WEEK-12

Date:08-10-2025

List of programs:

1. Write a C progam to implement various Operations on Binary Search Tree(Insertion,Search,Display).

1. **Aim:** To Write a C progam to implement various Operations on Binary Search Tree(Insertion,Search,Display).

Program:

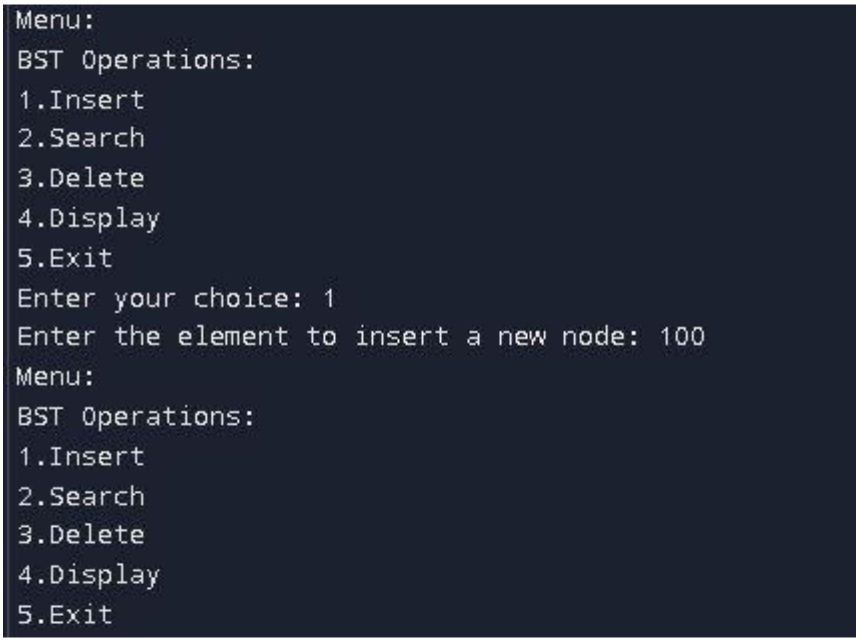
```
#include<stdio.h>

#include<stdlib.h>

struct node

{

   int data;

   struct node *left, *right;

};

struct node *newNode(int item)

{

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

   temp->data = item;

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

   return temp;

}

void inorder(struct node *root)

{

   if (root != NULL)

   {

      inorder(root->left);
```

```
        printf("%d ", root->data);
        inorder(root->right);
    }
}
struct node* insert(struct node *root, int key)
{
    if (root == NULL)
        return newNode(key);
    if (key <= root->data)
        root->left=insert(root->left, key);
    else
        if (key > root->data)
            root->right=insert(root->right, key);
    return root;
}
struct node *search(struct node *temp, int key)
{
    if(temp==NULL)
    {
        printf("No key found for value - %d\n", key);
        return temp;
    }
   if(temp->data == key)
        printf("Key %d found\n", key);
    else if(temp->data < key)
        search(temp->right, key);
    else
        search(temp->left, key);
```

