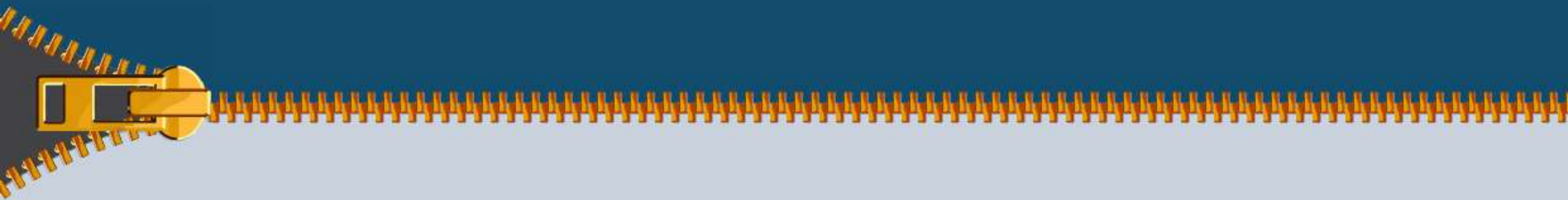


# STRUKTUR DATA

## TREE

fppt.com

# Latar Belakang

- Prioritas seorang programer ketika menyusun suatu data adalah menemukan cara agar proses pencarian, pemasukan dan penghapusan data dapat berjalan lancar.
- Array ➔ time consuming, memerlukan data movement
- Linked List ➔ Menggunakan pointer, namun data harus diproses secara sekuensial

Jadi bagaimana agar data dapat diatur secara dinamis sehingga proses dapat berlangsung efisien? ➔ **TREE**

fppt.com

# Definisi Tree

- Kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layaknya struktur sebuah pohon.
- Struktur pohon adalah suatu cara merepresentasikan suatu struktur hirarki (one-to-many) secara grafis yang mirip sebuah pohon, walaupun pohon tersebut hanya tampak sebagai kumpulan node-node dari atas ke bawah.

fppt.com

# Jenis Tree

1) Tree Statik : isi node-nodenya tetap karena bentuk pohonnya sudah ditentukan.

2) Tree Dinamik : isi nodenya berubah-ubah karena proses penambahan (insert) dan penghapusan (delete)

fppt.com

# Node Root

- Node root dalam sebuah tree adalah suatu node yang memiliki hiarki tertinggi dan dapat juga memiliki node-node anak.  Semua node dapat ditelusuri dari node root tersebut.
- Node root adalah node khusus yang tercipta pertama kalinya.
- Node-node lain di bawah node root saling terhubung satu sama lain dan disebut subtree

