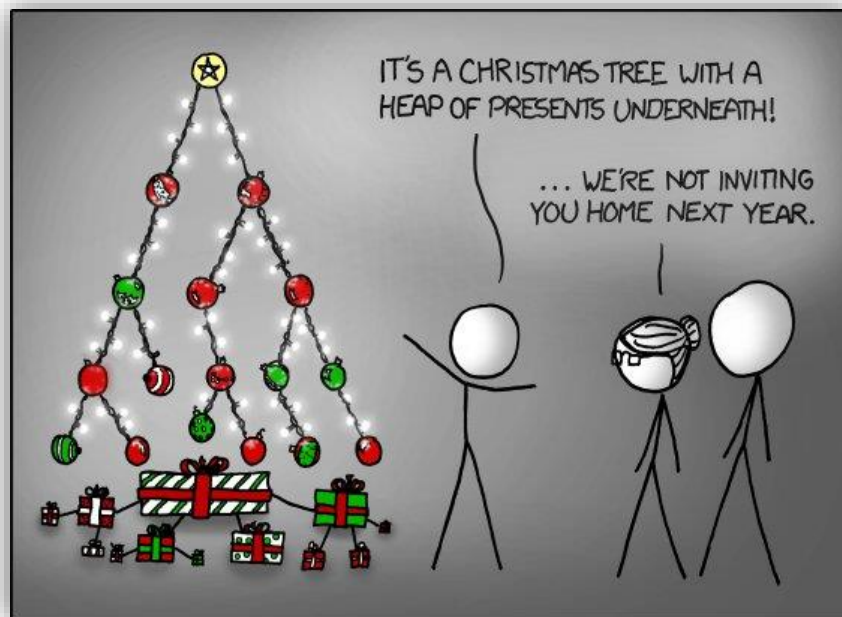


# Modul 11

## Binary Tree

PJ : Michael Kevin



Praktikum Informasi dan Struktur Data

Semester Gasal TA. 2024 / 2025

Program Studi Informatika

Universitas Atma Jaya Yogyakarta

## A. Tujuan

- Praktikan dapat memahami konsep dasar Binary Tree dan dapat menerapkannya dalam bentuk kode pada bahasa pemrograman C.
- Praktikan dapat menyelesaikan berbagai macam kasus yang melibatkan konsep Binary Tree.
- Praktikan dapat mengimplementasikan modul-modul sebelumnya dalam Binary Tree.

## B. Pengantar Binary Tree

Binary Tree adalah bagian dari struktur data yang terdiri dari beberapa nodes di mana setiap node tersebut memiliki 2 anak (children). Struktur ini sering digunakan dalam berbagai aplikasi komputer seperti :

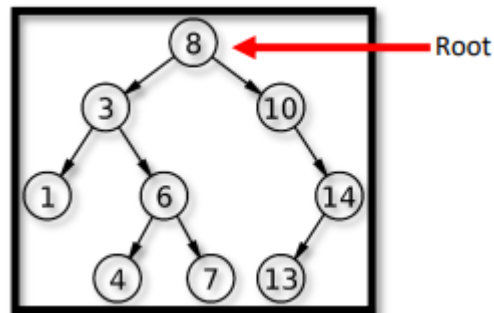
- 1) Search Engine
  - Digunakan dalam metode searching yang akan dibahas di modul selanjutnya.
- 2) Manajemen File Sistem
  - File system modern sering kali menggunakan struktur pohon, termasuk binary tree, untuk mengatur dan mengelola direktori dan file. Misalnya, sistem berkas seperti NTFS dan ext4 menggunakan struktur hierarki.
- 3) Kecerdasan Buatan (AI)
  - Dalam pembelajaran AI, decision trees adalah bentuk binary tree yang digunakan untuk membuat keputusan berdasarkan fitur dan atribut data. Mereka digunakan dalam berbagai algoritma klasifikasi dan regresi.

Binary tree dan variasinya sering menjadi pilihan karena efisiensinya dalam mengelola, mencari, dan memproses data dengan cara yang terstruktur dan mudah diakses.

