Laporan Binary Search Tree Kelompok 3

Oleh:

Muvidha Fatmawati Putri	(21091397011)
Diah Ayuning Tyas	(21091397013)
Shandy Ilham Alamsyah	(21091397015)
Fisma Meividianugraha Subani	(21091397017)
Alvin Febrianto	(21091397031)

Fakultas Vokasi

D4 Manajemen Informatika

UNIVERSITAS NEGERI SURABAYA

TAHUN AJARAN 2021/2022

```cpp
#include <iostream>
#define SPACE 10
using namespace std;

// deklarasi 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;
      }
```

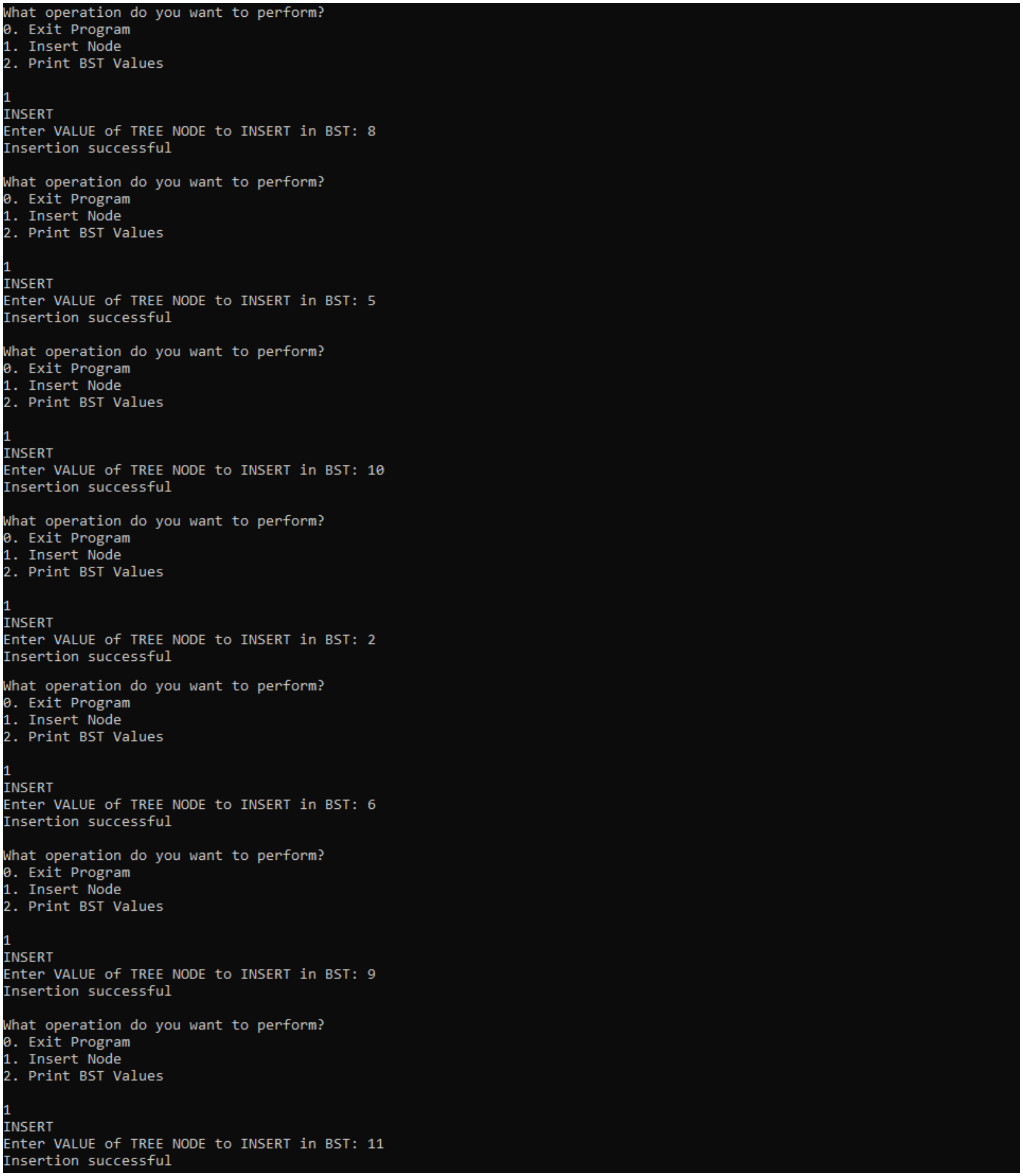
```
void print2D(TreeNode * r, int space) {
    // base case 1
    if (r == NULL)
      return;
    // memperluas jarak antara level 2
    space += SPACE;
    // proses anak kanan dulu 3
    print2D(r -> right, space);
    cout << endl;
    // 5
    for (int i = SPACE; i < space; i++)
      // 5.1
      cout << " ";
    // 6
    cout << r -> value << "\n";
    // proses anak kiri 7
    print2D(r -> left, space);
  }

  // simpul saat ini, kiri, kanan
  void printPreorder(TreeNode * r)
  {
    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);
  }

  // kiri, simpul terkini, kanan
  void printInorder(TreeNode * r)
  {
    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);
  }
```

```
// kiri, kanan, akar
void printPostorder(TreeNode * r)
{
  if (r == NULL)
    return;
  // pertama muncul kembali di subpohon kiri
  printPostorder(r -> left);
  // kemudian muncul kembali di subpohon kanan
  printPostorder(r -> right);
  // sekarang sepaket 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
  void printGivenLevel(TreeNode * r, int level) {
    if (r == NULL)
      return;
    else if (level == 0)
      cout << r -> value << " ";
    // level > 0
    else
    {
      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;
  }

};
```

