Practical No. 3

```
#include <iostream>
using namespace std;

class Node {
        int data;
        Node *left;
        Node *right;
        bool isRightThread;
        bool isLeftThread;
public:
        Node (int data) {
                this->data = data;
                this->left = NULL;
                this->right = NULL;
                this->isRightThread = false;
                this->isLeftThread = false;
        }

        int getData() {
                return this->data;
        }

        void bstInsert(int data) {
                if (data < this->data) {
                        if (!this->isLeftThread && this->left) {
                                this->left->bstInsert(data);
                        }
                        else {
                                Node* newnode = new Node(data);
                                newnode->left = this->left;
                                if (this->left) {
                                        newnode->isLeftThread = true;
                                }
                                newnode->right = this;
                                newnode->isRightThread = true;
                                this->left = newnode;
                                this->isLeftThread = false;
                        }
                }
                if (data > this->data) {
                        if (!this->isRightThread && this->right) {
                                this->right->bstInsert(data);
                }
                else {
                        Node* newnode = new Node(data);
                        newnode->right = this->right;
                        if (this->right) {
                                newnode->isRightThread = true;
                        }
                        newnode->left = this;
                        newnode->isLeftThread = true;
                        this->right = newnode;
                        this->isRightThread = false;
                }
        }
}

bool insert(int data, string path) {
        bool added;
        if (this->left) {
                added = this->left->insert(data, path + "left->");
                if (added) {
                        return added;
                }
        }
        else {
                char choice;
                cout<<"Do you want to insert node "<<data<<" at "<<path<<"left ? ";
                cin>>choice;
                if (choice == 'y') {
                        this->left = new Node(data);

                        this->left->right = this;
                        this->left->isRightThread = true;

                        return true;
                }
        }

        if (!this->isRightThread && this->right) {
                added = this->right->insert(data, path + "right->");
                if (added) {
                        return added;
                }
        }
        else {
                char choice;
                cout<<"Do you want to insert node "<<data<<" at "<<path<<"right ? ";
                cin>>choice;
