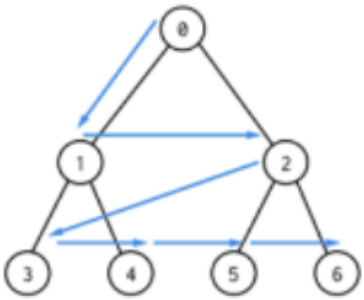
Nama : Yanuar Nurul Hilal
NIM : 222112418
Kelas : 2KS4

Penugasan Struktur Data Praktikum 9

1. Simpan ulang **Praktikum9B.c** dengan nama **Praktikum9B.c**, lalu lakukan modifikasi pada fungsi insert dan delete seperti yang kita lakukan pada kegiatan praktikum di atas.
2. Kemudian, tambahkan sebuah fungsi untuk menampilkan nama-nama mahasiswa yang ada pada tree dengan alur:




```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node{
    char data[30];
    int height;
    struct node *left;
    struct node *right;
};

typedef struct node *ptr;

ptr newNode(char data[]){
    ptr node = (ptr)malloc(sizeof(struct node));
    strcpy(node->data, data);
    node->height = 1;
    node->left = NULL;
    node->right = NULL;
    return node;
}
```

```c
void displayPreorder(ptr node){
    if (node == NULL)
        return;
    printf("%s ", node->data); // root
    displayPreorder(node->left); // subtree kiri
    displayPreorder(node->right); // subtree kanan
}

void displayInorder(ptr node){
    if (node == NULL)
        return;
    displayInorder(node->left); // subtree kiri
    printf("%s ", node->data); // root
    displayInorder(node->right); // subtree kanan
}

void displayPostorder(ptr node){
    if (node == NULL)
        return;
    displayPostorder(node->left); // subtree kiri
    displayPostorder(node->right); // subtree kanan
    printf("%s ", node->data); // root
}

int max(int a, int b){
    if (a > b)
        return a;
    else
        return b;
}

int getHeight(ptr N){
    if (N == NULL)
        return 0;
    return N->height;
}

// Hitung Balance factor untuk node N
int getBalanceFactor(ptr N){
    if (N == NULL)
        return 0;
    return getHeight(N->left) - getHeight(N->right);
```

```
}

ptr rightRotate(ptr T){
    ptr new_root = T->left;
    ptr orphan = new_root->right;
    // Lakukan rotasi
    new_root->right = T;
    T->left = orphan;
    // Update height
    T->height = max(getHeight(T->left), getHeight(T->right)) + 1;
    new_root->height = max(getHeight(new_root->left),
getHeight(new_root->right))
    + 1;
    // Return root baru
    return new_root;
}

ptr leftRotate(ptr T){
    ptr new_root = T->right;
    ptr orphan = new_root->left;
    // Lakukan rotasi
    new_root->left = T;
    T->right = orphan;
    // Update height
    T->height = max(getHeight(T->left), getHeight(T->right)) + 1;
    new_root->height = max(getHeight(new_root->left),
getHeight(new_root->right))
    + 1;
    // Return root baru return
    return new_root;
}

ptr insert(ptr root, char new_nama[]){
    // 1. Lakukan BST insert biasa, jika tree kosong maka data baru menjadi
root
    if (root == NULL)
        return (newNode(new_nama));
    // asumsi tidak boleh temp nilai yang sama dalam BST
    if (strcmp(new_nama, root->data) < 0)
        root->left = insert(root->left, new_nama);
    else if (strcmp(new_nama, root->data) > 0)
        root->right = insert(root->right, new_nama);
    // 2. Update height dari node baru sampai root
```

```
    root->height = 1 + max(getHeight(root->left), getHeight(root->right));
    // 3. Hitung balance factor untuk menentukan apakah node unbalanced
    int balance = getBalanceFactor(root);
    // Jika tidak balanced, return hasil rotation
    // Kasus 1: Left Left
    if (balance > 1 && strcmp(new_nama, root->left->data) < 0)
        return rightRotate(root);
    // Kasus 2: Right Right
    if (balance < -1 && strcmp(new_nama, root->right->data) > 0)
        return leftRotate(root);
    // Kasus 3: Right Left
    if (balance < -1 && strcmp(new_nama, root->right->data) < 0)
        {
        root->right = rightRotate(root->right);
        return leftRotate(root);
        }
    // Kasus 4: Left Right
    if (balance > 1 && strcmp(new_nama, root->left->data) > 0)
        {
        root->left = leftRotate(root->left);
        return rightRotate(root);
        } // return node jika balanced
    return root;
}

void search_node(ptr root, char data[]){
    int temp = 1;
    ptr cursor = root;
    while (strcmp(cursor->data, data) != 0)
    {
        if (cursor != NULL)
        {
            if (strcmp(data, cursor->data) > 0)
            {
                cursor = cursor->right;
            }
            else
            {
                cursor = cursor->left;
            }
            if (cursor == NULL)
            {
                printf("\nMahasiswa %s tidak ditemukan\n", data);
```

```
                temp = 0;
                break;
            }
        }
    }
