**Assignment 4**

**Name** : Komal Mahadev Potdar

**Roll No**.: 92

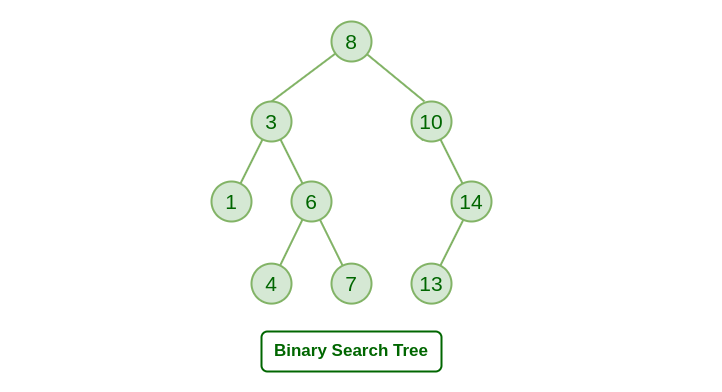
**PRN No**.: 12320165

**Div**: CS B SY

**Batch**: 3

**Binary Search Tree** is a node-based binary tree data structure which has the following properties:

* The left subtree of a node contains only nodes with keys lesser than the node’s key.
* The right subtree of a node contains only nodes with keys greater than the node’s key.
* The left and right subtree each must also be a binary search tree



**BST Traversal**

1. **Inorder Traversal**

* Traverse left subtree
* Visit the root and print the data.
* Traverse the right subtree

The inorder traversalof the BST gives the values of the nodes in sorted order. To get the decreasing order visit the right, root, and left subtree.

1. **Preorder Traversal**

* Visit the root and print the data.
* Traverse left subtree
* Traverse the right subtree

1. **Postorder Traversal**

* Traverse left subtree
* Traverse the right subtree
* Visit the root and print the data.

**1. Write a menu driven  program in C to perform following  operations  on BST**

**a. Create**

**b. Insert**

**c. Delete**

**d. Traversal**

**In-order with recursion and without recursion, pre-order  with recursion and without recursion, Post-order  with recursion and without recursion.**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node\* left;  struct Node\* right;  };  struct Node\* createNode(int data) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  newNode->data = data;  newNode->left = newNode->right = NULL;  return newNode;  }  struct StackNode {  struct Node\* data;  struct StackNode\* next;  };  struct StackNode\* createStackNode(struct Node\* data) {  struct StackNode\* stackNode = (struct StackNode\*)malloc(sizeof(struct StackNode));  stackNode->data = data;  stackNode->next = NULL;  return stackNode;  }  void push(struct StackNode\*\* root, struct Node\* data) {  struct StackNode\* stackNode = createStackNode(data);  stackNode->next = \*root;  \*root = stackNode;  }  int isEmpty(struct StackNode\* root) {  return !root;  }  struct Node\* pop(struct StackNode\*\* root) {  if (isEmpty(\*root))  return NULL;  struct StackNode\* temp = \*root;  \*root = (\*root)->next;  struct Node\* popped = temp->data;  free(temp);  return popped;  }  struct Node\* peek(struct StackNode\* root) {  if (isEmpty(root))  return NULL;  return root->data;  }  struct Node\* insert(struct Node\* root, int data) {  if (root == NULL) return createNode(data);  if (data < root->data)  root->left = insert(root->left, data);  else if (data > root->data)  root->right = insert(root->right, data);  return root;  }  struct Node\* insertMultiple(struct Node\* root, int arr[], int size) {  int i;  for (i = 0; i < size; i++)  root = insert(root, arr[i]);  return root;  }  void inorderRecursive(struct Node\* root) {  if (root != NULL) {  inorderRecursive(root->left);  printf("%d ", root->data);  inorderRecursive(root->right);  }  }  void preorderRecursive(struct Node\* root) {  if (root != NULL) {  printf("%d ", root->data);  preorderRecursive(root->left);  preorderRecursive(root->right);  }  }  void postorderRecursive(struct Node\* root) {  if (root != NULL) {  postorderRecursive(root->left);  postorderRecursive(root->right);  printf("%d ", root->data);  }  }  void inorderNonRecursive(struct Node\* root) {  struct StackNode\* stack = NULL;  struct Node\* current = root;  while (current != NULL || !isEmpty(stack)) {  while (current != NULL) {  push(&stack, current);  current = current->left;  }  current = pop(&stack);  printf("%d ", current->data);  current = current->right;  }  }  void preorderNonRecursive(struct Node\* root) {  if (root == NULL) return;  struct StackNode\* stack = NULL;  push(&stack, root);  while (!isEmpty(stack)) {  struct Node\* temp = pop(&stack);  printf("%d ", temp->data);  if (temp->right) push(&stack, temp->right);  if (temp->left) push(&stack, temp->left);  }  }  void postorderNonRecursive(struct Node\* root) {  if (root == NULL) return;  struct StackNode\* stack1 = NULL;  struct StackNode\* stack2 = NULL;  push(&stack1, root);  while (!isEmpty(stack1)) {  struct Node\* temp = pop(&stack1);  push(&stack2, temp);  if (temp->left) push(&stack1, temp->left);  if (temp->right) push(&stack1, temp->right);  }  while (!isEmpty(stack2)) {  struct Node\* temp = pop(&stack2);  printf("%d ", temp->data);  }  }  struct Node\* minValueNode(struct Node\* node) {  struct Node\* current = node;  while (current && current->left != NULL)  current = current->left;  return current;  }  struct Node\* deleteNode(struct Node\* root, int key) {  if (root == NULL) return root;  if (key < root->data)  root->left = deleteNode(root->left, key);  else if (key > root->data)  root->right = deleteNode(root->right, key);  else {  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;  }  struct Node\* temp = minValueNode(root->right);  root->data = temp->data;  root->right = deleteNode(root->right, temp->data);  }  return root;  }  int main() {  struct Node\* root = NULL;  int choice, data;  do {  printf("\n1. Insert\n");  printf("2. Inorder (Recursive)\n");  printf("3. Preorder (Recursive)\n");  printf("4. Postorder (Recursive)\n");  printf("5. Inorder (Non-Recursive)\n");  printf("6. Preorder (Non-Recursive)\n");  printf("7. Postorder (Non-Recursive)\n");  printf("8. Delete\n");  printf("9. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  switch (choice) {  case 1:  printf("Enter data to insert: ");  scanf("%d", &data);  root = insert(root, data);  break;  case 2:  printf("Inorder Traversal (Recursive): ");  inorderRecursive(root);  printf("\n");  break;  case 3:  printf("Preorder Traversal (Recursive): ");  preorderRecursive(root);  printf("\n");  break;  case 4:  printf("Postorder Traversal (Recursive): ");  postorderRecursive(root);  printf("\n");  break;  case 5:  printf("Inorder Traversal (Non-Recursive): ");  inorderNonRecursive(root);  printf("\n");  break;  case 6:  printf("Preorder Traversal (Non-Recursive): ");  preorderNonRecursive(root);  printf("\n");  break;  case 7:  printf("Postorder Traversal (Non-Recursive): ");  postorderNonRecursive(root);  printf("\n");  break;  case 8:  printf("Enter data to delete: ");  scanf("%d", &data);  root = deleteNode(root, data);  break;  case 9:  printf("Exiting...\n");  break;  default:  printf("Invalid choice. Please enter a valid option.\n");  }  } while (choice != 9);  return 0;  } |

Output:

