

Tugas : 05  
Nama : Fitri Romadhona  
NIM : 23050974179  
Kelas : PTI 2023E  
Dosen Pengampu : Riza Akhsani Setyo Prayoga, S.Kom., M.MT.  
Mata Kuliah : Struktur Data

**1. Source Code untuk menghapus sebuah node tertentu dan menghapus seluruh node.**

```
#include<iostream>
#include<string>

using namespace std;

// Struktur untuk Node pada Tree
struct Node {
    string data;
    Node* left;
    Node* right;
};

// Fungsi untuk membuat node baru
Node* createNode(string data) {
    Node* newNode = new Node();
    if (!newNode) {
        cout << "Gagal mengalokasikan memori\n";
        return NULL;
    }
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Fungsi untuk menambah node ke Tree (berdasarkan urutan leksikografis)
Node* insertNode(Node* root, string data) {
    if (root == NULL) {
        root = createNode(data);
        return root;
    }

    if (data < root->data) {
        root->left = insertNode(root->left, data);
    }
}
```

```

else if (data > root->data) {
    root->right = insertNode(root->right, data);
}

return root;
}

// Fungsi untuk mencari node terkecil (digunakan saat penghapusan node)
Node* minValueNode(Node* node) {
    Node* current = node;

    while (current && current->left != NULL)
        current = current->left;

    return current;
}

// Fungsi untuk menghapus node tertentu
Node* deleteNode(Node* root, string data) {
    if (root == NULL)
        return root;

    if (data < root->data)
        root->left = deleteNode(root->left, data);
    else if (data > root->data)
        root->right = deleteNode(root->right, data);
    else {
        // Node ditemukan
        if (root->left == NULL) {
            Node* temp = root->right;
            delete root;
            return temp;
        }
        else if (root->right == NULL) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// Fungsi untuk menghapus semua node dari Tree
void deleteAllNodes(Node* &root) {
    if (root == NULL) return;

```

```

deleteAllNodes(root->left);
deleteAllNodes(root->right);

cout << "Menghapus node: " << root->data << endl;
delete root;
root = NULL;
}

// Fungsi inorder traversal (untuk menampilkan isi tree secara berurutan)
void inorder(Node* root) {
    if (root == NULL)
        return;

    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = NULL;

    // Memasukkan data makanan
    root = insertNode(root, "Nasi Goreng");
    root = insertNode(root, "Bakso");
    root = insertNode(root, "Mie Ayam");
    root = insertNode(root, "Pizza");

    cout << "Inorder traversal sebelum penghapusan: ";
    inorder(root);
    cout << endl;

    // Menghapus node "Bakso"
    cout << "Menghapus node 'Bakso'\n";
    root = deleteNode(root, "Bakso");

    cout << "Inorder traversal setelah penghapusan 'Bakso': ";
    inorder(root);
    cout << endl;

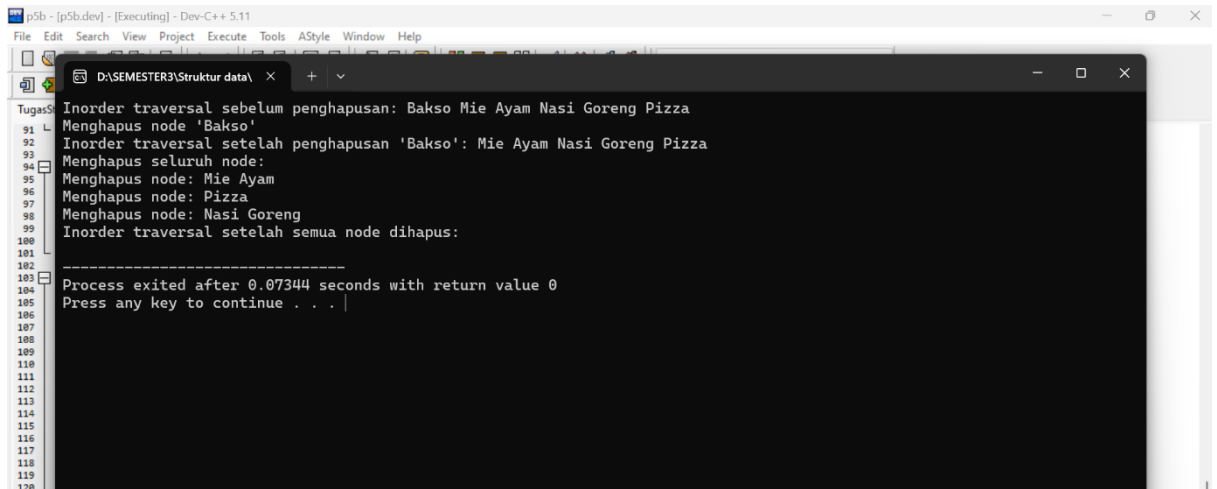
    // Menghapus seluruh node
    cout << "Menghapus seluruh node:\n";
    deleteAllNodes(root);

    cout << "Inorder traversal setelah semua node dihapus: ";
    inorder(root); // Seharusnya tidak ada output karena semua node telah dihapus
    cout << endl;

    return 0;
}

