**ASSIGNMENT – 33**

**1.PROBLEM STATEMENT**

Write a program in C to implement Binary Search Tree and include insertion, searching and display operations.

**2.ALGORITHMS**

Algorithm **Create\_Node**

**Input:** The element ‘item’ which is to be inserted into the tree.

**Output:** A node of the tree is created in memory with it’s data section holding ‘item’ and it’s address returned in a pointer ‘ptr’.

**Remarks:** Each node has three sections: a DATA section to hold integer data, a LCHILD section to point to the left subtree and a RCHILD section to point to the right subtree.

**Steps:**

1. A node is allocated in memory and its memory address is returned in a pointer ‘ptr’
2. ptr.DATA=item
3. ptr.LCHILD=ptr.RCHILD=null
4. **Return** ptr
5. **Stop**

Algorithm **Insert\_In\_BST**

**Input:** The address of the root node ‘root’ of the tree and the integer ‘item’ which is to be inserted.

**Output:** The element ‘item’ inserted at it’s appropriate position in the binary search tree.

**Remarks:** Recursive procedure

**Steps:**

1. **If**(root=null) **then** //if tree is empty
2. ptr=**Create\_Node**(item) //set root to ptr
3. **Return** ptr
4. **EndIf**
5. **If**(item<root.DATA) //if key is less than root data
6. Root.LCHILD=**Insert\_In\_BST**(root.LCHILD,item) //go to left subtree
7. **Else If**(item>root.DATA)
8. Root.RCHILD=**Insert\_In\_BST**(root.RCHILD,item)//go to right subtree
9. **Return** root
10. **Stop**

Algorithm **Display\_Inorder**

**Input:** The address of the root node of the tree which is to be displayed

**Output:** The contents of the tree displayed in inorder traversal

**Remarks:** Inorder traversal: LCHILD🡪root🡪RCHILD

**Steps:**

1. **If**(root≠null) **then** //tree is not empty
2. **Display\_Inorder**(root.LCHILD) //recursively print right subtree
3. **Print** the root.DATA element //print root data
4. **Display\_Inorder**(root.RCHILD) //recursively print left subtree
5. **EndIf**
6. **Return**
7. **Stop**

Algorithm **Display\_preorder**

**Input:** The address of the root node of the tree which is to be displayed

**Output:** The contents of the tree displayed in preorder traversal

**Remarks:** Inorder traversal: root🡪LCHILD🡪RCHILD

**Steps:**

1. **If**(root≠null) **then** //tree is not empty
2. **Print** the root.DATA element //print root data
3. **Display\_Inorder**(root.LCHILD) //recursively print left subtree
4. **Display\_Inorder**(root.RCHILD) //recursively print right subtree
5. **EndIf**
6. **Return**
7. **Stop**

Algorithm **Display\_postorder**

**Input:** The address of the root node of the tree which is to be displayed

**Output:** The contents of the tree displayed in postorder traversal

**Remarks:** Inorder traversal:LCHILD🡪RCHILD🡪root

**Steps:**

1. **If**(root≠null) **then**
2. **Display\_Inorder**(root.LCHILD) //recursively print left subtree
3. **Display\_Inorder**(root.RCHILD) //recursively print right subtree
4. **Print** the root.DATA element //print root data
5. **EndIf**
6. **Return**
7. **Stop**

Algorithm **Search\_BST**

**Input:** The address of the root node ‘root’ of the tree and the element ‘item’ which is to be inserted.

**Output:** If the element is found at a node, the node address is returned, otherwise null is returned.

**Remarks:**The key should be an integer

**Steps:**

1. **If**(root=null **OR** root.DATA=item) **then** //if root data matches key
2. **Return** root
3. **EndIf**
4. **If**(item>root.DATA) **then** //if key is greater than root data
5. **Return Search\_BST**(root.RCHILD,item) //search right subtree
6. **Else**
7. **Return Search\_BST**(root.LCHILD,item) //search left subtree
8. **Stop**

**3.SOURCE CODE**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

//structure to represent nodes of tree

typedef struct node

{

    int data;

    struct node\* lchild;

    struct node\* rchild;

}node;

//function to allocate node in heap

node\* createnode(int item)