fppt.com

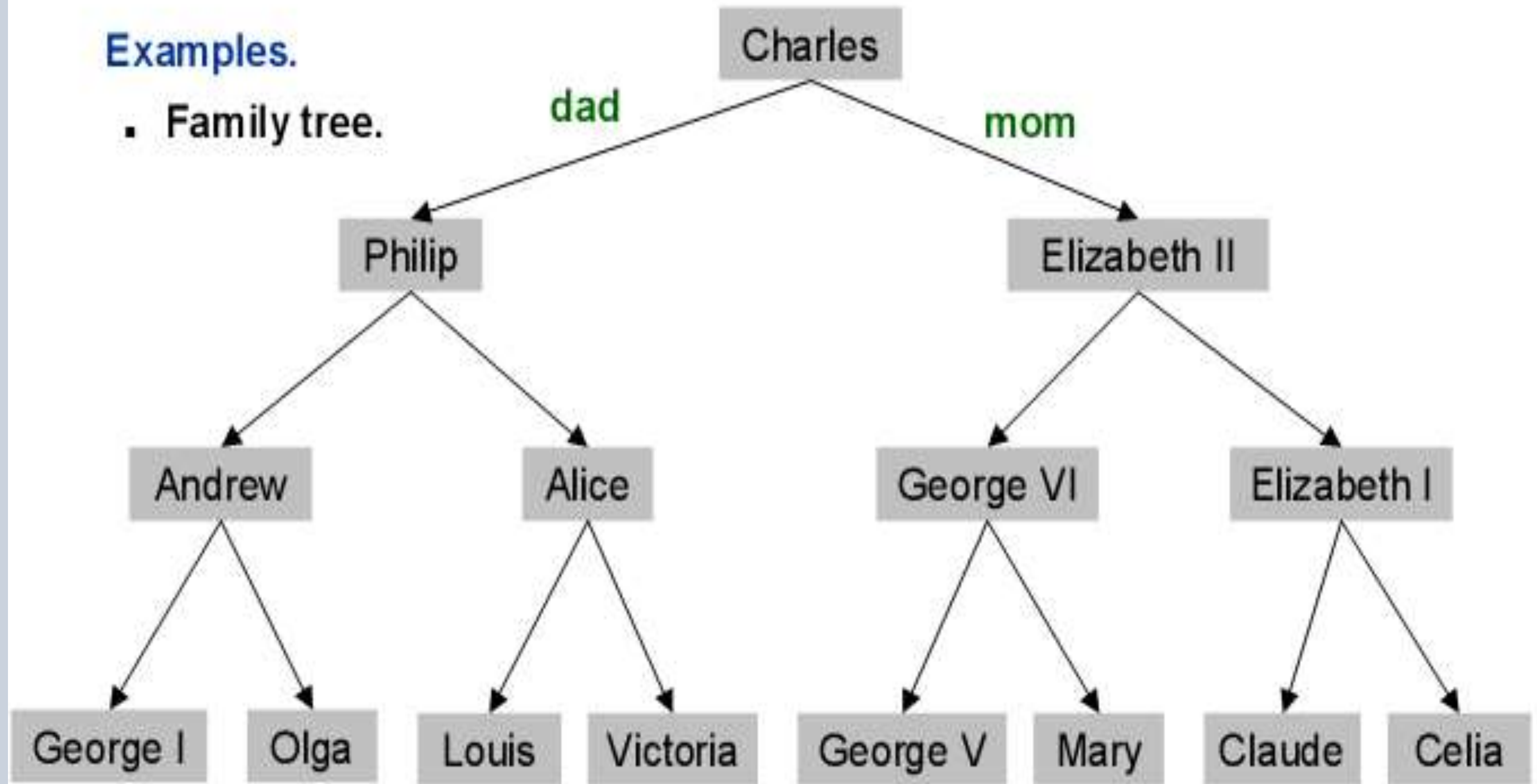
# Implementasi Tree

Contoh penggunaan struktur pohon :

- ✓ Silsilah keluarga
- ✓ Parse Tree (pada compiler)
- ✓ Struktur File
- ✓ Pertandingan

fppt.com

# Tree Example

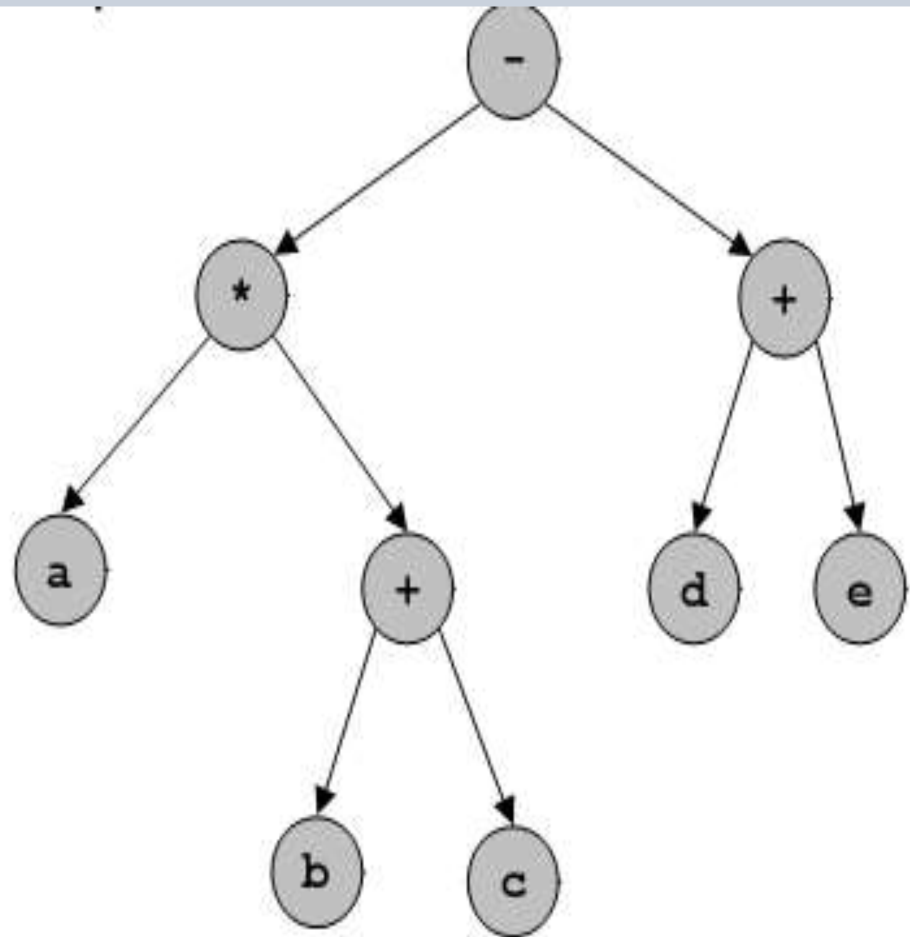


# Tree Example

Examples.

- Family tree.
- Parse tree.

  (a*(b+c))-(d+e)



