LAPORAN STRUKTUR DATA

KELOMPOK 6

FUNGSI UNTUK MEMBANGUN SEBUAH BINARY SEARCH TREE

Anggota Kelompok :

1. FIFIT SYAFAATY (21091397001)
2. NUR HASLINDA (21091397035)
3. RIZQI CAHYA ANGELITA (21091397047)
4. DWI RAMADHANIASARI (21091397057)
5. PUTRI DWINATRYSKA A.R.F (21091397075)

Konsep / Cara Kerja :

- **Binary Search Tree**

  Binary Search Tree merupakan struktur data pohon biner berbasis node yang memiliki property berikut :

  Sub Tree kiri dari sebuah node hanya berisi node dengan nilai lebih kecil dari nilai node.

  Sub Tree kanan dari sebuah node hanya berisi node dengan kunci nilai besar dari nilai node.

  Sub Tree kiri dan kanan masing masing juga harus berupa pohon pencarian biner.

  Properti Binary Search Tree di atas menyediakan urutan antar Nilai sehingga operasi seperti pencarian, minimum dan maksimum dapat dilakukan dengan cepat. Jika tidak ada pengurutan, maka kita mungkin harus membandingkan setiap Nilai untuk mencari Nilai yang diberikan.

Codingan C++ :

BST BSF.cpp

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

using namespace std;

//pendeklarasian class sebuah tree
class TreeNode {
  public:
    int value;
  TreeNode * left;
  TreeNode * right;

  TreeNode() {
    value = 0;
    left = NULL;
    right = NULL;
  }
  TreeNode(int v) {
    value = v;
    left = NULL;
    right = NULL;
  }
};

