

Laporan Binary Search Tree

Nama Kelompok :

- | | |
|---------------------------------|-------------|
| 1. Syarifah Saskia Aulia | 21091397012 |
| 2. Nadia Alfiani Raissa Pansera | 21091397014 |
| 3. Reiznu Ahmad Tjandrida | 21091397018 |
| 4. Widya Sari Wibowo | 21091397070 |

Kode

```
#include<iostream>
#define SPACE 10

using namespace std;

class pohonNode {
public:
    int nilai;
    pohonNode * kiri;
    pohonNode * kanan;

    pohonNode() {
        nilai = 0;
        kiri = NULL;
        kanan = NULL;
    }
    pohonNode(int v) {
        nilai = v;
        kiri = NULL;
        kanan = NULL;
    }
};

class BST {
public:
    pohonNode * root;
    BST() {
        root = NULL;
    }
    bool isEmpty() {
        if (root == NULL) {
            return true;
        } else {
            return false;
        }
    }

    void insertNode(pohonNode * node_baru) {
        if (root == NULL) {
            root = node_baru;
            cout << "Nilai Di Inputkan Sebagai Root Node!" << endl;
        } else {
            pohonNode * temp = root;
            while (temp != NULL) {
                if (node_baru->nilai == temp->nilai) {
                    cout << "Nilai Sudah Ada," <<
                        "Masukkan Nilai Lain!" << endl;
                }
            }
        }
    }
};
```

```

        return;
    } else if ((node_baru -> nilai < temp -> nilai) && (temp -> kiri == NULL)) {
        temp -> kiri = node_baru;
        cout << "Nilai Di Sisipkan Di Sebelah Kiri" << endl;
        break;
    } else if (node_baru -> nilai < temp -> nilai) {
        temp = temp -> kiri;
    } else if ((node_baru -> nilai > temp -> nilai) && (temp -> kanan == NULL)) {
        temp -> kanan = node_baru;
        cout << "Nilai Di Sisipkan Di Sebelah Kanan" << endl;
        break;
    } else {
        temp = temp -> kanan;
    }
}
}
}

pohonNode* insertRecursive(pohonNode *r, pohonNode *node_baru)
{
    if(r==NULL)
    {
        r=node_baru;
        cout <<"Penyisipan Berhasil"<<endl;
        return r;
    }

    if(node_baru->nilai < r->nilai)
    {
        r->kiri = insertRecursive(r->kiri,node_baru);
    }
    else if (node_baru->nilai > r->nilai)
    {
        r->kanan = insertRecursive(r->kanan,node_baru);
    }
    else
    {
        cout << "Nilai Duplikat Tidak Diperbolehkan" << endl;
        return r;
    }
    return r;
}

void print2D(pohonNode * r, int space) {
    if (r == NULL)
        return;
    space += SPACE; // Tingkatkan jarak antar level
    print2D(r -> kanan, space); // Memproses child dari yang kanan dulu
    cout << endl;
    for (int i = SPACE; i < space; i++)
        cout << " ";
}

```

```

    cout << r -> nilai << "\n";
    print2D(r -> kiri, space); // Memproses dari yang kiri
}

void printPreorder(pohonNode * r) //(node saat ini, kiri, kanan)
{
    if (r == NULL)
        return;
    /* Mencetak Data Node Pertama */
    cout << r -> nilai << " ";
    /* Kemudian berulang di subtree kiri */
    printPreorder(r -> kiri);
    /* Sekarang berulang di subtree Kanan */
    printPreorder(r -> kanan);
}

void printInorder(pohonNode * r) // (kiri, node saat ini, kanan)
{
    if (r == NULL)
        return;
    /* Perulangan pertama pada child sebelah kiri */
    printInorder(r -> kiri);
    /* Kemudian mencetak Data Node Pertama */
    cout << r -> nilai << " ";
    /* Sekaranh melakukan perulangan pada child sebelah kanan */
    printInorder(r -> kanan);
}

void printPostorder(pohonNode * r) //(kiri, kanan, Root)
{
    if (r == NULL)
        return;
    // Perulangan pertama pada subtree sebelah kiri
    printPostorder(r -> kiri);
    // Kemudian perulangan pada subtree sebelah kiri
    printPostorder(r -> kanan);
    // Mencetak node
    cout << r -> nilai << " ";
}

pohonNode * iterativeSearch(int v) {
    if (root == NULL) {
        return root;
    } else {
        pohonNode * temp = root;
        while (temp != NULL) {
            if (v == temp -> nilai) {
                return temp;
            } else if (v < temp -> nilai) {
                temp = temp -> kiri;
            } else {

