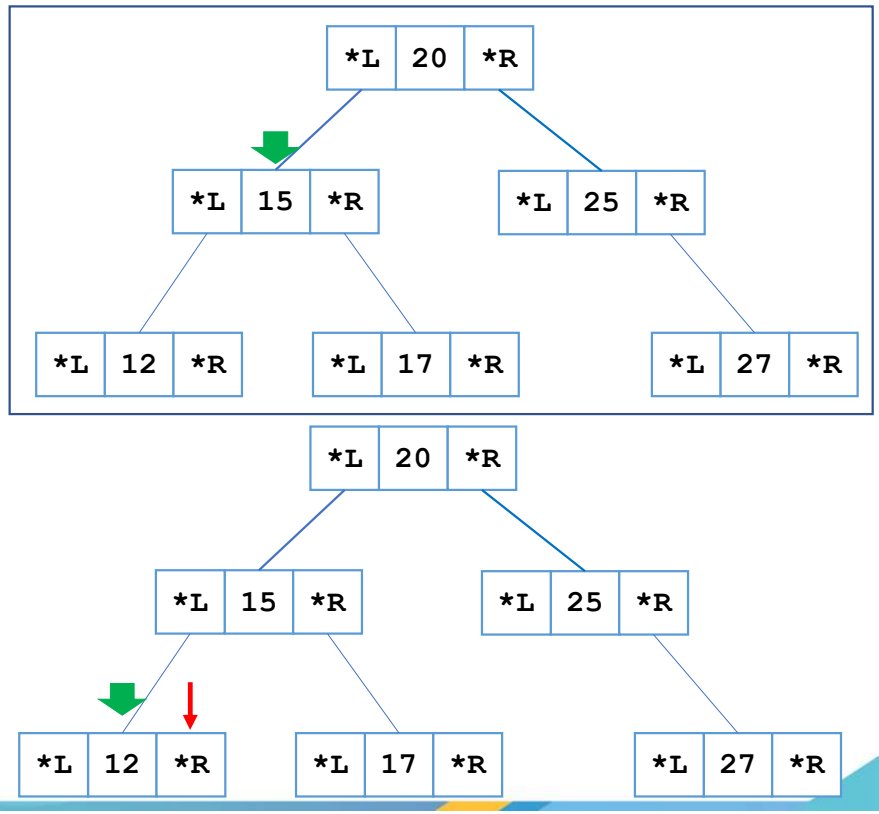
```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

Cetak = 12 15

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

Cetak = 12

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
```



inOrder Traversal #3

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

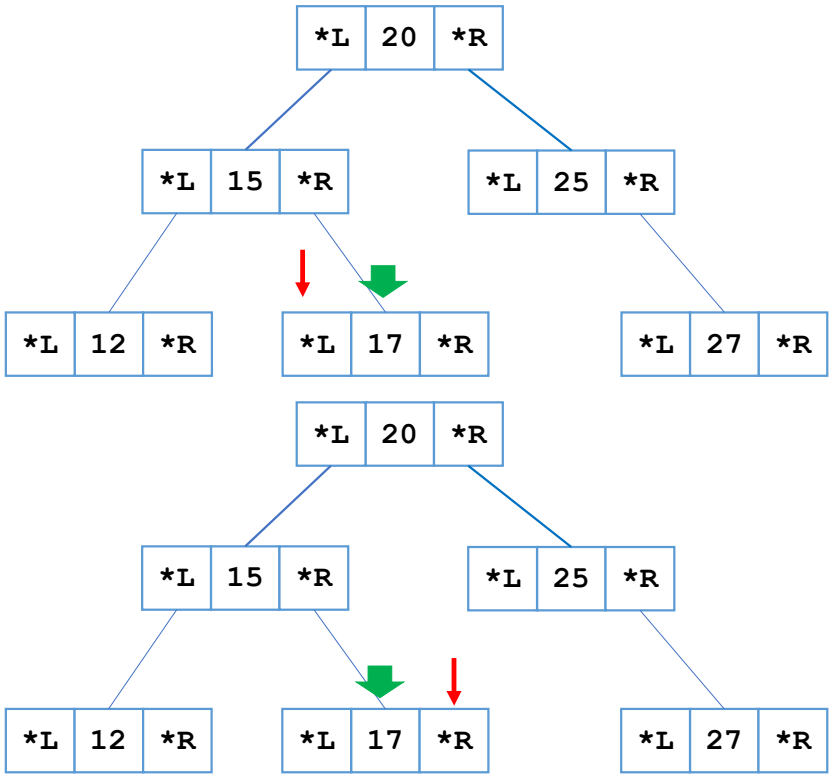
```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
```

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
```

Cetak = 12 15 17



inOrder Traversal #4

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

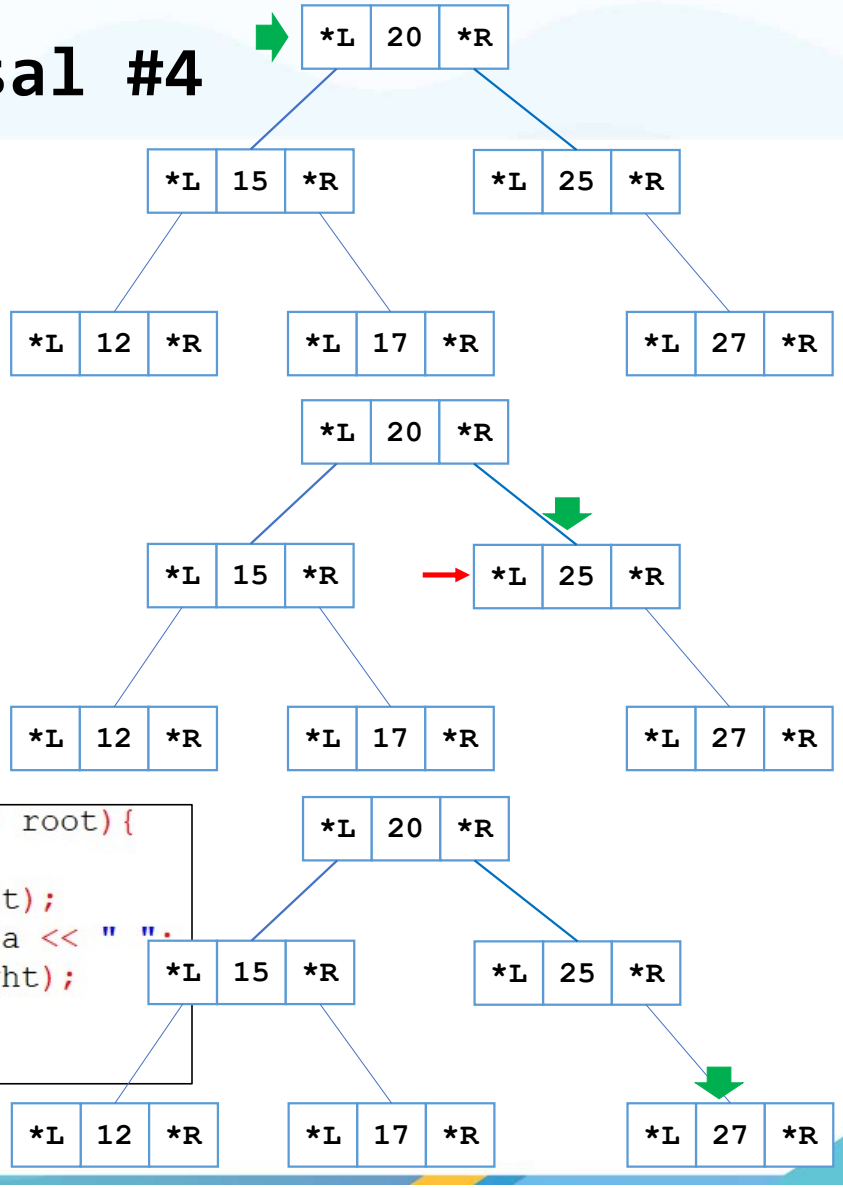
Cetak = 12 15 17 20

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

Cetak = 12 15 17 20 25

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```



inOrder Traversal #5

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

Cetak = 12 15 17 20

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

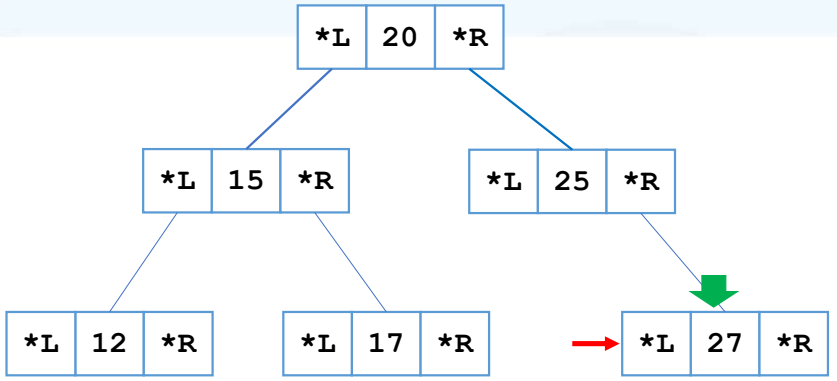
Cetak = 12 15 17 20 25

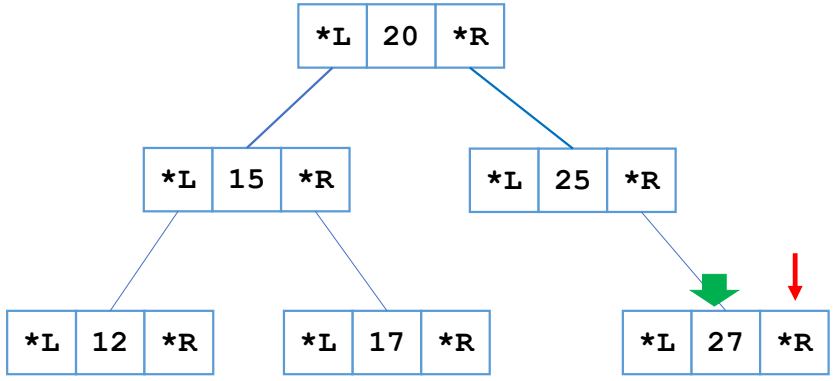
```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

