MODUL X

TREE LANJUTAN

A. TUJUAN

- Mahasiswa mampu menjelaskan mengenai algoritma Tree
- Mahasiswa mampu membuat dan mendeklarasikan struktur algoritma Tree
- Mahasiswa mampu menerapkan dan mengimplementasikan algoritma Tree

B. DASAR TEORI

Tree traversal adalah cara kunjungan node-node pada pohon biner. Ada tiga cara kunjungan dalam tree:

- Pre-order
- In-order
- Post-order

1. Pre-order
   a. Cetak data pada root
   b. Secara rekursif mencetak seluruh data pada subpohon kiri
   c. Secara rekursif mencetak seluruh data pada subpohon kanan

```
void preOrder(Node *root){
    if(root != NULL){
        printf("%d ",root->data);
        preOrder(root->kiri);
        preOrder(root->kanan);
    }
}
```

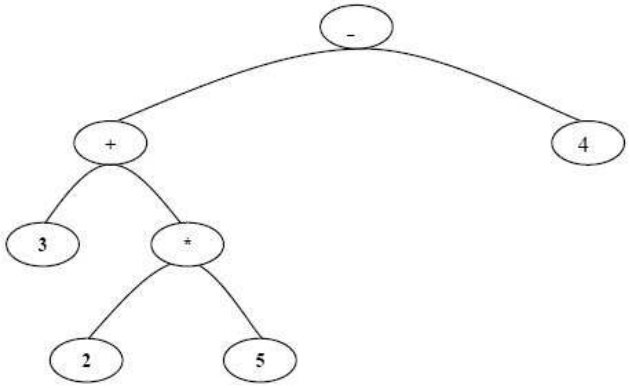
2. In-order
   a. Secara rekursif mencetak seluruh data pada subpohon kiri
   b. Cetak data pada root
   c. Secara rekursif mencetak seluruh data pada subpohon kanan

```
void inOrder(Node *root){
    if(root != NULL){
        inOrder(root->kiri);
        printf("%d ",root->data);
        inOrder(root->kanan);
    }
}
```

3. Post-order
   a. Secara rekursif mencetak seluruh data pada subpohon kiri
   b. Secara rekursif mencetak seluruh data pada subpohon kanan
   c. Cetak data pada root

```
void postOrder(Node *root){
    if(root != NULL){
        postOrder(root->kiri);
        postOrder(root->kanan);
        printf("%d ",root->data);
    }
g}
```

Contoh notasi matematika, misalkan suatu ekspresi berikut: 3 + 2 * 5 – 4



Prefiks: - + 3 * 2 5 4

Infiks: 3 + 2 * 5 - 4

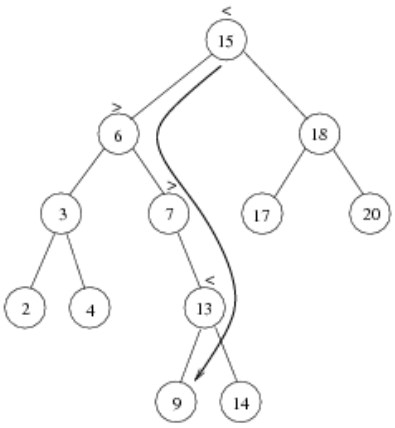
Postfiks: 3 2 5 * + 4 -

Sintaks progran pencarian data di Tree

```
void search(Node **root, int cari)
{
    if((*root) == NULL){
        printf("Data tidak ditemukan!");
    }
    else if(cari < (*root)->data)
        search(&(*root)->kiri,cari);
    else if(cari > (*root)->data)
        search(&(*root)->kanan,cari);
    else if(cari == (*root)->data)
        printf("Data ditemukan!");

}
```

Example: Search for 9 ...



1. Bandingkan 9 dengan 15; mengujungi kiri
2. Bandingkan 9 dengan 6; mengunjungi kanan
3. Bandingkan 9 dengan 7; mengunjungi kanan
4. Bandingkan 9 dengan 13; mengunjungi kiri
5. Bandingkan 9 dengan 9; data ditemukan

Penghitungan jumlah node dalam tree

```
int count(Tree *root)
{
if(root == NULL)
return 0;
return count(root->left) + count(root->right) + 1;
}
```

Penghitungan kedalaman tree

```
int height(Tree *root)
{
if (root == NULL)
return -1;
int u = height(root->left),
int v = height(root->right);
if (u > v)
return u+1;
else
return v+1;
}
```

Terdapat 3 kasus pada penghapusan :

(1) Node yang dihapus adalah daun

- Langsung dihapus

(2) Node yang dihapus hanya punya 1 anak

- Arahkan pointer (buat cabang baru) dari induk node langsung ke anaknya

(3) Node yang dihapus punya 2 anak

- Ganti nilai dari node yang dihapus dengan nilai terkecil dari subpohon kanannya
- Delete element terkecil

AVL Tree adalah pohon yang setiap node nya memiliki AVL property.