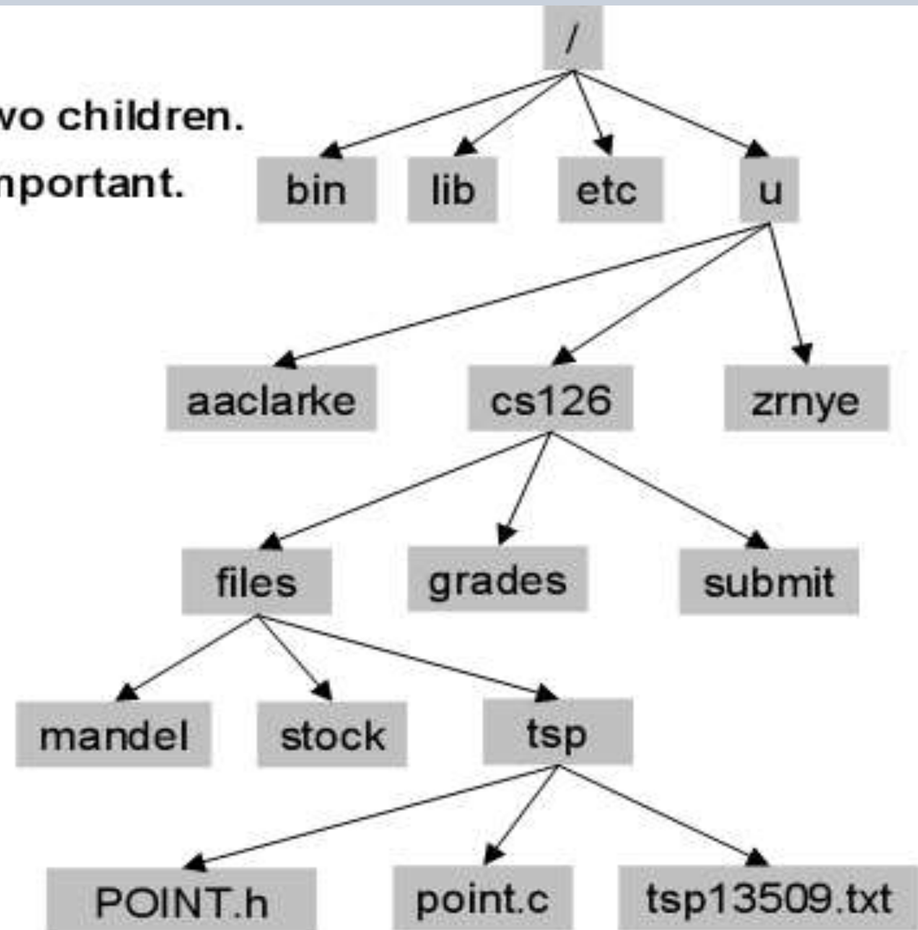
# Tree Example

**Trees.**

- Nodes need not have exactly two children.
- Order of children may not be important.

