NAME-YASH GANDHI SE IT BATCH A 14
EXP-6

Binary Search Tree

Questions:

a. For the given sequence of keys build a BST

```
spit@DB-Lab-406-U16:~$ cd Desktop
spit@DB-Lab-406-U16:~/Desktop$ gcc bst.c
spit@DB-Lab-406-U16:~/Desktop$ ./a.out
Enter the number of data: 5
Enter data: 4
Enter data: 2
Enter data: 1
Enter data: 5
Enter data: 3

 Tree :
1 2 3 4 5

 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
1

Enter element to search: 4
  4 is present in the tree


 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2
```

```
 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
1

Enter element to search: 6
KEY ABSENT
```

b. Perform Deletion Operation in BST show all three cases of deletion

//BY PREDECESSOR

```
 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2

Enter element to delete: 4
Tree after deletion
1 2 3 5
Root-3


 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
3

Enter element to insert: 4
Tree after insertion
1 2 3 4 5

 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2

Enter element to delete: 1
Tree after deletion
2 3 4 5
Root-3


 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2

Enter element to delete: 5
Tree after deletion
2 3 4
Root-3
```

```
 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2

Enter element to delete: 6
KEY ABSENT
```

```
 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2

Enter element to delete: 4
Tree after deletion
2 3
Root-3


 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
3

Enter element to insert: 1
Tree after insertion
1 2 3

 ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
0
```

PROGRAM-

```
#include<stdio.h>
#include<stdlib.h>


typedef struct node  //structure for node
{
        int data;
        struct node*left;  //left and right child
        struct node*right;
}node;


node* create_bst(node* root,int key)  //function to create binary tree
{

        if(root==NULL)
        {
          root=(node*)malloc(sizeof(node));
     root->left=NULL;
          root->right=NULL;
          root->data=key;
                return root;
        }

        if(key<(root->data))
        {
        root->left=create_bst(root->left,key);
        return root;
        }
        else
        {
        root->right=create_bst(root->right,key);
        return root;
        }
}

node* form(node* root)  //function to insert
{
```

```
   int n,i,k;
   printf("Enter the number of data: ");
   scanf("%d",&n);
    for(i=0;i<n;i++)
        {
        printf("Enter data: ");
        scanf("%d",&k);
        root=create_bst(root,k);
        }

  return root;

}

void inorder(node*root)  //print inorder
{

        if(root!=NULL)//inorder representation
        {
                inorder(root->left);
                printf("%d ",root->data);
                inorder(root->right);
        }
}

node* get_min(node*root) //find predecessor
{

    node*temp=root;
    while(temp->right!=NULL)
    {
       temp=temp->right;
    }
    return temp;
}

void find(node*root,int x)
{
```

```
        if(root==NULL)
        {
                printf("KEY ABSENT\n");
                return ;

        }
        if(x==root->data)
        {
                printf("  %d is present in the tree\n",root->data);
                return;
        }
        if(x<(root->data))
        {
                find(root->left,x);
        }
        if(x>(root->data))
        {
           find(root->right,x);

        }
}


node* delete(node*root,int x) //delete and replace predecessor
{

   node*temp;

   if(root==NULL) //if node not present
   {
       printf("KEY ABSENT\n");
      return root;
   }

        if(x<root->data)  //searching of node
        {
           root->left=delete(root->left,x);
           return root;
        }
```

```
        else if(x>root->data)
        {
           root->right=delete(root->right,x);
           return root;
        }

        else
        {
        if(root->left==NULL && root->right==NULL)  //leaf node -delete
        {
              node*temp;
           temp=root;
           free(temp);
           return NULL;
        }
        else if(root->left==NULL) //single right child -delete
        {
              node*temp;
           temp=root;
           root=root->right;
           free(temp);
           return root;
        }
        else if(root->right==NULL) //single left child -delete
        {
              node*temp;
           temp=root;
           root=root->left;
           free(temp);
           return root;
        }

        temp=get_min(root->left);  //Root with 2 child -delete
        root->data=temp->data;
        root->left=delete(root->left,temp->data);

        }
}
```

```
void main()
{
  node* root=NULL;  //initialize root

  root=form(root);  //form tree

  printf("\n Tree :\n");
  inorder(root);



  int c=1;

  while(c==1)
  {
     int choice,x;  //function calls as per user

        printf("\n\n ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to
Exit\n"); //search and delete options

        scanf("%d",&choice);
        switch(choice)
        {

        case 0: c=0;
                break;

        case 1:
                printf("\nEnter element to search: ");
                scanf("%d",&x);

                 find(root,x);
           break;

    case 2:
        printf("\nEnter element to delete: ");
```

```
            scanf("%d",&x);

        root=delete(root,x);

        printf("Tree after deletion \n");
            inorder(root);

            printf("\nRoot-%d \n",root->data);
            break;

        case 3:
        printf("\nEnter element to insert: ");
            scanf("%d",&x);

        root=create_bst(root,x);

            printf("Tree after insertion \n");
            inorder(root);


        break;

        }

  }

}
```

OUTPUT

```
spit@DB-Lab-406-U16:~$ cd Desktop
spit@DB-Lab-406-U16:~/Desktop$ gcc bst.c
spit@DB-Lab-406-U16:~/Desktop$ ./a.out
Enter the number of data: 5
Enter data: 4
Enter data: 2
Enter data: 1
Enter data: 5
```

Enter data: 3

Tree :
1 2 3 4 5

ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
1

Enter element to search: 4
4 is present in the tree

ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2

Enter element to delete: 4
Tree after deletion
1 2 3 5
Root-3

ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
3

Enter element to insert: 4
Tree after insertion
1 2 3 4 5

ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2

Enter element to delete: 1
Tree after deletion
2 3 4 5
Root-3

ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2

Enter element to delete: 5
Tree after deletion
2 3 4
Root-3

ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
2

Enter element to delete: 4
Tree after deletion
2 3
Root-3

ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
3

Enter element to insert: 1
Tree after insertion
1 2 3

ENTER..... 1 to Search 2 to Delete 3 to Insert and 0 to Exit
0