{

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

    ptr->lchild=ptr->rchild=NULL;

    ptr->data=item;

    return ptr;

}

//function to insert a node in BST

node\* insert(node\* root,int item)

{

    node \*ptr;

    if(root==NULL) //if tree is empty

    {

        ptr=createnode(item);

        return ptr;

    }

    if(item<root->data)

        root->lchild=insert(root->lchild,item);

    else if(item>root->data)

        root->rchild=insert(root->rchild,item);

    return root;

}

//function to traverse BST by inorder

void disp\_inorder(node \*root)

{

    if(root!=NULL) //if tree is not empty

    {

        disp\_inorder(root->lchild);

        printf("%d ",root->data);

        disp\_inorder(root->rchild);

    }

}

//function to traverse BST by preorder

void disp\_preorder(node \*root)

{

    if(root!=NULL) //if tree is not empty

    {

        printf("%d ",root->data);

        disp\_preorder(root->lchild);

        disp\_preorder(root->rchild);

    }

}

//function to traverse BST by preorder

void disp\_postorder(node \*root)

{

    if(root!=NULL) //if tree is not empty

    {

        disp\_postorder(root->lchild);

        disp\_postorder(root->rchild);

        printf("%d ",root->data);

    }

}

//function to search an item in BST

node\* search\_BST(node \*root,int item)

{

    if(root==NULL || root->data==item)

        return root;

    if(item>root->data)

        return search\_BST(root->rchild,item);

    return search\_BST(root->lchild,item);

}

int main(void)

{

    bool found=false;

    node \*root=NULL,\*root2=NULL;

    int ch,item;

    printf("To perfrom the following operations on a Binary Search Tree: ");

    while(1)

    {

        printf("\nMENU: ");

        printf("\n1.INSERT\n2.SEARCH\n3.DISPLAY\n4.EXIT");

        printf("\nEnter your choice: ");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1:

            printf("Enter the item to be inserted: ");

            scanf("%d",&item);

            root=insert(root,item);

            printf("Element of BST: \n");

            if(root!=NULL)

            {

                printf("INORDER TRAVERSAL: ");

                disp\_inorder(root);

                printf("\nPREORDER TRAVERSAL: ");

                disp\_preorder(root);

                printf("\nPOSTORDER TRAVERSAL: ");

                disp\_postorder(root);

            }

            else

                printf("Tree is empty");

            break;

            case 2:

            printf("Enter the key to be searched: ");

            scanf("%d",&item);

            if(root!=NULL)

                root2=search\_BST(root,item);

            else

                printf("Tree is empty");

            if(root2==NULL)

                printf("Element %d not found in the tree",item);

            else

                printf("Element %d found in the tree",item);

            break;

            case 3:

            if(root!=NULL)

            {

                printf("INORDER TRAVERSAL: ");

                disp\_inorder(root);

                printf("\nPREORDER TRAVERSAL: ");

                disp\_preorder(root);

                printf("\nPOSTORDER TRAVERSAL: ");

                disp\_postorder(root);

            }

            else

                printf("Tree is empty");

            break;

            case 4:

            exit(1);