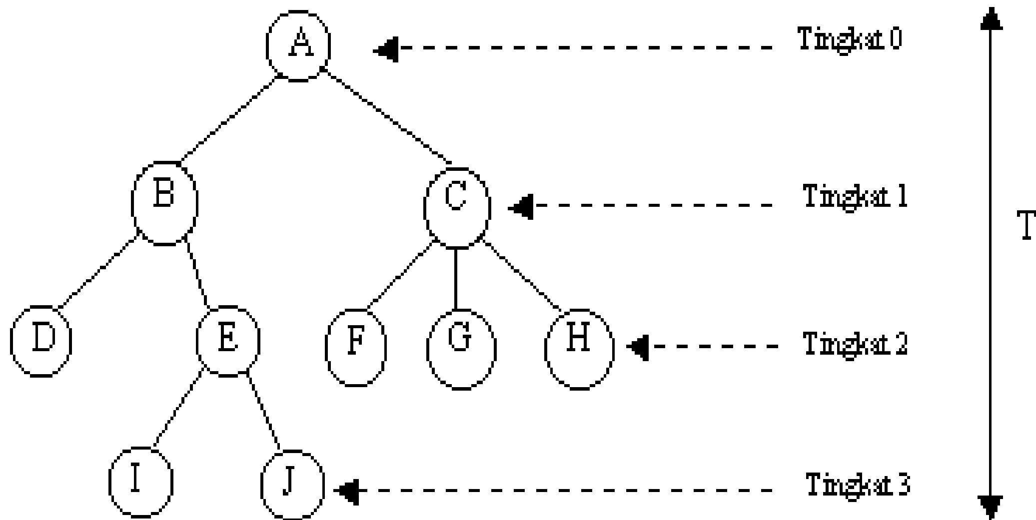
**Examples.**

- Family tree.
- Parse tree.
- Unix file hierarchy.
  - not binary

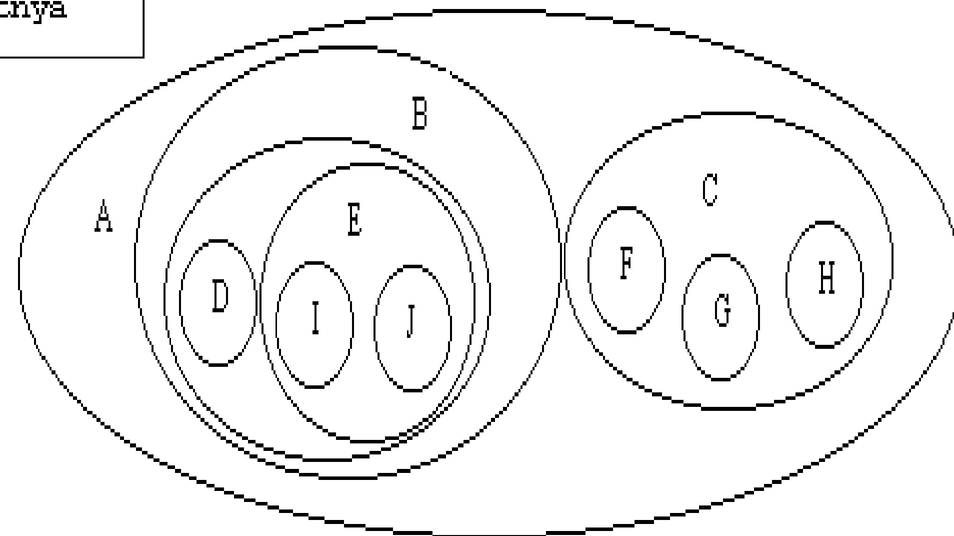



fppt.com

# Representasi Tree



Gambar 6.1. Contoh sebuah Pohon beserta Tingkatnya

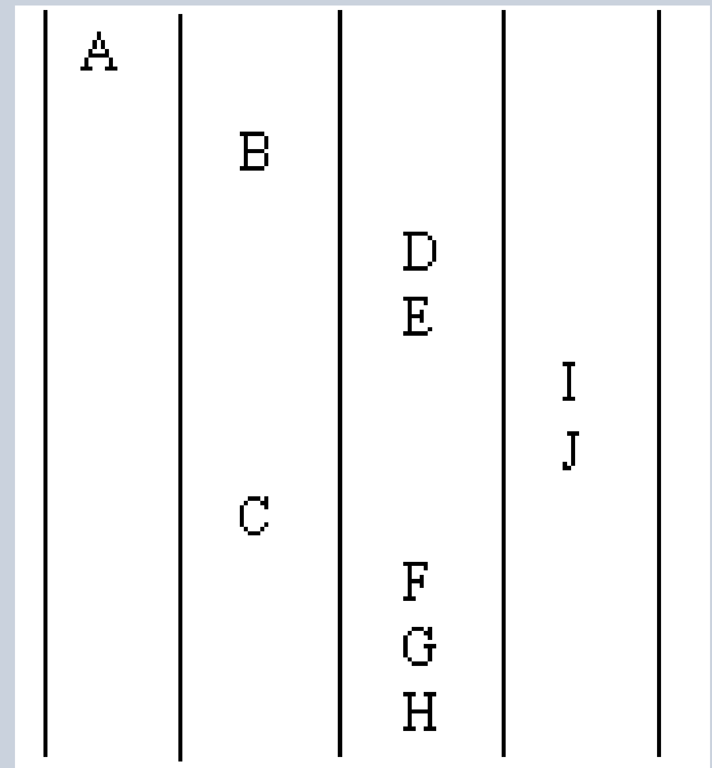


Gambar 6.2. Diagram Venn

fppt.com

# Representasi Tree

- Notasi Tingkat
- Notasi Kurung
  - (A(B(D,E(I,J)),C(F,G,H)))



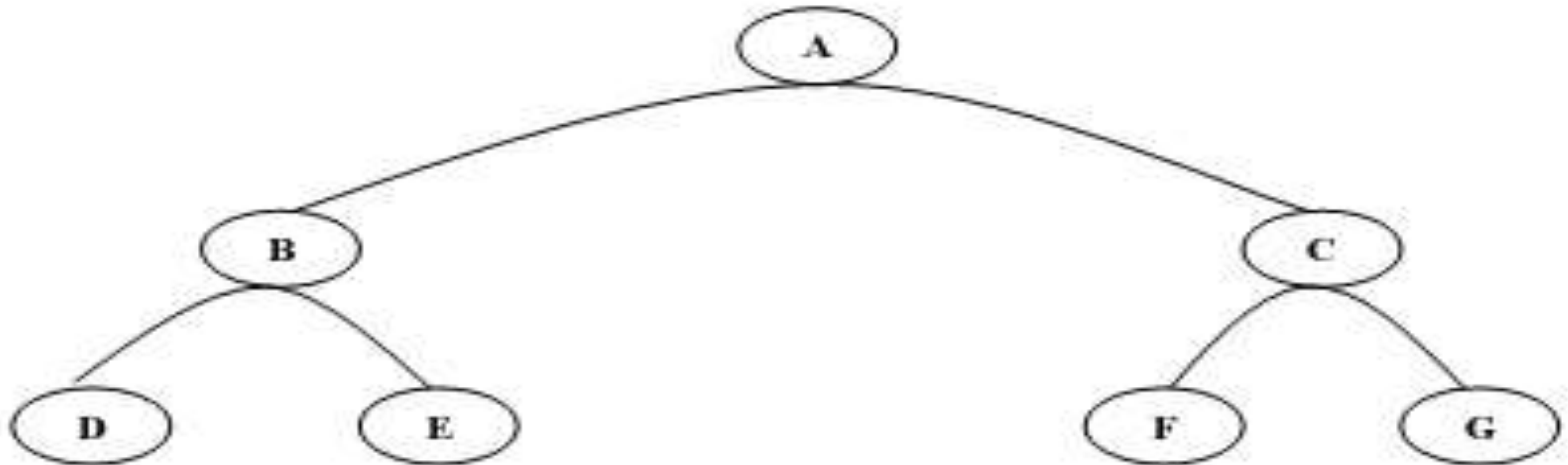

fppt.com

# Terminologi Tree

| | |
|---|---|
| Predecesor | Node yang berada diatas node tertentu. |
| Successor | Node yang berada dibawah node tertentu. |
| Ancestor | Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama |
| Descendant | Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama |
| Parent | Predecessor satu level di atas suatu node. |
| Child | Successor satu level di bawah suatu node. |
| Sibling | Node-node yang memiliki parent yang sama |
| Subtree | Suatu node beserta descendantnya. |
| Size | Banyaknya node dalam suatu tree |
| Height | Banyaknya tingkatan dalam suatu tree |
| Root | Node khusus yang tidak memiliki predecessor. |
| Leaf | Node-node dalam tree yang tidak memiliki successor. |
| Degree | Banyaknya child dalam suatu node |

