Code Console

```
#include<iostream>

#include<string.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

using namespace std;

// Define a structure to be used as the tree node

struct TreeNode

{

   int      Key;

   float    fValue;

   int      iValue;

   char     cArray[7];

   TreeNode *left;

   TreeNode *right;

};


class Tree

{

   private:

      TreeNode *root;

   public:

      Tree();

      ~Tree();

      bool isEmpty();


      TreeNode *SearchTree(int Key);

      TreeNode *SearchNode(TreeNode *Node,int Key);
```

```
        void left_to_right(TreeNode *&p);
        void right_to_left(TreeNode *&p);
        int Insert(TreeNode *&temp,TreeNode *newNode);
        int Insert_with_ivalue(int Key, float f, int i, char *cA);
        int Insert_without_ivalue(int Key, float f, char *cA);

        void del(TreeNode *&p, int x);
        void del(TreeNode *&p);
        void Delete(int x);

        void PrintOne(TreeNode *T);
        void PrintTree();

        void splite(TreeNode *, TreeNode *&,TreeNode *&,int);
        void Split(Tree *,Tree *,int);

        void join(TreeNode *&main,TreeNode *a,TreeNode *b);
        void join(TreeNode *&a,TreeNode *b);
        void Join(Tree *a,Tree *b);
    private:
        void ClearTree(TreeNode *T);
        TreeNode *clone(TreeNode * T);
        void PrintAll(TreeNode *T);
};
/// ****************************************************
///             initailize Tree       (1)
Tree::Tree()
{
    root = NULL;
    return;
}
/// ****************************************************
/// ****************************************************
///              destroy Tree             (2)
Tree::~Tree()
{
   ClearTree(root);
   return;
}
void Tree::ClearTree(TreeNode *T)
{
   if(T==NULL) return;
   if(T->left != NULL) ClearTree(T->left);
   if(T->right != NULL) ClearTree(T->right);
   delete T;
   return;
}
/// ****************************************************
/// ****************************************************
///              IsEmpty           (3)
bool Tree::isEmpty()
{
   return(root==NULL);
}
/// ****************************************************
/// ****************************************************
///           return copy of node        (4)
TreeNode *Tree::clone(TreeNode * T)
{
   TreeNode *clone;

   clone = new TreeNode();
   *clone = *T;
   clone->left = NULL;
   clone->right = NULL;
   return clone;
}
/// ****************************************************
/// ****************************************************
///            search       (5)
TreeNode *Tree::SearchNode(TreeNode *Node,int Key)
{
   if (Node == NULL) return Node;
   else {
      if (Key < Node->Key)
         SearchNode(Node->left,Key);
      if (Key > Node->Key)
         SearchNode(Node->right,Key);
      else return(clone(Node));
   }
}
TreeNode *Tree::SearchTree(int Key)
{
   TreeNode * temp = root;
   SearchNode(temp,Key);
}
/// ****************************************************
/// ****************************************************
///          insert node       (6)
void Tree::left_to_right(TreeNode  *&p){
   TreeNode *q=p->left;
   p->left=q->right;
   q->right=p;
   p=q;
}
/*******************************/
```

```
void Tree::right_to_left(TreeNode *&p){
   TreeNode *q=p->right;
   p->right=q->left;
   q->left=p;
   p=q;
}


int Tree::Insert(TreeNode *&temp,TreeNode *newNode)
{
   if(temp == NULL){
         temp = new TreeNode();
         temp = clone(newNode);
   }
      else{
         if( newNode->Key < temp->Key ){
            Insert(temp->left, newNode);
            if(temp->left->iValue < temp->iValue) {
               left_to_right(temp);
            }
         }
         else{
            Insert(temp->right, newNode);
            if(temp->right->iValue < temp->iValue) {
               right_to_left(temp);
            }
         }
      }
  return(true);
}
int Tree::Insert_with_ivalue(int Key, float f, int i, char *cA)
{
   TreeNode *newNode;
```

```
    newNode = new TreeNode();
    newNode->Key = Key;
    newNode->fValue = f;
    newNode->iValue = i;
    strcpy(newNode->cArray, cA);
    newNode->left = newNode->right = NULL;

    TreeNode *temp = root;
    return(Insert(root,newNode));
}
int Tree::Insert_without_ivalue(int Key, float f, char *cA)
{
    TreeNode *newNode;
    newNode = new TreeNode();
    newNode->Key = Key;
    newNode->fValue = f;
    newNode->iValue = rand() % 1000 + 1 ;
    strcpy(newNode->cArray, cA);
    newNode->left = newNode->right = NULL;

    TreeNode *temp = root;
    return(Insert(root,newNode));
}
/// *****************************************************
/// *****************************************************
///          Deletete node       (7)