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

//fungsi untuk menambahkan node baru
  void insertNode(TreeNode * new_node) {
    //jika root masih kosong
    if (root == NULL) {
      //pengalokasian memori dari node yang telah dibuat
      root = new_node;
      cout << "Value Inserted as root node!" << endl;
    } else {
      TreeNode * temp = root;
      while (temp != NULL) {
        if (new_node -> value == temp -> value) {
          cout << "Value Already exist," <<
            "Insert another value!" << endl;
          return;
        } else if ((new_node -> value < temp -> value) && (temp -> left == NULL)) {
          temp -> left = new_node;
          cout << "Value Inserted to the left!" << endl;
          break;
        } else if (new_node -> value < temp -> value) {
          temp = temp -> left;
        } else if ((new_node -> value > temp -> value) && (temp -> right == NULL)) {
          temp -> right = new_node;
          cout << "Value Inserted to the right!" << endl;
          break;
        } else {
          temp = temp -> right;
        }
      }
```

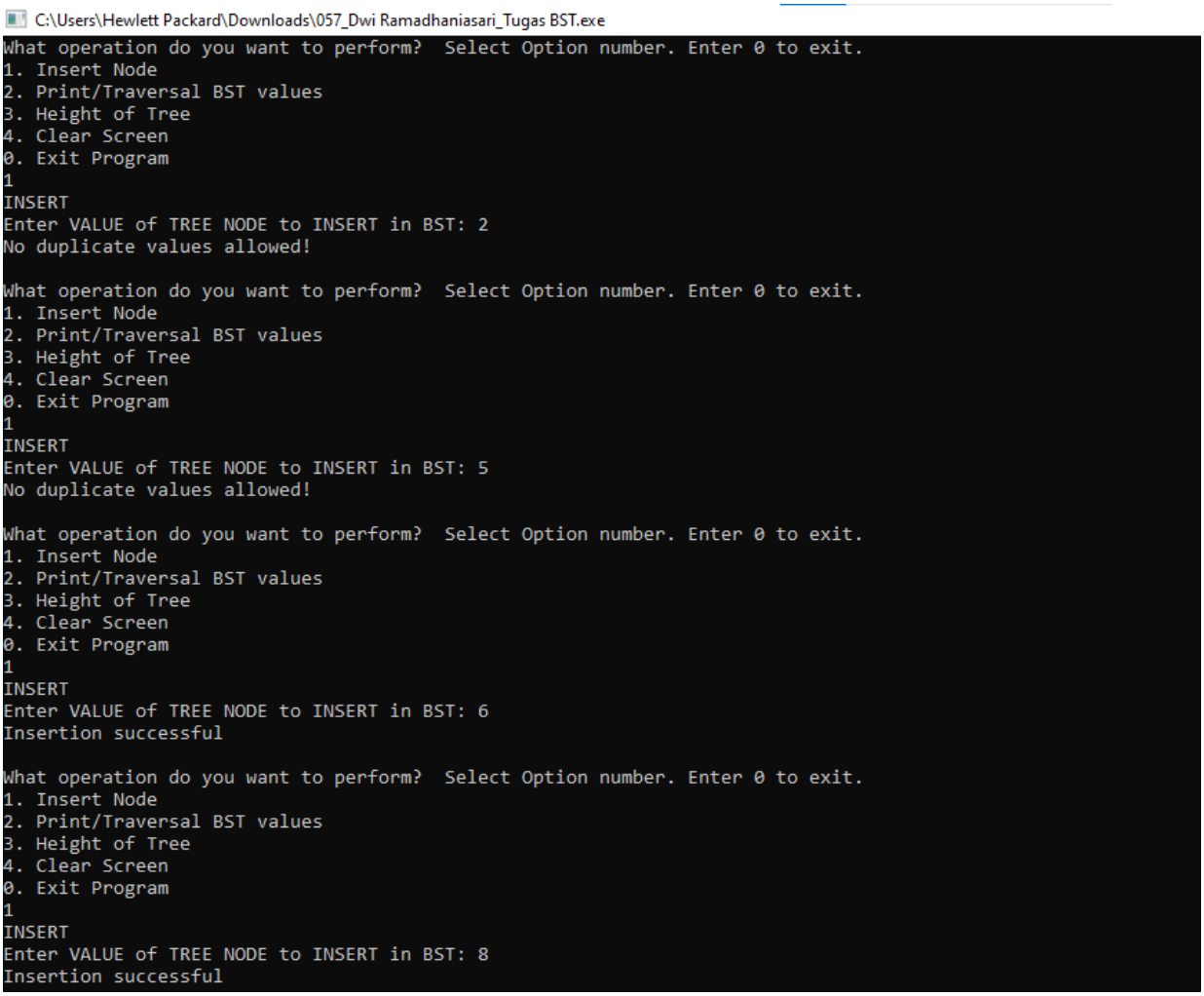
```
    }
  }
   TreeNode* insertRecursive(TreeNode *r, TreeNode *new_node)
   {
       if(r==NULL)
       {
           r=new_node;
           cout <<"Insertion successful"<<endl;
           return r;
       }

       if(new_node->value < r->value)
       {
           r->left = insertRecursive(r->left,new_node);
       }
       else if (new_node->value > r->value)
       {
           r->right = insertRecursive(r->right,new_node);
       }
      else
       {
           cout << "No duplicate values allowed!" << endl;
           return r;
       }
       return r;
   }

  void print2D(TreeNode * r, int space) {
    if (r == NULL) // Base case  1
      return;
    space += SPACE; // memperluas jarak antara level 2
    print2D(r -> right, space); // Proses anak kanan dulu 3
    cout << endl;
```

```
    for (int i = SPACE; i < space; i++) // 5
      cout << " "; // 5.1
    cout << r -> value << "\n"; // 6
    print2D(r -> left, space); // proses anak kiri 7
  }

  void printPreorder(TreeNode * r) //(simpul saat ini, Kiri, Kanan)
  {
    if (r == NULL)
      return;
    /* mencetak data simpul pertama*/
    cout << r -> value << " ";
    /* kemudian muncul kembali di subpohon kiri */
    printPreorder(r -> left);
    /* sekarang muncul kembali di subpohon kanan*/
    printPreorder(r -> right);
  }

  void printInorder(TreeNode * r) //  (kiri, simpul terkini, kanan)
  {
    if (r == NULL)
      return;
    /* pertama muncul kembali di anak kiri*/
    printInorder(r -> left);
    /* kemudian cetak data di simpul */
    cout << r -> value << " ";
    /* sekarang muncul kembali di anak kanan*/
    printInorder(r -> right);
  }
  void printPostorder(TreeNode * r) //(kiri, kanan, akar)
  {
    if (r == NULL)
      return;
```