fppt.com

# Terminologi Tree




Ancestor (F) = C,A

Descendant (C) = F,G

Parent (D) = B

Child (A) = B,C

Sibling (F) = G

Size = 7

fppt.com

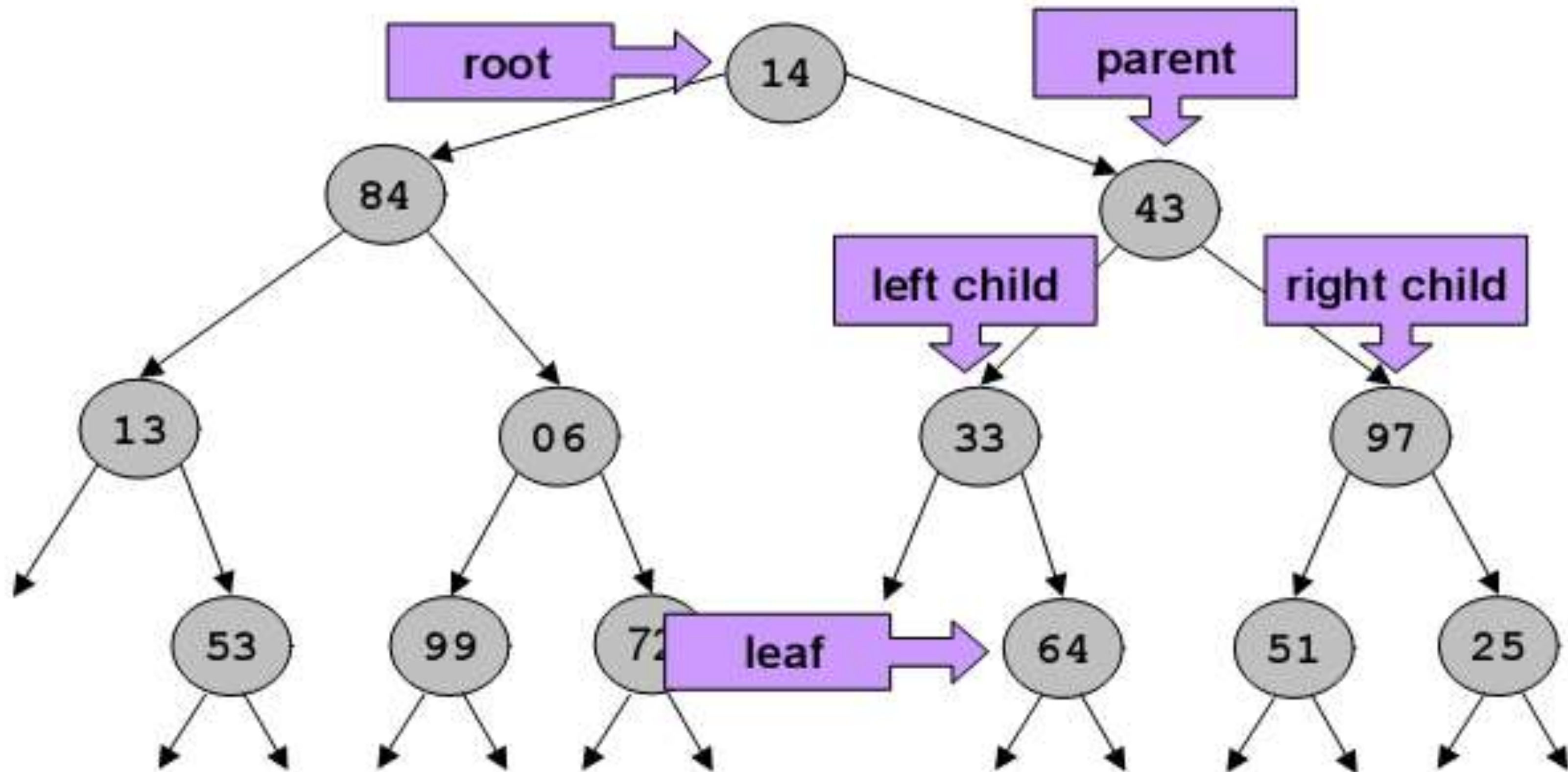
# Binary Tree

- Suatu tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal **dua subtree** dan kedua subtree tersebut harus **terpisah**.

- Tiap node dalam binary tree hanya boleh memiliki **paling banyak** dua child.

fppt.com

# Ilustrasi Binary Tree




fppt.com

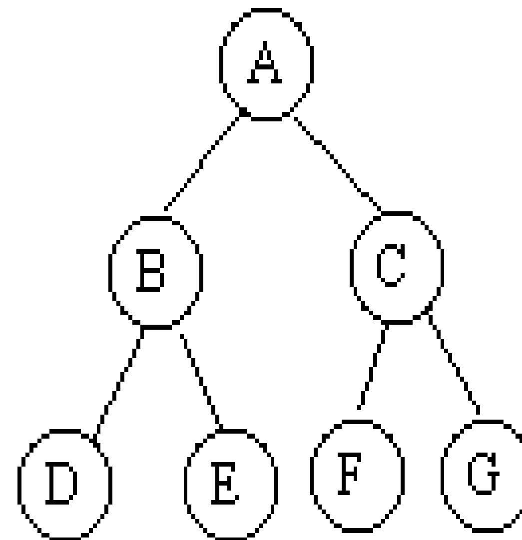
# Node pada binary tree

- Jumlah maksimum node pada setiap tingkat adalah $2^n$
- Node pada binary tree maksimum berjumlah $2^n$-1

Tingkat ke-0, jumlah max=$2^0$ --→

Tingkat ke-1, jumlah max=$2^1$ --->

Tingkat ke-2, jumlah max=$2^2$ -→




Gambar 6.4. Pohon Biner Tingkat 2 Lengkap

fppt.com

# Implementasi Program

- Tree dapat dibuat dengan menggunakan linked list secara rekursif.
- Linked list yang digunakan adalah <u>double linked list non circular</u>
- Data yang pertama kali masuk akan menjadi node root.
- Data yang lebih kecil dari data node root akan masuk dan menempati node kiri dari node root, sedangkan jika lebih besar dari data node root, akan masuk dan menempati node di sebelah kanan node root.