void Tree::del(TreeNode *&p, int x){
        if (p == NULL) return;
        if (p->Key == x) del(p);
        else
            if (x < p->Key) del(p->left,x);
        else del(p->right,x);
    }
/******************************/
void Tree::del(TreeNode *&p){
        if (p->left == NULL && p->right == NULL) {
            delete p;
            p = NULL;
            return;
        }
        if (p->left == NULL && p->right != NULL){
            right_to_left(p);
            del(p->left);
            return;
        }
        if (p->left != NULL && p->right == NULL){
            left_to_right(p);
            del(p->right);
            return;
        }
        if (p->left->iValue < p->right->iValue) {
            left_to_right(p);
            del(p->right);
            } else {
                right_to_left(p);
                del(p->left);
            }
        //update(p);
}
void Tree::Delete(int x){
    del(root,x);
}
/// ****************************************************
/// ****************************************************
///          Print            (8)
void Tree::PrintOne(TreeNode *T)
{
   cout << T->Key << "\t\t" << T->fValue << "\t\t" << T->iValue << "\t\t"
      << T->cArray << "\n";
}
void Tree::PrintAll(TreeNode *T)
{
   if(T != NULL)
   {
      PrintOne(T);
      PrintAll(T->left);
      //PrintOne(T);
      PrintAll(T->right);
   }
}
void Tree::PrintTree()
{
   PrintAll(root);
}
///****************************************************
///****************************************************
///          split
void Tree::splite(TreeNode *tree, TreeNode *&right_tree,TreeNode *&left_tree,int key )
{
   if(tree==NULL)
      left_tree=right_tree=NULL;
   else if (tree->Key == key){
      left_tree = tree->left;
      right_tree = tree->right;
   }else{
        if(key<tree->Key){
          right_tree = tree;
          splite(tree->left,right_tree->left,left_tree,key);
        }
        else{
          left_tree = tree;
          splite(tree->right,right_tree,left_tree->right,key);
        }
    }
}
void Tree::Split(Tree *the_left,Tree *the_right,int x){
    splite(root,the_left->root,the_right->root,x);
}
void Tree::join(TreeNode *&main,TreeNode *a,TreeNode *b){
    join(main,a);
    join(main,b);
}
void Tree::join(TreeNode *&a,TreeNode *b){
    if (b != NULL){

        TreeNode *c = new(TreeNode);
        c = clone(b);
        Insert(a,c);
        join(a,b->left);
        join(a,b->right);
    }
}
void Tree::Join(Tree *a,Tree *b){
    join(root,a->root,b->root);
}
/// ****************************************************
///          main
int main(void)
{
//-----------------------------------------------------------------------------------------------
    cout<<"try of split\n";


    Tree    *theTree;
    Tree     *the_right_tree;
    Tree     *the_left_tree;
    TreeNode       *newNode;
    // Do initialization stuff
    theTree = new Tree();
    the_left_tree=new Tree();
    the_right_tree=new Tree();


    cout <<"Building tree...\n";
    theTree->Insert_with_ivalue(8, 2.3f, 2, "Node1");
    theTree->Insert_with_ivalue(4, 3.4f, 4, "Node2");
    theTree->Insert_with_ivalue(12, 4.5f, 8, "Node3");
    theTree->Insert_with_ivalue(2, 5.6f, 16, "Node4");
    theTree->Insert_with_ivalue(6, 6.7f, 32, "Node5");
    theTree->Insert_with_ivalue(10, 7.8f, 64, "Node6");
    theTree->Insert_with_ivalue(14, 8.9f, 128, "Node7");
    theTree->Insert_with_ivalue(1, 9.0f, 256, "Node8");
    theTree->Insert_with_ivalue(3, 0.9f, 512, "Node9");
    theTree->Insert_with_ivalue(5, 9.8f, 1024, "Node10");
    theTree->Insert_with_ivalue(7, 8.7f, 2048, "Node11");
    theTree->Insert_with_ivalue(9, 7.6f, 4096, "Node12");
    theTree->Insert_with_ivalue(11, 6.5f, 8192, "Node13");
    theTree->Insert_with_ivalue(13, 5.4f, 16384, "Node14");
    theTree->Insert_with_ivalue(15, 4.3f, 32768, "Node15");
```

```
    theTree->Split(the_right_tree,the_left_tree,6);
    cout<<"\nthe first tree\n";
    the_left_tree->PrintTree();
    cout<<"\nthe scond tree\n";
    the_right_tree->PrintTree();
    cout <<"Done.\nPress Enter to continue...";
    cin.get();

    cout <<"----------------------------------------------------\n";
//---------------------------------------------------------------------------------------------



    cout<<"try of join function\n";


    Tree    *theTree;
    Tree    *theTree1;
    Tree    *theTree2;
    TreeNode        *newNode;


    // Do initialization stuff
    theTree = new Tree();
    theTree1 = new Tree();
    theTree2 = new Tree();
    //theTree1->Insert_with_ivalue(8, 2.3f, 2, "Node1");
    theTree2->Insert_with_ivalue(4, 3.4f, 4, "Node2");
    //theTree1->Insert_with_ivalue(12, 4.5f, 8, "Node3");
    theTree2->Insert_with_ivalue(2, 5.6f, 16, "Node4");