Cetak = 12 15 17 20 25 27

```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```

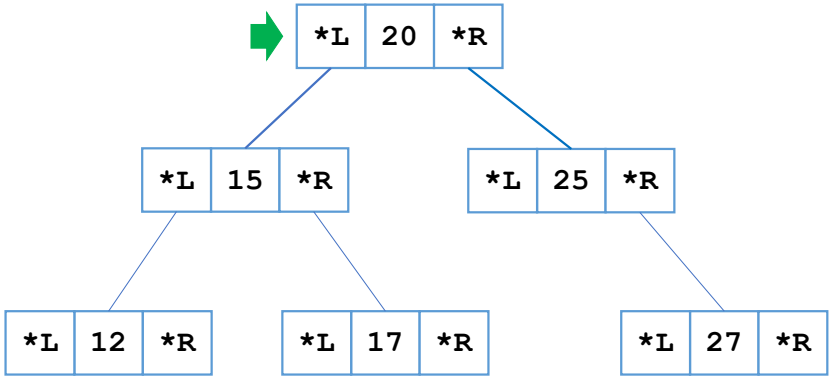
```
void Tree::inOrder(Node * root){
    if(root != NULL){
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
```





postOrder Traversal

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```



postOrder Traversal #1

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

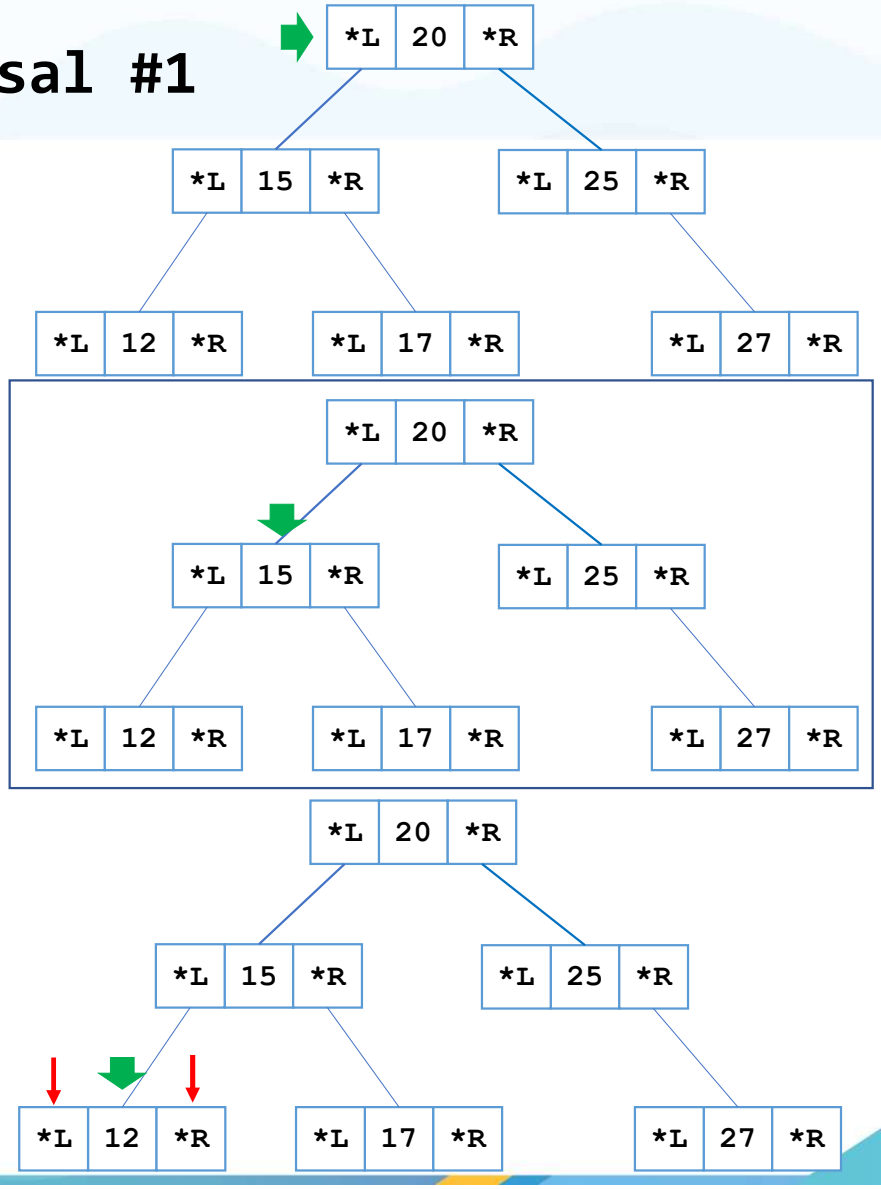
```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

Cetak = 12

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
```

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
```



postOrder Traversal #2

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

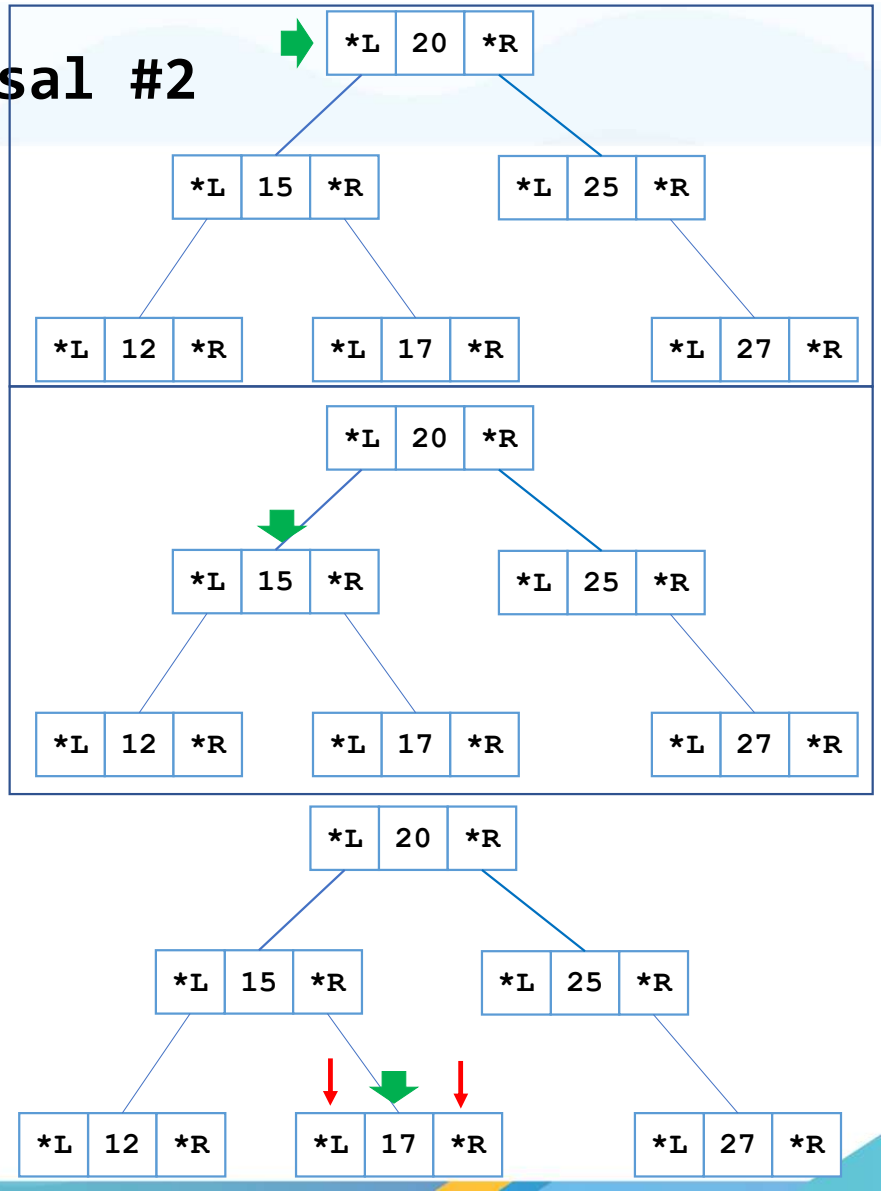
Cetak = 12 17 15

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

Cetak = 12 17

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
```

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
```

