

**Name : Zeeshan Ali**

**Roll no: SU92-BSITM-F22-019**

**Data Structure and Algorithm(Lab)**

**Final Paper**

**Question no 01:**

**Code:**

```
// Question no 01
#include<iostream>
#include <queue>
using namespace std;
class BST{
    int data;
    BST* left;
    BST* right;
public:
    BST();
    BST(int);
    BST* insert(BST* ,int );
    BST* Delete (BST*,int);
    void preorder(BST*);
    void inorder(BST*);
    void postorder(BST*);
    void DFS_preorder(BST*);
    void DFS_inorder(BST*);
    void DFS_postorder(BST*);
    void BFS_LevelOrder(BST*);
};
```

```

BST :: BST(){
    data=0;
    left=NULL;
    right=NULL;
}

BST :: BST(int value){
    data=value;
    left=NULL;
    right=NULL;
}

BST* BST::insert(BST* root,int value){
    if(root==NULL){
        return new BST(value);
    }
    if(value<=root->data){
        root->left=insert(root->left,value);
    }
    else if(value>root->data){
        root->right=insert(root->right,value);
    }
    return root;
}

void BST::preorder(BST* root){
    if(root==NULL){
        return;
    }
    cout<<root->data<<" ";

```

```

        preorder(root->left);
        preorder(root->right);
    }

void BST::inorder(BST* root){
    if(root==NULL){
        return;
    }
    inorder(root->left);
    cout<<root->data<<" ";
    inorder(root->right);
}

void BST::postorder(BST* root){
    if(root==NULL){
        return;
    }
    postorder(root->left);
    postorder(root->right);
    cout<<root->data<<" ";
}

BST* BST :: Delete (BST* root , int value){
    if(root==NULL){
        return root;
    }
    else{
        if(value<=root->data){
            root->left=Delete(root->left,value);
        }
    }
}

```

```

        else if(value>root->data){
            root->right=Delete(root->right,value);
        }else {
if (root->left == NULL) {
    BST* temp = root->right;
    delete root;
    return temp;
} else if (root->right == NULL) {
    BST* temp = root->left;
    delete root;
    return temp;
}
    BST* temp = root->right;
    while (temp->left != NULL) {
        temp = temp->left;
    }
    root->data = temp->data;
    root->right = Delete(root->right, temp->data);
}
return root;
}
}

void BST::BFS_LevelOrder(BST* root) {
    if (root == NULL) {
        return;
    }
}

```

```

queue<BST*> q;
q.push(root);

while (q.empty()!=NULL) {
    BST* current = q.front();
    cout << current->data << " ";
    if (current->left != NULL) {
        q.push(current->left);
    }
    if (current->right != NULL) {
        q.push(current->right);
    }
    q.pop();
}
}

int main(){
    BST b, *root=NULL;
    root=b.insert(root,10);
    b.insert(root,5);
    b.insert(root,15);
    b.insert(root,3);
    b.insert(root,7);
    b.insert(root,12);
    b.insert(root,18);
    b.insert(root,2);
    b.insert(root,4);
    b.insert(root,6);

```

```

        b.insert(root,8);
        b.insert(root,11);
        b.insert(root,14);
        b.insert(root,13);
        b.insert(root,19);
        b.insert(root,20);
        b.insert(root,25);
// Now inserting the said nodes
        b.insert(root,25);
        b.insert(root,12);
cout << "a. In Order Traversal" << endl;
b.inorder(root);
cout << "\nb. Pre Order Traversal" << endl;
b.preorder(root);
cout << "\nc. Post Order Traversal" << endl;
b.postorder(root);
cout << "\nd. Level Order Traversal" << endl;
b.BFS_LevelOrder(root);
        cout << "\ne. Node to be deleted is 15 " << endl;
        int key_to_be_deleted = 15;
        root = b.Delete(root,key_to_be_deleted);
        cout << "Node Has been deleted " << endl;

        return 0;
}

```

**Output:**

A screenshot of a C++ IDE (Dev-C++) showing a program execution. The main window displays the output of an AVL tree program. The output lists five operations: a. In Order Traversal, b. Pre Order Traversal, c. Post Order Traversal, d. Level Order Traversal, and e. Node to be deleted is 15. The output shows the resulting tree structure for each operation. The IDE interface includes a menu bar, a toolbar, and a status bar at the bottom showing the current line and column numbers.

```
D:\University Relative\3rd Semester\Data Structure & Algorithm Lab\New folder\Ques 1.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug Ques 1.cpp Ques 2.cpp Ques 3.cpp
129 b.insert(root,6);
130 b.insert(root,8);
D:\University Relative\3rd Semester\Data Structure & Algorithm Lab\New folder\Ques 1.exe
a. In Order Traversal
2 3 4 5 6 7 8 10 11 12 12 13 14 15 18 19 20 25 25
b. Pre Order Traversal
10 5 3 2 4 7 6 8 15 12 11 12 14 13 18 19 20 25 25
c. Post Order Traversal
2 4 3 6 8 7 5 12 11 13 14 12 25 25 20 19 18 15 10
d. Level Order Traversal
10 5 15 3 7 12 18 2 4 6 8 11 14 19 12 13 20 25 25
e. Node to be deleted is 15
Node Has been deleted
-----
Process exited after 0.5527 seconds with return value 0
Press any key to continue . . .
```

## Question no 02:

### Code:

/\*You'll need to implement the member functions of the AVLTree

class (e.g., insert, getHeight, getBalanceFactor, rightRotate, leftRotate, and Show).