```

```

        temp = temp -> kanan;
    }
}
return NULL;
}
}

pohonNode * recursiveSearch(pohonNode * r, int val) {
    if (r == NULL || r -> nilai == val)
        return r;

    else if (val < r -> nilai)
        return recursiveSearch(r -> kiri, val);

    else
        return recursiveSearch(r -> kanan, val);
}

int height(pohonNode * r) {
    if (r == NULL)
        return -1;
    else {
        /* Menghitung tinggi dari setiap subtree */
        int lheight = height(r -> kiri);
        int rheight = height(r -> kanan);

        /* Menggunakan nilai yang paling besar */
        if (lheight > rheight)
            return (lheight + 1);
        else return (rheight + 1);
    }
}

/* Mencatak node yang sudah diberi level */
void printGivenLevel(pohonNode * r, int level) {
    if (r == NULL)
        return;
    else if (level == 0)
        cout << r -> nilai << " ";
    else // level > 0
    {
        printGivenLevel(r -> kiri, level - 1);
        printGivenLevel(r -> kanan, level - 1);
    }
}

void printLevelOrderBFS(pohonNode * r) {
    int h = height(r);
    for (int i = 0; i <= h; i++)
        printGivenLevel(r, i);
}

```

```

pohonNode * minnilaiNode(pohonNode * node) {
    pohonNode * current = node;
    /* Loop kebawah untuk menemukan leaf paling kiri */
    while (current -> kiri != NULL) {
        current = current -> kiri;
    }
    return current;
}

pohonNode * deleteNode(pohonNode * r, int v) {
    if (r == NULL) {
        return NULL;
    }
    // Jika key yang akan dihapus lebih kecil dari key root,
    // kemudian letakkan subtree di sebelah kiri
    else if (v < r -> nilai) {
        r -> kiri = deleteNode(r -> kiri, v);
    }
    // Jika key yang akan dihapus lebih besar dari key root,
    // kemudian letakkan subtree di sebelah kanan
    else if (v > r -> nilai) {
        r -> kanan = deleteNode(r -> kanan, v);
    }
    // Jika key nya sama dengan key root, maka Ini adalah node yang akan dihapus
    else {
        // node dengan satu child atau tidak sama sekali
        if (r -> kiri == NULL) {
            pohonNode * temp = r -> kanan;
            delete r;
            return temp;
        } else if (r -> kanan == NULL) {
            pohonNode * temp = r -> kiri;
            delete r;
            return temp;
        } else {
            // Node dengan dua child: Dapatkan penerus inorder (terkecil di subtree sebelah
            // kanan)
            pohonNode * temp = minnilaiNode(r -> kanan);
            // Salin data penerus inorder ke node ini
            r -> nilai = temp -> nilai;
            // Hapus penerus inorder
            r -> kanan = deleteNode(r -> kanan, temp -> nilai);
            //deleteNode(r->kanan, temp->nilai);
        }
    }
    return r;
}

};

```

```

int main() {
    BST obj;
    int option, val;

    do {
        cout<<endl;
        cout << "Pilih Operasi Yang Di Inginkan " <<
            " Pilih Angka. Ketik 0 Untuk Keluar." << endl;
        cout << "1. Masukkan Node" << endl;
        cout << "2. Cari Node" << endl;
        cout << "3. Hapus Node" << endl;
        cout << "4. Print/Traversal Nilai BST" << endl;
        cout << "5. Height of Tree" << endl;
        cout << "6. Clear Screen" << endl;
        cout << "0. Exit Program" << endl;
        cout << "Masukkan Angka : ";

        cin >> option;
        cout<<endl;
        //Node n1;
        pohonNode * node_baru = new pohonNode();

        switch (option) {
            case 0:
                break;
            case 1:
                cout <<"MASUKKAN NILAI"<<endl;
                cout <<"Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: ";
                cin >> val;
                node_baru->nilai = val;
                obj.root= obj.insertRecursive(obj.root,node_baru);
                //obj.insertNode(node_baru);
                cout<<endl;
                break;

            case 2:
                cout << "CARI" << endl;
                cout << "Masukkan Nilai Dari TREE NODE Yang Ingin Dicari Dalam BST: ";
                cin >> val;
                //node_baru = obj.iterativeSearch(val);
                node_baru = obj.recursiveSearch(obj.root, val);
                if (node_baru != NULL) {
                    cout << "Nilai Ditemukan" << endl;
                } else {
                    cout << "Nilai Tidak Ditemukan" << endl;
                }
                break;
            case 3:
                cout << "HAPUS" << endl;