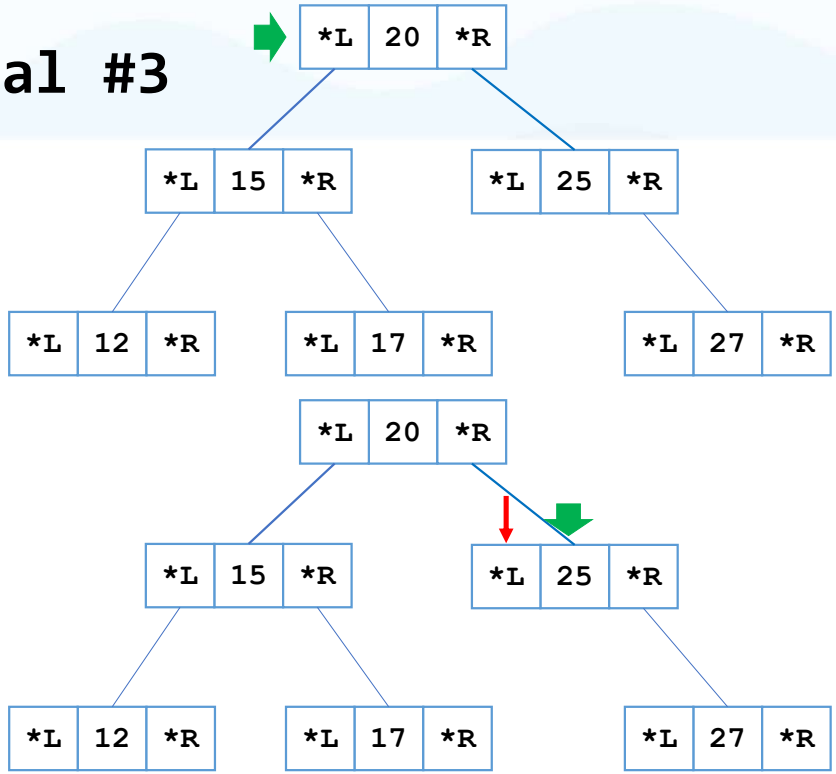


postOrder Traversal #3

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```



postOrder Traversal #4

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

Cetak = 12 17 15 27 25 20

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

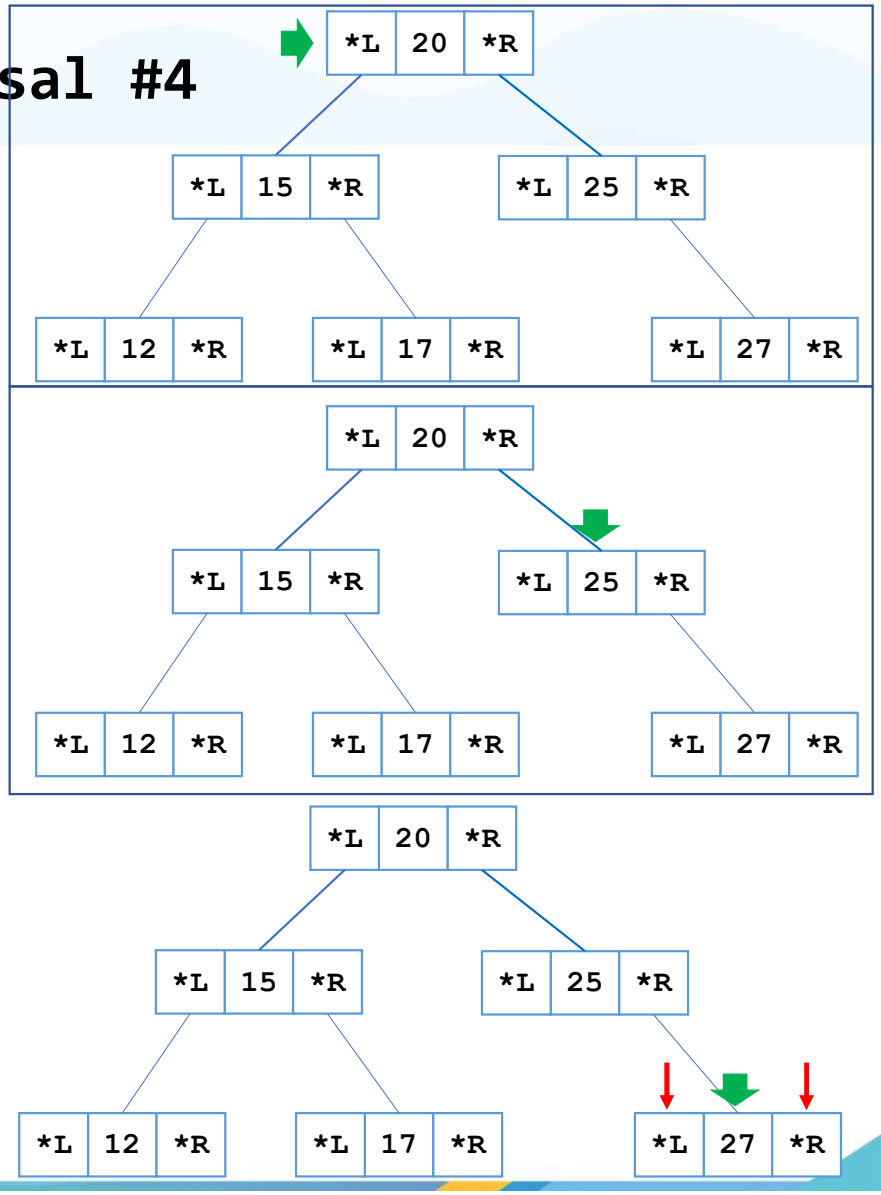
Cetak = 12 17 15 27 25

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

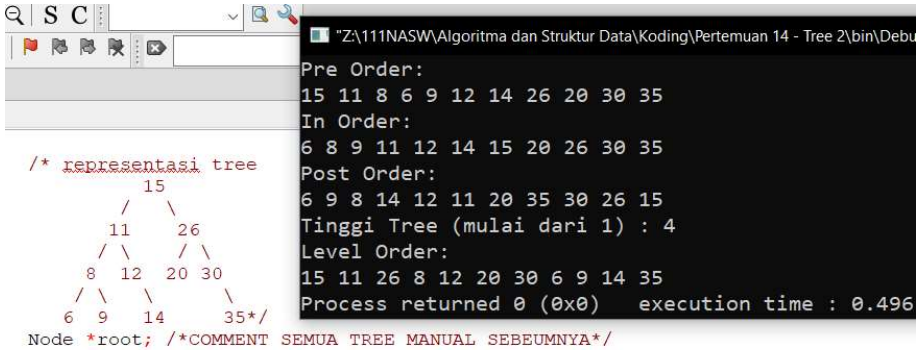
Cetak = 12 17 15 27

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
```

```
void Tree::postOrder(Node * root){
    if(root != NULL){
        postOrder(root->left);
        postOrder(root->right);
```



main.cpp insertBinaryRoot




```
/* representasi tree
          15
         /   \
        11    26
       / \    / \
      8  12  20 30
     / \   \      \
    6  9   14     35*/
Node *root; /*COMMENT SEMUA TREE MANUAL SEBEUMNYA*/
Tree T;
root = NULL;
root = T.insertBinaryRoot(root,15);
root = T.insertBinaryRoot(root,11); root = T.insertBinaryRoot(root,26);
root = T.insertBinaryRoot(root,8); root = T.insertBinaryRoot(root,12);
root = T.insertBinaryRoot(root,20); root = T.insertBinaryRoot(root,30);
root = T.insertBinaryRoot(root,6); root = T.insertBinaryRoot(root,9);
root = T.insertBinaryRoot(root,14); root = T.insertBinaryRoot(root,35);

cout << "Pre Order: " << endl; T.preOrder(root);
cout << "\nIn Order: " << endl; T.inOrder(root);
cout << "\nPost Order: " << endl; T.postOrder(root);

int tinggi;
tinggi = T.heightNode(root);
cout <<"\nTinggi Tree (mulai dari 1) : " << tinggi;

cout << "\nLevel Order: " << endl;
T.levelOrder(root);

return 0;
```

insertBinaryRoot

```
Node *Tree::insertBinaryRoot(Node * root, int nilai){
    if(root == NULL){
        root = new Node(nilai);
    }else if(nilai <= root->data){
        root->left = insertBinaryRoot(root->left,nilai);
    }else{
        root->right = insertBinaryRoot(root->right,nilai);
    }
    return root;
}
```

Contoh insert Binary Tree sudah ada pada file Pertemuan 14-15 – Tree.pdf
Slide 20-23

insertBinaryRoot(root, 12)

Sudah ada Tree 20, 15 , 25 lalu akan insertBinaryRoot(root, 12)

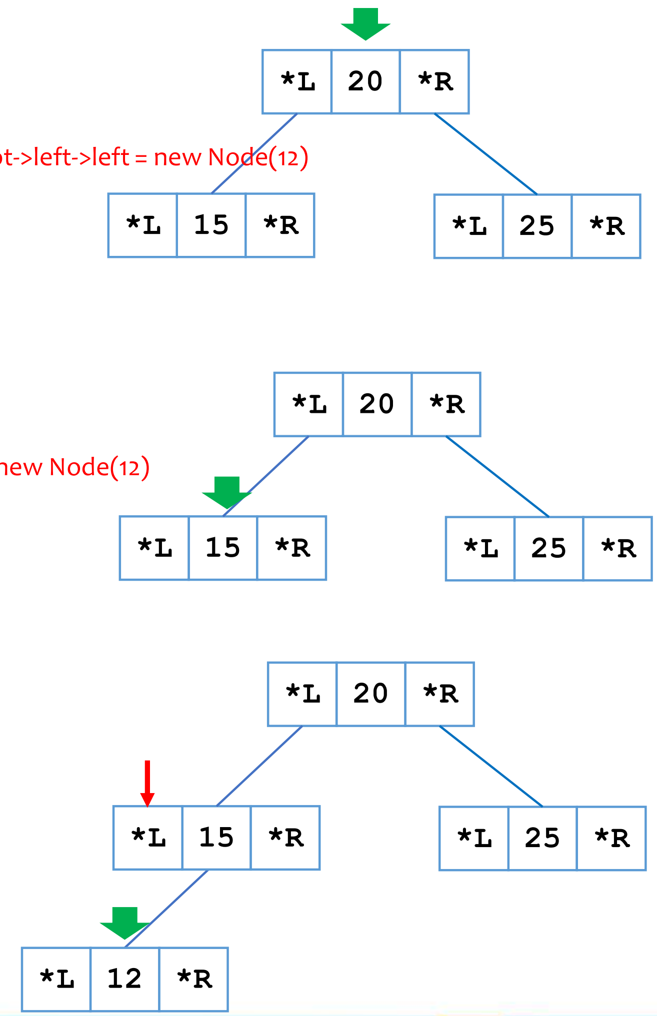
```
Node *Tree::insertBinaryRoot(Node * root, int nilai){
    if(root == NULL){
        root = new Node(nilai);
    }else if(nilai <= root->data){
        root->left = insertBinaryRoot(root->left,nilai);
    }else{
        root->right = insertBinaryRoot(root->right,nilai);
    }
    return root;
}
```

root->left->left = new Node(12)

```
Node *Tree::insertBinaryRoot(Node * root, int nilai){
    if(root == NULL){
        root = new Node(nilai);
    }else if(nilai <= root->data){
        root->left = insertBinaryRoot(root->left,nilai);
    }else{
        root->right = insertBinaryRoot(root->right,nilai);
    }
    return root;
}
```

root->left = new Node(12)