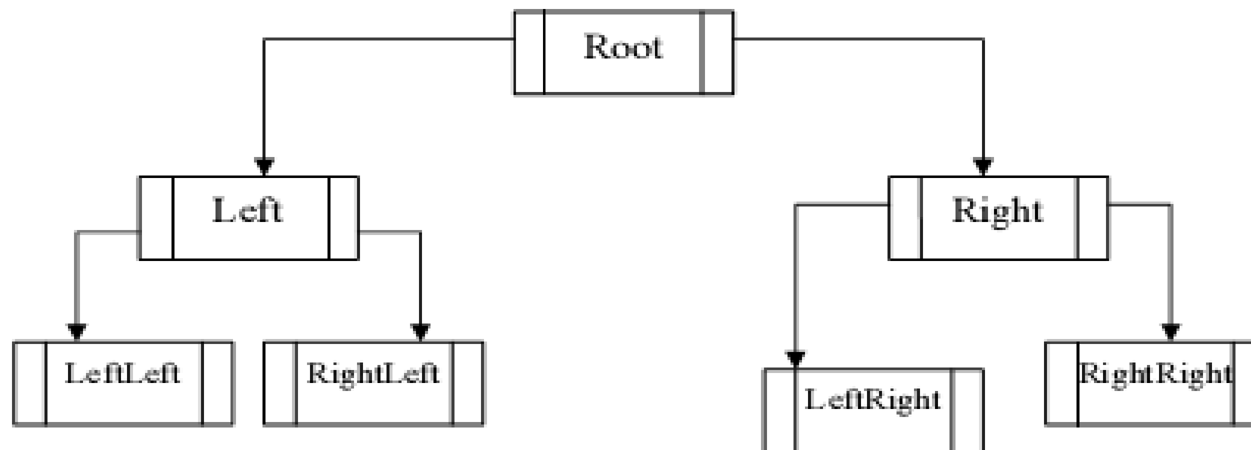
fppt.com

# Implementasi Program

**Deklarasi struct**

```
typedef struct Tree{
                    int data;
                    Tree *left;
                    Tree *right;
               }
```

Ilustrasi:




Deklarasi variabel:

```
Tree *pohon;
```

fppt.com

# Operasi-operasi Tree

- Create: membentuk sebuah tree baru yang kosong.
  - `pohon = NULL;`
- Clear: menghapus semua elemen tree.
  - `pohon = NULL;`
- Empty: mengetahui apakah tree kosong atau tidak
  - `int` **`isEmpty`** `(Tree *pohon){`
    - `if(pohon == NULL) return 1;`
    - `else return 0;`
  - `}`

fppt.com

# Operasi-operasi Tree

- **Insert**: menambah node ke dalam Tree secara rekursif.  Jika data yang akan dimasukkan lebih besar daripada elemen root, maka akan diletakkan di node sebelah kanan, sebaliknya jika lebih kecil maka akan diletakkan di node sebelah kiri.  Untuk data pertama akan menjadi elemen root.
- **Find**: mencari node di dalam Tree secara rekursif sampai node tersebut ditemukan dengan menggunakan variable bantuan ketemu.  Syaratnya adalah tree tidak boleh kosong.
- **Traverse**: yaitu operasi kunjungan terhadap node-node dalam pohon dimana masing-masing node akan dikunjungi sekali.
- **Count**: menghitung jumlah node dalam Tree
- **Height** : mengetahui kedalaman sebuah Tree
- **Find Min dan Find Max** : mencari nilai terkecil dan terbesar pada Tree
- **Child** : mengetahui anak dari sebuah node (jika punya)

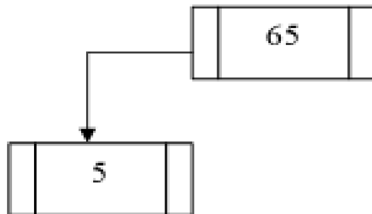
fppt.com

# Ilustrasi Insert

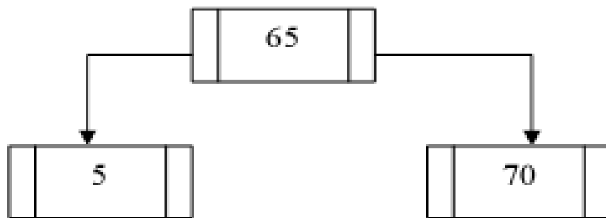
Ilustrasi operasi Insert:

1. Insert(root, 65)

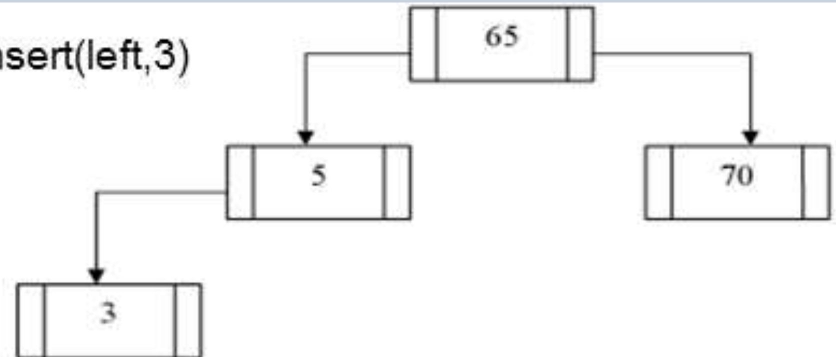
2. Insert(left, 5)



