**Batch: A1 Roll No.: 1911004**

**Experiment / assignment / tutorial No. 7**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:**  Implementation of BST & Binary tree traversal techniques. |

**Objective:** To Understand and Implement Binary Search Tree, Preorder, Postorder and Inorder Traversal Techniques.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 1 | Explain the different data structures used in problem solving |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. <https://www.geeksforgeeks.org/binary-tree-data-structure/>
3. <https://www.thecrazyprogrammer.com/2015/03/c-program-for-binary-search-tree-insertion.html>

**Abstract**:

**A tree** is a non- linear data structure used to represent hierarchical relationship existing among several data items. It is a finite set of one or more data items such that, there is a special data item called the root of the tree. Its remaining data items are partitioned into number of mutually exclusive subsets, each of which is itself a tree, and they are called subtrees.

**A binary tree** is a finite set of nodes. It is either empty or It consists a node called root with two disjoint binary trees-Left subtree, Right subtree. The Maximum degree of any node is 2

**A Binary Search Tree** is a node-based binary tree data structure in which 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.

**Related Theory: -**

1. **In order Traversal of BST**

* LRN
* Traverse the left subtree of root in Post order
* Traverse the right subtree of root in Post order
* Visit the Root

1. **Preorder Traversal of BST**

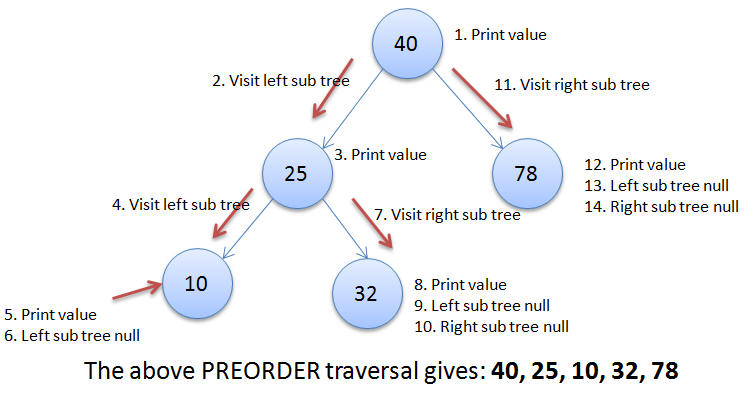
* NLR
* Visits the Root
* Traverse the left subtree of root in Preorder
* Traverse the right subtree of root in Preorder

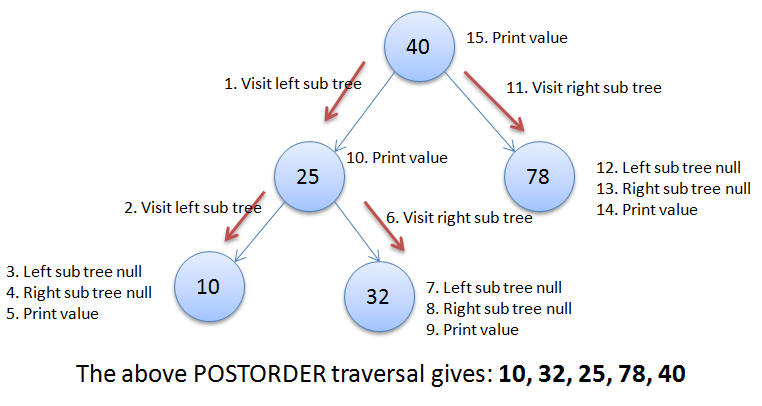
1. **Post order Traversal of BST**

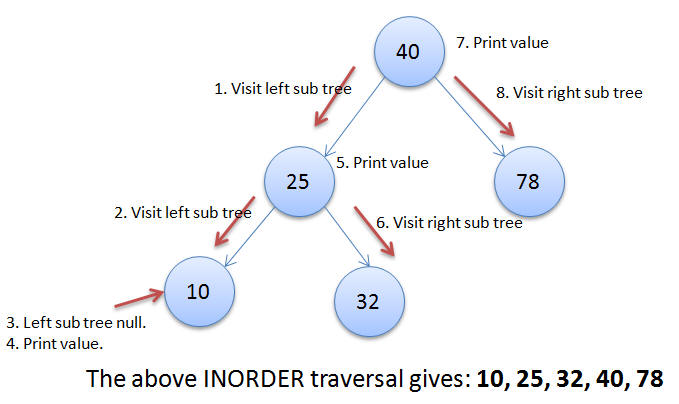
* LNR
* Traverse the left subtree of root in In order
* Visit the Root
* Traverse the right subtree of root in In order