                if (choice == 'y') {
                        Node* newnode = new Node(data);
                        if (this->right) {
                                newnode->right = this->right;
                                newnode->isRightThread = true;
                        }
                        this->right = newnode;
                        this->isRightThread = false;
                        return true;
                }
        }
        return added;
}

void inorder() {
        Node* temp = this;
        Node* parentofTemp = NULL;
        bool isComplete = false;
        do {
                parentofTemp = temp;
                while (temp != NULL && !parentofTemp->isLeftThread) {
                        parentofTemp = temp;
                        temp = temp->left;
                }
                temp = parentofTemp;
                cout<<temp->data<<", ";
                while (temp->isRightThread) {
                        temp = temp->right;
                        cout<<temp->data<<", ";
                }
                if (temp->right) {
                        temp = temp->right;
                }
                else {
                        isComplete = true;
                }
        } while(!isComplete);
}

void preorder() {
        Node* temp = this;
```

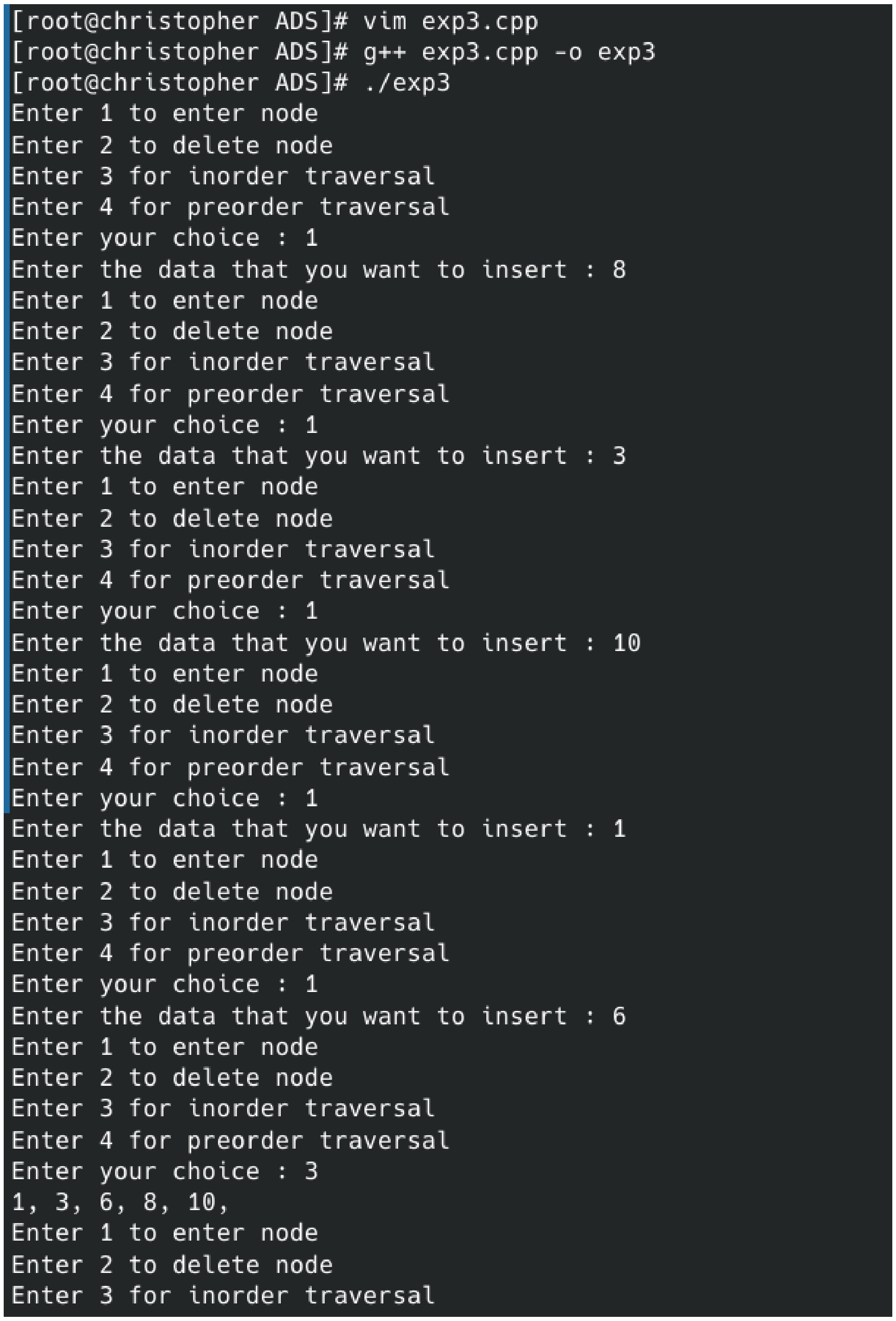
```
                Node* parentofTemp = NULL;
                bool isComplete = false;
                do {
                        parentofTemp = temp;
                        while (temp != NULL && !parentofTemp->isLeftThread) {
                                cout<<temp->data<<", ";
                                if (temp->left)
                                parentofTemp = temp;
                                temp = temp->left;
                        }
                        temp = parentofTemp;
                        while (temp->isRightThread) {
                                temp = temp->right;
                        }
                        if (temp->right) {
                                temp = temp->right;
                                if (temp->isLeftThread) {
                                        cout<<temp->data<<", ";
                                }
                        }
                        else {
                                isComplete = true;
                        }
                } while(!isComplete);
        }

        static Node* deleteNode(Node* parent,Node* toBeDel) {
      Node *temp;
      bool isLeft;
      if (parent) {
         if (parent->left == toBeDel) {
            isLeft = true;
         }
         else {
            isLeft = false;
         }
      }
      if ((!toBeDel->isLeftThread && toBeDel->left) && (!toBeDel->isRightThread &&
toBeDel->right)) {
         Node* auxparent = toBeDel;
         temp = toBeDel->right;
         while (temp->left != toBeDel) {
            auxparent = temp;
            temp = temp->left;
```

```
    }
    temp->left = toBeDel->left;