## C. Istilah-istilah Binary Tree

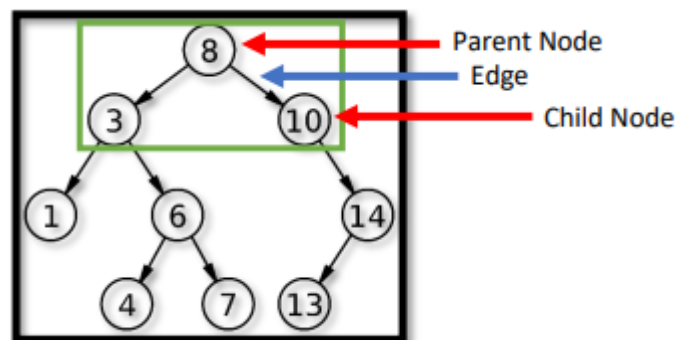
### 1) Root

- Adalah node teratas dari sebuah Tree yang tidak memiliki Parent Node, setiap Tree pasti memiliki satu root.



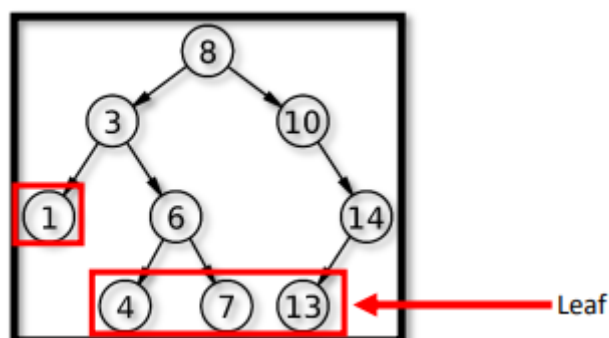
### 2) Edge

- Adalah hubungan Parent Node dengan Child Node yang berfungsi melambangkan hubungan antar node.



### 3) Leaf

- Adalah node dalam binary tree yang tidak mempunyai child node. Setiap Tree pasti mempunyai minimal 1 Leaf. Leaf juga merupakan node terbawah dari suatu tree.

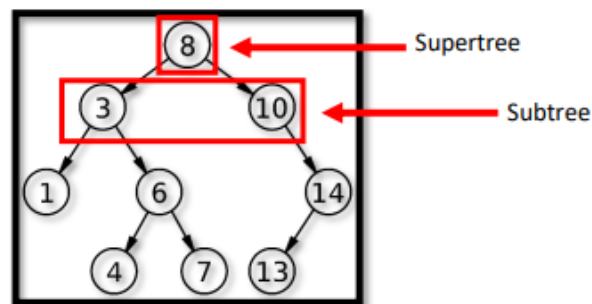


#### 4) Supertree

- Adalah bagian yang menggambarkan pohon yang lebih besar, dimana pohon lain menjadi subtreenya.

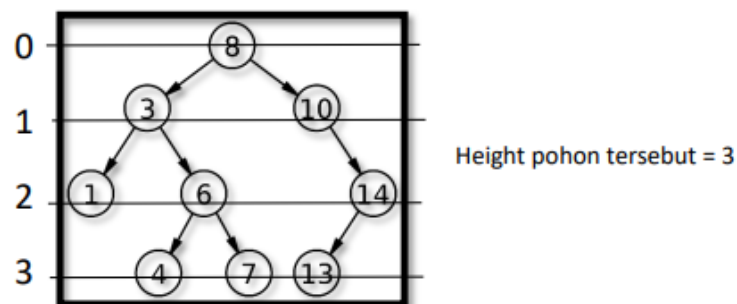
#### 5) Subtree

- Adalah suatu bagian dari pohon lain yang merupakan cabang dari pohon yang lebih besar.



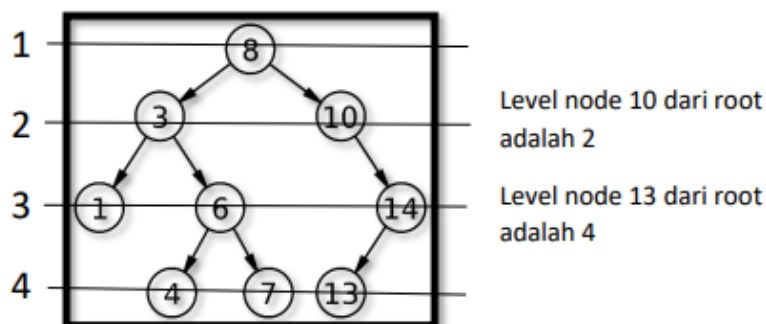
#### 6) Height / Depth

- Adalah jarak dari root ke jalur yang memiliki edge terbanyak.