```
Node *Tree::insertBinaryRoot(Node * root, int nilai){
    if(root == NULL){
        root = new Node(nilai);
    }else if(nilai <= root->data){
        root->left = insertBinaryRoot(root->left,nilai);
    }else{
        root->right = insertBinaryRoot(root->right,nilai);
    }
    return root;
}
```



insertBinaryRoot(root, 35)

Sudah ada Tree 20, 15 , 25, 12 lalu akan insertBinaryRoot(root, 35)

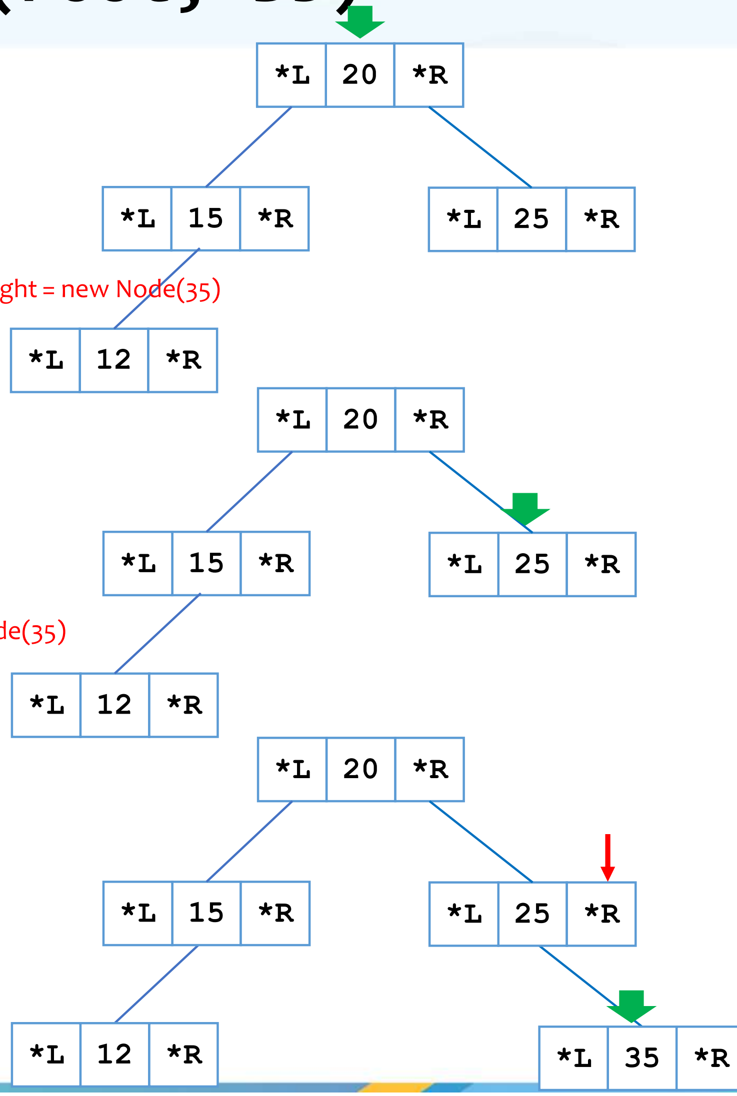
```
Node *Tree::insertBinaryRoot(Node * root, int nilai){
    if(root == NULL){
        root = new Node(nilai);
    }else if(nilai <= root->data){
        root->left = insertBinaryRoot(root->left,nilai);
    }else{
        root->right = insertBinaryRoot(root->right,nilai);
    }
    return root;
}
```

root->right->right = new Node(35)

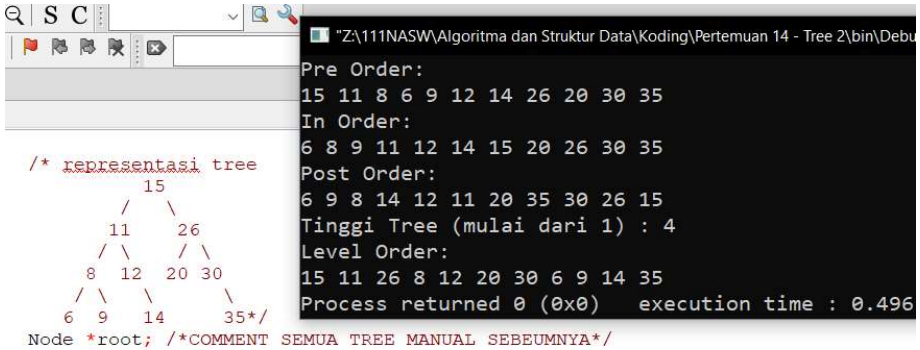
```
Node *Tree::insertBinaryRoot(Node * root, int nilai){
    if(root == NULL){
        root = new Node(nilai);
    }else if(nilai <= root->data){
        root->left = insertBinaryRoot(root->left,nilai);
    }else{
        root->right = insertBinaryRoot(root->right,nilai);
    }
    return root;
}
```

root->right = new Node(35)

```
Node *Tree::insertBinaryRoot(Node * root, int nilai){
    if(root == NULL){
        root = new Node(nilai);
    }else if(nilai <= root->data){
        root->left = insertBinaryRoot(root->left,nilai);
    }else{
        root->right = insertBinaryRoot(root->right,nilai);
    }
    return root;
}
```



main.cpp




```
/* representasi tree
          15
        /   \
       11    26
      / \    / \
     8  12  20 30
    / \   \       \
   6  9   14      35*/
Node *root; /*COMMENT SEMUA TREE MANUAL SEBEUMNYA*/
Tree T;
root = NULL;
root = T.insertBinaryRoot(root,15);
root = T.insertBinaryRoot(root,11); root = T.insertBinaryRoot(root,26);
root = T.insertBinaryRoot(root,8); root = T.insertBinaryRoot(root,12);
root = T.insertBinaryRoot(root,20); root = T.insertBinaryRoot(root,30);
root = T.insertBinaryRoot(root,6); root = T.insertBinaryRoot(root,9);
root = T.insertBinaryRoot(root,14); root = T.insertBinaryRoot(root,35);

cout << "Pre Order: " << endl; T.preOrder(root);
cout << "\nIn Order: " << endl; T.inOrder(root);
cout << "\nPost Order: " << endl; T.postOrder(root);

int tinggi;
tinggi = T.heightNode(root);
cout <<"\nTinggi Tree (mulai dari 1) : " << tinggi;

cout << "\nLevel Order: " << endl;
T.levelOrder(root);

return 0;
```

heightNode #1

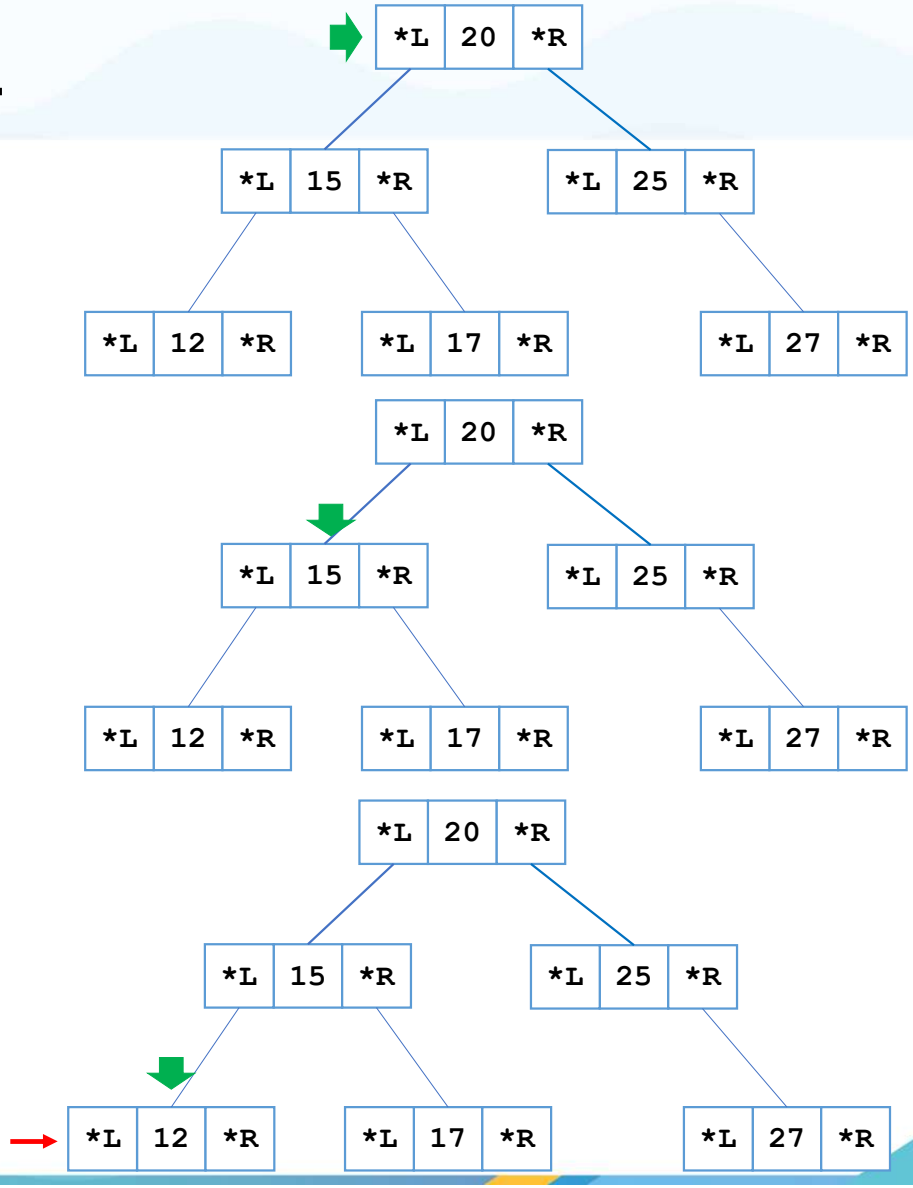
```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 0

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
```



heightNode #2

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

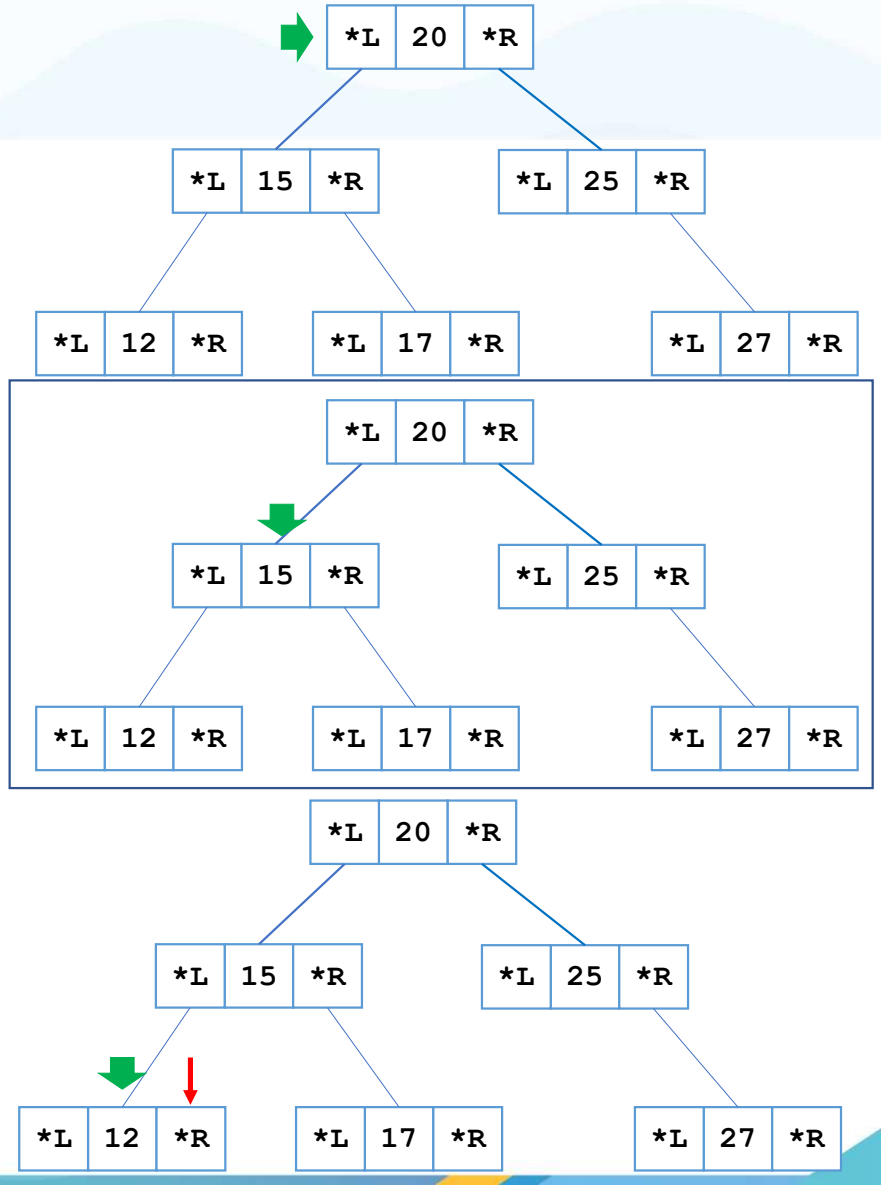
```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 1