                        temp->isLeftThread = toBeDel->isLeftThread;

                        if (auxparent != toBeDel) {
                auxparent->left = temp->right;
                        }

                        if (toBeDel->right != temp) {
                temp->right = toBeDel->right;
                                    temp->isRightThread = toBeDel->isRightThread;
                        }


                        auxparent = toBeDel;
                        Node* temp2 = toBeDel->left;
                        while (temp2->right != toBeDel) {
                                    auxparent = temp2;
                                    temp2 = temp2->right;
                        }
                        temp2->right = temp;

    delete toBeDel;
    toBeDel = temp;
}
else if (!toBeDel->isLeftThread && toBeDel->left) {
    temp = toBeDel->left;
                        temp->right = toBeDel->right;
                        temp->isRightThread = toBeDel->isRightThread;
    delete toBeDel;
    toBeDel = temp;
}
else if (!toBeDel->isRightThread && toBeDel->right) {
    temp = toBeDel->right;
                        temp->left = toBeDel->left;
                        temp->isLeftThread = toBeDel->isLeftThread;
    delete toBeDel;
    toBeDel = temp;
}
else {
    delete toBeDel;
    toBeDel = NULL;
}
if (parent) {
```

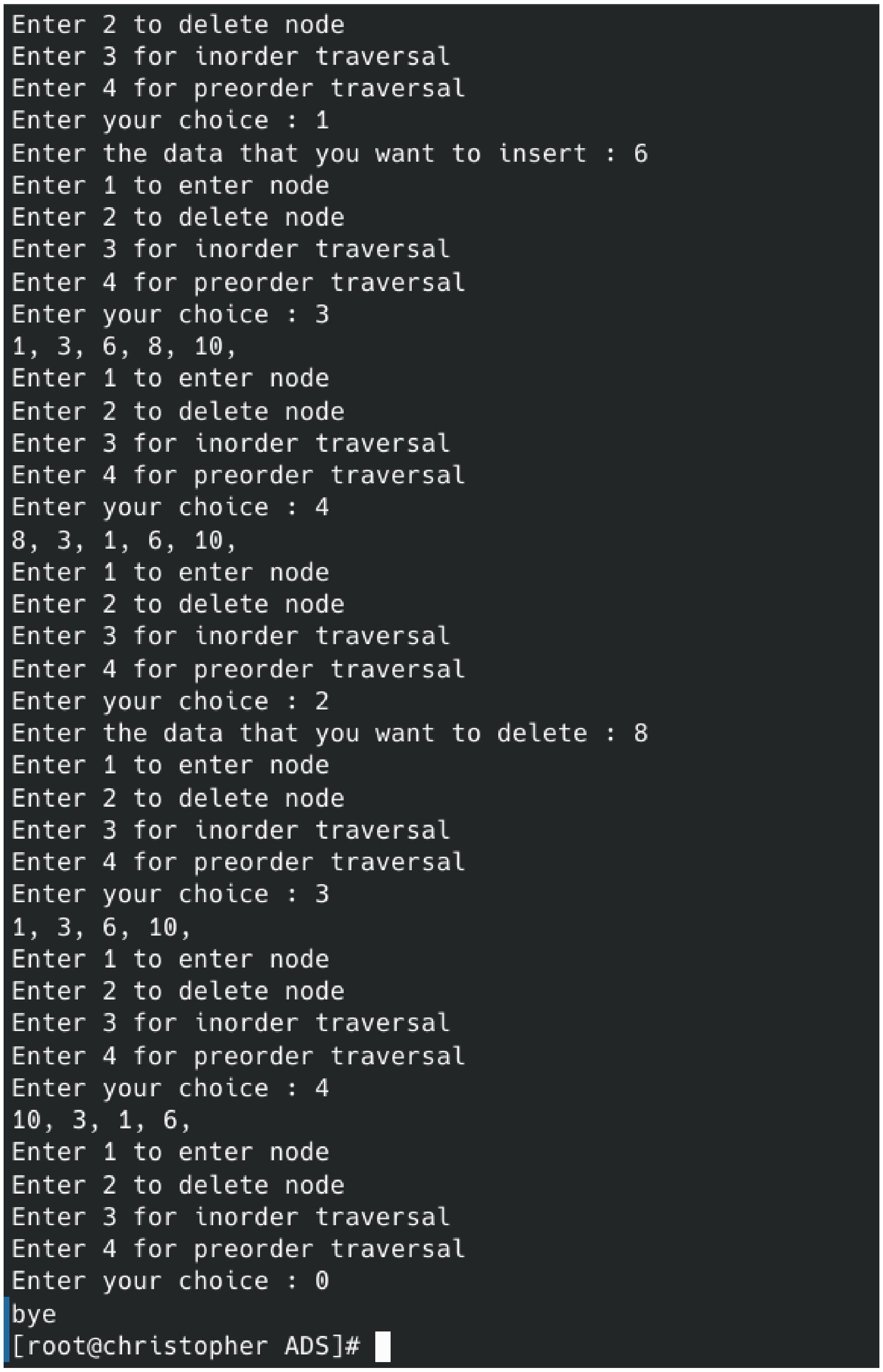
```
            if (isLeft) {
                parent->left = toBeDel;
            }
            else {
                parent->right = toBeDel;
            }
            return parent;
        }
        return toBeDel;
    }

            Node* deleteWrapper(int data) {
        if (data < this->data) {
            if (this->left && !this->isLeftThread) {
                if (data == this->left->data) {
                    return deleteNode(this, this->left);
                } else {
                    return this->left->deleteWrapper(data);
                }
            }
            else {
                return NULL;
            }
        }
        else if (data > this->data) {
            if (this->right && !this->isRightThread) {
                if (data == this->right->data) {
                    return deleteNode(this, this->right);
                } else {
                    return this->right->deleteWrapper(data);
                }
            }
            else {
                return NULL;
            }
        }
        return NULL;
    }
};


class BinaryTree {
            Node* root;
public:
```

```
BinaryTree() {
        root = NULL;
}

void bstInsert(int data) {
        if (root) {
                root->bstInsert(data);
        }
        else {
                root = new Node(data);
        }
}

void insert(int data) {
        if (root) {
                bool isAdded = root->insert(data, "root->");
                if(!isAdded) {
                        cout<<"Node not added"<<endl;
                }
        }
        else {
                root = new Node(data);
        }

}

void inorder() {
        if (root) {
                root->inorder();