#### 7) Level

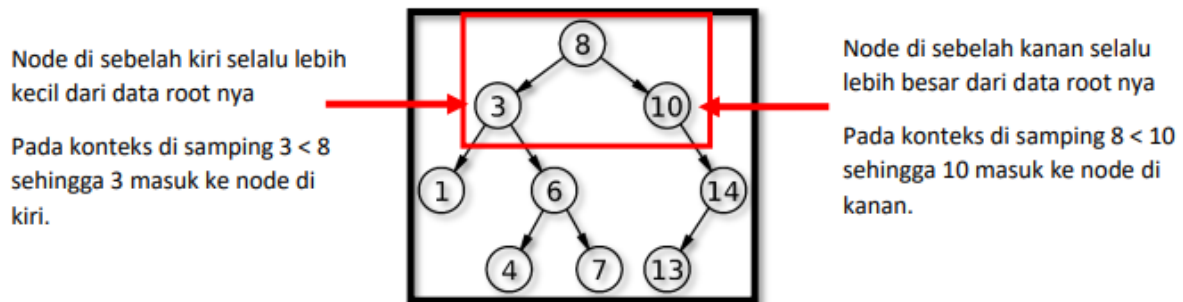
- Adalah jarak dari root ke sebuah node tertentu.



- Perbedaan Height dan Level hanya pada titik mulai perhitungan, dimana height pada pohon kosong memiliki height -1 dan level 0.

## D. Konsep Binary Search Tree

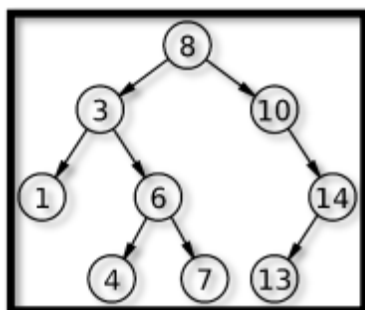
- Konsep yang akan digunakan dalam praktikum kali ini adalah konsep Binary Search Tree. Konsep ini memiliki sifat berpola yakni seluruh Left Child harus memiliki nilai lebih kecil daripada Parent Node dan Right Child sebaliknya.



## E. Penelusuran Binary Tree

Penelusuran pada Binary Tree sering juga disebut Traversal yang merupakan cara menelusuri setiap node yang ada pada Binary Search Tree dengan mengimplementasikan konsep Recursive sehingga perulangan yang dilakukan menjadi lebih sederhana dalam bentuk code. Adapun berbagai penelusuran dalam Binary Tree :

- 1) Pre Order : Root – Left Child – Right Child
- 2) In Order : Left Child – Root – Right Child
- 3) Post Order : Left Child – Right Child – Root



**Pre Order:** 8 – 3 – 1 – 6 – 4 – 7 – 10 – 14 – 13

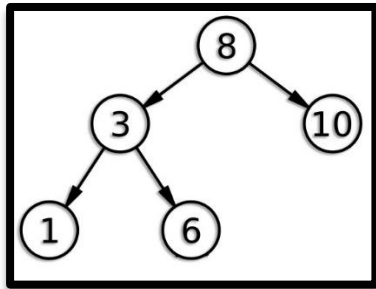
**In Order:** 1 – 3 – 4 – 6 – 7 – 8 – 10 – 13 – 14

**Post Order:** 1 – 4 – 7 – 6 – 3 – 13 – 14 – 10 – 8

## F. Jenis Binary Tree berdasarkan children yang dimiliki

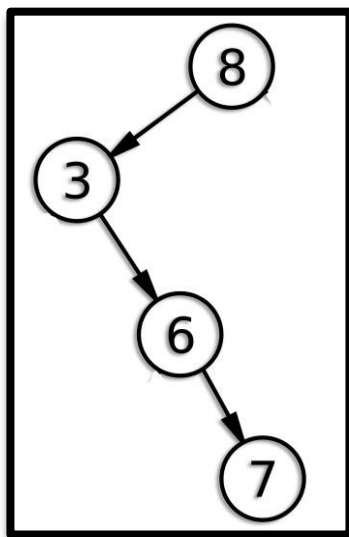
### 1. Full Binary Tree

Suatu Binary Tree dapat dikatakan full jika semua node mempunyai 0 atau 2 children. Kita dapat menyebut Full Binary Tree jika semua node kecuali node leaf, mempunyai 2 children.



