# **Name: Abdurrahman Qureshi**

# **Roll No: 242466**

Practical No: 6

1) Demonstrate Insertion and Deletion operation on Binary Search Tree

CODE

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

struct node

{

    int data;

    struct node \*left;

    struct node \*right;

};

struct node \*tree;

void create\_tree();

int insertElement();

void preorderTraversal();

void inorderTraversal();

void postorderTraversal();

int deleteElement();

int main()

{

    int c, val;

    struct node \*ptr;

    create\_tree(tree);

    do

    {

        printf("\n \*\*\*\*\*\*MAIN MENU\*\*\*\*\*\*\* \n");

        printf("\n 1. Insert Element");

        printf("\n 2. Preorder Traversal");

        printf("\n 3. Inorder Traversal");

        printf("\n 4. Postorder Traversal");

        printf("\n 5. Delete an element");

        printf("\n\n Enter your option : ");

        scanf("%d", &c);

        switch (c)

        {

        case 1:

            insertElement();

            break;

        case 2:

            preorderTraversal();

            break;

        case 3:

            inorderTraversal();

            break;

        case 4:

            postorderTraversal();

            break;

        case 5:

            deleteElement();

            break;

        }

    } while (c != 6);

    return 0;

}

void create\_tree()

{

    tree = NULL;

}

int insertElement()

{

    struct node \*ptr, \*nodeptr, \*parentptr;

    int val;

    ptr = (struct node \*)malloc(sizeof(struct node));

    printf("\n Enter the value of the new node : ");

    scanf("%d", &val);

    ptr->data = val;

    ptr->left = NULL;

    ptr->right = NULL;

    if (tree == NULL)

    {

        tree = ptr;

        tree->left = NULL;

        tree->right = NULL;

    }

    else

    {

        parentptr = NULL;

        nodeptr = tree;

        while (nodeptr != NULL)

        {

            parentptr = nodeptr;

            if (val < nodeptr->data)

                nodeptr = nodeptr->left;

            else

                nodeptr = nodeptr->right;

        }

        if (val < parentptr->data)

            parentptr->left = ptr;

        else

            parentptr->right = ptr;

    }

    return tree;

}

void preorderTraversal()

{

    if (tree != NULL)

    {

        printf("%d\t", tree->data);

        preorderTraversal(tree->left);

        preorderTraversal(tree->right);

    }

}

void inorderTraversal()

{

    if (tree != NULL)

    {

        inorderTraversal(tree->left);

        printf("%d\t", tree->data);

        inorderTraversal(tree->right);

    }

}

void postorderTraversal()

{

    if (tree != NULL)

    {

        postorderTraversal(tree->left);

        postorderTraversal(tree->right);

        printf("%d\t", tree->data);

    }

}

int deleteElement()

{

    struct node \*cur, \*parent, \*suc, \*psuc, \*ptr;

    int val;

    if (tree->left == NULL)

    {

        printf("\n The tree is empty ");

        return (tree);

    }

    printf("\n Enter the element to be deleted : ");

    scanf("%d", &val);

    parent = tree;

    cur = tree->left;

    while (cur != NULL && val != cur->data)

    {

        parent = cur;

        cur = (val < cur->data) ? cur->left : cur->right;

    }

    if (cur == NULL)

    {

        printf("\n The value to be deleted is not present in the tree");

        return (tree);

    }

    if (cur->left == NULL)

        ptr = cur->right;

    else if (cur->right == NULL)

        ptr = cur->left;

    else

    {

        psuc = cur;

        cur = cur->left;

        while (suc->left != NULL)

        {

            psuc = suc;

            suc = suc->left;

        }

        if (cur == psuc)

        {

            suc->left = cur->right;

        }

        else

        {

            suc->left = cur->left;

            psuc->left = suc->right;

            suc->right = cur->right;

        }

        ptr = suc;

