**LAPORAN PRAKTIKUM**

**“Pertemuan ke-10: Post Test - Sequence”**

Diajukan untuk memenuhi salah satu praktikum Mata Struktur Data Informatika yang di ampu oleh:

 Dr., Ardiansyah, S.T., M.Cs.

Disusun Oleh:

Mohammad Farid Hendianto 2200018401

A / Selasa 09.30 – 11.30 Komputasi Dasar

**PROGRAM STUDI INFORMATIKA**

**UNIVERSITAS AHMAD DAHLAN**

**FAKULTAS TEKNOLOGI INDUSTRI**

**TAHUN 2023**

1. Dalam contoh di landasan teori buatlah implementasi dalam bahasa pemrogaman.

Jawab:

Dengan bahasa pemrograman c++, berikut adalah source code kodingan.



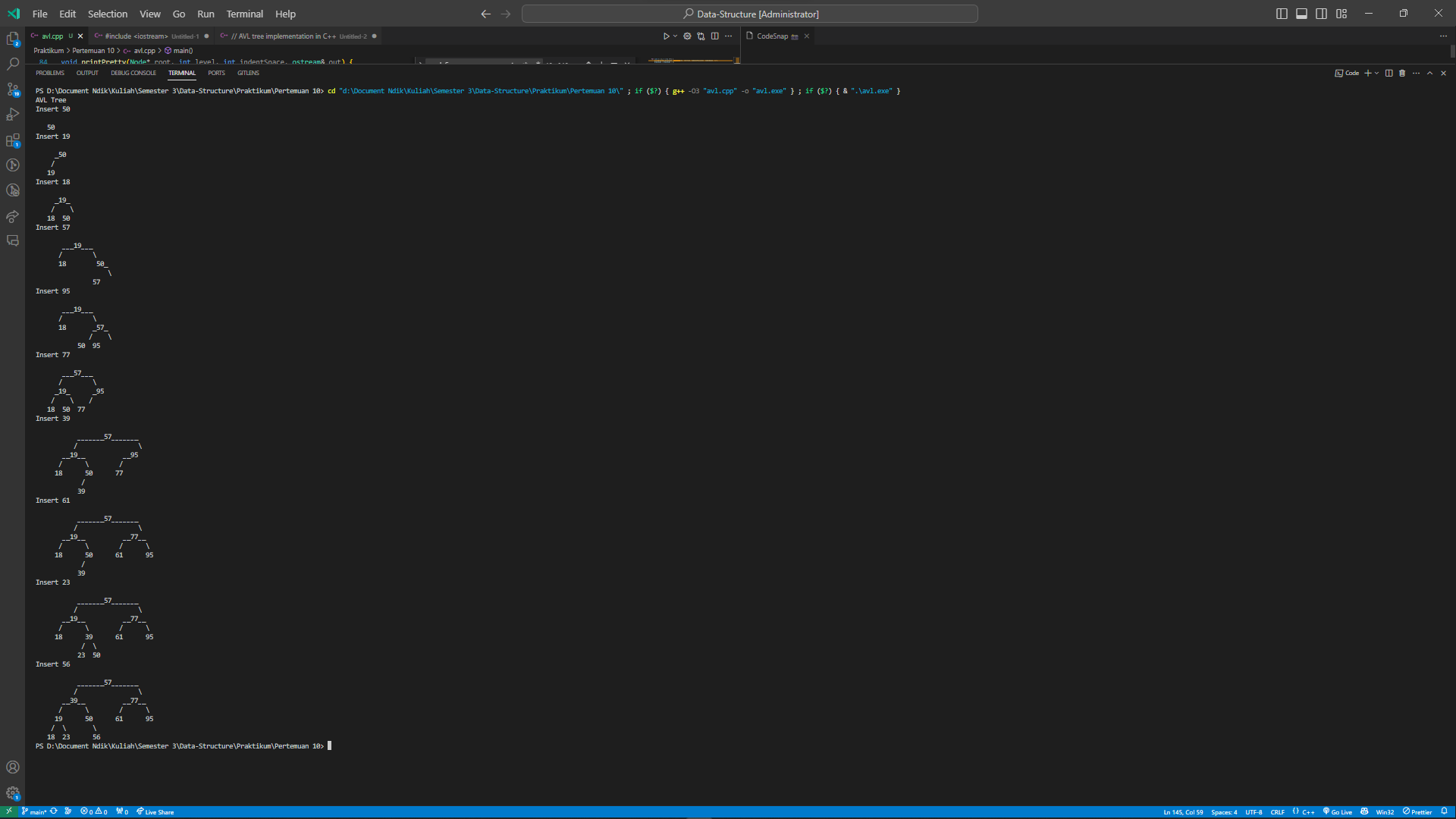


**Program jalan (Screenshot input output)**

**Input**

|  |
| --- |
| cout << "AVL Tree" << endl;  cout << "Insert 50" << endl;  root = insert(root, 50);printPretty(root, 1, 3, cout);  cout << "Insert 19" << endl;  root = insert(root, 19);printPretty(root, 1, 3, cout);  cout << "Insert 18" << endl;  root = insert(root, 18);printPretty(root, 1, 3, cout);  cout << "Insert 57" << endl;  root = insert(root, 57);printPretty(root, 1, 3, cout);  cout << "Insert 95" << endl;  root = insert(root, 95);printPretty(root, 1, 3, cout);  cout << "Insert 77" << endl;  root = insert(root, 77);printPretty(root, 1, 3, cout);  cout << "Insert 39" << endl;  root = insert(root, 39);printPretty(root, 1, 3, cout);  cout << "Insert 61" << endl;  root = insert(root, 61);printPretty(root, 1, 3, cout);  cout << "Insert 23" << endl;  root = insert(root, 23);printPretty(root, 1, 3, cout);  cout << "Insert 56" << endl;  root = insert(root, 56);printPretty(root, 1, 3, cout); |