### 2. Degenerate Tree

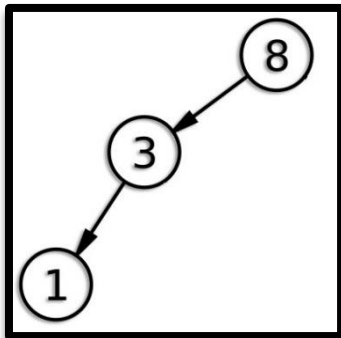
Binary Tree yang merupakan Degenerate Tree adalah Binary Tree yang setiap node mempunyai 0 atau 1 child (left / right).



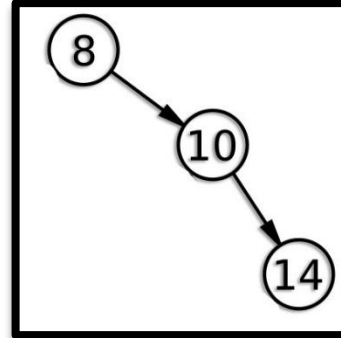
### 3. Skewed Binary Tree

Skewed Binary Tree adalah Binary Tree yang setiap node hanya memiliki 1 node yaitu hanya kanan / hanya kiri.

*Left-Skewed Binary Tree*



*Right-Skewed Binary Tree*

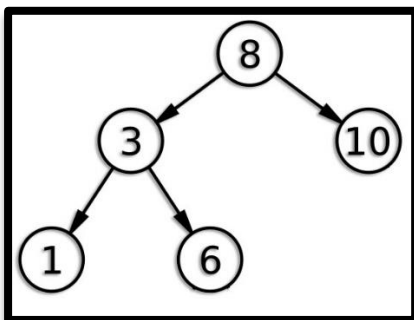


## G. Jenis Binary Tree berdasarkan completion of levels

### 1. Complete Binary Tree

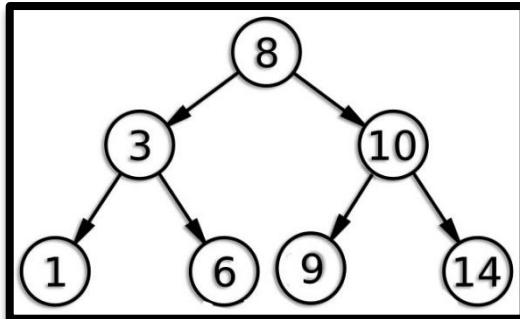
Complete Binary Tree sangatlah mirip dengan Full Binary Tree, namun ada beberapa perbedaan, yaitu :

- Semua level kecuali level terbawah harus penuh.
- Semua leaf harus miring ke kiri.
- Elemen leaf terakhir mungkin tidak mempunyai right sibling, Complete Binary Tree tidak harus menjadi Full Binary Tree.



## 2. Perfect Binary Tree

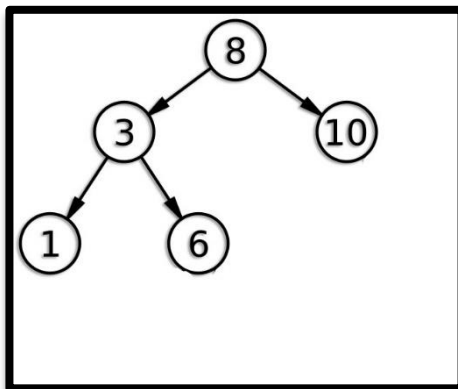
Perfect Binary Tree adalah Binary Tree yang semua node mempunyai 2 children kecuali leaf node dan semua leaf node harus berada pada level yang sama.



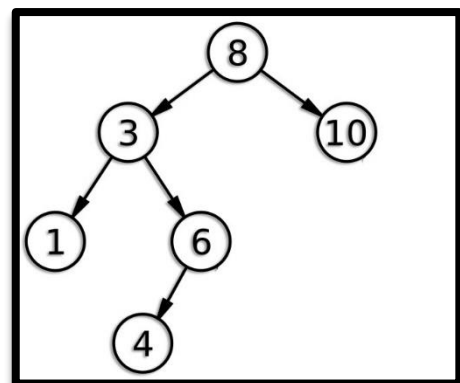
## 3. Balanced Binary Tree

Adalah kondisi Binary Tree dimana perbedaan ketinggian antara left sub-tree & right sub-tree tidak lebih dari 1.