    }

    if (parent->left == cur)

        parent->left = ptr;

    else

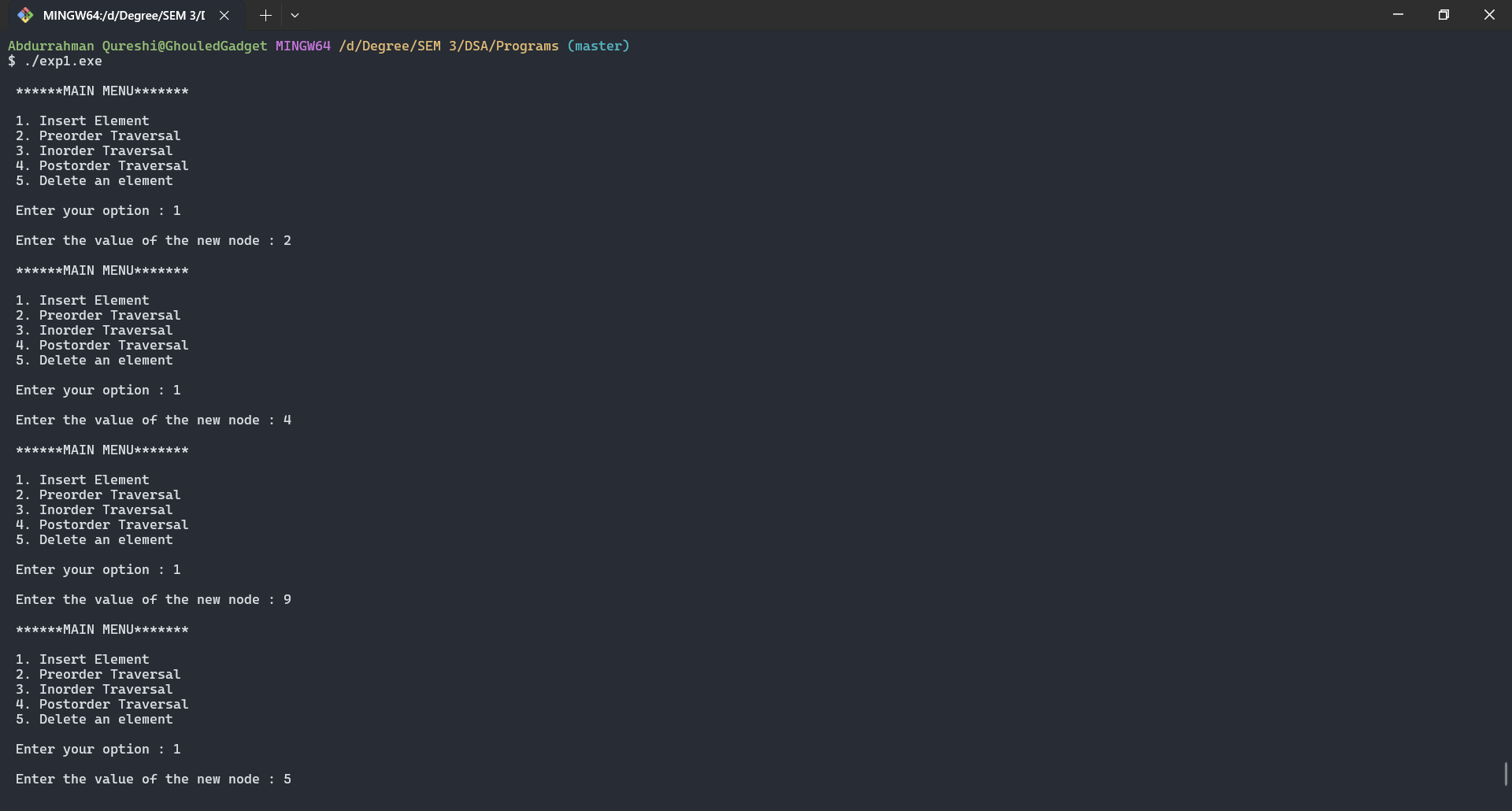
        parent->right = ptr;

