bst.h file:

#pragma once  
#include <iostream>  
#include <string>  
#include <vector>  
using namespace std;

class treenode {//node in a BST  
public:  
   string county\_name;  
   double population\_size;  
   treenode\* lchild, \* rchild; //left and right children pointers  
};

class bst {  
public:  
   bst(); // store the data file(“county\_data.txt”) into initial bst  
   ~bst(); //de-allocates dynamic memory allocate for tree  
   bool empty(); // checks to see if the tree is empty  
   void insert(const string& item, const double& population); //inserts a county record into bst based on county\_name.  
   void insert(treenode\*&, const string& item, const double& population); //see insert description above  
   void del\_name(string item); //deletes a county record based on county name.  
   void del\_name(treenode\*& loc\_ptr, string item); //see del description above  
   treenode\* search\_tree(treenode\*, string item); //returns pointer to node with county name  
   treenode\* search\_tree(string item); //see search\_tree description above  
   treenode\* inorder\_succ(treenode\*); //return pointer to inorder successor (based on county name).  
   void county\_ranges(const string& min\_name, const string& max\_name); //prints all county names  
   //to the screen between min\_name and max\_name, inclusive.  
   void print\_tree(); //prints each county record to the screen sorted by county name.  
   void print\_tree(treenode\*);  
   void sorted\_info();//prints each county record to a file called “county\_info.txt” sorted by county\_name.  
   void createSort(treenode\* node, vector<string>& v);  
   void print\_range(treenode\* root, const string& min, const string& max);

private:  
   treenode\* root;  
};

bst.cpp file:

#include "bst.h"  
#include <sstream>  
#include <fstream>

bst::bst() {  
   //create input file stream object  
   ifstream in("county\_data.txt");  
   string line;  
   while (getline(in, line)) {//get data line by line  
       vector<string> v; //crate vecctor object to hold seprate values from string  
       size\_t pos = 0;  
       string delimeter = " ";  
       string token;  
       while ((pos = line.find(delimeter)) != string::npos) {  
           token = line.substr(0, pos);  
           v.push\_back(token);  
           line.erase(0, pos + delimeter.length());  
       }  
       //last element in vector is pupulation of contry  
       double popul = stod(line);//convert string to double  
       //get name of county from vector  
       string name = v.at(0);  
       if (v.size() > 1) {  
           name.append(" ");  
           name.append(v.at(1));  
       }  
       //clear vector  
       v.clear();  
       // create node in tree  
       insert(name, popul);  
   }  
   // close file object when done reading  
   in.close();  
}

bst::~bst() {  
   while (root != 0){  
       del\_name(root->county\_name);  
   }  
}

bool bst::empty() {  
   return (root == 0);  
}

void bst::insert(const string& item, const double& population) {  
   // call overloaded function to insert item  
   insert(root, item, population);  
}

void bst::insert(treenode\*& loc\_ptr, const string& item, const double& population) {  
   // if root node is 0 then insert in root node  
   if (loc\_ptr == 0) {  
       loc\_ptr = new treenode;  
       loc\_ptr->lchild = loc\_ptr->rchild = 0;  
       loc\_ptr->county\_name = item;  
       loc\_ptr->population\_size = population;  
   }  
   // add node acording to county name in binary tree  
   else if (loc\_ptr->county\_name.compare(item) > 0) {  
       insert(loc\_ptr->lchild, item, population);  
   }  
   else if (loc\_ptr->county\_name.compare(item) < 0) {  
       insert(loc\_ptr->rchild, item, population);  
   }  
   // print message when county is already in list  
   else {  
       cout << "the county is already in the tree\n";  
   }  
}

treenode\* bst::search\_tree(string item) {  
   // search tree from root node  
   return search\_tree(root, item);  
}

treenode\* bst::search\_tree(treenode\* loc\_ptr, string item) {  
   //check if current node is not empty  
   if (loc\_ptr != 0) {  
       //if current node contians county name return pointer to current node  
       if (loc\_ptr->county\_name.compare(item) == 0) {  
           return loc\_ptr;  
       }  
       // else if corrent node's county name is greater then given contry name move to left side  
       else if (loc\_ptr->county\_name.compare(item) > 0) {  
           return search\_tree(loc\_ptr->lchild, item);  
       }  
       // else move to right side  
       else {  
           return search\_tree(loc\_ptr->rchild, item);  
       }  
   }  
   // if current node is NULL then return current node  
   else {  
       return loc\_ptr;  
   }  
}

void bst::del\_name(string item) {  
   //search from root to delete item  
   del\_name(root, item);  
}