```

```

    cout << "Masukkan Nilai Dari TREE NODE Yang Ingin Dihapus Dalam BST: ";
    cin >> val;
    node_baru = obj.iterativeSearch(val);
    if (node_baru != NULL) {
        obj.deleteNode(obj.root, val);
        cout << "Nilai Dihapus" << endl;
    } else {
        cout << "Nilai Tidak Ditemukan" << endl;
    }
    break;
case 4:
    cout << "PRINT 2D: " << endl;
    obj.print2D(obj.root, 5);
    cout << endl;
    cout << "Print Level Order BFS: \n";
    obj.printLevelOrderBFS(obj.root);
    cout << endl;
    cout << "PRE-ORDER: ";
    obj.printPreorder(obj.root);
    cout << endl;
    cout << "IN-ORDER: ";
    obj.printInorder(obj.root);
    cout << endl;
    cout << "POST-ORDER: ";
    obj.printPostorder(obj.root);
    break;
case 5:
    cout << "TREE HEIGHT" << endl;
    cout << "Height : " << obj.height(obj.root) << endl;
    break;
case 6:
    system("cls");
    break;
default:
    cout << "Masukkan Angka Yang Sesuai " << endl;
}

} while (option != 0);

return 0;
}

```


Hasil Program

Input – Memasukkan Nilai Node

```
C:\Users\reizn\OneDrive\Docu  X + v

Pilih Operasi Yang Di Inginkan  Pilih Angka. Ketik 0 Untuk Keluar.
1. Masukkan Node
2. Cari Node
3. Hapus Node
4. Print/Traversal Nilai BST
5. Height of Tree
6. Clear Screen
0. Exit Program
Masukkan Angka : 1|
```

Pilih angka 1, lalu tekan *Enter*

```
C:\Users\reizn\OneDrive\Docu  X + v

Pilih Operasi Yang Di Inginkan  Pilih Angka. Ketik 0 Untuk Keluar.
1. Masukkan Node
2. Cari Node
3. Hapus Node
4. Print/Traversal Nilai BST
5. Height of Tree
6. Clear Screen
0. Exit Program
Masukkan Angka : 1

MASUKKAN NILAI
Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: |
```

Masukkan Angka BST.

```
MASUKKAN NILAI
Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: 2
Penyisipan Berhasil
```

```
MASUKKAN NILAI
Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: 5
Penyisipan Berhasil
```

MASUKKAN NILAI

Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: 6
Penyisipan Berhasil

MASUKKAN NILAI

Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: 8
Penyisipan Berhasil

MASUKKAN NILAI

Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: 9
Penyisipan Berhasil

MASUKKAN NILAI

Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: 10
Penyisipan Berhasil

MASUKKAN NILAI

Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: 11
Penyisipan Berhasil

Jika sudah memasukkan sesuai yang kita inginkan, lalu kita pilih angka no 4 untuk melihat hasilnya seperti gambar dibawah ini.

Pilih Operasi Yang Di Inginkan Pilih Angka. Ketik 0 Untuk Keluar.

1. Masukkan Node
 2. Cari Node
 3. Hapus Node
 4. Print/Traversal Nilai BST
 5. Height of Tree
 6. Clear Screen
 0. Exit Program
- Masukkan Angka : 4|

Pilih Operasi Yang Di Inginkan Pilih Angka. Ketik 0 Untuk Keluar.

1. Masukkan Node
2. Cari Node
3. Hapus Node
4. Print/Traversal Nilai BST
5. Height of Tree
6. Clear Screen
0. Exit Program

Masukkan Angka : 4

PRINT 2D:



Print Level Order BFS:

2 5 6 8 9 10 11

PRE-ORDER: 2 5 6 8 9 10 11

IN-ORDER: 2 5 6 8 9 10 11

POST-ORDER: 11 10 9 8 6 5 2

Penjelasan Kode

```
6 class pohonNode {
7     public:
8         int nilai;
9         pohonNode * kiri;
10        pohonNode * kanan;
11
12    pohonNode() {
13        nilai = 0;
14        kiri = NULL;
15        kanan = NULL;
16    }
17    pohonNode(int v) {
18        nilai = v;
19        kiri = NULL;
20        kanan = NULL;
21    }
22 };
```

