

MODUL 9: TREE BAGIAN 1

1.1. Deskripsi Singkat

Tree atau pohon merupakan struktur data yang tidak linear yang digunakan untuk mempresentasikan data yang bersifat hirarki antara elemen-elemennya. Definisi tree yaitu sekumpulan simpul (node) yang saling dihubungkan oleh sebuah vektor (edge) yang tidak membentuk sirkuit. Terdapat banyak jenis tree. Salah satunya yang paling penting adalah pohon biner (*binary tree*) karena banyak aplikasinya. Pada praktikum ini kita akan mempelajari Binary Tree dan Binary Search Tree.

1.2. Tujuan Praktikum

- 1) Mahasiswa mampu mengimplementasikan binary tree dengan menggunakan bahasa C
- 2) Mahasiswa mampu mengimplementasikan binary search tree dengan menggunakan bahasa C

1.3. Material Praktikum

Kegiatan pada modul ini memerlukan material berupa software editor dan compiler (atau IDE) untuk bahasa pemrograman C.

1.4. Kegiatan Praktikum

A. Binary Tree

Binary Tree adalah tree yang pada masing-masing simpulnya hanya dapat memiliki maksimum 2 (dua) node child, tidak boleh lebih. Terdapat beberapa operasi yang dapat dilakukan pada binary tree, diantaranya adalah penambahan simpul baru dan penelusuran atau traversal pohon. Kita akan mencoba membuat program yang menjalankan kedua operasi tersebut.

1. Simpul-simpul pada tree, dapat dibuat menggunakan structure. Structure tersebut dapat menyimpan data tertentu, serta menyimpan informasi lokasi anak kanan dan anak kiri. Contohnya seperti ini.

```
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
```

2. Untuk membuat simpul baru, yang kita lakukan adalah mengalokasikan memori untuk structure simpul di atas dan melakukan assignment nilai pada `data`, `*left`, `*right`, dengan cara seperti ini.

```
struct node *newNode(int data)
{
    struct node *node = (struct node*)malloc(sizeof(struct
node));

    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return node;
}
```

3. Untuk mencoba membuat simpul baru, panggil fungsi **`newNode`** di atas pada fungsi **`main`**. Contohnya adalah sebagai berikut.

```
int main()
{

    struct node* root = newNode(8);

    root->left = newNode(3);
    root->left->right = newNode(1);
    root->left->left = newNode(6);
    root->left->right->right = newNode(4);
    root->left->right->left = newNode(7);

    root->right = newNode(10);
    root->right->left = newNode(14);
    root->right->left->right = newNode(13);

    return 0;
}
```

Simpan potongan program pada ketiga tahapan di atas dengan nama **Praktikum9A.c**. Pastikan tidak ada error yang muncul.

4. Untuk menampilkan isi dari pohon yang telah kita buat, kita dapat menelusuri satu-persatu simpul yang terdapat pada pohon (traversal). Terdapat tiga cara untuk melintasi sebuah tree, yaitu preorder, inorder, dan postorder. Ketik ulang dan pahami cara kerja potongan program untuk menelusuri pohon di bawah ini.

```
void displayPreorder(struct node* node)
{
    if(node == NULL)
        return;

    printf("%d ", node->data); //root
    displayPreorder(node->left); //subtree kiri
    displayPreorder(node->right); //subtree kanan
}

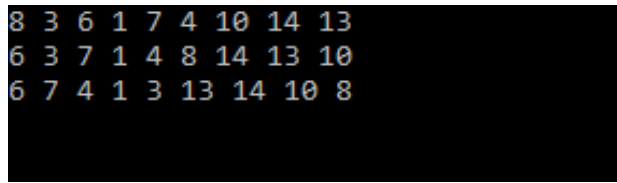
void displayInorder(struct node* node)
{
    if(node == NULL)
        return;

    displayInorder(node->left); //subtree kiri
    printf("%d ", node->data); //root
    displayInorder(node->right); //subtree kanan
}

void displayPostorder(struct node* node)
{
    if(node == NULL)
        return;

    displayPostorder(node->left); //subtree kiri
    displayPostorder(node->right); //subtree kanan
    printf("%d ", node->data); //root
}
```

Tambahkan potongan program di atas pada **Praktikum9A.c**, lalu panggil fungsi-fungsi traversal tersebut melalui fungsi main. Jika berhasil, maka data pada seluruh node yang ada pada tree akan muncul di layar seperti pada gambar berikut:



```
8 3 6 1 7 4 10 14 13
6 3 7 1 4 8 14 13 10
6 7 4 1 3 13 14 10 8
```

B. Binary Search Tree

Kekurangan dari Binary Tree adalah data pada node-node yang ada tidak teratur. Sehingga akan sulit bila ingin melakukan pencarian/search node tertentu dalam binary tree karena bisa jadi program harus mengunjungi seluruh node pada tree untuk dapat menemukan node yang dicari. Binary Search Tree adalah Binary Tree yang isi nodenya sudah teratur. Binary search tree dibuat untuk mengatasi kelemahan pada binary tree. Binary Search Tree memiliki sifat dimana semua left child harus lebih kecil dari pada right child dan parentnya. Pada dasarnya operasi dalam Binary Search Tree sama dengan Binary Tree biasa, hanya saja saat melakukan insert, update, dan delete, kita harus memastikan urutan node pada tree tetap terjaga. Pada praktikum ini kita akan mencoba memodifikasi fungsi insert pada **Praktikum9A.c** agar pohon yang terbentuk adalah Binary Search Tree.