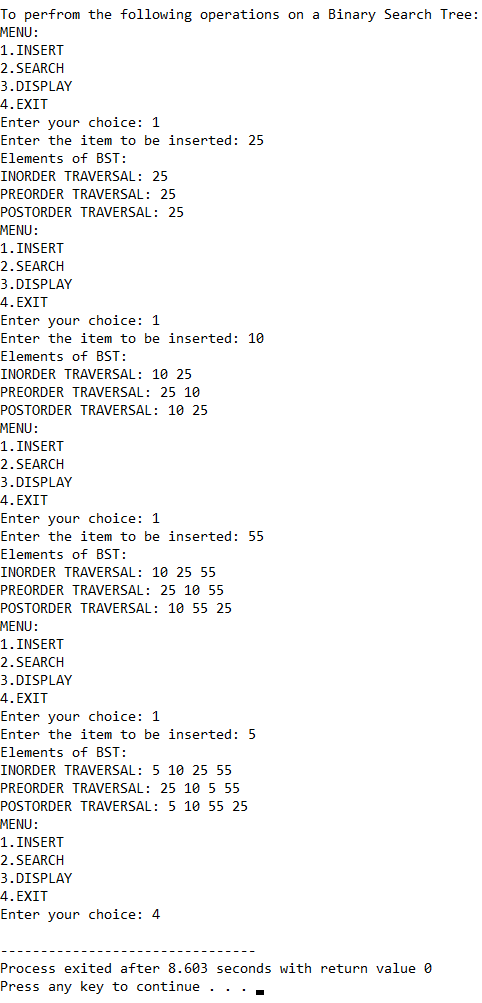
        }

    }

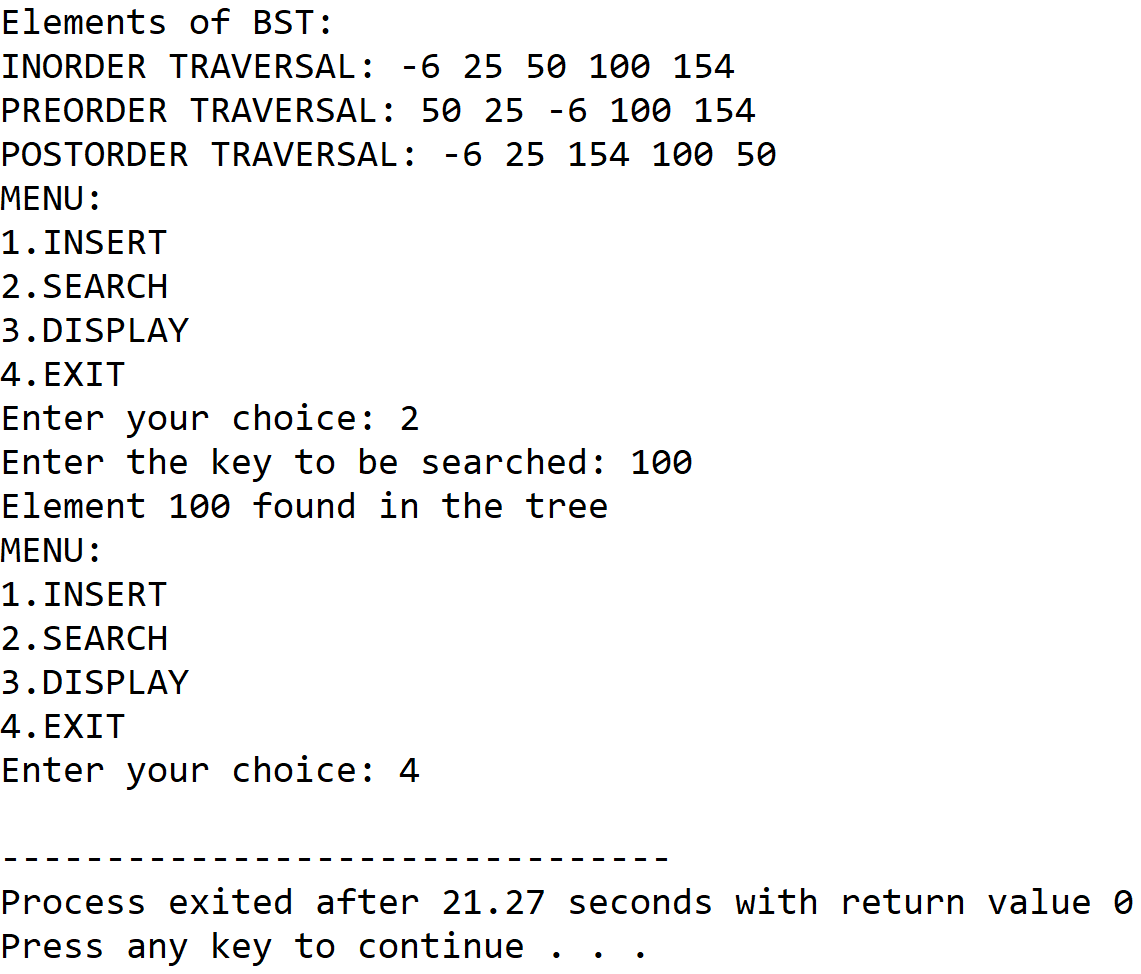
}

**4.OUTPUT**

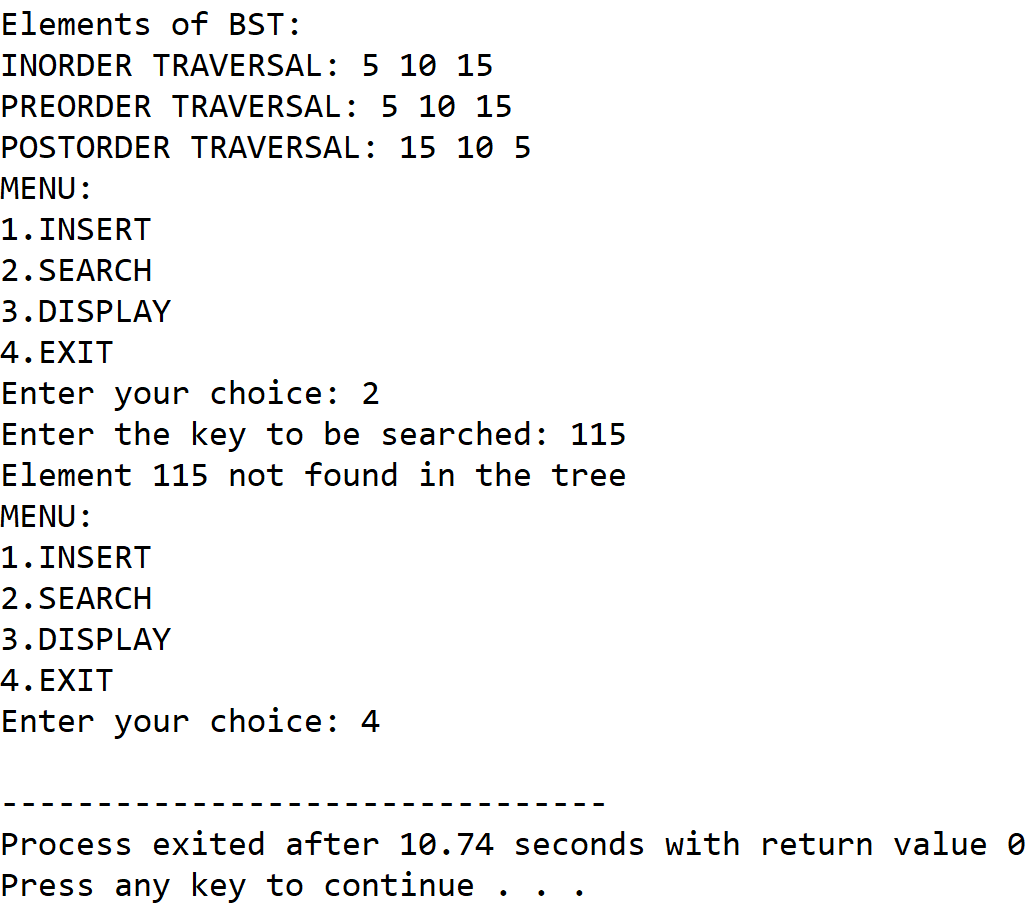
**SET 1:** Insertion

****

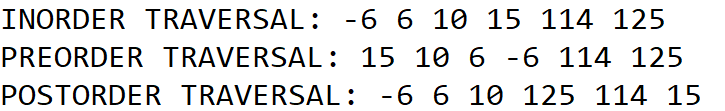
**SET 2:** Search successful



**SET 3:** Search unsuccessful



**SET 4**: Display



**5.DISCUSSIONS**

**Variable Description**

* **root,root2:** pointers to hold root nodes of the binary search tree
* **item:** to hold the integer to be inserted into the tree
* **ptr:** pointer to nodes of the tree

**Limitations**

* The left child and right child pointers of the leaf nodes never point to any subtree, and thus lead to wastage of space

**Uses**

* The program can be used to build a binary search tree with integer data, it is a very efficient datastructure specifically for searching a integer data from a large number of nodes. For example, it can be used to implement a database which requires frequent searching of data.

**Future Scope**

* The lchild and rchild pointers of the leaf nodes can be pointed to the root node of the tree to facilitate circular traversal.