*Balanced Tree*



*Unbalanced Tree*





## H. Struktur, Prosedur, dan Fungsi Binary Tree

Pada modul kali ini, kita akan membuat aplikasi sederhana yang dapat menampung struktur penyimpanan data yang dapat melakukan operasi-operasi yang dibutuhkan terhadap suatu data. *Ingat! modul ini hanya membahas code Binary Search Tree di mana node kiri selalu lebih kecil dan node kanan sebaliknya.*

### 1) Struct Binary Tree

Pada guided kali ini, setiap node mempunyai 1 tipe data integer dan 2 pointer yaitu pointer ke kiri untuk Left Child dan ke kanan untuk Right Child.

```
1 typedef int infotype;
2 typedef struct TreeNode* address;
3 typedef struct TreeNode* BinaryTree;
4
5 typedef struct TreeNode {
6     infotype data;
7     address left;
8     address right;
9 } TreeNode;
```

### 2) Create Empty

Sama hal nya dengan membuat node kosong pada linked list.

```
1 void createEmpty(BinaryTree* root) {
2     *root = NULL;
3 }
```

### 3) Is Empty

Sama hal nya dengan mengecek apakah node kosong pada linked list.

```
1 bool isEmpty(BinaryTree root) {
2     return root == NULL;
3 }
```

#### 4) Is Leaf

Mengecek apakah node tersebut ialah daun atau tidak, jika ia tidak mempunyai node di kiri dan juga tidak mempunyai node di kanan, maka bisa dikatakan bahwa node tersebut adalah daun / leaf.

```

1  bool isLeaf(BinaryTree node) {
2      return node->left == NULL && node->right == NULL;
3  }

```

#### 5) Alokasi Data

Membentuk node dari data yang dimasukkan, dan menginisialisasikan empty pada left & right child.

```

1  address alokasi(infotype data) {
2      address newNode = (address)malloc(sizeof(TreeNode));
3      newNode->data = data;
4      newNode->left = NULL;
5      newNode->right = NULL;
6      return newNode;
7  }

```

#### 6) Insert Node

Dengan menerapkan recursive function, kita mencari node yang kosong & sesuai dengan data yang kita miliki untuk memasukkan node baru. Jika node yang sedang ditelusuri kosong, maka letakkan data pada node tersebut, jika value node baru lebih kecil dari data yang dimiliki, maka recursifkan lagi ke node kiri, sebaliknya, recursifkan ke node kanan.


```

1  void insertNode(BinaryTree* root, address newNode) {
2      if (isEmpty(*root))
3          *root = newNode;
4      else if (newNode->data < (*root)->data)
5          insertNode(&(*root)->left, newNode);
6      else
7          insertNode(&(*root)->right, newNode);
8  }

```

### 7) Pre Order

Melakukan print seluruh data dengan urutan Root – Left Child – Right Child.



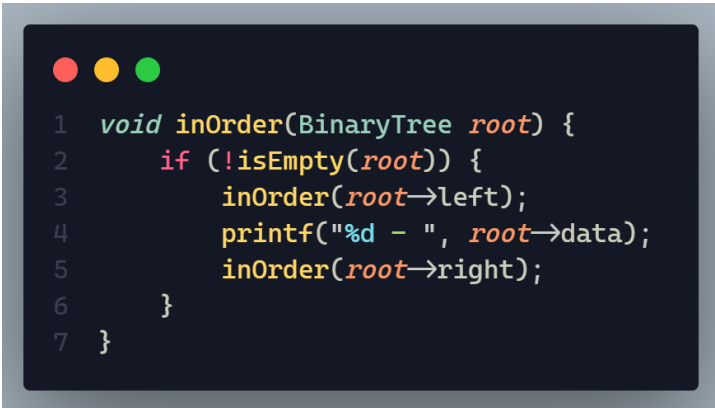
```

1 void preOrder(BinaryTree root) {
2     if (!isEmpty(root)) {
3         printf("%d - ", root->data);
4         preOrder(root->left);
5         preOrder(root->right);
6     }
7 }

```

### 8) In Order

Melakukan print seluruh data dengan urutan Left Child – Root – Right Child.