```

## Hasil yang Ditampilkan



```
p5b - [p5b.dev] - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help

D:\SEMESTER3\Struktur data\
Tugas5
91 Inorder traversal sebelum penghapusan: Bakso Mie Ayam Nasi Goreng Pizza
92 Menghapus node 'Bakso'
93 Inorder traversal setelah penghapusan 'Bakso': Mie Ayam Nasi Goreng Pizza
94 Menghapus seluruh node:
95 Menghapus node: Mie Ayam
96 Menghapus node: Pizza
97 Menghapus node: Nasi Goreng
98 Inorder traversal setelah semua node dihapus:
99
100 -----
101 Process exited after 0.07344 seconds with return value 0
102 Press any key to continue . . .
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
```

## 2. Source Code untuk searching node.

```
#include <iostream>
#include <string>

using namespace std;

// Struktur untuk Node pada Tree
struct Node {
    string data;
    Node* left;
    Node* right;
};

// Fungsi untuk membuat node baru
Node* createNode(string data) {
    Node* newNode = new Node();
    if (!newNode) {
        cout << "Gagal mengalokasikan memori\n";
        return NULL;
    }
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Fungsi untuk menambah node ke Tree (berdasarkan urutan leksikografis)
Node* insertNode(Node* root, string data) {
    if (root == NULL) {
        root = createNode(data);
        return root;
    }
}
```

```

    if (data < root->data) {
        root->left = insertNode(root->left, data);
    }
    else if (data > root->data) {
        root->right = insertNode(root->right, data);
    }

    return root;
}

// Fungsi untuk melakukan pencarian node
bool searchNode(Node* root, string key) {
    if (root == NULL)
        return false;

    if (root->data == key)
        return true;

    if (key < root->data)
        return searchNode(root->left, key);
    else
        return searchNode(root->right, key);
}

// Fungsi inorder traversal (untuk menampilkan isi tree secara berurutan)
void inorder(Node* root) {
    if (root == NULL)
        return;

    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = NULL;

    // Memasukkan data makanan
    root = insertNode(root, "Nasi Goreng");
    root = insertNode(root, "Bakso");
    root = insertNode(root, "Mie Ayam");
    root = insertNode(root, "Pizza");

    cout << "Inorder traversal dari Tree: ";
    inorder(root);
    cout << endl;

    // Mencari node tertentu
    string searchItem;
    cout << "Masukkan nama makanan yang ingin dicari: "; Tahu

```

```

cin >> searchItem;

if (searchNode(root, searchItem))
    cout << searchItem << " ditemukan dalam Tree.\n";
else
    cout << searchItem << " tidak ditemukan dalam Tree.\n";

return 0;
}

```

## Hasil yang Ditampilkan

```

D:\SEMESTER3\Struktur data\TugasStrukturData05.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TugasStrukt
92 // Inorder traversal sebelum penghapusan: Bakso Mie Ayam Nasi Goreng Pizza
93 // Menghapus node 'Bakso'
94 // Inorder traversal setelah penghapusan 'Bakso': Mie Ayam Nasi Goreng Pizza
95 // Menghapus seluruh node:
96 // Menghapus node: Mie Ayam
97 // Menghapus node: Pizza
98 // Menghapus node: Nasi Goreng
99 // Inorder traversal setelah semua node dihapus:
100 //
101 //
102 //
103 //
104 //
105 //
106 //
107 //
108 //
109 //
110 //
111 //
112 //
113 //
114 //
115 //
116 //
117 //
118 //
119 //
120 //
121 //
122 //
123 //
124 //
125 //
126 //

```

### 3. Source Code untuk program AVL tree dengan tampilan : 1. Insert 2. Delete 3. Tranverse 4. Exit.

```

#include <iostream>
#include <string>

using namespace std;

// Struktur untuk Node pada AVL Tree
struct Node {
    string data;
    Node* left;
    Node* right;
    int height;
};

```

```

// Fungsi untuk mendapatkan tinggi dari node
int getHeight(Node* node) {
    return (node == NULL) ? 0 : node->height;
}

// Fungsi untuk mendapatkan faktor keseimbangan
int getBalance(Node* node) {
    return (node == NULL) ? 0 : getHeight(node->left) - getHeight(node->right);
}

// Fungsi untuk membuat node baru
Node* createNode(string data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    newNode->height = 1; // Node baru ditambahkan sebagai daun
    return newNode;
}

// Rotasi kanan
Node* rightRotate(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;

    x->right = y;
    y->left = T2;

    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;

    return x; // Mengembalikan root baru
}

// Rotasi kiri
Node* leftRotate(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;

    y->left = x;
    x->right = T2;

    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;

    return y; // Mengembalikan root baru
}

// Fungsi untuk menyisipkan data ke dalam AVL Tree
Node* insert(Node* node, string data) {
    // Normal BST insertion

