**Assignment-1**

* **Insertion of Node in BST**

**// C program to demonstrate insert**

**// operation in binary**

**// search tree.**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node {**

**int key;**

**struct node \*left, \*right;**

**};**

**// A utility function to create a new BST node**

**struct node\* newNode(int item)**

**{**

**struct node\* temp**

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

**temp->key = item;**

**temp->left = temp->right = NULL;**

**return temp;**

**}**

**// A utility function to do inorder traversal of BST**

**void inorder(struct node\* root)**

**{**

**if (root != NULL) {**

**inorder(root->left);**

**printf("%d \n", root->key);**

**inorder(root->right);**

**}**

**}**

**/\* A utility function to insert**

**a new node with given key in**

**\* BST \*/**

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

**{**

**/\* If the tree is empty, return a new node \*/**

**if (node == NULL)**

**return newNode(key);**

**/\* Otherwise, recur down the tree \*/**

**if (key < node->key)**

**node->left = insert(node->left, key);**

**else if (key > node->key)**

**node->right = insert(node->right, key);**

**/\* return the (unchanged) node pointer \*/**

**return node;**

**}**

**// Driver Code**

**int main()**

**{**

**/\* Let us create following BST**

**50**

**/ \**

**30 70**

**/ \ / \**

**20 40 60 80 \*/**

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

**root = insert(root, 50);**

**insert(root, 30);**

**insert(root, 20);**

**insert(root, 40);**

**insert(root, 70);**

**insert(root, 60);**

**insert(root, 80);**

**// print inoder traversal of the BST**

**inorder(root);**

**return 0;**

**}**

* **Deletion of Node in BST**

**// C program to demonstrate**

**// delete operation in binary**

**// search tree**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct node {**

**int key;**

**struct node \*left, \*right;**

**};**

**// A utility function to create a new BST node**

**struct node\* newNode(int item)**

**{**

**struct node\* temp**

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

**temp->key = item;**

**temp->left = temp->right = NULL;**

**return temp;**

**}**

**// A utility function to do inorder traversal of BST**

**void inorder(struct node\* root)**

**{**

**if (root != NULL) {**

**inorder(root->left);**

**printf("%d ", root->key);**

**inorder(root->right);**

**}**

**}**

**/\* A utility function to**

**insert a new node with given key in**

**\* BST \*/**

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

**{**

**/\* If the tree is empty, return a new node \*/**

**if (node == NULL)**

**return newNode(key);**

**/\* Otherwise, recur down the tree \*/**

**if (key < node->key)**

**node->left = insert(node->left, key);**

**else**

**node->right = insert(node->right, key);**

**/\* return the (unchanged) node pointer \*/**

**return node;**

**}**

**/\* Given a non-empty binary search**

**tree, return the node**

**with minimum key value found in**

**that tree. Note that the**

**entire tree does not need to be searched. \*/**

**struct node\* minValueNode(struct node\* node)**

**{**

**struct node\* current = node;**

**/\* loop down to find the leftmost leaf \*/**

**while (current && current->left != NULL)**

**current = current->left;**

**return current;**

**}**

**/\* Given a binary search tree**

**and a key, this function**

**deletes the key and**

**returns the new root \*/**

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

**{**

**// base case**

**if (root == NULL)**

**return root;**

**// If the key to be deleted**

**// is smaller than the root's**

**// key, then it lies in left subtree**

**if (key < root->key)**

**root->left = deleteNode(root->left, key);**

**// If the key to be deleted**

**// is greater than the root's**

**// key, then it lies in right subtree**

**else if (key > root->key)**

**root->right = deleteNode(root->right, key);**

**// if key is same as root's key,**

**// then This is the node**

**// to be deleted**

**else {**

**// node with only one child or no child**

**if (root->left == NULL) {**

**struct node\* temp = root->right;**

**free(root);**

**return temp;**

**}**

**else if (root->right == NULL) {**

**struct node\* temp = root->left;**

**free(root);**

**return temp;**

**}**

**// node with two children:**

**// Get the inorder successor**

**// (smallest in the right subtree)**

**struct node\* temp = minValueNode(root->right);**

**// Copy the inorder**

**// successor's content to this node**

**root->key = temp->key;**

**// Delete the inorder successor**

**root->right = deleteNode(root->right, temp->key);**

**}**

**return root;**

**}**

**// Driver Code**

**int main()**

**{**

**/\* Let us create following BST**

**50**

**/ \**

**30 70**

**/ \ / \**

**20 40 60 80 \*/**

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