```
    // pertama muncul kembali di subpohon kiri
    printPostorder(r -> left);
    // kemudian muncul kembali di subpohon kanan
    printPostorder(r -> right);
    // sekarang sepakat dengan simpul
    cout << r -> value << " ";
  }

  TreeNode * iterativeSearch(int v) {
    if (root == NULL) {
      return root;
    } else {
      TreeNode * temp = root;
      while (temp != NULL) {
        if (v == temp -> value) {
          return temp;
        } else if (v < temp -> value) {
          temp = temp -> left;
        } else {
          temp = temp -> right;
        }
      }
      return NULL;
    }
  }

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

    else if (val < r -> value)
      return recursiveSearch(r -> left, val);
```

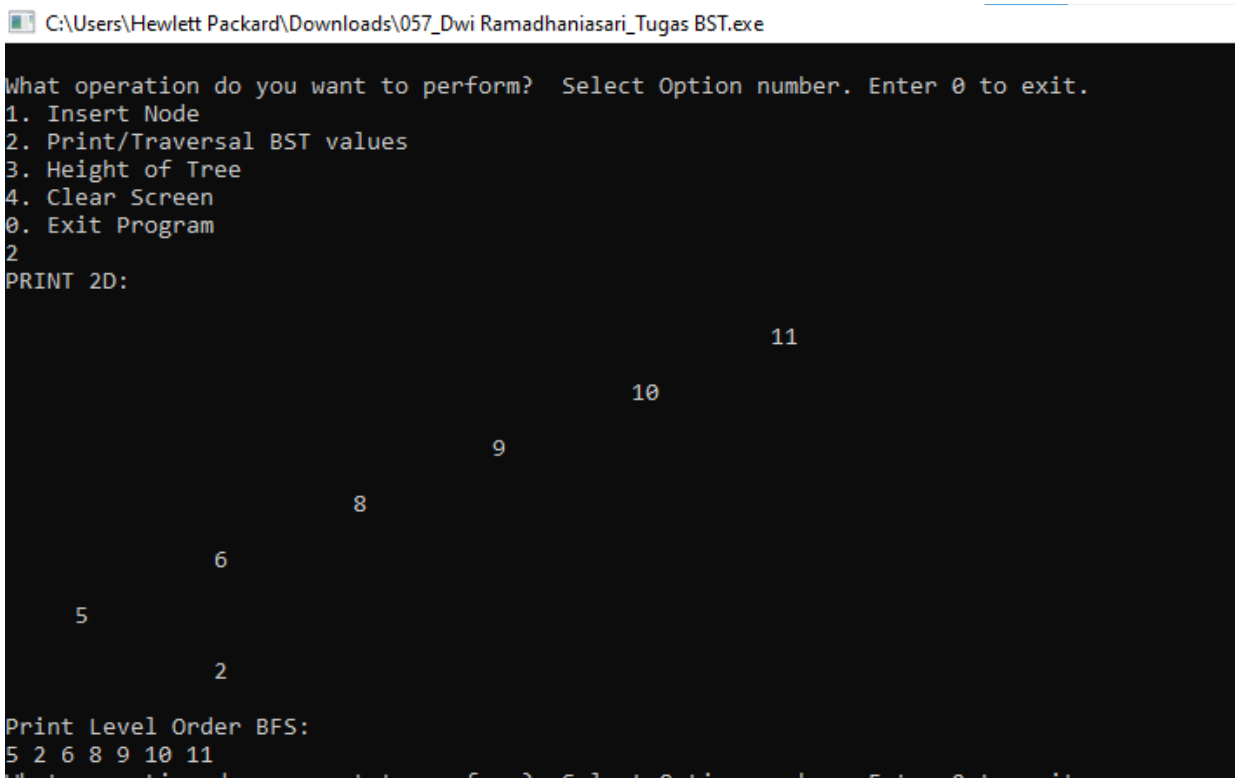
```
    else
      return recursiveSearch(r -> right, val);
  }