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
return 1+max(0,0)
return 1
```

leftH = 0
rightH = 0

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
```



heightNode #3

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

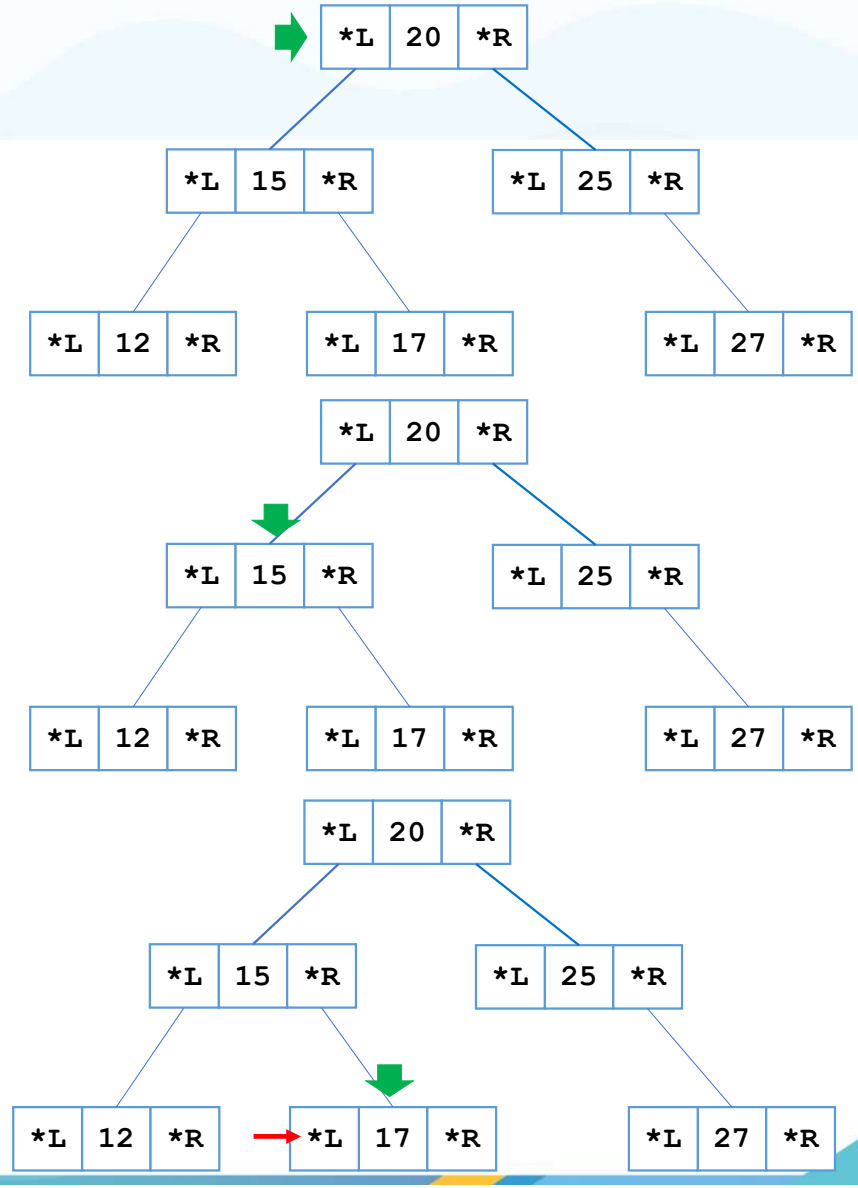
```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 1

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 0

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
```



heightNode #4

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 2

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 1
rightH = 1

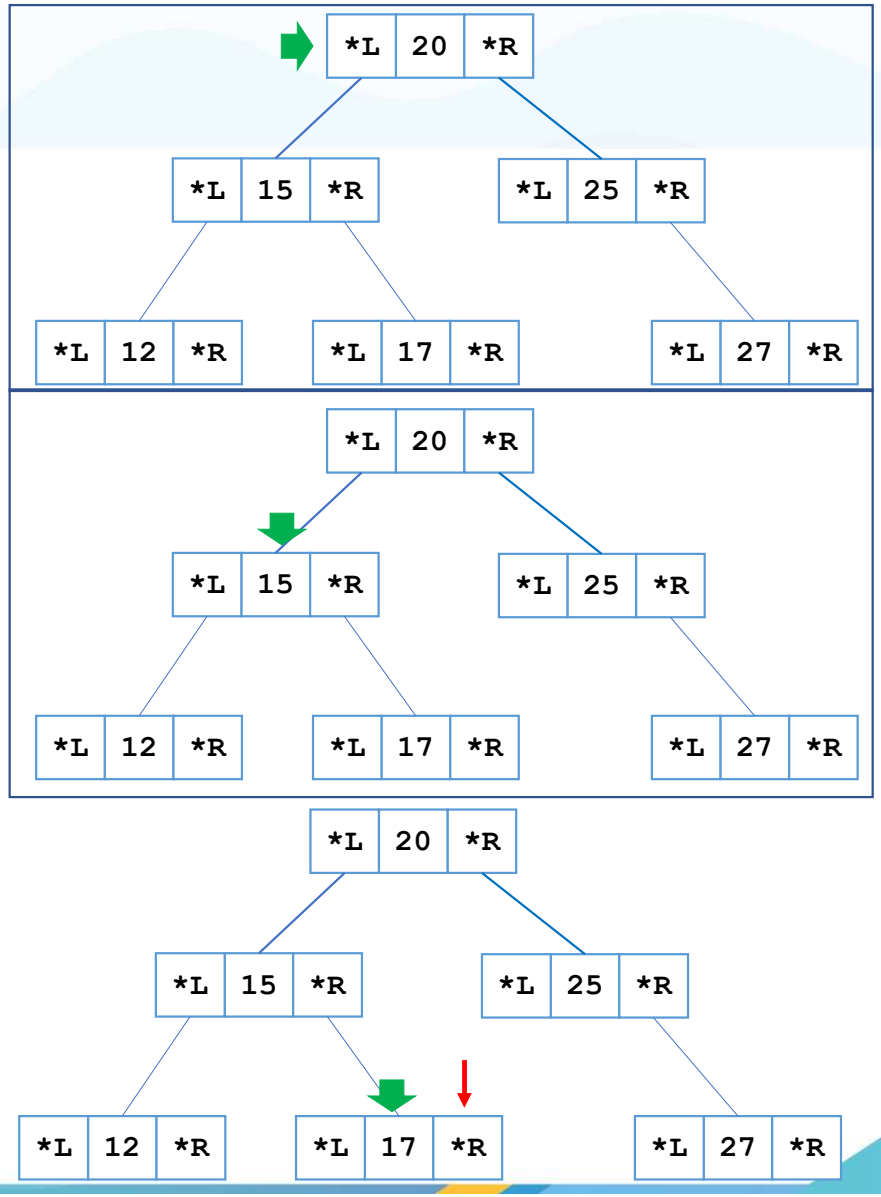
return 1+max(1,1)
return 2

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 0
rightH = 0

return 1+max(0,0)
return 1

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
```



heightNode #5

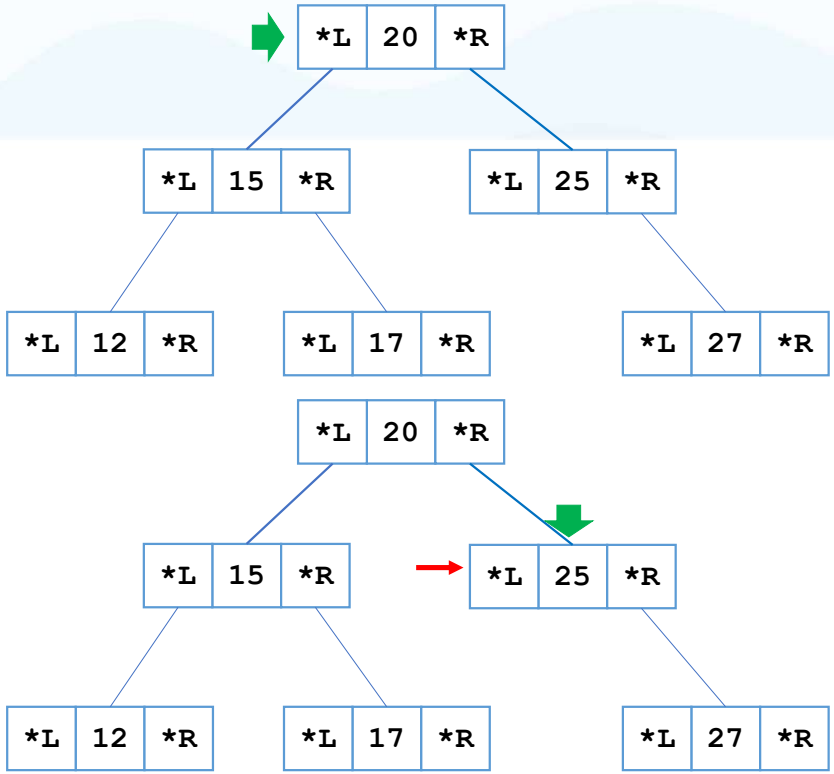
```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 2