```

```

if (node == NULL)
    return createNode(data);

if (data < node->data)
    node->left = insert(node->left, data);
else if (data > node->data)
    node->right = insert(node->right, data);
else // Duplicate keys tidak diperbolehkan
    return node;

// Memperbarui tinggi node ini
node->height = 1 + max(getHeight(node->left), getHeight(node->right));

// Mendapatkan faktor keseimbangan node ini
int balance = getBalance(node);

// Jika node tidak seimbang, lakukan rotasi
// Kasus Kiri Kiri
if (balance > 1 && data < node->left->data)
    return rightRotate(node);

// Kasus Kanan Kanan
if (balance < -1 && data > node->right->data)
    return leftRotate(node);

// Kasus Kiri Kanan
if (balance > 1 && data > node->left->data) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
}

// Kasus Kanan Kiri
if (balance < -1 && data < node->right->data) {
    node->right = rightRotate(node->right);
    return leftRotate(node);
}

return node; // Kembalikan node (tidak berubah)
}

// Fungsi untuk mencari node dengan nilai terkecil
Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left != NULL)
        current = current->left;
    return current;
}

// Fungsi untuk menghapus node dari AVL Tree
Node* deleteNode(Node* root, string data) {

```



```

// Normal BST deletion
if (root == NULL)
    return root;

if (data < root->data)
    root->left = deleteNode(root->left, data);
else if (data > root->data)
    root->right = deleteNode(root->right, data);
else {
    // Node ditemukan
    if ((root->left == NULL) || (root->right == NULL)) {
        Node* temp = (root->left) ? root->left : root->right;

        if (temp == NULL) { // Node dengan satu anak atau tidak ada anak
            temp = root;
            root = NULL;
        } else
            *root = *temp; // Copy isi node anak
        delete temp;
    } else {
        Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
}

// Jika tree hanya memiliki satu node
if (root == NULL)
    return root;

// Memperbarui tinggi node ini
root->height = 1 + max(getHeight(root->left), getHeight(root->right));

// Mendapatkan faktor keseimbangan node ini
int balance = getBalance(root);

// Jika node tidak seimbang, lakukan rotasi
// Kasus Kiri Kiri
if (balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(root);

// Kasus Kiri Kanan
if (balance > 1 && getBalance(root->left) < 0) {
    root->left = leftRotate(root->left);
    return rightRotate(root);
}

// Kasus Kanan Kanan
if (balance < -1 && getBalance(root->right) <= 0)
    return leftRotate(root);

```

```

// Kasus Kanan Kiri
if (balance < -1 && getBalance(root->right) > 0) {
    root->right = rightRotate(root->right);
    return leftRotate(root);
}

return root; // Kembalikan node (tidak berubah)
}

// Fungsi untuk melakukan inorder traversal
void inorder(Node* root) {
    if (root != NULL) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

// Fungsi utama
int main() {
    Node* root = NULL;
    int choice;
    string food;

    do {
        cout << "\nMenu:\n";
        cout << "1. Insert\n";
        cout << "2. Delete\n";
        cout << "3. Traverse\n";
        cout << "4. Exit\n";
        cout << "Pilih opsi: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan nama makanan: ";
                cin >> food;
                root = insert(root, food);
                break;
            case 2:
                cout << "Masukkan nama makanan yang ingin dihapus: ";
                cin >> food;
                root = deleteNode(root, food);
                break;
            case 3:
                cout << "Inorder traversal dari AVL Tree: ";
                inorder(root);
                cout << endl;
                break;
        }
    } while (choice != 4);
}

```

```

        case 4:
            cout << "Keluar dari program.\n";
            break;
        default:
            cout << "Pilihan tidak valid. Silakan coba lagi.\n";
        }
    } while (choice != 4);

    return 0;
}

```

## Hasil yang Ditampilkan

```

p5b - [p5b.dev] - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
D:\SEMESTER3\Struktur data\ TugasStruk
182 Menu:
183 1. Insert
184 2. Delete
185 3. Traverse
186 4. Exit
187
188 Pilih opsi: 1
189 Masukkan nama makanan: Bakso
190
191 Menu:
192 1. Insert
193 2. Delete
194 3. Traverse
195 4. Exit
196
197 Pilih opsi: 2
198 Masukkan nama makanan yang ingin dihapus: Pizza
199
200 Menu:
201 1. Insert
202 2. Delete
203 3. Traverse
204 4. Exit
205
206 Pilih opsi: 3
207 Inorder traversal dari AVL Tree: Bakso
208
209
210
211
212
213
214
215
216

```

```

p5b - [p5b.dev] - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
D:\SEMESTER3\Struktur data\ TugasStruk
182 Pilih opsi: 1
183 Masukkan nama makanan: Bakso
184
185 Menu:
186 1. Insert
187 2. Delete
188 3. Traverse
189 4. Exit
190
191 Pilih opsi: 2
192 Masukkan nama makanan yang ingin dihapus: Pizza
193
194 Menu:
195 1. Insert
196 2. Delete
197 3. Traverse
198 4. Exit
199
200 Pilih opsi: 3
201 Inorder traversal dari AVL Tree: Bakso
202
203 Menu:
204 1. Insert
205 2. Delete
206 3. Traverse
207 4. Exit
208
209 Pilih opsi: 4
210 Keluar dari program.
211
212 -----
213 Process exited after 57.45 seconds with return value 0
214 Press any key to continue . . .
215
216
217
218
219
220
221
222
223

```