5. Tambahkan fungsi insert berikut pada **Praktikum9A.c**. Setiap kali kita akan menambahkan simpul baru pada pohon, program akan mencari posisi yang tepat untuk simpul baru tersebut. Program akan membandingkan data pada simpul baru dengan data pada simpul-simpul yang sudah ada pada pohon.

```
struct node* insert(struct node* root, int newData)
{
    if(root == NULL)
    {
        root = newNode(newData);
    }
    else
    {
        int is_left = 0;
        struct node* cursor = root;
        struct node* prev = NULL;
        while(cursor != NULL)
        {
```

```

        prev = cursor;
        if(newData < cursor->data)
        {
            is_left = 1;
            cursor = cursor->left;
        }
        else if(newData > cursor->data)
        {
            is_left = 0;
            cursor = cursor->right;
        }
    }

    if(is_left == 1)
        prev->left = newNode(newData);
    else
        prev->right = newNode(newData);
}

return root;
}

```

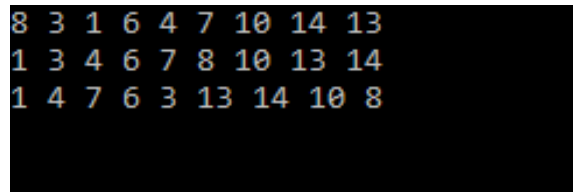
6. Modifikasi potongan program pada fungsi main. Sebelumnya kita menambahkan simpul baru secara langsung pada posisi yang kita inginkan. Sekarang, simpul-simpul baru ditambahkan dengan menggunakan fungsi **insert**.

```

struct node* root = newNode(8);
root = insert(root, 3);
root = insert(root, 1);
root = insert(root, 6);
root = insert(root, 4);
root = insert(root, 7);
root = insert(root, 10);
root = insert(root, 14);
root = insert(root, 13);

```

Setelah itu, coba jalankan program **Praktikum9A.c**. Jika berhasil, maka output yang ditampilkan di layar seperti gambar berikut:



```
8 3 1 6 4 7 10 14 13
1 3 4 6 7 8 10 13 14
1 4 7 6 3 13 14 10 8
```

7. Sekarang setelah node-node telah terurut, maka akan lebih mudah untuk melakukan pencarian sebuah node. Tambahkan potongan program untuk pencarian node berikut pada **Praktikum9A.c**. Ketik ulang sambil dipahami cara kerjanya.

```
void search_node(struct node* root, int data)
{
    struct node* cursor = root;

    while(cursor->data != data){
        if(cursor != NULL){
            if(data > cursor->data){
                cursor = cursor->right;
            }
            else {
                cursor = cursor->left;
            }

            if(cursor == NULL){
                printf("\nNode %d tidak ditemukan", data);
            }
        }
    }

    printf("\nNode %d ditemukan", data);
    return;
}
```

Cobalah untuk melakukan pencarian beberapa node, melalui fungsi main. Contoh:

```
search_node(root, 6);
search_node(root, 100);
```

8. Jika sudah berhasil melakukan pencarian sebuah node, maka menghapus sebuah node dari pohon adalah sebuah hal yang mudah. Tambahkan potongan program berikut pada **Praktikum9A.c**.

```
struct node* delete_node(struct node* root, int
deletedData)
{
    if(root == NULL)
        return;

    struct node* cursor;
    if(deletedData > root->data)
        root->right = delete_node(root->right,deletedData);
    else if(deletedData < root->data)
        root->left = delete_node(root->left, deletedData);
    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;
            }
            root->data = cursor->data;
            root->right = delete_node(root->right, cursor-
>data);
```

```

        }
    }
}
return root;
}

```

Node yang dihapus, digantikan dengan predecessor atau successor node tersebut pada urutan inorder. Pada fungsi `delete_node` di atas, node yang dihapus digantikan oleh predecessor atau successor? Pastikan Anda telah memahami cara kerja fungsi tersebut. Kemudian cobalah untuk menghapus beberapa node dari pohon melalui fungsi `main`. Contoh:

```

root=delete_node(root, 13);
root=delete_node(root, 3);

```

1.5. Penugasan

Sekarang, Anda telah memahami cara menyimpan data menggunakan struktur data tree dan melakukan operasi sederhana pada tree yang telah dibuat. Modifikasi program pada **Praktikum9A.c** sehingga data yang disimpan pada Binary Search Tree, bukan lagi sebuah angka bertipe integer, melainkan data nama mahasiswa dengan tipe **char[30]**. Simpan hasil modifikasi Anda pada file **Praktikum9B.c**.