|  |  |
| --- | --- |
|  | SVKM’s NMIMS  School of Technology Management & Engineering Navi Mumbai Campus |
| Department of Computer Engineering |

|  |  |
| --- | --- |
| Name: Yash Patil | SAP ID: 70472200140 |
| Semester: III | Year: II |
| Subject: DSA | Roll No.: A176 |
| Practical: 8 | Date: 25/09/23 |
| Batch: 1 |  |

Aim: –

Implementation of Binary Search Tree: Insertion, deletion and Search operation on tree data structure.

Theory: –

# Code/Implementation: -

#include<stdio.h> #include<malloc.h> struct node

{ int data; struct node\* left; struct node\* right;

};

struct node\* createNode(int data)

{

struct node \*n; n = (struct node \*) malloc(sizeof(struct node)); n->data = data; n->left = NULL;

n->right = NULL; return n;

}

void insert(struct node \*root, int key)

{

struct node \*prev = NULL;

while(root!=NULL

{ prev = root; if(key==root->data)

{

printf("Cannot insert %d, already in BST", key); return;

}

else if(key<root->data)

{

root = root->left;

} else {

root = root->right;

}

}

struct node\* new = createNode(key); if(key<prev->data)

{

prev->left = new;

} else

{

prev->right = new;

}

}

void preOrder(struct node\* root)

{

if(root!=NULL)

{

printf("%d ", root->data); preOrder(root->left);

preOrder(root->right);

}

}

void postOrder(struct node\* root)

{

if(root!=NULL)

{

postOrder(root->left); postOrder(root->right);

printf("%d ", root->data);

}

}

void inOrder(struct node\* root)

{

if(root!=NULL)

{

inOrder(root->left); printf("%d ", root->data);

inOrder(root->right);

}

}

int isBST(struct node\* root)

{

static struct node \*prev = NULL; if(root!=NULL)

{ if(!isBST(root->left))

return 0;

if(prev!=NULL && root->data <= prev->data)

return 0; prev = root;

return isBST(root->right);

} else return 1; }

struct node \* searchIter(struct node\* root, int key)

{

while(root!=NULL)

{

if(key == root->data)

{ return root;

}

else if(key<root->data) root = root->left; else

root = root->right;

}

return NULL;

}

struct node \*inOrderPredecessor(struct node\* root)

{

if(root->left!=NULL)

{ root = root->left;

while (root->right!=NULL)

{

root = root->right;

} } else { root = root->right;

while (root->left!=NULL)

{

root = root->left;

}

} return root; }

struct node \*deleteNode(struct node \*root, int value)

{ struct node\* iPre; if (root == NULL)

return NULL;

if (root->left==NULL&&root->right==NULL)

{ free(root);

return NULL;

}

if (value < root->data) root-> left = deleteNode(root->left,value); else if (value > root->data) root-> right = deleteNode(root->right,value); else

{

iPre = inOrderPredecessor(root); root->data = iPre->data; root->left = deleteNode(root->left, iPre->data);

} return root;

} int main() { int val,ch;

struct node \*root=NULL,\*temp=NULL;

printf("Enter Root Value:"); scanf("%d",&val); root = createNode(val); do{ printf("\n 1: Insert\n 2: Search\n 3: Delete\n 4: Inorder\n 5: Postorder\n 6: Preorder\n 7: Exit\nEnter choice: "); scanf("%d",&ch); switch(ch)

{

case 1: printf("Enter Root Value:"); scanf("%d",&val); insert(root,val);break; case 2: printf("Enter Search Value:"); scanf("%d",&val); temp=searchIter(root,val); if(temp==NULL) printf("\n Node not found"); else

printf("\n Node found with address: %d",temp);break;

case 3:printf("\n Enter Value to delete:"); scanf("%d",&val); deleteNode(root,val); case 4:inOrder(root);break; case 5:postOrder(root);break;