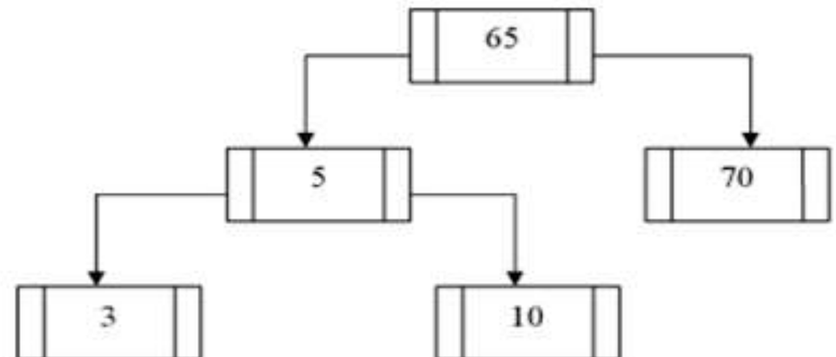

3. Insert(right, 70)




4. insert(left,3)




5. Insert(right, 10)



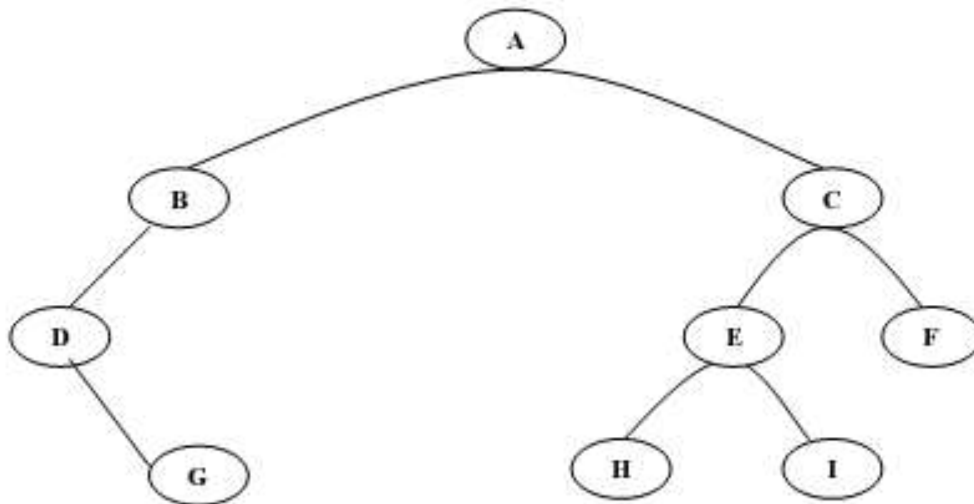
# Recursive Insert

```
void tambah(Tree **root,int databaru){
	if((*root) == NULL){
		Tree *baru;
		baru = new Tree;
		baru->data = databaru;
		baru->left = NULL;
		baru->right = NULL;
		(*root) = baru;
		(*root)->left = NULL;
		(*root)->right = NULL;
	}
	else if(databaru < (*root)->data)
		tambah(&(*root)->left,databaru);
	else if(databaru > (*root)->data)
		tambah(&(*root)->right,databaru);
	else if(databaru == (*root)->data)
		printf("Data sudah ada!");
}
```

fppt.com

# Ilustrasi Transverse

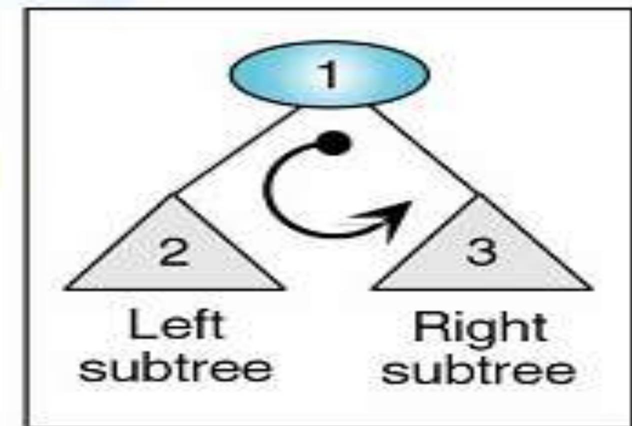
Misal terdapat Tree sebagai berikut:




1. Kunjungan PreOrder (notasi prefiks)

   Hasil kunjungan: "ABDGCEHIF"

```
void preOrder(Tree *root){
      if(root != NULL){
            printf("%d ",root->data);
            preOrder(root->left);
            preOrder(root->right);
      }
}
```




(a) Preorder traversal

fppt.c

# Searching in Tree

```
Tree *cari(Tree *root,int data){
      if(root==NULL)
return NULL;
      else if(data < root->data)
return (cari(root->left,data));
      else if(data > root->data)
return (cari(root->right,data));
      else if(data == root->data)
return root;
}
```