**Output**

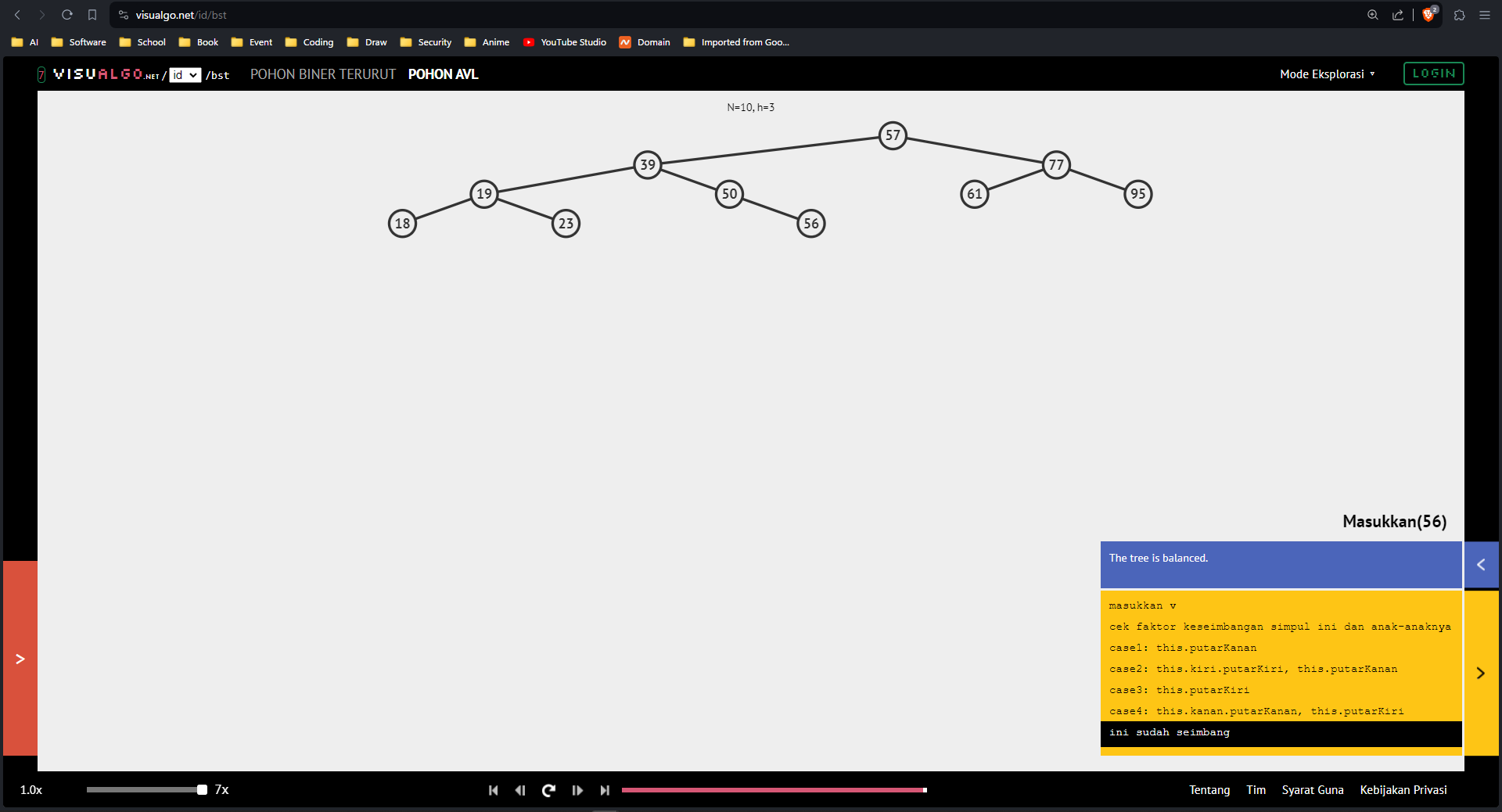


Berikut output lebih jelasnya

|  |
| --- |
| AVL Tree  Insert 50  50  Insert 19  \_50  /  19  Insert 18  \_19\_  / \  18 50  Insert 57  \_\_\_19\_\_\_  / \  18 50\_  \  57  Insert 95  \_\_\_19\_\_\_  / \  18 \_57\_  / \  50 95  Insert 77  \_\_\_57\_\_\_  / \  \_19\_ \_95  / \ /  18 50 77  Insert 39  \_\_\_\_\_\_\_57\_\_\_\_\_\_\_  / \  \_\_19\_\_ \_\_95  / \ /  18 50 77  /  39  Insert 61  \_\_\_\_\_\_\_57\_\_\_\_\_\_\_  / \  \_\_19\_\_ \_\_77\_\_  / \ / \  18 50 61 95  /  39  Insert 23  \_\_\_\_\_\_\_57\_\_\_\_\_\_\_  / \  \_\_19\_\_ \_\_77\_\_  / \ / \  18 39 61 95  / \  23 50  Insert 56  \_\_\_\_\_\_\_57\_\_\_\_\_\_\_  / \  \_\_39\_\_ \_\_77\_\_  / \ / \  19 50 61 95  / \ \  18 23 56 |

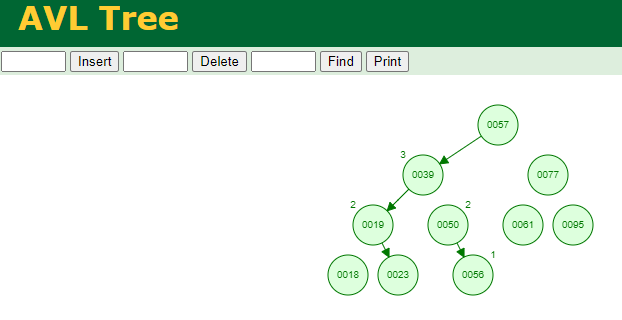
Pembuktian

|  |
| --- |
| let data = [50, 19, 18, 57, 95, 77, 39, 61, 23, 56];  for (let i = 0; i < data.length; i++) {  setTimeout(function() {  $('#v-insert').val(data[i]);  $('#insert-go').click();  }, i \* 100);  } |



<https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>

|  |
| --- |
| // Buat struktur data array  let data = [50, 19, 18, 57, 95, 77, 39, 61, 23, 56];  // Dapatkan elemen input dan tombol dari DOM  let insertInput = document.querySelectorAll('#AlgorithmSpecificControls input[type="text"]')[0];  let insertButton = document.querySelectorAll('#AlgorithmSpecificControls input[type="button"]')[0];  // Fungsi untuk memasukkan data secara otomatis  function insertDataAutomatically() {  // Lakukan untuk setiap item data  data.forEach((item, index) => {  // Setel nilai input dengan item data  insertInput.value = item;  // Buat event click  let clickEvent = new Event('click');  // Kirim event click ke tombol insert  insertButton.dispatchEvent(clickEvent);  });  }  // Tambahkan event listener ke tombol insert  insertButton.addEventListener('click', function() {  // Dapatkan nilai dari input dan ubah menjadi integer  let inputValue = parseInt(insertInput.value);  // Tambahkan nilai input ke array data  data.push(inputValue);  // Kosongkan input  insertInput.value = '';  // Cetak data ke console untuk verifikasi  console.log(data);  });  // Panggil fungsi untuk memasukkan data secara otomatis  insertDataAutomatically(); |