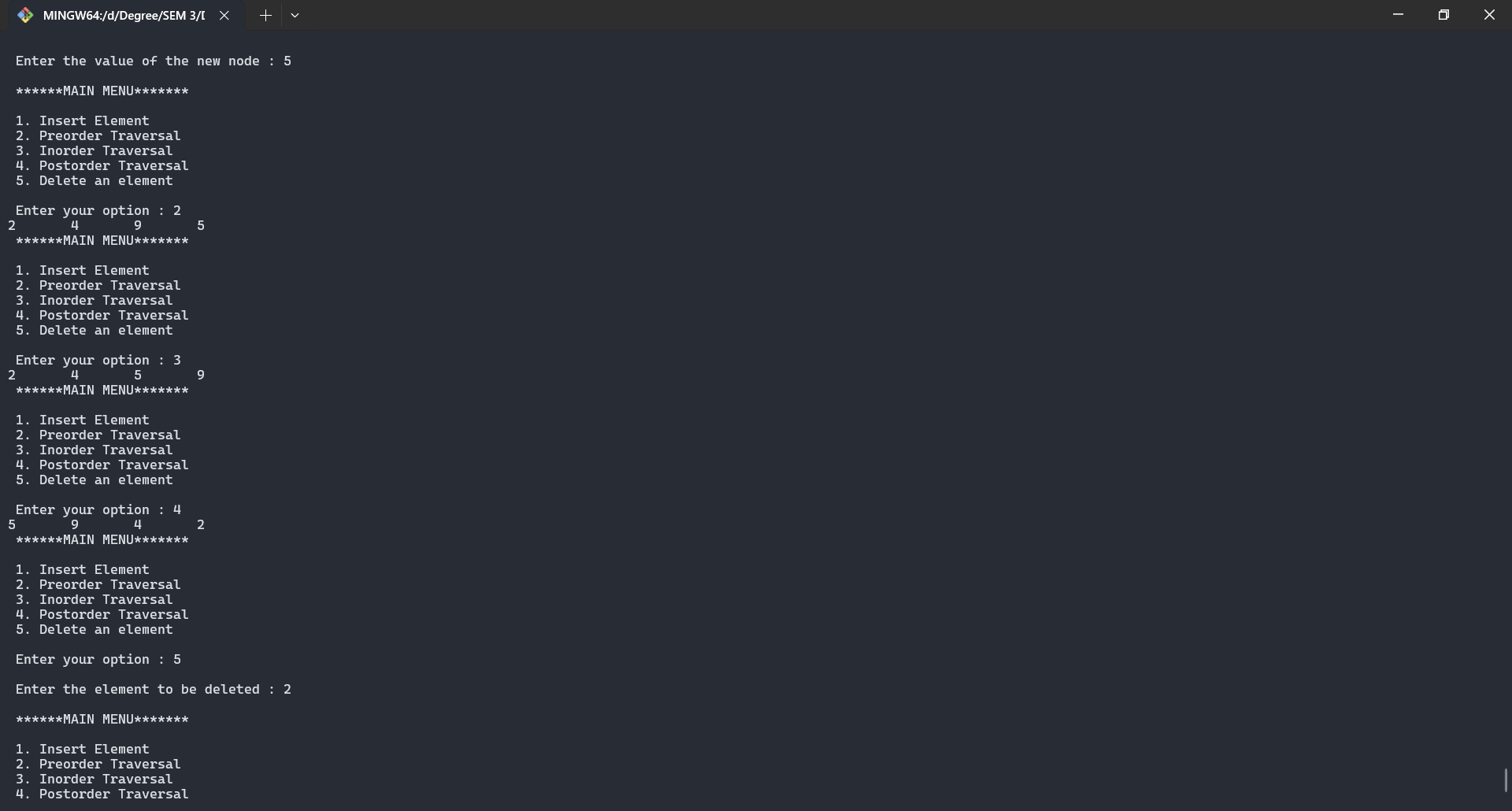
    free(cur);