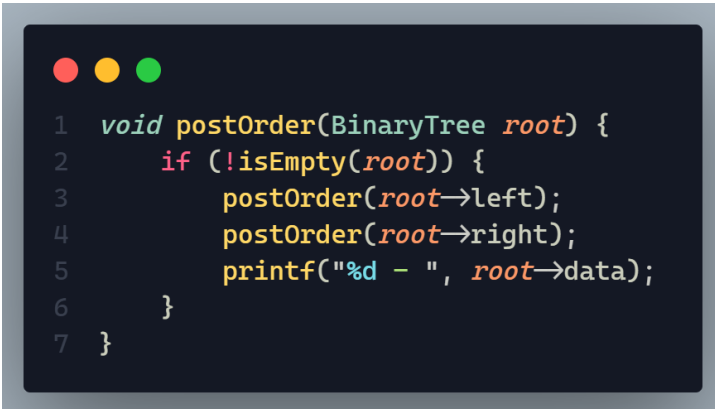
```

1 void inOrder(BinaryTree root) {
2     if (!isEmpty(root)) {
3         inOrder(root->left);
4         printf("%d - ", root->data);
5         inOrder(root->right);
6     }
7 }

```

### 9) Post Order

Melakukan print seluruh data dengan urutan Left Child – Right Child – Root.



```

1 void postOrder(BinaryTree root) {
2     if (!isEmpty(root)) {
3         postOrder(root->left);
4         postOrder(root->right);
5         printf("%d - ", root->data);
6     }
7 }

```

### 10) Height

Mengukur ketinggian tree, jika tree kosong, maka ketinggiannya -1, jika menemui daun / leaf, maka ketinggian tersebut 0, jika menemukan node yang bukan leaf, maka tambahkan 1 dengan height tertinggi dari kedua leafnya. Fungsi fmax digunakan untuk mencari nilai maximal dengan membandingkan 2 nilai.

```

1  int treeHeight(BinaryTree root) {
2      if (isEmpty(root))
3          return -1;
4      else if (isLeaf(root))
5          return 0;
6      else
7          return (1 + fmax(treeHeight(root->left), treeHeight(root->right)));
8  }

```

### 11) Is Found

Digunakan untuk mengecek apakah node dengan value yang dicari ada pada tree dengan output true / false, proses tersebut mempunyai urutan :

- Jika node kosong, maka value yang dicari tidak ada.
- Jika value node yang dicari sama dengan value node yang dikunjungi, berarti data yang dicari ada.
- Jika value node yang dicari lebih kecil daripada node yang dikunjungi, rekursifkan ke kiri.
- Sebaliknya, rekursifkan ke kanan.

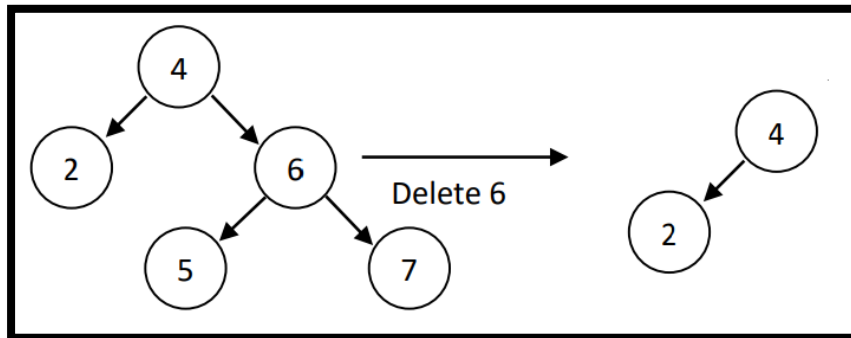
```

1  bool isFound(BinaryTree root, int data) {
2      if (isEmpty(root))
3          return false;
4      else if (root->data == data)
5          return true;
6      else if (data < root->data)
7          return isFound(root->left, data);
8      else
9          return isFound(root->right, data);
10 }

```

## 12) Delete Node

Delete node pada Binary Tree memiliki cara yang unik, apabila data yang ingin di delete ada di tengah suatu tree (bukan leaf), deleting node dengan cara biasa akan menghilangkan data dibawahnya.



Tentu ini akan menjadi masalah jika kita hanya ingin menghapus node 6 jika node tersebut bukanlah leaf.

```

1 void deleteNode(BinaryTree* root, int data) {
2     if (isEmpty(*root))
3         return;
4     else if (data < (*root)→data)
5         deleteNode(&(*root)→left, data);
6     else if (data > (*root)→data)
7         deleteNode(&(*root)→right, data);
8     else {
9         address temp = *root;
10        *root = NULL;
11        free(temp);
12    }
13 }
  
```

Code diatas belumlah sempurna karena masih akan menghapus node dibawahnya.