    if (temp == 1)
    {
    printf("\nMahasiswa %s ditemukan", data);
    }
}

ptr delete_node(ptr root, char deleted[])
{
    if (root == NULL)
        return NULL;
    ptr cursor;
    if (strcmp(deleted, root->data) > 0)
        root->right = delete_node(root->right, deleted);
    else if (strcmp(deleted, root->data) < 0)
        root->left = delete_node(root->left, deleted);
    else
    {
        // 1 CHILD
        if (root->left == NULL)
        {
            cursor = root->right;
            free(root);
            root = cursor;
        }
        else if (root->right == NULL)
        {
            cursor = root->left;
            free(root);
            root = cursor;
        } // 2 CHILDS NODE
        else
        {
            cursor = root->right;
            while (cursor->left != NULL)
            {
                cursor = cursor->left;
            }
            strcpy(root->data, cursor->data);
```

```
            root->right = delete_node(root->right, cursor->data);
        }
    }
    // Jika setelah dilakukan delete, tree kosong maka return root
        if (root == NULL)
    return root;
    // 2. Update height dari node
    root->height = 1 + max(getHeight(root->left), getHeight(root->right));
    // 3. Hitung balance factor untuk menentukan apakah root unbalanced
    int balance = getBalanceFactor(root);
    // Jika tidak balanced, return hasil rotation
    // Kasus 1: Left Left
    if (balance > 1 && getBalanceFactor(root->left) >= 0)
        return rightRotate(root);
    // Kasus 2: Right Right
    if (balance < -1 && getBalanceFactor(root->right) <= 0)
        return leftRotate(root);
    // Kasus 3: Right Left
    if (balance < -1 && getBalanceFactor(root->right) > 0)
    {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    // Kasus 4: Left Right
    if (balance > 1 && getBalanceFactor(root->left) < 0)
    {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    return root;
}

void view(ptr root, int level)
{
    // Jika root kosong maka return tanpa nilai
    if (root == NULL)
    {
        return;
    }
    // Jika level 1 maka akan menampilkan root
    if (level == 1)
    {
        printf("%s ", root->data);
```

```
    }
    // Menampilkan node dimulai dari sisi kiri ke kanan
    // dengan rekursi
    else if (level > 1)
    {
        view(root->left, level - 1);
        view(root->right, level - 1);
    }
}

void print_bylevel(ptr root)
{
    // Menghitung level untuk setiap node
    int level = 1 + max(getHeight(root->left), getHeight(root->right));
    // Menampilkan setiap node di setiap levelnya dengan memanggil
    // void view
    for (int i = 1; i <= level; i++)
    {
        view(root, i);
        printf("\n");
    }
}

int main(int argc, char const *argv[])
{
    system("cls");
    ptr root = NULL;
    root = insert(root, "Stegen");
    root = insert(root, "Kounde");
    root = insert(root, "Araujo");
    root = insert(root, "Balde");
    root = insert(root, "Pedri");
    root = insert(root, "Gavi");
    root = insert(root, "Ansu");
    root = insert(root, "Dembele");
    root = insert(root, "Lewy");
    // display secara Preorder, Inorder, dan Postorder
    printf("\nPreorder : ");
    displayPreorder(root);
    printf("\nInorder : ");
    displayInorder(root);
    printf("\nPostorder : ");
    displayPostorder(root);
```

```
    printf("\n\n===Tampilan Data Pada Setiap Level===\n");
    print_bylevel(root);
    root = delete_node(root, "Dembele");
    root = delete_node(root, "Lewy");
    printf("\n===AVL Tree Setelah Delete Dembele & Lewy===\n");
    printf("\nPreorder : ");
    displayPreorder(root);
    printf("\n\nInorder : ");
    displayInorder(root);
    printf("\n\nPostorder : ");
    displayPostorder(root);
    printf("\n\n===Cari Node Pedri & Ferran dalam AVL Tree===\n");
    search_node(root, "Pedri");
    search_node(root, "Ferran");
    printf("\n");
}
```

Output :

```
Preorder : Kounde Balde Araujo Ansu Gavi Dembele Pedri Lewy Stegen
Inorder : Ansu Araujo Balde Dembele Gavi Kounde Lewy Pedri Stegen
Postorder : Ansu Araujo Dembele Gavi Balde Lewy Stegen Pedri Kounde

===Tampilan Data Pada Setiap Level===
Kounde
Balde Pedri
Araujo Gavi Lewy Stegen
Ansu Dembele

===AVL Tree Setelah Delete Dembele & Lewy===

Preorder : Kounde Balde Araujo Ansu Gavi Pedri Stegen

Inorder : Ansu Araujo Balde Gavi Kounde Pedri Stegen

Postorder : Ansu Araujo Gavi Balde Stegen Pedri Kounde

===Cari Node Pedri & Ferran dalam AVL Tree===

Mahasiswa Pedri ditemukan
Mahasiswa Ferran tidak ditemukan
```