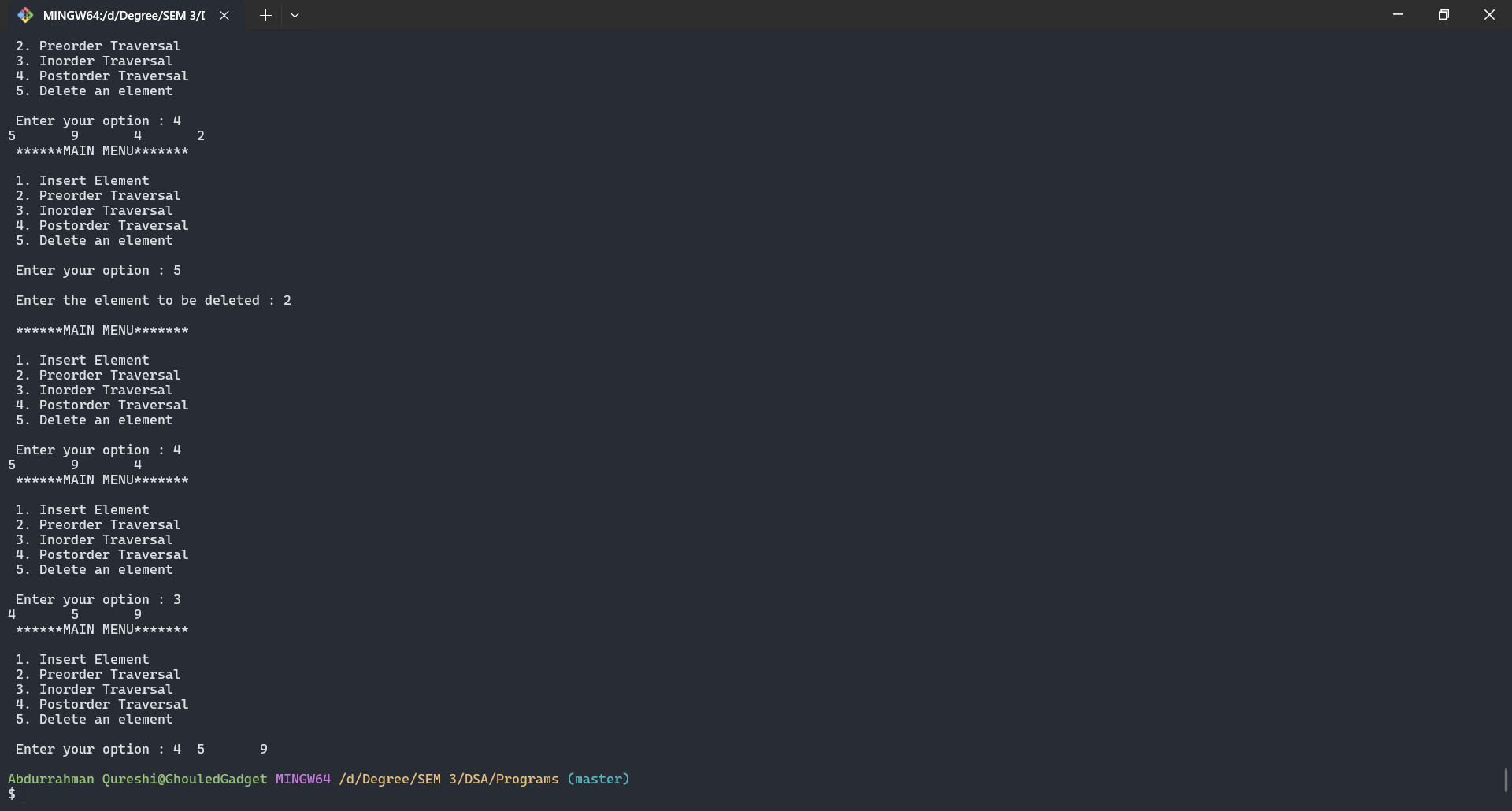
    return tree;

}

OUTPUT







Tools used :

Software: Dev c++

Hardware: Lab Computers

References: Mam notes.

Conclusion

Binary Search Tree (or BST) is a special kind of binary tree in which the values of all the nodes of the left subtree of any node of the tree are smaller than the value of the node. Also, the values of all the nodes of the right subtree of any node are greater than the value of the node.