AVL Property :

- Sebuah node memiliki AVL property jika *height* (tinggi) subpohon kiri & subpohon kanan node tersebut sama atau berbeda 1.
- *Height* (tinggi) pohon adalah jarak dari root menuju daun terbawah yang dimiliki pohon tersebut

C. LATIHAN

```
#include <stdio.h>
#include <conio.h>
typedef struct Node{
    int data;
    Node *kiri;
    Node *kanan;
};
int count;
void tambah(Node **root, int databaru)
{
    if((*root) == NULL){
        Node *baru;
        baru = new Node;
        baru->data = databaru;
        baru->kiri = NULL;
        baru->kanan = NULL;
        (*root) = baru;
        (*root)->kiri = NULL;
        (*root)->kanan = NULL;
        printf("Data bertambah!");
        count++;
    }
    else if(databaru < (*root)->data)
        tambah(&(*root)->kiri,databaru);
    else if(databaru > (*root)->data)
        tambah(&(*root)->kanan,databaru);
    else if(databaru == (*root)->data)
        printf("Data sudah ada!");
}
void preOrder(Node *root){
    if(root != NULL){
        printf("%d ",root->data);
        preOrder(root->kiri);
        preOrder(root->kanan);
    }
}
void inOrder(Node *root){
    if(root != NULL){
        inOrder(root->kiri);
        printf("%d ",root->data);
        inOrder(root->kanan);
    }
}
void postOrder(Node *root){
    if(root != NULL){
        postOrder(root->kiri);
        postOrder(root->kanan);
        printf("%d ",root->data);
    }
}
```

```
void search(Node **root, int cari)
{
    if((*root) == NULL){
        printf("Data tidak ditemukan!");
    }
    else if(cari < (*root)->data)
        search(&(*root)->kiri,cari);
    else if(cari > (*root)->data)
        search(&(*root)->kanan,cari);
    else if(cari == (*root)->data)
        printf("Data ditemukan!");

}
void hapus(Node **root, int del)
{
    if((*root) == NULL){
        printf("Data tidak ada!");
    }
    else if(del < (*root)->data)
        hapus(&(*root)->kiri,del);
    else if(del > (*root)->data)
        hapus(&(*root)->kanan,del);
    else if(del == (*root)->data)
    {
        (*root)=NULL;
        printf("Data dihapus!");
    }
}
main(){
    int pil,c,cari,del;
    Node *pohon,*t;
    pohon = NULL;
    do{
        int data;
        printf("MENU\n");
        printf("1. Tambah\n");
        printf("2. Lihat pre-order\n");
        printf("3. Lihat in-order\n");
        printf("4. Lihat post-order\n");
        printf("5. Search\n");
        printf("6. Kosongkan Tree\n");
        printf("7. Hapus data\n");
        printf("8. Exit\n");
        printf("Pilihan : ");
        scanf("%d",&pil);
        switch(pil){
            case 1:     printf("Data baru : ");
                scanf("%d", &data);
                tambah(&pohon,data);
                break;
            case 2:     if(pohon!=NULL) preOrder(pohon);
                else printf("Masih kosong!");
                break;
            case 3:     if(pohon!=NULL) inOrder(pohon);
                else printf("Masih kosong!");
                break;
            case 4:     if(pohon!=NULL) postOrder(pohon);
                else printf("Masih kosong!");
                break;
            case 5:     printf("Cari data : ");
                scanf("%d", &cari);
                search(&pohon,cari);
                break;
            case 6:
```

```
                pohon = NULL;
                printf("Node dihapus semua");
                break;
            case 7:     printf("Hapus data : ");
                scanf("%d", &del);
                hapus(&pohon,del);
                break;
            }
        getch();
    }while(pil!=8);
}
```

Rubah Program tree diatas menjadi :
-inputan data dalam bentuk karakter
-fungsi delete memperhatikan 3 kondisi
-masukkan fungsi perhitungan jumlah node dan perhitungan ke dalam tree

D. TUGAS

Buatlah program AVL tree dengan tampilan minimal:

1. Insert
2. Delete
3. Tranverse
4. Exit