    //theTree1->Insert_with_ivalue(6, 6.7f, 32, "Node5");
    theTree2->Insert_with_ivalue(10, 7.8f, 64, "Node6");
    //theTree1->Insert_with_ivalue(14, 8.9f, 128, "Node7");
    theTree2->Insert_with_ivalue(1, 9.0f, 256, "Node8");
    //theTree1->Insert_with_ivalue(3, 0.9f, 512, "Node9");
    theTree2->Insert_with_ivalue(5, 9.8f, 1024, "Node10");
    //theTree1->Insert_with_ivalue(7, 8.7f, 2048, "Node11");
    theTree2->Insert_with_ivalue(9, 7.6f, 4096, "Node12");
    //theTree1->Insert_with_ivalue(11, 6.5f, 8192, "Node13");
    theTree2->Insert_with_ivalue(13, 5.4f, 16384, "Node14");
    theTree1->Insert_with_ivalue(15, 4.3f, 32768, "Node15");

    cout<<"the first tree\n";
    theTree1->PrintTree();

    cout<<"\nthe scond tree\n";
    theTree2->PrintTree();

    cout<<"\nthe result tree\n";
    theTree->PrintTree();

    cout<"----------------------------------\n";
    cout<"----------------------------------\n";
    theTree->Join(theTree1,theTree2);
    cout<"----------------------------------\n";
    cout<"----------------------------------\n";

    cout<<"\nthe result tree\n";
    theTree->PrintTree();

//-----------------------------------------------------------------------------------------------
    Tree    *theTree;
    TreeNode        *newNode;

    theTree = new Tree();

    cout <<"Building tree...\n";
```

```
theTree->Insert_with_ivalue(8, 2.3f, 32768, "Node1");
theTree->Insert_with_ivalue(4, 3.4f, 16384, "Node2");
theTree->Insert_with_ivalue(12, 4.5f, 8192, "Node3");
theTree->Insert_with_ivalue(2, 5.6f, 4096, "Node4");
theTree->Insert_with_ivalue(6, 6.7f, 2048, "Node5");
theTree->Insert_with_ivalue(10, 7.8f, 1024, "Node6");
theTree->Insert_with_ivalue(14, 8.9f, 512, "Node7");
theTree->Insert_with_ivalue(1, 9.0f, 256, "Node8");
theTree->Insert_with_ivalue(3, 0.9f, 128, "Node9");
theTree->Insert_with_ivalue(5, 9.8f, 64, "Node10");
theTree->Insert_with_ivalue(7, 8.7f, 32, "Node11");
theTree->Insert_with_ivalue(9, 7.6f, 16, "Node12");
theTree->Insert_with_ivalue(11, 6.5f, 8, "Node13");
theTree->Insert_with_ivalue(13, 5.4f, 4, "Node14");
theTree->Insert_with_ivalue(15, 4.3f, 2, "Node15");

cout <<"All nodes inserted\n";

cout <<"---------------------------------------------------\n";
theTree->PrintTree();
cout <<"Press Enter to continue...";
cin.get();
cout <<"---------------------------------------------------\n";

cout <<"---------------------------------------------------\n";
cout <<"Testing the search function\n";
newNode = theTree->SearchTree(13);
if(newNode != NULL)
{
   theTree->PrintOne(newNode);
   delete newNode;
}
```

```
else
   cout <<"Search key not found.\n";

newNode = theTree->SearchTree(6);
if(newNode != NULL)
{
   theTree->PrintOne(newNode);
   delete newNode;
}
else
   cout <<"Search key not found.\n";

newNode = theTree->SearchTree(1);
if(newNode != NULL)
{
   theTree->PrintOne(newNode);
   delete newNode;
}
else
   cout <<"Search key not found.\n";

newNode = theTree->SearchTree(25);
if(newNode != NULL)
{
   theTree->PrintOne(newNode);
   delete newNode;
}
else
   cout <<"Search key not found.\n";

cout <<"-----------------------------------------------------\n";
cout <<"Testing Deletete function\n";
cout <<"----------------------------------------------------\n";
cout <<"Testing Deleteting a leaf...\n";
theTree->Delete(4);
theTree->PrintTree();
 cout <<"Press Enter to continue...";
cin.get();
cout <<"----------------------------------------------------\n";

cout <<"----------------------------------------------------\n";
cout <<"Testing Deleteting a node with 2 children...\n";
theTree->Delete(7);
theTree->PrintTree();
cout <<"Press Enter to continue...";
cin.get();
cout <<"----------------------------------------------------\n";

cout <<"----------------------------------------------------\n";
cout <<"Testing Deleteting a node with 1 child...\n";
theTree->Delete(1);
theTree->PrintTree();
cout <<"Press Enter to continue...";
cin.get();
cout <<"----------------------------------------------------\n";

cout <<"----------------------------------------------------\n";
cout <<"Testing trying to Deletete a node that is not in the tree...\n";
theTree->Delete(20);
theTree->PrintTree();
cout <<"Press Enter to continue...";
cin.get();
cout <<"----------------------------------------------------\n";
```

```
    cout <<"-----------------------------------------------------\n";
    cout <<"Testing Deleteting the root...\n";
    theTree->Delete(15);
    theTree->PrintTree();
    cout <<"Done.\nPress Enter to continue...";
    cin.get();

    cout <<"-----------------------------------------------------\n";
//-----------------------------------------------------------------------------------------------
    return 0;
```