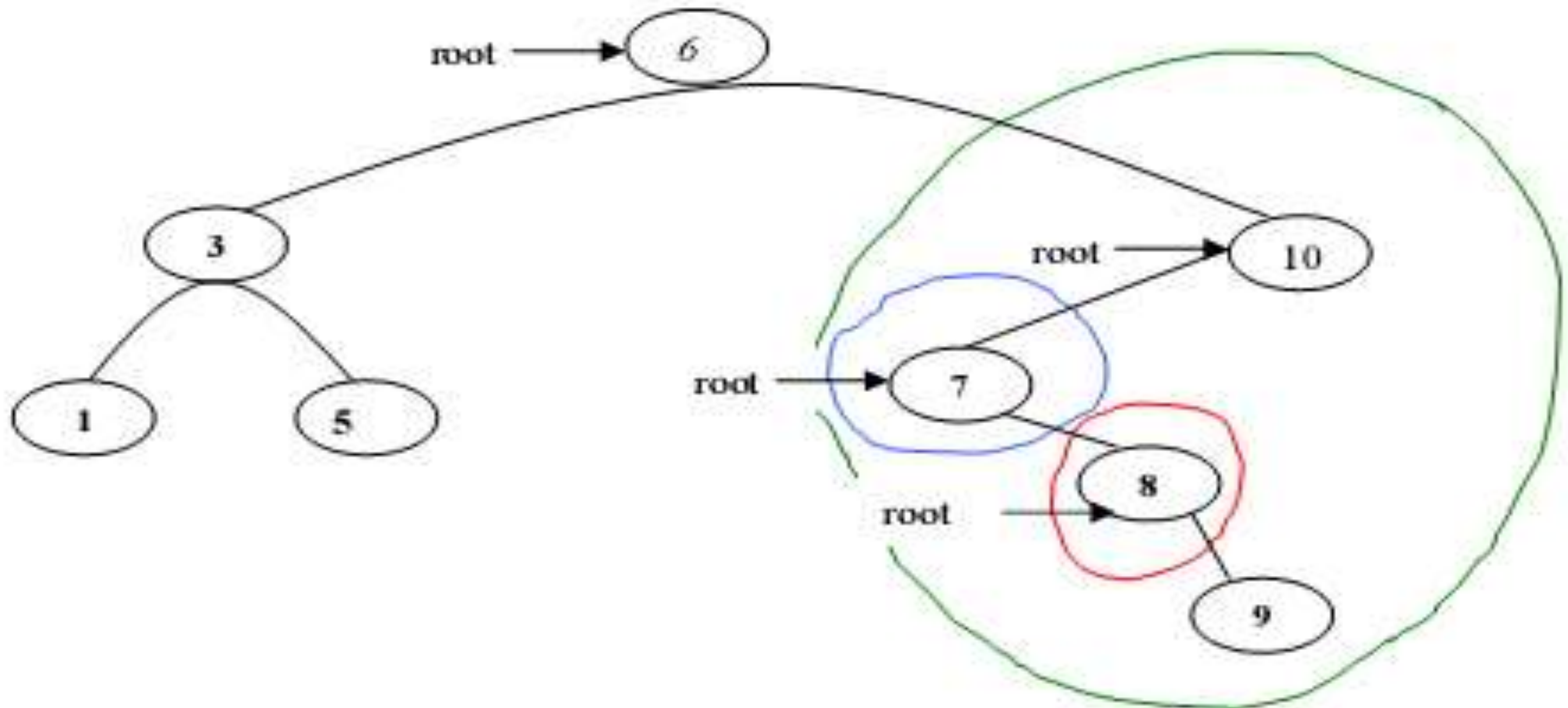
- Pencarian dilakukan secara rekursif, dimulai dari node root.
- Jika data yang dicari lebih kecil daripada data node root, maka pencarian dilakukan di sub node sebelah kiri, sedangkan jika data yang dicari lebih besar daripada data node root, maka pencarian dilakukan di sub node sebelah kanan
- Jika data yang dicari sama dengan data suatu node berarti kembalikan node tersebut dan berarti data ditemukan.

fppt.com

# Ilustrasi Searching

Ilustrasi:

Misal dicari data 8



# Keterangan Searching

- Root = 6 dan 8 > 6, maka akan dicari di sub node bagian kanan root.
- Root = 10 dan 8 < 10, maka akan dicari di sub node bagian kiri root.
- Root = 7 dan 8 > 7, maka akan dicari di sub node bagian kanan root.
- Root = 8, berarti 8 = 8, maka akan dikembalikan node tersebut dan dianggap ketemu!

fppt.com

# Jumlah Node Tree

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

Penghitungan jumlah node dalam tree dilakukan dengan cara mengunjungi setiap node, dimulai dari root ke subtree kiri, kemudian ke subtree kanan dan  masing-masing node dicatat jumlahnya, dan terakhir jumlah node yang ada di subtree kiri dijumlahkan dengan jumlah node yang ada di subtree kanan ditambah 1 yaitu node root.

fppt.com

# Kedalaman (height) Node Tree

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

- Penghitungan kedalaman dihitung dari setelah root, yang dimulai dari subtree bagian kiri kemudian ke subtree bagian kanan.
- Untuk masing-masing kedalaman kiri dan kanan akan dibandingkan, jika ternyata subtree kiri lebih dalam, maka yang dipakai adalah jumlah kedalaman subtree kiri, demikian sebaliknya.
- Hal ini didasarkan pada prinsip binary tree, dimana tree-nya selalu memiliki maksimal 2 node anak.

fppt.com

# Find Min Node

```
Tree *FindMin(Tree *root)
{
	if(root == NULL)
		return NULL;
	else
		if(root->left == NULL)
			return root;
		else
			return FindMin(root-
>left);
}
```

- **Penggunaan:**

➔ `Tree *t = FindMin(pohon);`

fppt.com

# Mencari Leaf (daun)

```
void leaf(Tree *root){
      if(root == NULL) printf("kosong!");
      if(root->left!=NULL) leaf(root->left);
      if(root->right!=NULL) leaf(root->right);
   if(root->right == NULL && root->left == NULL)
printf("%d ",root->data);
}
```

fppt.com

- Abdul Kadir. Teori Dan Aplikasi Struktur Data Menggunakan C++. Andi Publisher.
- Antonius Rachmat C. Http://Lecturer.Ukdw.Ac.Id/Anton
- Jayanti Yusmah Sari. Struktur Data, Teknik Informatika Uho


fppt.com

# thank you

fppt.com