**Diagram for :**

* **Preorder Traversal of BST**
* **Post order Traversal of BST**
* **In order Traversal of BST**







**Algorithm for Implementation of BST & Binary tree traversal techniques:**

* **Insert()-**

1. **If tree = null**

**Allocate memory for tree**

**Set tree data = val**

**Set tree left = tree right = null**

**Else**

**If val < tree data**

**Insert(tree left, val)**

**Else**

**Insert(tree right, val)**

* **PreOrder()**

1. **start**
2. **Repeat Steps a to d if TREE != NULL**
3. **Write TREE DATA**
4. **PREORDER(TREE LEFT)**
5. **PREORDER(TREE RIGHT)**
6. **END**

* **InOrder()**

1. **start**
2. **Repeat Steps a to d if TREE != NULL**
3. **INORDER(TREE LEFT)**
4. **Write TREE DATA**
5. **INORDER(TREE RIGHT)**

* **PostOrder()**

1. **Start**
2. **Repeat Steps a to d if TREE != NULL**
3. **POSTORDER(TREE LEFT)**
4. **POSTORDER(TREE RIGHT)**
5. **Write TREE DATA**
6. **End**

**Implementation Details:**

1. **Enlist all the Steps followed and various options explored.**

Using menu driven program approach, where we ask user for this choice of operation to do and call that function . Also user can insert , delete / display the tree in the required order of choice.

* void Insert(node \*\*root,int n) :- This function inserts a node in the tree.
* void Inorder(node \*root):- this function traverses the tree in inorder format.
* void preorder(node \*root):- this function traverses the tree in preorder format.
* void postorder(node \*root):- this function traverses the tree in postorder format.

1. **Assumptions made for Input:**

* We have assumed that the data type of the values in binary tree nodes are integer type.
* Each node has at-most 2 children
* the left child is greater than the right one always

1. **Built-In Functions Used:**

* malloc()
* scanf()
* printf()