**Hint :** ada 4 kondisi berbeda node yang akan dihapus, yaitu jika node tersebut adalah leaf, node tersebut hanya mempunyai 1 child kiri, node tersebut hanya mempunyai 1 child kanan, dan jika node tersebut mempunyai 2 child. Prosedur diatas hanya berlangsung dengan benar jika node tersebut adalah leaf.

## I. Perbedaan Binary Tree & Linked List

Aspek	Linked List	Binary Tree
Struktur	Rangkaian node dihubungkan secara linear.	Rangkaian node berbentuk hierarki dengan node akar dan cabang-cabang.
Tipe Node	Setiap node memiliki satu pointer yaitu next.	Setiap node memiliki dua pointer yaitu left dan right.
Penelusuran	Dilakukan secara satu arah.	Dilakukan dengan beberapa cara yaitu preorder, inorder, dan postorder.
Kompleksitas Waktu Akses	$O(n)$ karena penelusuran dilakukan secara sekuensial.	$O(\log n)$ untuk balanced binary search tree hingga $O(n)$ jika pohon berbentuk seperti linked list.
Penerapan	Digunakan untuk struktur linear seperti stack atau queue.	Digunakan untuk representasi hierarki, pencarian cepat, dan ekspresi matematika / system file.
Tipe Prosedur & Fungsi	Menggunakan fungsi rekursif atau iteratif.	Menggunakan fungsi rekursif.
Kelebihan	Mudah diimplementasikan dan cocok untuk aplikasi dengan struktur data linear.	Efisien untuk pencarian dan penyimpanan data yang terstruktur.
Kekurangan	Waktu akses lebih lambat karena harus melewati node satu per satu.	Lebih kompleks dalam implementasi dan manajemen dibanding linked list.

## J. Guided

Setelah memahami konsep-konsep Binary Tree, mari kita membuat program binary tree sempurna yang dapat melakukan Insert, Delete, Show, Cek dengan sempurna.

### 1. Insert

Input data ke Binary Tree menggunakan angka inputan sebagai nilai pembanding untuk masuk ke tree kiri / kanan.

### 2. Show & Cek

Print semua data pada tree dengan 3 macam penelusuran yaitu preorder, inorder, dan postorder. Buat juga fungsi untuk mencari apakah data sudah ada dalam tree atau belum.

### 3. Delete

Hapus node yang diinputkan pada tree, jika node tidak ada maka tidak melakukan apa-apa. Kerjakan catatan opsional pada code dibawah untuk membantu mengerjakan UGD.

- Header.h

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <conio.h>
4  #include <stdbool.h>
5  #include <math.h>
6
7  typedef int infotype;
8  typedef struct TreeNode* address;
9  typedef struct TreeNode* BinaryTree;
10
11 typedef struct TreeNode {
12     infotype data;
13     address left;
14     address right;
15 } TreeNode;
16
17 //Inisialisasi Tree
18 void createEmpty(BinaryTree* root);
19 bool isEmpty(BinaryTree root);
20 bool isLeaf(BinaryTree node);
21
22 //Insert Tree
23 address alokasi(infotype data);
24 void insertNode(BinaryTree* root, address newNode);
25
26 //Read Tree
27 void preOrder(BinaryTree root);
28 void inOrder(BinaryTree root);
29 void postOrder(BinaryTree root);
30 int treeHeight(BinaryTree root);
31
32 //Search TreeNode
33 bool isFound(BinaryTree root, int data);
34
35 //Delete TreeNode
36 void deleteNode(BinaryTree* root, int data);
37

```



- Main.c

```

1  #include "header.h"
2
3  int main() {
4      BinaryTree BT;
5      infotype data;
6
7      int menu = -1;
8
9      createEmpty(&BT);
10
11     while (menu != 0) {
12         system("cls");
13         printf("---[ Binary Search Tree ]---\n");
14         printf("[1] Insert Node\n");
15         printf("[2] Delete Node\n");
16         printf("[3] Show pohon\n");
17         printf("[4] Cek apakah node ada\n");
18         printf("[0] Exit\n");
19         printf(">>> "); scanf("%d", &menu);
20         switch(menu) {
21             case 1:
22                 printf("\nInput data: "); scanf("%d", &data);
23                 insertNode(&BT, alokasi(data));
24                 break;
25
26             case 2:
27                 printf("\nDelete data: "); scanf("%d", &data);
28                 deleteNode(&BT, data);
29                 break;
30
31             case 3:
32                 printf("\nPreOrder   : "); preOrder(BT);
33                 printf("\nInOrder    : "); inOrder(BT);
34                 printf("\nPostOrder  : "); postOrder(BT);
35                 printf("\n\nHeight   : %d", treeHeight(BT));
36                 break;
37
38             case 4:
39                 printf("\nCari data yang ingin dicari : "); scanf("%d", &data);
40                 printf("\nData %s", isFound(BT, data) ? "ditemukan" : "tidak ditemukan");
41                 break;
42
43             case 0:
44                 printf("\n\t[*] Nama / NPM");
45                 break;
46
47             default:
48                 printf("\n\t[!] Menu tidak tersedia");
49                 break;
50         } getch();
51     }
52     return 0;
53 }
54

```

- Source.c