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 0

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```



heightNode #6

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 2

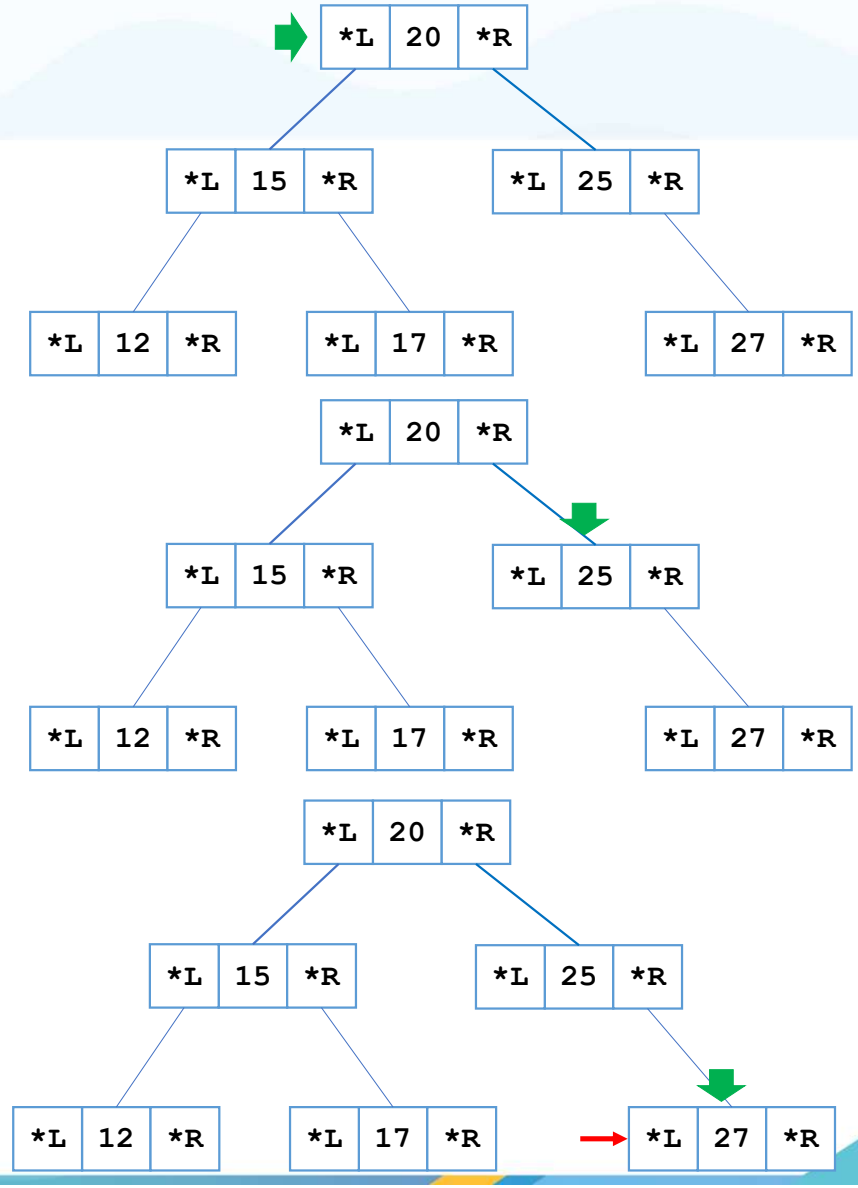
```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 0

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 0

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
```



heightNode #7

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 2
rightH = 2

return 1+max(2,2)
return 3

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

leftH = 0
rightH = 1

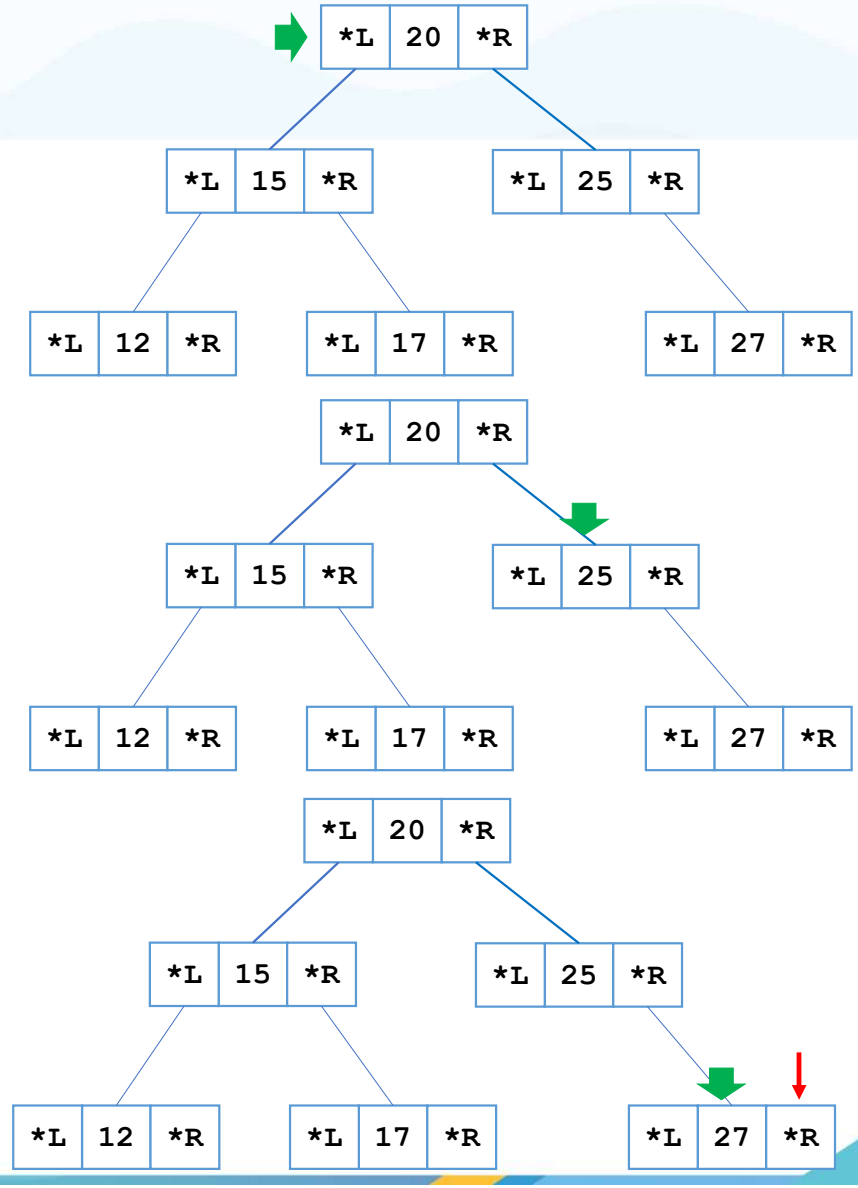
return 1+max(0,1)
return 2

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
    }
}
```

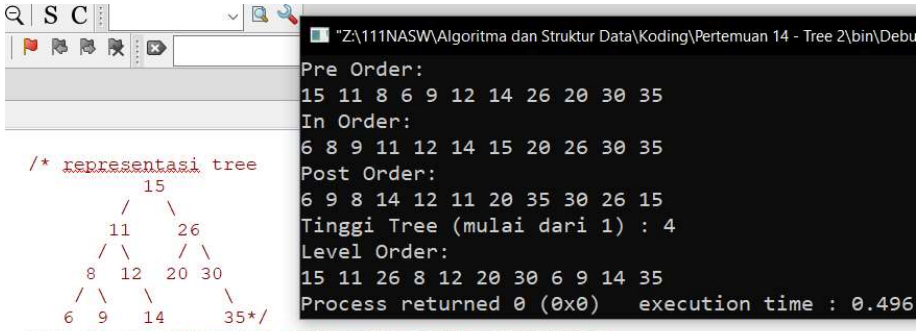
leftH = 0
rightH = 0

return 1+max(0,0)
return 1

```
int Tree::heightNode(Node * root){
    if(root == NULL){
        return 0;
    }else{
        int leftH=heightNode(root->left);
        int rightH= heightNode(root->right);

        return 1+max(leftH, rightH);
```



main.cpp



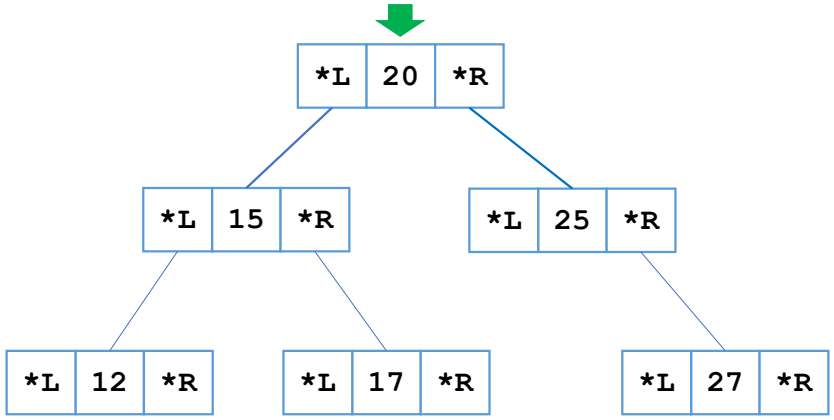

```
/* representasi tree
          15
        /   \
       11    26
      / \    / \
     8  12  20 30
    / \   \      \
   6  9   14     35*/
Node *root; /*COMMENT SEMUA TREE MANUAL SEBEUMNYA*/
Tree T;
root = NULL;
root = T.insertBinaryRoot(root,15);
root = T.insertBinaryRoot(root,11); root = T.insertBinaryRoot(root,26);
root = T.insertBinaryRoot(root,8); root = T.insertBinaryRoot(root,12);
root = T.insertBinaryRoot(root,20); root = T.insertBinaryRoot(root,30);
root = T.insertBinaryRoot(root,6); root = T.insertBinaryRoot(root,9);
root = T.insertBinaryRoot(root,14); root = T.insertBinaryRoot(root,35);

cout << "Pre Order: " << endl; T.preOrder(root);
cout << "\nIn Order: " << endl; T.inOrder(root);
cout << "\nPost Order: " << endl; T.postOrder(root);

int tinggi;
tinggi = T.heightNode(root);
cout <<"\nTinggi Tree (mulai dari 1) : " << tinggi;

cout << "\nLevel Order: " << endl;
T.levelOrder(root);