**Program source code for Implementation of BST & Binary tree traversal techniques :**

**# include <stdio.h>**

**# include <stdlib.h>**

**struct node {**

**int data;**

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

**};**

**//new Node**

**struct node \* new\_node() {**

**struct node \*temp;**

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

**temp->left = 0;**

**temp->right = 0;**

**return temp;**

**}**

**//inserting elements in binary search tree**

**void insert(struct node \*root, struct node \* newNode) {**

**if ( newNode->data < root->data) {**

**if (root->left == 0)**

**root->left = newNode;**

**else**

**insert(root->left, newNode);**

**}**

**if ( newNode->data > root->data) {**

**if (root->right == 0)**

**root->right = newNode;**

**else**

**insert(root->right, newNode);**

**}**

**}**

**//search the node in binary Search Tree**

**struct node \*search(struct node \*root, int key, struct node \*\*parent) {**

**struct node \*temp;**

**temp = root;**

**while (temp != 0) {**

**if (temp->data == key) {**

**printf("\nThe %d Element is Present ", temp->data);**

**return temp;**

**}**

**\*parent = temp;**

**if (temp->data > key)**

**temp = temp->left;**

**else**

**temp = temp->right;**

**}**

**return 0;**

**}**

**//displays the tree in inorder fashion**

**void inorder(struct node \*temp) {**

**if (temp != 0) {**

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

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

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

**}**

**}**

**//displays the tree in preorder fashion**

**void preorder(struct node \*temp) {**

**if (temp != 0) {**

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

**preorder(temp->left);**

**preorder(temp->right);**

**}**

**}**

**//displays the tree in postorder fashion**

**void postorder(struct node \*temp) {**

**if (temp != 0) {**

**postorder(temp->left);**

**postorder(temp->right);**

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

**}**

**}**

**int main() {**

**int choice;**

**int ans ;**

**int key;**

**struct node \* newNode, \*root, \*tmp, \*parent;**

**struct node \* new\_node();**

**root = 0;**

**printf("\nProgram For Binary Search Tree ");**

**do {**

**printf("\n1.Insert");**

**printf("\n2.Search");**

**printf("\n3.Recursive Traversals");**

**printf("\n0.Exit");**

**printf("\nEnter your choice : ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**do {**

**newNode = new\_node();**

**printf("\nEnter The Element ");**

**scanf("%d", & newNode->data);**

**if (root == 0) //Tree is not insertd**

**root = newNode;**

**else**

**insert(root, newNode);**

**printf("\nDo you want to enter more Elements?(y=1/n=0) ");**

**scanf("%d",&ans);**

**} while (ans!=0);**

**break;**

**case 2:**

**printf("\nEnter Element to be searched :");**

**scanf("%d", &key);**

**tmp = search(root, key, &parent);**

**printf("\nParent of node %d is %d", tmp->data, parent->data);**

**break;**

**case 3:**

**if (root == 0)**

**printf("Tree Is Not inserted");**

**else {**

**printf("\nThe Inorder display : ");**

**inorder(root);**

**printf("\nThe Preorder display : ");**

**preorder(root);**

**printf("\nThe Postorder display : ");**

**postorder(root);**

**}**

**break;**

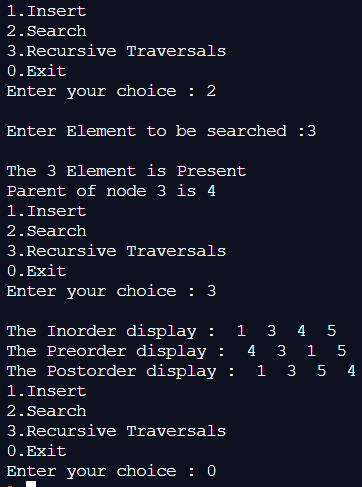
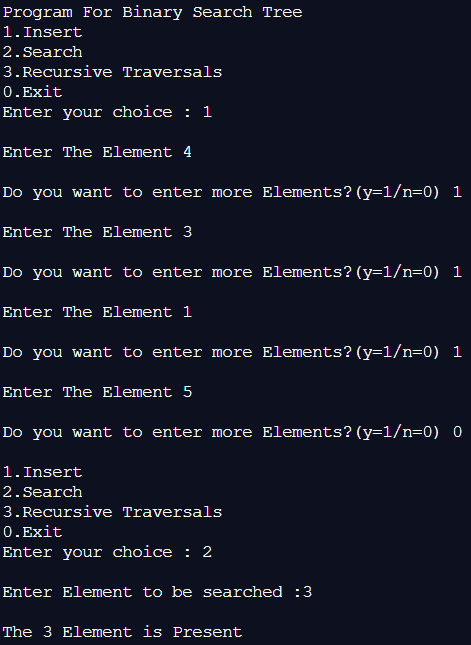
**}**

**} while (choice != 0);**

**return 0;**

**}**

**Output Screenshots for Each Operation:**



* **Explain the Importance of the approach followed by you**
* Insertion , deletion and searching / traversing of data is easier with BST.
* It can be easily traversed in the required order by following the required & appropriate algorithm.
* Searching the data element also takes lesser time. Thus it is easier to insert, delete, search / traverse a binary search tree**.**

**Conclusion:**

We implemented a BST along with various types of BST traversing techniques in C ; thus obtaining correct necessary output**.**

**Post Lab Questions:**

1. **Illustrate 2 Applications of Trees.**

* Store hierarchical data, like folder structure, organization structure, XML/HTML data.
* is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item
* BST is used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries

1. **Compare and Contrast between B Tree and B+ Tree?**

|  |  |
| --- | --- |
| **B Tree** | **B+ Tree** |
| **In a B tree, search keys and data are stored in internal or leaf nodes.** | **In B+-tree, data is stored only in leaf nodes.** |
| **Insertion in B tree is more complicated than B+ tree.** | **Insertion in B+ tree is less complicated than B tree.** |
| **B tree the leaf node cannot be stored using a linked list** | **In a B+ tree, leaf nodes data are ordered as a sequential linked list** |
| **B tree has no redundant value** | **B+ trees store redundant search key** |