**Penjelasan program**

Program ini adalah implementasi dari struktur data AVL Tree dalam bahasa pemrograman C++. AVL Tree adalah jenis binary search tree yang self-balancing, yang berarti setiap kali node ditambahkan atau dihapus, tree akan menyeimbangkan dirinya sendiri.

Berikut adalah penjelasan fungsi dan class dalam program:

* `class Node`: Class ini digunakan untuk merepresentasikan node dalam AVL Tree. Setiap node memiliki kunci (key), tinggi (height), dan dua pointer ke node anak kiri dan kanan.
* `height(Node \*N)`: Fungsi ini mengembalikan tinggi dari node N. Jika N adalah nullptr, maka mengembalikan 0.
* `max(int a, int b)`: Fungsi ini mengembalikan nilai maksimum antara a dan b.
* `rotateRight(Node \*y) dan rotateLeft(Node \*x)`: Fungsi ini melakukan rotasi ke kanan dan ke kiri pada node y dan x. Rotasi ini digunakan untuk menyeimbangkan AVL Tree.
* `getBalance(Node \*N)`: Fungsi ini mengembalikan balance factor dari node N, yaitu perbedaan tinggi antara subtree kiri dan kanan.
* `insert(Node \*node, int key)`: Fungsi ini memasukkan node baru dengan kunci tertentu ke dalam AVL Tree. Setelah penambahan, fungsi ini juga memeriksa balance factor dan melakukan rotasi jika diperlukan. Setelah memasukkan node, fungsi ini juga melakukan pengecekan dan rotasi jika diperlukan untuk menjaga tree tetap seimbang. Rotasi dilakukan berdasarkan kondisi balance factor dari node:
* if (balance > 1 && key < node->left->key) return rotateRight(node);: Jika balance factor lebih dari 1 dan kunci node yang akan dimasukkan lebih kecil dari kunci node anak kiri, maka dilakukan rotasi ke kanan pada node tersebut. Ini disebut kasus Left-Left.
* if (balance < -1 && key > node->right->key) return rotateLeft(node);: Jika balance factor kurang dari -1 dan kunci node yang akan dimasukkan lebih besar dari kunci node anak kanan, maka dilakukan rotasi ke kiri pada node tersebut. Ini disebut kasus Right-Right.
* if (balance > 1 && key > node->left->key) { node->left = rotateLeft(node->left); return rotateRight(node); }: Jika balance factor lebih dari 1 dan kunci node yang akan dimasukkan lebih besar dari kunci node anak kiri, maka dilakukan rotasi ke kiri pada node anak kiri, diikuti dengan rotasi ke kanan pada node tersebut. Ini disebut kasus Left-Right.
* if (balance < -1 && key < node->right->key) { node->right = rotateRight(node->right); return rotateLeft(node); }: Jika balance factor kurang dari -1 dan kunci node yang akan dimasukkan lebih kecil dari kunci node anak kanan, maka dilakukan rotasi ke kanan pada node anak kanan, diikuti dengan rotasi ke kiri pada node tersebut. Ini disebut kasus Right-Left.
* `printBranches, printNodes, printLeaves, printPretty`: Fungsi-fungsi ini digunakan untuk mencetak AVL Tree ke console dengan format yang indah dan mudah dibaca.
* `main()`: Fungsi utama program ini. Membuat AVL Tree dan memasukkan beberapa node ke dalamnya, lalu mencetak tree setelah setiap penambahan.

Program ini mencetak AVL Tree setelah setiap penambahan node, sehingga Anda dapat melihat bagaimana tree menyeimbangkan dirinya sendiri setelah setiap operasi insert.

Untuk mengakses source kodingan, dapat melihat link berikut:

<https://github.com/IRedDragonICY/Data-Structure>