return 0;
```

printCurrentLevel

T.printCurrentLevel(root, 0)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

Cetak: 20



printCurrentLevel

T.printCurrentLevel(root, 1)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

printCurrentLevel(root->left, 0)

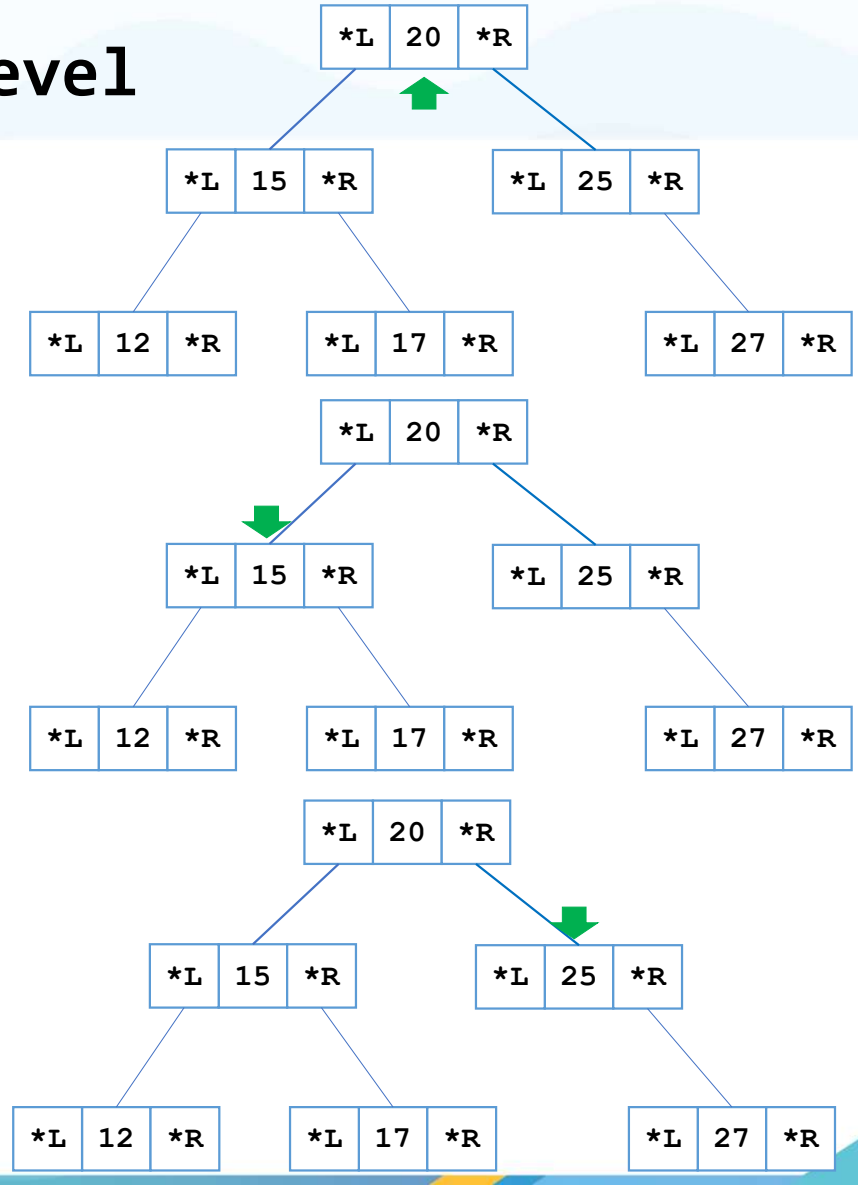
```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

Cetak: 15

printCurrentLevel(root->right, 0)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

Cetak: 15 25



printCurrentLevel

T.printCurrentLevel(root, 2) #1

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

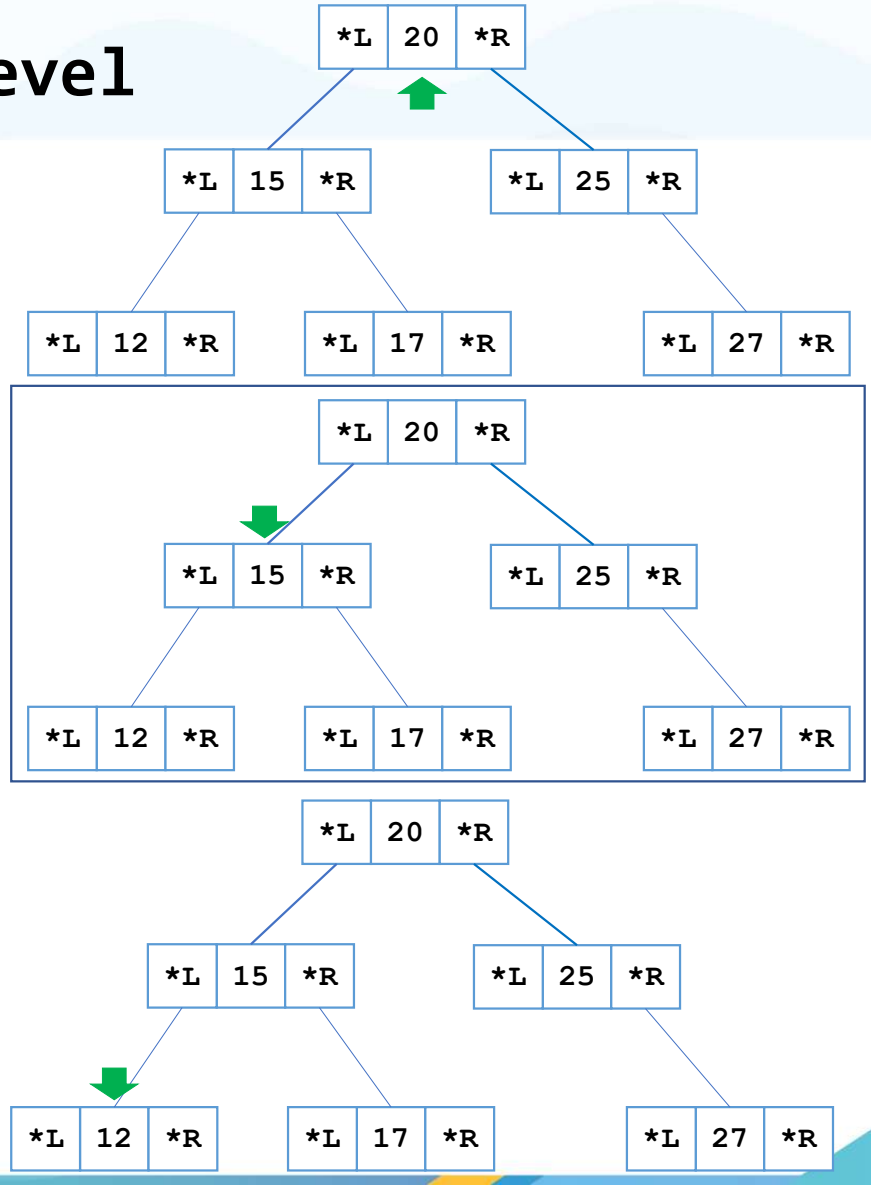
printCurrentLevel(root->left, 1)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

printCurrentLevel(root->left, 0)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

Cetak: 12



printCurrentLevel

T.printCurrentLevel(root, 2) #2

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

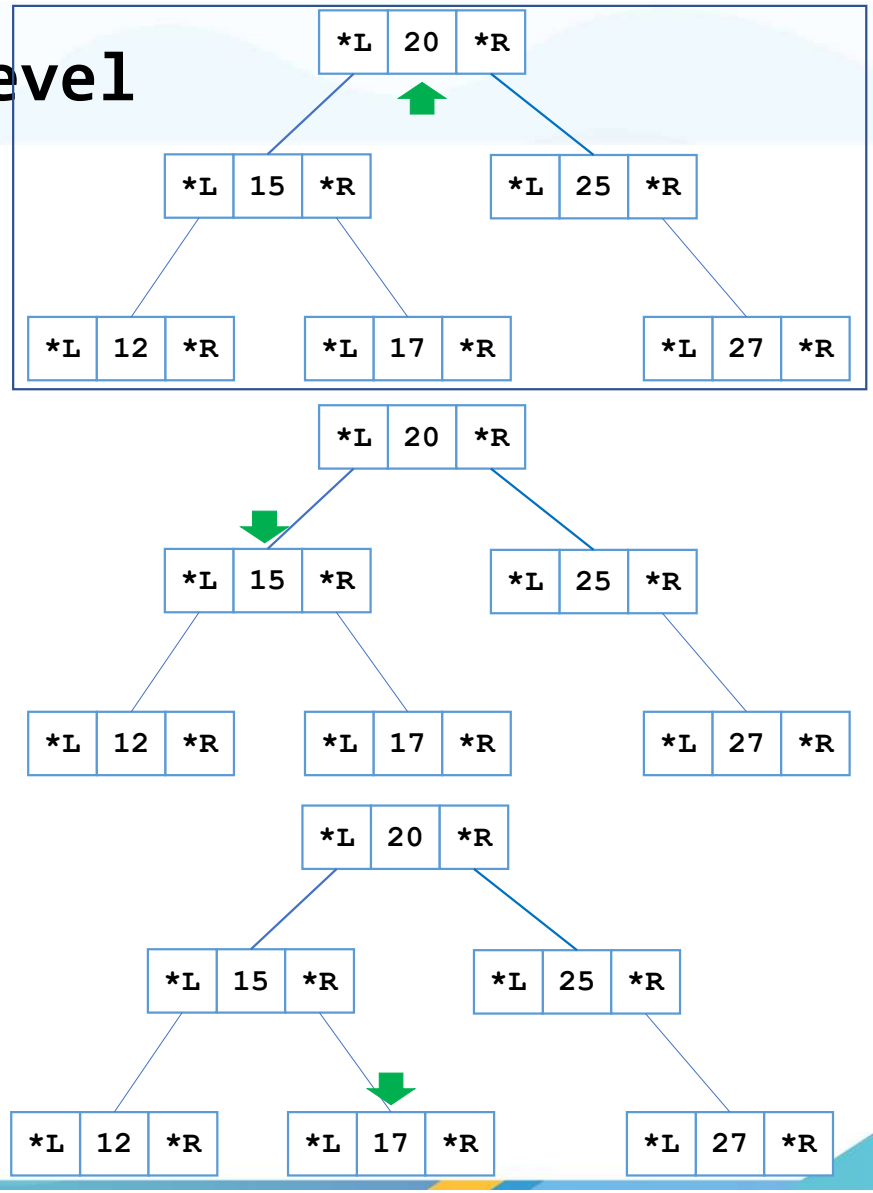
printCurrentLevel(root->left, 1)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

printCurrentLevel(root->right, 0)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