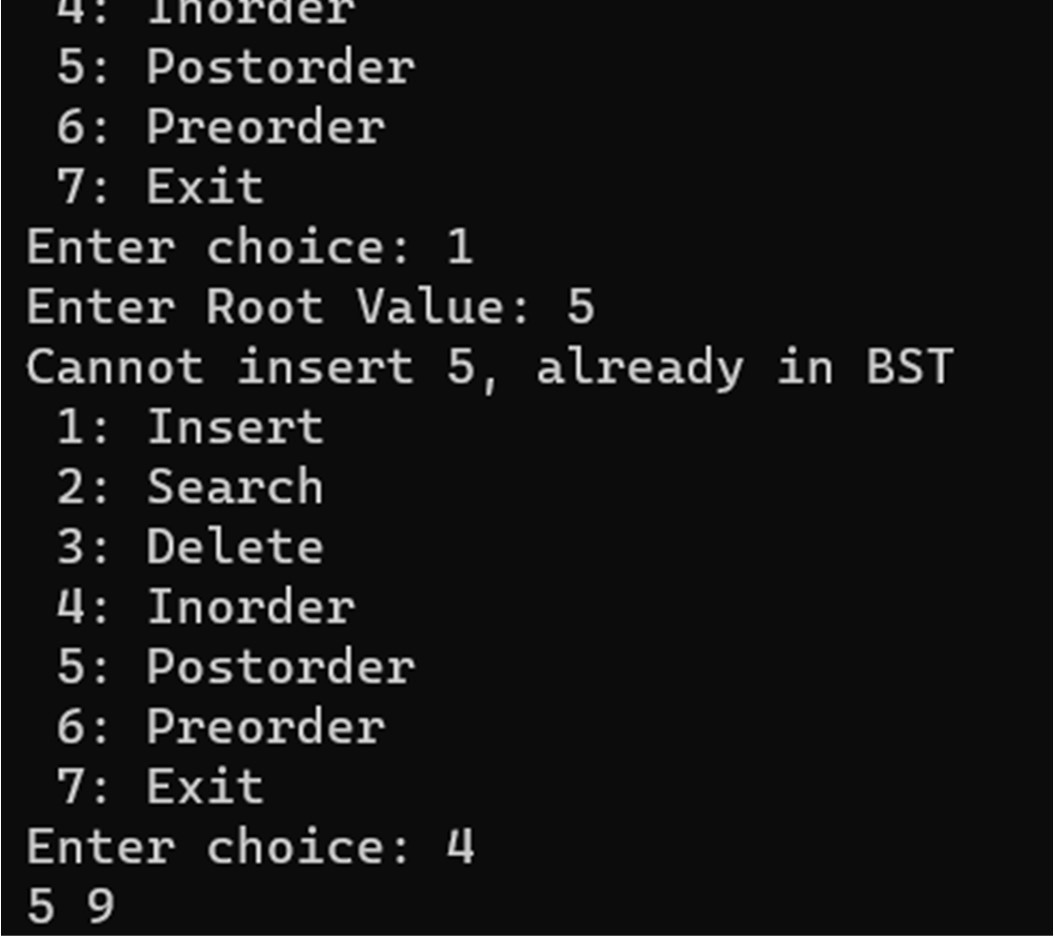
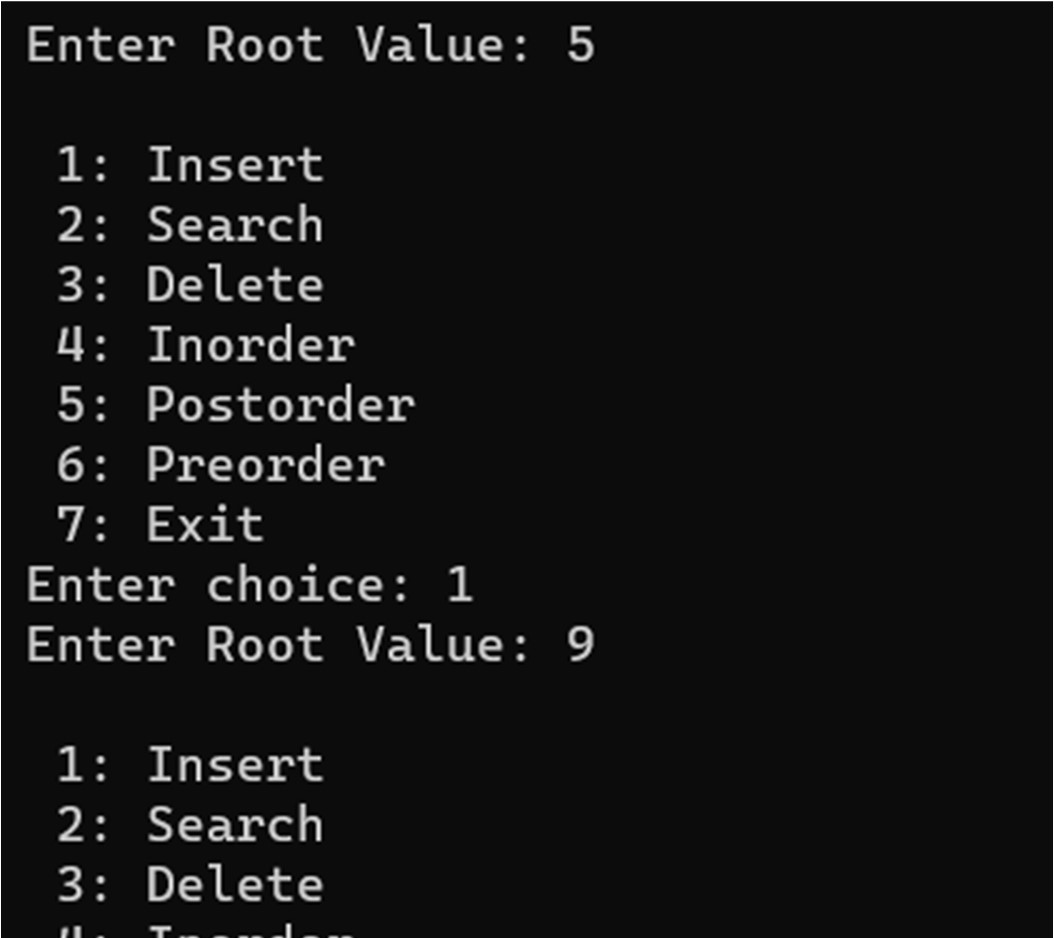
case 6:preOrder(root);break;

}

}while(ch<=6); return 0;

}

# Output: -



Conclusion: - Implemented Binary Search Tree: Insertion, deletion and Search operation on tree data structure.

Outcome: - Implemented Binary Search Tree: Insertion, deletion and Search operation on tree data structure.