  int height(TreeNode * r) {
    if (r == NULL)
      return -1;
    else {
      /* tinggi komputer satu sama lain/ persaaman tinggi sampul kanan / kiri*/
      int lheight = height(r -> left);
      int rheight = height(r -> right);

      /* menggunakan yang terbesar*/
      if (lheight > rheight)
        return (lheight + 1);
      else return (rheight + 1);
    }
  }

  /* mencetak simpul pada level */
  void printGivenLevel(TreeNode * r, int level) {
    if (r == NULL)
      return;
    else if (level == 0)
      cout << r -> value << " ";
    else // level > 0
    {
      printGivenLevel(r -> left, level - 1);
      printGivenLevel(r -> right, level - 1);
    }
  }
  void printLevelOrderBFS(TreeNode * r) {
    int h = height(r);

    for (int i = 0; i <= h; i++)
      printGivenLevel(r, i);
  }

  TreeNode * minValueNode(TreeNode * node) {
    TreeNode * current = node;
    /* perulangan untuk menemukan daun paling kiri */
    while (current -> left != NULL) {
      current = current -> left;
    }
    return current;
  }

};

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

  do {
    cout << "What operation do you want to perform? " <<
      " Select Option number. Enter 0 to exit." << endl;
    cout << "1. Insert Node" << endl;
    cout << "2. Print/Traversal BST values" << endl;
    cout << "3. Height of Tree" << endl;
    cout << "4. Clear Screen" << endl;
    cout << "0. Exit Program" << endl;

    cin >> option;
    //simpul n1;
    TreeNode * new_node = new TreeNode();

    switch (option) {

    case 0:
      break;
    case 1:
        cout <<"INSERT"<<endl;
          cout <<"Enter VALUE of TREE NODE to INSERT in BST: ";
          cin >> val;
          new_node->value = val;
          obj.root= obj.insertRecursive(obj.root,new_node);
          cout<<endl;
            break;


    case 2:
      cout << "PRINT 2D: " << endl;
      obj.print2D(obj.root, 5);
      cout << endl;
      cout << "Print Level Order BFS: \n";
      obj.printLevelOrderBFS(obj.root);
      cout << endl;

      break;
    case 3:
      cout << "TREE HEIGHT" << endl;
      cout << "Height : " << obj.height(obj.root) << endl;
      break;
    case 4:
      system("cls");
      break;
    default:
      cout << "Enter Proper Option number " << endl;
    }

  } while (option != 0);

  return 0;
}
```

Input & Output :



C:\Users\Hewlett Packard\Downloads\057_Dwi Ramadhaniasari_Tugas BST.exe

```
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 8
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Print/Traversal BST values
3. Height of Tree
4. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 9
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Print/Traversal BST values
3. Height of Tree
4. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 10
Insertion successful

What operation do you want to perform?  Select Option number. Enter 0 to exit.
1. Insert Node
2. Print/Traversal BST values
3. Height of Tree
4. Clear Screen
0. Exit Program
1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 11
Insertion successful
```


C:\Users\Hewlett Packard\Downloads\057_Dwi Ramadhaniasari_Tugas BST.exe
What operation do you want to perform? Select Option number. Enter 0 to exit.
1. Insert Node
2. Print/Traversal BST values
3. Height of Tree
4. Clear Screen
0. Exit Program
2
PRINT 2D:
11
10
9
8
6
5
2
Print Level Order BFS:
5 2 6 8 9 10 11