Cetak: 12 17



printCurrentLevel

T.printCurrentLevel(root, 2) #3

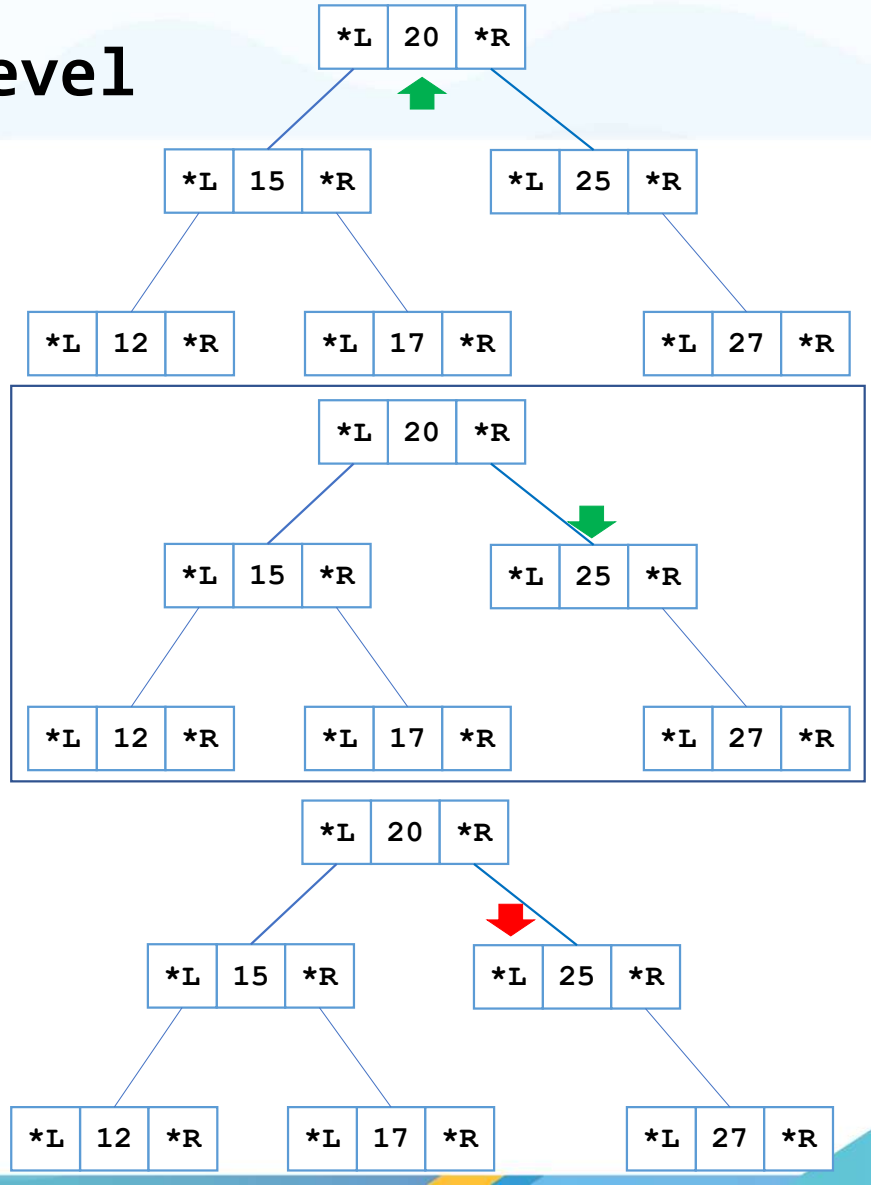
```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

printCurrentLevel(root->right, 1)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

printCurrentLevel(root->left, 0)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```



printCurrentLevel

T.printCurrentLevel(root, 2) #4

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

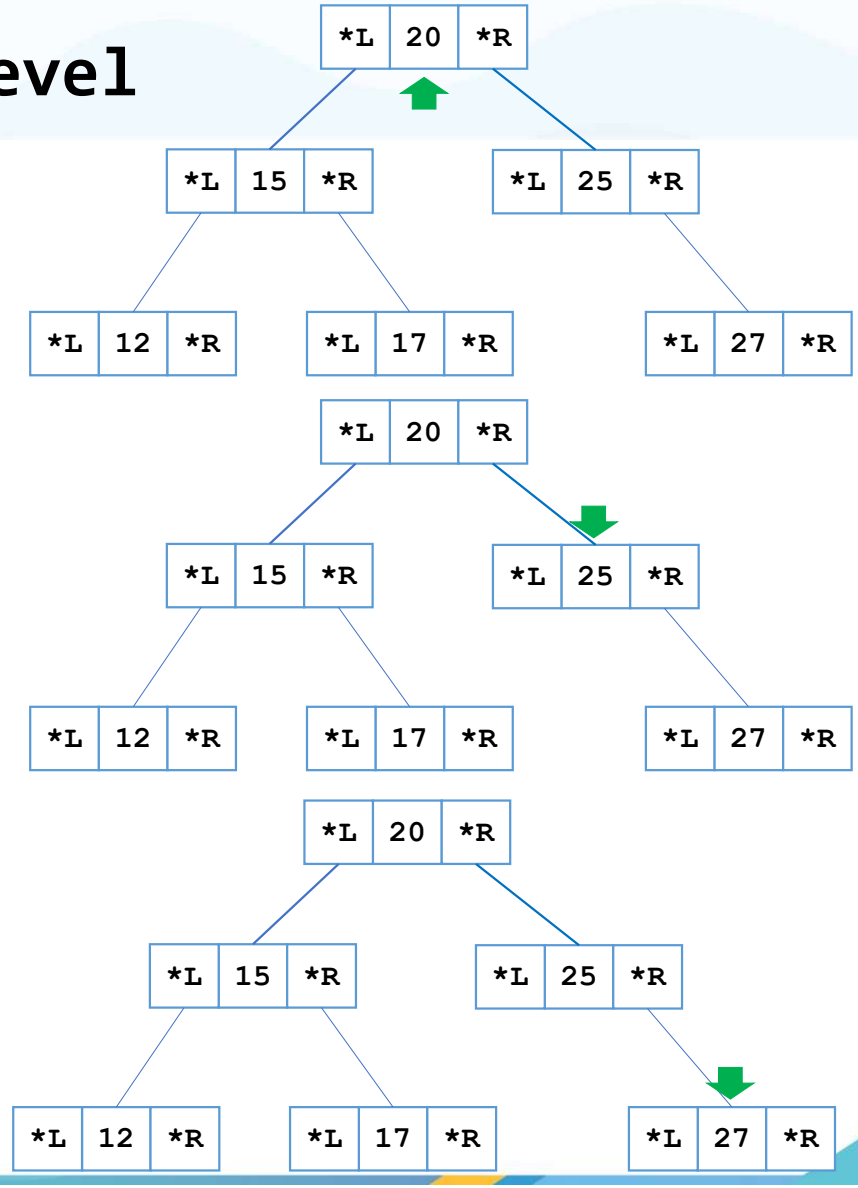
printCurrentLevel(root->right, 1)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

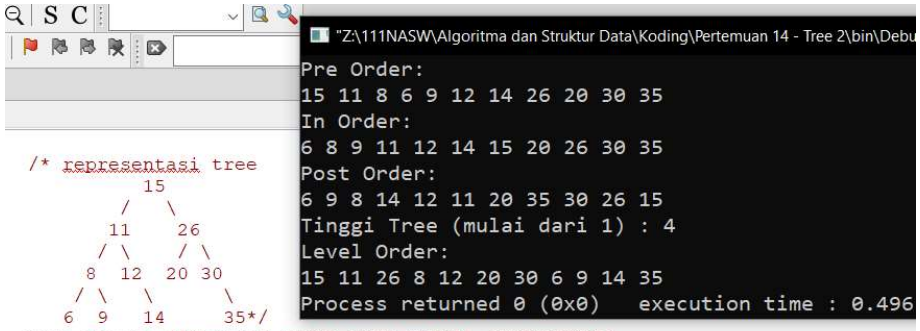
printCurrentLevel(root->right, 0)

```
void Tree::printCurrentLevel(Node *root, int level){
    if(root != NULL){
        if(level == 0){
            cout << root->data << " ";
        }else if(level > 0){
            printCurrentLevel(root->left,level-1);
            printCurrentLevel(root->right,level-1);
        }
    }
}
```

Cetak: 12 17 27



main.cpp



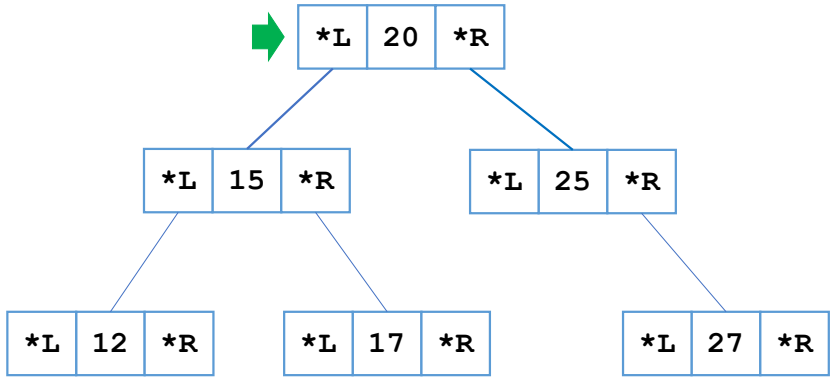

```
/* representasi tree
          15
        /   \
       11    26
      / \    / \
     8  12  20 30
    / \   \      \
   6  9   14     35*/
Node *root; /*COMMENT SEMUA TREE MANUAL SEBEUMNYA*/
Tree T;
root = NULL;
root = T.insertBinaryRoot(root,15);
root = T.insertBinaryRoot(root,11); root = T.insertBinaryRoot(root,26);
root = T.insertBinaryRoot(root,8); root = T.insertBinaryRoot(root,12);
root = T.insertBinaryRoot(root,20); root = T.insertBinaryRoot(root,30);
root = T.insertBinaryRoot(root,6); root = T.insertBinaryRoot(root,9);
root = T.insertBinaryRoot(root,14); root = T.insertBinaryRoot(root,35);

cout << "Pre Order: " << endl; T.preOrder(root);
cout << "\nIn Order: " << endl; T.inOrder(root);
cout << "\nPost Order: " << endl; T.postOrder(root);

int tinggi;
tinggi = T.heightNode(root);
cout <<"\nTinggi Tree (mulai dari 1) : " << tinggi;

cout << "\nLevel Order: " << endl;
T.levelOrder(root);

return 0;
```

levelOrder

levelOrder(root)

```
void Tree::levelOrder(Node *root){
    int h = heightNode(root);
    for(int i=0;i<h;i++){ //jika i=1; i<=h
        printCurrentLevel(root,i);
    }
}
```

h = 3
i=0; 0<3
    printCurrentLevel(root,0) #Cetak : 20
i=1; 1<3
    printCurrentLevel(root,1) #Cetak : 20 15 25
i=2; 2<3
    printCurrentLevel(root,2) #Cetak : 20 15 25 12 17 27
i=3; 3<3 loop end



Referensi

Utama:

1. Introduction to Algorithms 4th edition, Thomas H. Cormen, Charles E. Leiserson, Ronald E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, 2022
2. Introduction to the Design and Analysis of Algorithms 3rd Edition, Anany Levitin, Pearson, 2011
3. Data Structures and Algorithms in C++, Michael T. Goodrich, Roberto Tamasia, David M. Mount, John Wiley & Sons, 2011

Pendukung:

1. Data Structures and Algorithms in C++ 4th Edition, Adam Drozdek, Cengage Learning, 2013

TERIMA KASIH

ANY QUESTIONS?