                cout<<endl;
        }
}
void preorder() {
        if (root) {
                root->preorder();
                cout<<endl;
        }
}

void deleteNode(int data) {
        if (root) {
                if (root->getData() == data) {
                        root = Node::deleteNode(NULL, root);
                }
                else {
                    Node* result = root->deleteWrapper(data);
                    if (!result) {
                        cout<<"Node not found hence not deleted"<<endl;
                    }
                }
            }
        }
};




int main() {
        BinaryTree tree;
        int data;
        int choice = 1;
        while (choice) {
            cout<<"Enter 1 to enter node"<<endl;
            cout<<"Enter 2 to delete node"<<endl;
            cout<<"Enter 3 for inorder traversal"<<endl;
            cout<<"Enter 4 for preorder traversal"<<endl;
            cout<<"Enter your choice : ";
            cin>>choice;
            if (choice == 1) {
                cout<<"Enter the data that you want to insert : ";
                cin>>data;
                tree.bstInsert(data);
            }
            else if (choice == 2) {
                cout<<"Enter the data that you want to delete : ";
                cin>>data;
                tree.deleteNode(data);
            }
            else if (choice == 3) {
                tree.inorder();
            }
            else if (choice == 4) {
                tree.preorder();
            }
            else {
                    cout<<"bye"<<endl;
                }
            }
            return 0;
}
```

Output:



```
[root@christopher ADS]# vim exp3.cpp
[root@christopher ADS]# g++ exp3.cpp -o exp3
[root@christopher ADS]# ./exp3
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 1
Enter the data that you want to insert : 8
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 1
Enter the data that you want to insert : 3
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 1
Enter the data that you want to insert : 10
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 1
Enter the data that you want to insert : 1
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 1
Enter the data that you want to insert : 6
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 3
1, 3, 6, 8, 10,
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
```



```
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 1
Enter the data that you want to insert : 6
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 3
1, 3, 6, 8, 10,
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 4
8, 3, 1, 6, 10,
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 2
Enter the data that you want to delete : 8
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 3
1, 3, 6, 10,
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 4
10, 3, 1, 6,
Enter 1 to enter node
Enter 2 to delete node
Enter 3 for inorder traversal
Enter 4 for preorder traversal
Enter your choice : 0
bye
[root@christopher ADS]#
```