Pada baris 6, terdapat sebuah class **pohonNode** yang berfungsi untuk menampung variabel public yang berisi nilai awal yang nantinya akan digunakan untuk program **Binary Search Tree**.

```
24 class BST {
25     public:
26         pohonNode * root;
27     BST() {
28         root = NULL;
29     }
30     bool isEmpty() {
31         if (root == NULL) {
32             return true;
33         } else {
34             return false;
35         }
36     }
```

Class BST berisi banyak fungsi dan variabel yang akan digunakan untuk program **Binary Search Tree**.

```

38 void insertNode(pohonNode * node_baru) {
39     if (root == NULL) {
40         root = node_baru;
41         cout << "Nilai Di Inputkan Sebagai Root Node!" << endl;
42     } else {
43         pohonNode * temp = root;
44         while (temp != NULL) {
45             if (node_baru -> nilai == temp -> nilai) {
46                 cout << "Nilai Sudah Ada," <<
47                     "Masukkan Nilai Lain!" << endl;
48                 return;
49             } else if ((node_baru -> nilai < temp -> nilai) && (temp -> kiri == NULL)) {
50                 temp -> kiri = node_baru;
51                 cout << "Nilai Di Sisipkan Di Sebelah Kiri" << endl;
52                 break;
53             } else if (node_baru -> nilai < temp -> nilai) {
54                 temp = temp -> kiri;
55             } else if ((node_baru -> nilai > temp -> nilai) && (temp -> kanan == NULL)) {
56                 temp -> kanan = node_baru;
57                 cout << "Nilai Di Sisipkan Di Sebelah Kanan" << endl;
58                 break;
59             } else {
60                 temp = temp -> kanan;
61             }
62         }
63     }
64 }

```

Pada baris ke 38, terdapat sebuah fungsi yang berguna untuk memasukkan angka node yang di dalam fungsi tersebut terdapat 2 buah parameter, yaitu **pohonNode** dan **node_baru**.

Pada baris 39-42, terdapat perintah if yang berfungsi untuk mengecek apakah nilai dari variabel root bernilai null ?. Jika root bernilai null, maka nilai root akan ditimpa dengan nilai baru, yaitu **node_baru**. Dan menampilkan teks seperti pada baris ke 41.

Pada baris 42-63 terdapat perintah else yang didalamnya terdapat perulangan **while** yang di dalamnya terdapat sebuah struktur pemilihan **if else**

```

65 | pohonNode* insertRecursive(pohonNode *r, pohonNode *node_baru)
66 | {
67 |     if(r==NULL)
68 |     {
69 |         r=node_baru;
70 |         cout << "Penyisipan Berhasil"<<endl;
71 |         return r;
72 |     }
73 |
74 |     if(node_baru->nilai < r->nilai)
75 |     {
76 |         r->kiri = insertRecursive(r->kiri,node_baru);
77 |     }
78 |     else if (node_baru->nilai > r->nilai)
79 |     {
80 |         r->kanan = insertRecursive(r->kanan,node_baru);
81 |     }
82 |     else
83 |     {
84 |         cout << "Nilai Duplikat Tidak Diperbolehkan" << endl;
85 |         return r;
86 |     }
87 |     return r;
88 | }

```

Untuk kodingan diatas, berfungsi untuk menyisipkan nilai pada tree node. Dan terdapat logika untuk mengisi node dari kiri dan kanan tree node. Juga terdapat pengecekan jika nilai mengalami duplikasi.

```

90 | void print2D(pohonNode * r, int space) {
91 |     if (r == NULL)
92 |         return;
93 |     space += SPACE; // Tingkatkan jarak antar level
94 |     print2D(r -> kanan, space); // Memproses child dari yang kanan dulu
95 |     cout << endl;
96 |     for (int i = SPACE; i < space; i++)
97 |         cout << " ";
98 |     cout << r -> nilai << "\n";
99 |     print2D(r -> kiri, space); // Memproses dari yang kiri
100 | }

```

Berfungsi untuk mencetak node dalam bentuk 2 Dimensi yang prosesnya dimulai dari kiri.

```

102 void printPreorder(pohonNode * r) //(node saat ini, kiri, kanan)
103 {
104     if (r == NULL)
105         return;
106     /* Mencetak Data Node Pertama */
107     cout << r -> nilai << " ";
108     /* Kemudian berulang di subtree kiri */
109     printPreorder(r -> kiri);
110     /* Sekarang berulang di subtree Kanan */
111     printPreorder(r -> kanan);
112 }
113