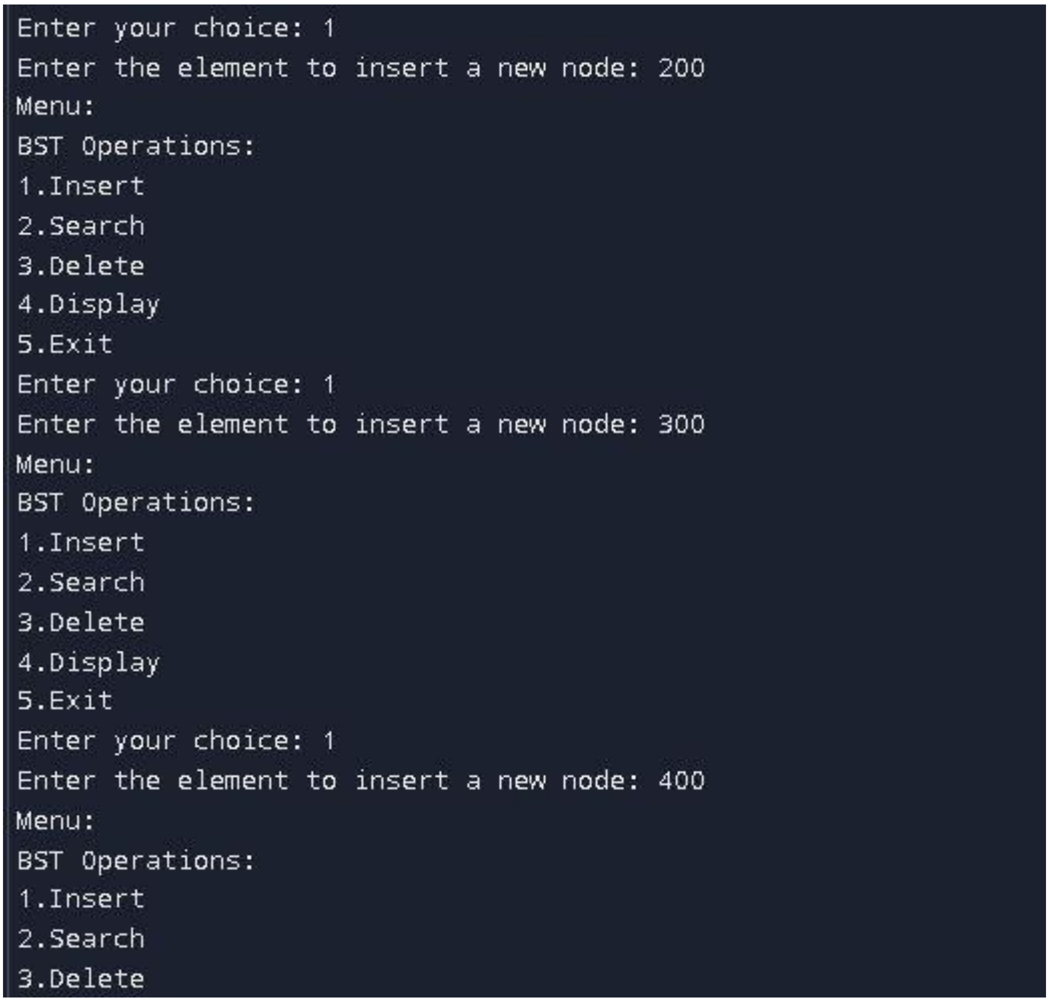
```
}
struct node *getInSuccessor(struct node *temp)
{
    while(temp->left != NULL)
        temp = temp->left;
    return temp;
}
struct node * deletion(struct node *root, int delKey)
{
    struct node *temp;
    if(root==NULL)
    {
        printf("Unable to delete. No such key exists.\n");
        return root;
    }
    else if(delKey > root->data)
        root->right = deletion(root->right, delKey);
    else if(delKey < root->data)
        root->left = deletion(root->left, delKey);
    else
    {
        if(root->left == NULL)
        {
            temp = root->right;
            free(root);
            return temp;
        }
        else if(root->right == NULL)
        {
          temp = root->left;
          free(root);
          return temp;
        }
        temp = getInSuccessor(root->right);
        root->data = temp->data;
        root->right = deletion(root->right, temp->data);
      }
      return root;
    }
    int main()
    {
      int ch,n;
      struct node *root = NULL;
      while(1)
      {
        printf("Menu:\nBST
Operations:\n1.Insert\n2.Search\n3.Delete\n4.Display\n5.Exit\n");
        printf("Enter your choice: ");
        scanf("%d",&ch);
       switch(ch)
        {
          case 1:printf("Enter the element to insert a new node: ");
                 scanf("%d",&n);
      if(root==NULL)
                 root=insert(root, n);
              else
```

```
                  insert(root,n);
              break;
         case 2: printf("Enter the element to search: ") ;
               scanf("%d",&n);
               search(root,n);
               break ;
         case 3:printf("Enter node value which you want to delete: ");
              scanf("%d",&n);
              deletion(root, n);
              break ;
         case 4: printf("Inorder Traversal of BST: ");
               inorder(root);
               break;
         case 5: exit(0);
               break;
         default: printf("Wrong Choice\n");
      }
   }
}
```

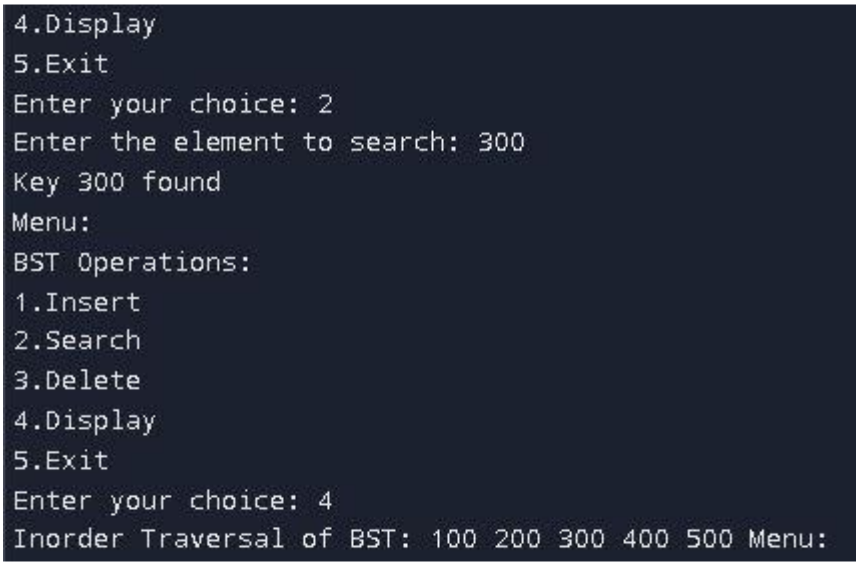
Output:



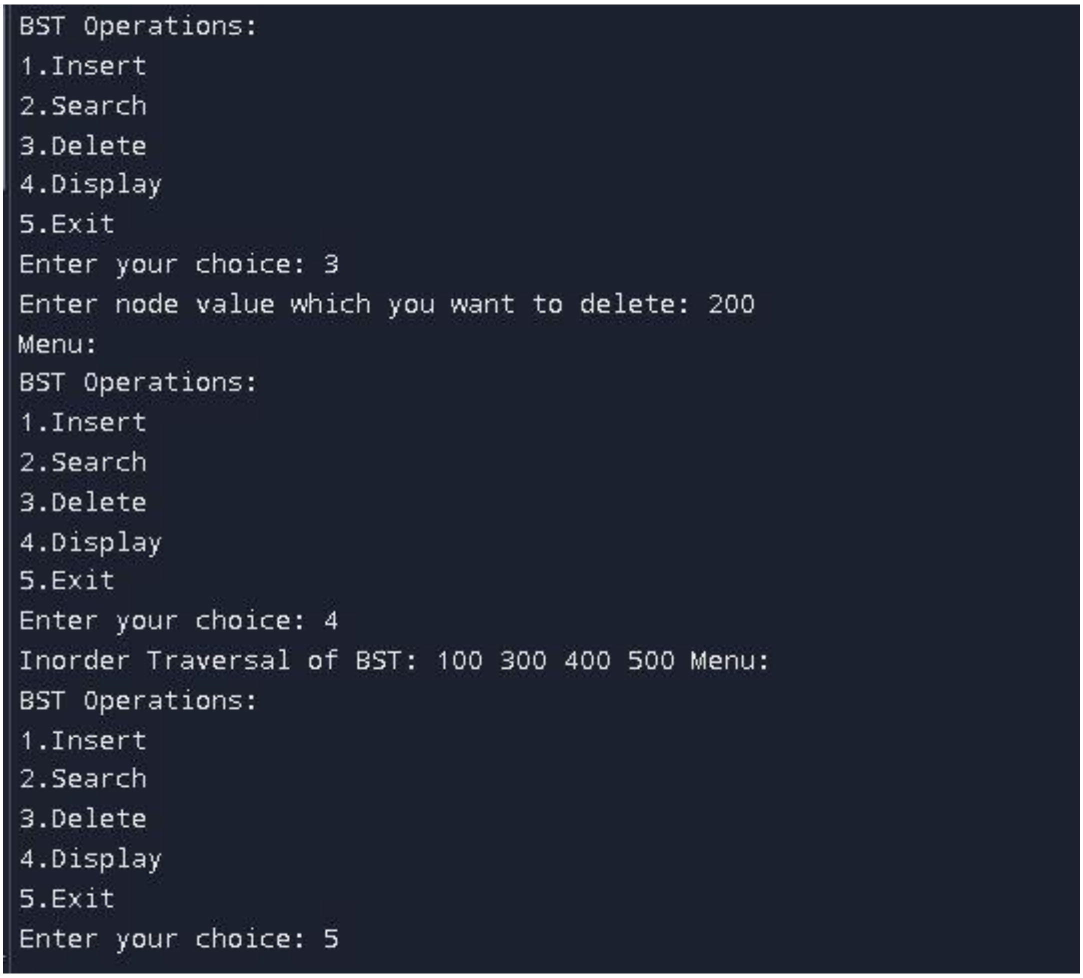
```
Menu:
BST Operations:
1.Insert
2.Search
3.Delete
4.Display
5.Exit
Enter your choice: 1
Enter the element to insert a new node: 100
Menu:
BST Operations:
1.Insert
2.Search
3.Delete
4.Display
5.Exit
```



```
Enter your choice: 1
Enter the element to insert a new node: 200
Menu:
BST Operations:
1.Insert
2.Search
3.Delete
4.Display
5.Exit
Enter your choice: 1
Enter the element to insert a new node: 300
Menu:
BST Operations:
1.Insert
2.Search
3.Delete
4.Display
5.Exit
Enter your choice: 1
Enter the element to insert a new node: 400
Menu:
BST Operations:
1.Insert
2.Search
3.Delete
```



```
4.Display
5.Exit
Enter your choice: 2
Enter the element to search: 300
Key 300 found
Menu:
BST Operations:
1.Insert
2.Search
3.Delete
4.Display
5.Exit
Enter your choice: 4
Inorder Traversal of BST: 100 200 300 400 500 Menu:
```



```
BST Operations:
1.Insert
2.Search
3.Delete
4.Display
5.Exit
Enter your choice: 3
Enter node value which you want to delete: 200
Menu:
BST Operations:
1.Insert
2.Search
3.Delete
4.Display
5.Exit
Enter your choice: 4
Inorder Traversal of BST: 100 300 400 500 Menu:
BST Operations:
1.Insert
2.Search
3.Delete
4.Display
5.Exit
Enter your choice: 5
```

Inferences:

- In binary search tree, the **insertion operation** ensures that every new element is placed in its correct position according to the BST property, maintaining the order automatically.
- The **search operation** efficiently locates an element by recursively or iteratively comparing it with the root and traversing either the left or right subtree — this reduces the average search time to **O(log n)** for balanced trees.
- The **display (traversal)** operation uses **recursive functions** such as in-order, pre-order, and post-order to visit and print all nodes in different sequences.
- **In-order traversal** displays the elements in **ascending (sorted)** order, verifying that the structure is a valid BST.