**root = insert(root, 50);**

**root = insert(root, 30);**

**root = insert(root, 20);**

**root = insert(root, 40);**

**root = insert(root, 70);**

**root = insert(root, 60);**

**root = insert(root, 80);**

**printf("Inorder traversal of the given tree \n");**

**inorder(root);**

**printf("\nDelete 20\n");**

**root = deleteNode(root, 20);**

**printf("Inorder traversal of the modified tree \n");**

**inorder(root);**

**printf("\nDelete 30\n");**

**root = deleteNode(root, 30);**

**printf("Inorder traversal of the modified tree \n");**

**inorder(root);**

**printf("\nDelete 50\n");**

**root = deleteNode(root, 50);**

**printf("Inorder traversal of the modified tree \n");**

**inorder(root);**

**return 0;**

**}**

* **Tree Traversal**

1. **Inorder**

**// C program for different tree traversals**

**#include <stdio.h>**

**#include <stdlib.h>**

**/\* A binary tree node has data, pointer to left child**

**and a pointer to right child \*/**

**struct node {**

**int data;**

**struct node\* left;**

**struct node\* right;**

**};**

**/\* Helper function that allocates a new node with the**

**given data and NULL left and right pointers. \*/**

**struct node\* newNode(int data)**

**{**

**struct node\* node**

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

**node->data = data;**

**node->left = NULL;**

**node->right = NULL;**

**return (node);**

**}**

**/\* Given a binary tree, print its nodes in inorder\*/**

**void printInorder(struct node\* node)**

**{**

**if (node == NULL)**

**return;**

**/\* first recur on left child \*/**

**printInorder(node->left);**

**/\* then print the data of node \*/**

**printf("%d ", node->data);**

**/\* now recur on right child \*/**

**printInorder(node->right);**

**}**

**/\* Driver code\*/**

**int main()**

**{**

**struct node\* root = newNode(1);**

**root->left = newNode(2);**

**root->right = newNode(3);**

**root->left->left = newNode(4);**

**root->left->right = newNode(5);**

**// Function call**

**printf("\nInorder traversal of binary tree is \n");**

**printInorder(root);**

**getchar();**

**return 0;**

**}**

**2)Preorder**

**// C program for different tree traversals**

**#include <stdio.h>**

**#include <stdlib.h>**

**/\* A binary tree node has data, pointer to left child**

**and a pointer to right child \*/**

**struct node {**

**int data;**

**struct node\* left;**

**struct node\* right;**

**};**

**/\* Helper function that allocates a new node with the**

**given data and NULL left and right pointers. \*/**

**struct node\* newNode(int data)**

**{**

**struct node\* node**

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

**node->data = data;**

**node->left = NULL;**

**node->right = NULL;**

**return (node);**

**}**

**/\* Given a binary tree, print its nodes in preorder\*/**

**void printPreorder(struct node\* node)**

**{**

**if (node == NULL)**

**return;**

**/\* first print data of node \*/**

**printf("%d ", node->data);**

**/\* then recur on left subtree \*/**

**printPreorder(node->left);**

**/\* now recur on right subtree \*/**

**printPreorder(node->right);**

**}**

**/\* Driver code\*/**

**int main()**

**{**

**struct node\* root = newNode(1);**

**root->left = newNode(2);**

**root->right = newNode(3);**

**root->left->left = newNode(4);**

**root->left->right = newNode(5);**

**// Function call**

**printf("\nPreorder traversal of binary tree is \n");**

**printPreorder(root);**

**getchar();**

**return 0;**

**}**

**3)Post Order**

**// C program for different tree traversals**

**#include <stdio.h>**

**#include <stdlib.h>**

**/\* A binary tree node has data, pointer to left child**

**and a pointer to right child \*/**

**struct node {**

**int data;**

**struct node\* left;**

**struct node\* right;**

**};**

**/\* Helper function that allocates a new node with the**

**given data and NULL left and right pointers. \*/**

**struct node\* newNode(int data)**

**{**

**struct node\* node**

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

**node->data = data;**

**node->left = NULL;**

**node->right = NULL;**

**return (node);**

**}**

**/\* Given a binary tree, print its nodes according to the**

**"bottom-up" postorder traversal. \*/**

**void printPostorder(struct node\* node)**

**{**

**if (node == NULL)**

**return;**

**// first recur on left subtree**

**printPostorder(node->left);**

**// then recur on right subtree**

**printPostorder(node->right);**

**// now deal with the node**

**printf("%d ", node->data);**

**}**

**/\* Driver code\*/**

**int main()**

**{**

**struct node\* root = newNode(1);**

**root->left = newNode(2);**

**root->right = newNode(3);**

**root->left->left = newNode(4);**

**root->left->right = newNode(5);**

**// Function call**

**printf("\nPostorder traversal of binary tree is \n");**

**printPostorder(root);**

**getchar();**

**return 0;**

**}**