```

1  #include "header.h"
2
3  void createEmpty(BinaryTree* root) {
4      *root = NULL;
5  }
6
7  bool isEmpty(BinaryTree root) {
8      return root == NULL;
9  }
10
11 bool isLeaf(BinaryTree node) {
12     return node->left == NULL && node->right == NULL;
13 }
14
15 address alokasi(infotype data) {
16     address newNode = (address)malloc(sizeof(TreeNode));
17     newNode->data = data;
18     newNode->left = NULL;
19     newNode->right = NULL;
20     return newNode;
21 }
22
23 void insertNode(BinaryTree* root, address newNode) {
24     if (isEmpty(*root))
25         *root = newNode;
26     else if (newNode->data < (*root)->data)
27         insertNode(&(*root)->left, newNode);
28     else
29         insertNode(&(*root)->right, newNode);
30 }
31
32 void preOrder(BinaryTree root) {
33     if (!isEmpty(root)) {
34         printf("%d - ", root->data);
35         preOrder(root->left);
36         preOrder(root->right);
37     }
38 }
39
40 void inOrder(BinaryTree root) {
41     if (!isEmpty(root)) {
42         inOrder(root->left);
43         printf("%d - ", root->data);
44         inOrder(root->right);
45     }
46 }
47
48 void postOrder(BinaryTree root) {
49     if (!isEmpty(root)) {
50         postOrder(root->left);
51         postOrder(root->right);
52         printf("%d - ", root->data);
53     }
54 }

```

```

55
56 int treeHeight(BinaryTree root) {
57     if (isEmpty(root))
58         return -1;
59     else if (isLeaf(root))
60         return 0;
61     else
62         return(1 + fmax(treeHeight(root->left), treeHeight(root->right)));
63 }
64
65 bool isFound(BinaryTree root, int data) {
66     if (isEmpty(root))
67         return false;
68     else if (root->data == data)
69         return true;
70     else if (data < root->data)
71         return isFound(root->left, data);
72     else
73         return isFound(root->right, data);
74 }
75
76 void deleteNode(BinaryTree* root, int data) {
77     if (isEmpty(*root))
78         return;
79     else if (data < (*root)->data)
80         deleteNode(&(*root)->left, data);
81     else if (data > (*root)->data)
82         deleteNode(&(*root)->right, data);
83     else {
84         address temp = *root;
85         *root = NULL;
86         free(temp);
87
88         /*          OPSIONAL
89         Prosedur deleteNode ini belum sempurna karena masih menghapus
90         seluruh node di bawah node yang ditemukan, ada alternatif lain
91         untuk melakukan penghapusannya tanpa menghapus keseluruhan
92         pohon di bawahnya. Lakukan searching mandiri untuk mencari
93         tau hal tersebut dan kalian boleh letakkan code di sini
94         karena akan membantu UGD kalian.
95
96         */
97     }
98 }

```

**K. Format Penamaan**

GD11\_X\_YYYYY.zip

X = Kelas

Y = 5 NPM Terakhir

**Catatan :**

1. menyempurnakan prosedur deleteNode memanglah opsional dan tidak mengurangi nilai apapun jika tidak dikerjakan, namun itu merupakan salah satu soal UGD yang akan dikeluarkan.
2. Diperbolehkan untuk mengedit / menambahkan catatan dalam guided berupa text maupun fungsi-fungsi lain sebagai cheatsheet dalam mengerjakan UGD.

**Masih bingung dengan cara kerja Binary Tree?** Kunjungi website simulasi binary tree di : <https://visualgo.net/id/bst>

“Computer Scientists draw the root at the top of their trees because they've never been outside to see what a real tree looks like”