```

Untuk mencetak node yang belum di urutkan.

```

114 void printInorder(pohonNode * r) // (kiri, node saat ini, kanan)
115 {
116     if (r == NULL)
117         return;
118     /* Perulangan pertama pada child sebelah kiri */
119     printInorder(r -> kiri);
120     /* Kemudian mencetak Data Node Pertama */
121     cout << r -> nilai << " ";
122     /* Sekaranh melakukan perulangan pada child sebelah kanan */
123     printInorder(r -> kanan);
124 }

```

Untuk mencetak node yang sudah di urutkan.

```

125 void printPostorder(pohonNode * r) //(kiri, kanan, Root)
126 {
127     if (r == NULL)
128         return;
129     // Perulangan pertama pada subtree sebelah kiri
130     printPostorder(r -> kiri);
131     // Kemudian perulangan pada subtree sebelah kiri
132     printPostorder(r -> kanan);
133     // Mencetak node
134     cout << r -> nilai << " ";
135 }
136

```

Untuk mencetak node dari urutan yang paling besar.

```

137 pohonNode * iterativeSearch(int v) {
138     if (root == NULL) {
139         return root;
140     } else {
141         pohonNode * temp = root;
142         while (temp != NULL) {
143             if (v == temp -> nilai) {
144                 return temp;
145             } else if (v < temp -> nilai) {
146                 temp = temp -> kiri;
147             } else {
148                 temp = temp -> kanan;
149             }
150         }
151         return NULL;
152     }
153 }

```

Untuk melakukan pencarian pada node.

```

166 int height(pohonNode * r) {
167     if (r == NULL)
168         return -1;
169     else {
170         /* Menghitung tinggi dari setiap subtree */
171         int lheight = height(r -> kiri);
172         int rheight = height(r -> kanan);
173
174         /* Menggunakan nilai yang paling besar */
175         if (lheight > rheight)
176             return (lheight + 1);
177         else return (rheight + 1);
178     }
179 }

```

Untuk menemukan nilai tengah dari sebuah node.

```

270 switch (option) {
271     case 0:
272         break;
273     case 1:
274         cout << "MASUKKAN NILAI" << endl;
275         cout << "Masukkan Nilai Dari TREE NODE Untuk Di Masukkan Dalam BST: ";
276         cin >> val;
277         node_baru->nilai = val;
278         obj.root = obj.insertRecursive(obj.root, node_baru);
279         //obj.insertNode(node_baru);
280         cout << endl;
281         break;
282 }

```



```

282 case 2:
283     cout << "CARI" << endl;
284     cout << "Masukkan Nilai Dari TREE NODE Yang Ingin Dicari Dalam BST: ";
285     cin >> val;
286     //node_baru = obj.iterativeSearch(val);
287     node_baru = obj.recursiveSearch(obj.root, val);
288     if (node_baru != NULL) {
289         cout << "Nilai Ditemukan" << endl;
290     } else {
291         cout << "Nilai Tidak Ditemukan" << endl;
292     }
293     break;
294 case 3:
295     cout << "HAPUS" << endl;
296     cout << "Masukkan Nilai Dari TREE NODE Yang Ingin Dihapus Dalam BST: ";
297     cin >> val;
298     node_baru = obj.iterativeSearch(val);
299     if (node_baru != NULL) {
300         obj.deleteNode(obj.root, val);
301         cout << "Nilai Dihapus" << endl;
302     } else {
303         cout << "Nilai Tidak Ditemukan" << endl;
304     }
305     break;
306 case 4:
307     cout << "PRINT 2D: " << endl;
308     obj.print2D(obj.root, 5);
309     cout << endl;
310     cout << "Print Level Order BFS: \n";
311     obj.printLevelOrderBFS(obj.root);
312     cout << endl;
313     cout << "PRE-ORDER: ";
314     obj.printPreorder(obj.root);
315     cout << endl;
316     cout << "IN-ORDER: ";
317     obj.printInorder(obj.root);
318     cout << endl;
319     cout << "POST-ORDER: ";
320     obj.printPostorder(obj.root);
321     break;
322 case 5:
323     cout << "TREE HEIGHT" << endl;
324     cout << "Height : " << obj.height(obj.root) << endl;
325     break;
326 case 6:
327     system("cls");
328     break;
329 default:
330     cout << "Masukkan Angka Yang Sesuai " << endl;
331 }
332
333 } while (option != 0);
334

```

Kodingan diatas merupakan sebuah logika struktur pemilihan selain if else yang berfungsi untuk melakukan proses sesuai inputan yang di masukkan oleh user.