These functions are responsible for inserting nodes into the

AVL tree, balancing it, and Showing the tree structure.\*/

```
#include <iostream>
```

```
using namespace std;
```

```
class AVL {
```

```
    int data;
```

```
    AVL *left, *right;
```

```
    int height;
```

```
public:
```

```
    AVL();
```

```
    AVL(int);
```

```

AVL* Insert(AVL*, int);
AVL* Delete(AVL*, int);
void Inorder(AVL*);
void preorder(AVL*);
void postorder(AVL*);
int getHeight(AVL* node);
int getBalance(AVL* node);
AVL* rightRotate(AVL* y);
AVL* leftRotate(AVL* x);
AVL* minValueNode(AVL* node);
};

AVL::AVL() {
    data = 0;
    left = right = NULL;
    height = 1;
}

AVL::AVL(int value) {
    data = value;
    left = right = NULL;
    height = 1;
}

AVL* AVL::Insert(AVL* root, int value) {
    if (root == NULL) {

        return new AVL(value);
    }

    if (value < root->data) {

```



```

    root->left = Insert(root->left, value);
} else if (value > root->data) {

    root->right = Insert(root->right, value);
}
root->height = 1 + max(getHeight(root->left), getHeight(root->right));
int balance = getBalance(root);

if (balance > 1 && value < root->left->data) {
    return rightRotate(root);
}
if (balance < -1 && value > root->right->data) {
    return leftRotate(root);
}
if (balance > 1 && value > root->left->data) {
    root->left = leftRotate(root->left);
    return rightRotate(root);
}
if (balance < -1 && value < root->right->data) {
    root->right = rightRotate(root->right);
    return leftRotate(root);
}
return root;
}

int AVL::getHeight(AVL* node) {
    if (node == NULL) {

```

```

        return 0;
    }
    return node->height;
}

int AVL::getBalance(AVL* node) {
    if (node == NULL) {
        return 0;
    }
    return getHeight(node->left) - getHeight(node->right);
}

AVL* AVL::rightRotate(AVL* y) {
    AVL* x = y->left;
    AVL* T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = 1 + max(getHeight(y->left), getHeight(y->right));
    x->height = 1 + max(getHeight(x->left), getHeight(x->right));
    return x;
}

AVL* AVL::leftRotate(AVL* x) {
    AVL* y = x->right;
    AVL* T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = 1 + max(getHeight(x->left), getHeight(x->right));
    y->height = 1 + max(getHeight(y->left), getHeight(y->right));
    return y;
}

```

```

}

AVL* AVL::minValueNode(AVL* node) {
    AVL* current = node;

    while (current->left != NULL) {
        current = current->left;
    }

    return current;
}

void AVL::Inorder(AVL* root) {
    if (root == NULL) {
        return;
    }
    Inorder(root->left);
    cout << root->data << " ";
    Inorder(root->right);
}

void AVL::preorder(AVL* root) {
    if (root == NULL) {
        return;
    }
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}

```

```

void AVL::postorder(AVL* root) {
    if (root == NULL) {
        return;
    }
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
}

int main() {
    AVL avl, *root = NULL;
    root = avl.Insert(root, 25);
    avl.Insert(root,15);
    avl.Insert(root,35);
    avl.Insert(root,10);
    avl.Insert(root,20);
    avl.Insert(root,30);
    avl.Insert(root,40);
    avl.Insert(root,5);
    avl.Insert(root,12);
    avl.Insert(root,18);
    avl.Insert(root,22);
    avl.Insert(root,28);
    avl.Insert(root,32);
    avl.Insert(root,38);
    avl.Insert(root,45);
    avl.Insert(root,2);
    avl.Insert(root,8);
}