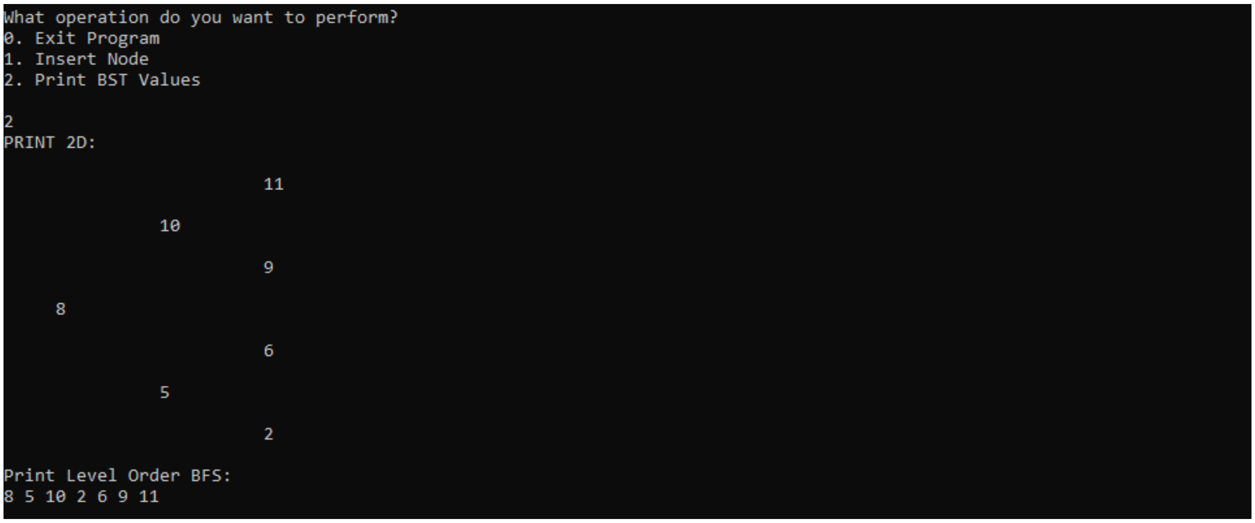
```
int main() {
  BST obj;
  int option, val;

  do {
    cout << "What operation do you want to perform?" << endl;
    cout << "0. Exit Program" << endl;
      cout << "1. Insert Node" << endl;
    cout << "2. Print BST Values" << endl;
    cout << 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 << "\n\n";
      break;
    default:
      cout << "Enter Proper Option Number\n\n";
    }
  }
  while (option != 0);

  return 0;
}
```

- **Contoh Input:**



```
What operation do you want to perform?
0. Exit Program
1. Insert Node
2. Print BST Values

1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 8
Insertion successful

What operation do you want to perform?
0. Exit Program
1. Insert Node
2. Print BST Values

1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 5
Insertion successful

What operation do you want to perform?
0. Exit Program
1. Insert Node
2. Print BST Values

1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 10
Insertion successful

What operation do you want to perform?
0. Exit Program
1. Insert Node
2. Print BST Values

1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 2
Insertion successful
What operation do you want to perform?
0. Exit Program
1. Insert Node
2. Print BST Values

1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 6
Insertion successful

What operation do you want to perform?
0. Exit Program
1. Insert Node
2. Print BST Values

1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 9
Insertion successful

What operation do you want to perform?
0. Exit Program
1. Insert Node
2. Print BST Values

1
INSERT
Enter VALUE of TREE NODE to INSERT in BST: 11
Insertion successful
```

- **Hasil Output:**



```
What operation do you want to perform?
0. Exit Program
1. Insert Node
2. Print BST Values

2
PRINT 2D:

                         11

               10

                         9

     8

                         6

               5

                         2

Print Level Order BFS:
8 5 10 2 6 9 11
```