```

```
    avl.Insert(root,26);
    avl.Insert(root,36);
    avl.Insert(root,42);
    avl.Insert(root,24);
    avl.Insert(root,39);
    avl.Insert(root,8);
    avl.Insert(root,23);

    cout << "AVL Tree after all the balancing " << endl;

    cout << "Inorder traversal" << endl;

    avl.Inorder(root);

    cout << "\nPreorder traversal" << endl;

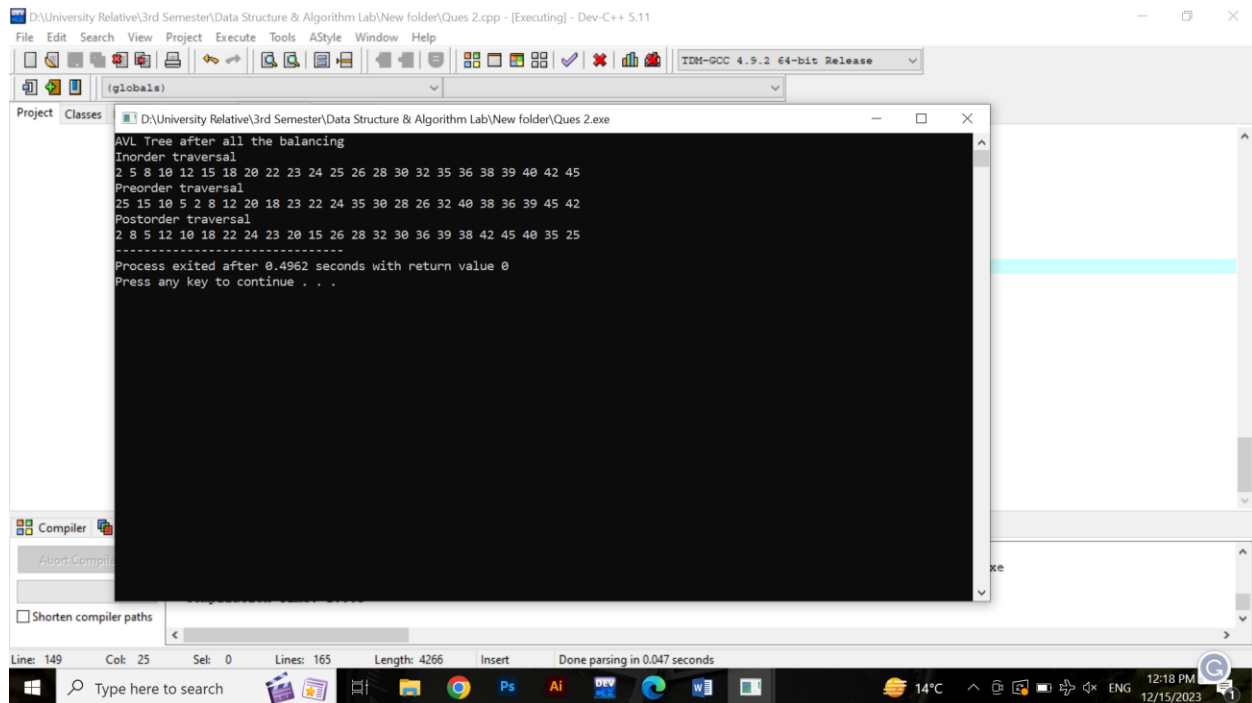
    avl.preorder(root);

    cout << "\nPostorder traversal" << endl;

    avl.postorder(root);

    return 0;
}
```

**Output:**



### Question no 03:

#### Code:

/\*Define a function RemoveDuplicateNode(Node \*head) which remove the duplicate node ( repeating ) from the sorted link list and display the resultant list. Sample:

1 -> 2 -> 3 -> 3 -> 8

Output: 1 -> 2 -> 3 -> 8\*/

```
#include <iostream>
```

```
using namespace std;
```

```
struct Node {
```

```
    int data;
```

```
    Node* next;
```

```
    Node(int value) {
```

```
        data = value;
```

```
        next = NULL;
```

```
    }
```

```

};

void RemoveDuplicateNode(Node* head) {
    if (head == NULL) {
        return;
    }
    Node* current = head;
    while (current != NULL && current->next != NULL) {
        if (current->data == current->next->data) {

            Node* duplicate = current->next;
            current->next = current->next->next;
            delete duplicate;
        } else {

            current = current->next;
        }
    }
}

void DisplayList(Node* head) {
    while (head != NULL) {
        cout << head->data;
        if (head->next != NULL) {
            cout << " -> ";
        }
        head = head->next;
    }
    cout << endl;
}

```

```

}

int main() {

    Node* head = new Node(1);

    head->next = new Node(2);

    head->next->next = new Node(3);

    head->next->next->next = new Node(3);

    head->next->next->next->next = new Node(8);

    cout << "Original List: ";

    DisplayList(head);

    RemoveDuplicateNode(head);

    cout << "List after removing duplicates: ";

    DisplayList(head);

    return 0;

}

```

### Output:

The screenshot shows the Dev-C++ IDE with a project named 'Ques 3.exe' being executed. The output window displays the following text:

```

Original List: 1 -> 2 -> 3 -> 3 -> 8
List after removing duplicates: 1 -> 2 -> 3 -> 8
-----
Process exited after 0.4782 seconds with return value 0
Press any key to continue . . .

```

Below the output window, the 'About Compilation' dialog box is visible, showing the following details:

- Output Filename: D:\University Relative\3rd Semester\Data Structure & Algorithm Lab\New folder\Ques 3.exe
- Output Size: 1.83339405059814 MiB
- Compilation Time: 1.03s

The status bar at the bottom of the IDE indicates the current line is 11, column is 22, and the file length is 1449 characters. The system tray shows the date and time as 12:11 PM on 12/15/2023.