void bst::del\_name(treenode\*& loc\_ptr, string item) {  
   // delete a node based on given county name  
   // check if current node is NULL  
   if (loc\_ptr == 0) {  
       cout << "item not in tree\n";  
   }  
   // if current node's county name is greater then given county name move to left in search space  
   else if (loc\_ptr->county\_name.compare(item) > 0) {  
       del\_name(loc\_ptr->lchild, item);  
   }  
   // if current node's county name is less then given county name move to right in search space  
   else if (loc\_ptr->county\_name.compare(item) < 0) {  
       del\_name(loc\_ptr->rchild, item);  
   }  
   else {//delete current node  
       treenode\* ptr;  
       //check if current node have only right child  
       if (loc\_ptr->lchild == 0) {  
           //delete current node and assign right child to current node  
           ptr = loc\_ptr->rchild;  
           delete loc\_ptr;  
           loc\_ptr = ptr;  
       }  
       //check if current node have only left child  
       else if (loc\_ptr->rchild == 0) {  
           //delete current node and assign left child to current node  
           ptr = loc\_ptr->lchild;  
           delete loc\_ptr;  
           loc\_ptr = ptr;  
       }  
       // current node have both left and right child  
       else {  
           // get inorder successor node  
           ptr = inorder\_succ(loc\_ptr);  
           //copy successor's data to current node  
           loc\_ptr->county\_name = ptr->county\_name;  
           loc\_ptr->population\_size = ptr->population\_size;  
           //delete successor node  
           del\_name(loc\_ptr->rchild, ptr->county\_name);  
       }  
   }  
}

treenode\* bst::inorder\_succ(treenode\* loc\_ptr) {  
   // get the right node  
   treenode\* ptr = loc\_ptr->rchild;  
   // from right node get the left most node that is successor to the given node  
   while (ptr->lchild != 0) {  
       ptr = ptr->lchild;  
   }  
   //return the successor node  
   return ptr;  
}

void bst::print\_tree() {  
   // print entire tree by calling auxilary function  
   print\_tree(root);  
}

void bst::print\_tree(treenode\* loc\_ptr) {  
   if (loc\_ptr != 0) {// if current node is not NULL  
       // move to left node  
       print\_tree(loc\_ptr->lchild);  
       // print current node's data  
       cout << loc\_ptr->county\_name << ", " << loc\_ptr->population\_size << endl;  
       // move to right node  
       print\_tree(loc\_ptr->rchild);  
   }  
}

void bst::county\_ranges(const string& min\_name, const string& max\_name) {  
   // print all county between minimum value and maximum value, inclusive  
   print\_range(root, min\_name, max\_name);  
}

void bst::print\_range(treenode\* node, const string& min, const string& max) {  
   if (node != 0) {// if current node is not NULL  
       // if min is less then current node's county name then move to left node  
       if (node->county\_name.compare(min) > 0) {  
           print\_range(node->lchild, min, max);  
       }  
       //if current node is between min and max value print current node's data  
       if (node->county\_name.compare(min) >= 0) {  
           if (node->county\_name.compare(max) <= 0) {  
               cout << node->county\_name << ", " << node->population\_size << endl;  
           }  
       }  
       // move to right node  
       // if max is greater then current node's county name then move to right node  
       if (node->county\_name.compare(max) < 0) {  
           print\_range(node->rchild, min, max);  
       }  
   }  
}

void bst::sorted\_info() {  
   // create output file object to write data to file  
   ofstream f;  
   f.open("county\_info.txt");  
   // set iterator to root  
   treenode\* i = root;  
   //create a vector object to store sorted data  
   vector<string> v;  
   createSort(root, v);  
   for(int i=0;i<v.size();i++){  
       string data = v.at(i);  
       f << data << endl;  
   }  
   //close file object  
   f.close();  
}

void bst::createSort(treenode\* node, vector<string> &v) {  
   if (node != 0){  
       createSort(node->lchild,v);  
       string str = node->county\_name + " " ;  
       stringstream s;  
       s << node->population\_size;  
       str = str + s.str();  
       v.push\_back(str);  
       createSort(node->rchild,v);  
   }  
}

bst\_driver.cpp file:

#include <iostream>  
#include "bst.h"  
using namespace std;

int main() {  
   cout << "Test1: default constructor\n";  
   bst myTree;  
   myTree.print\_tree();  
   cout << "End Test1\n";

   cout << "Test2:insert\n";  
   myTree.insert("New-county", 234658);  
   myTree.print\_tree();  
   cout << "End Test2\n";

   cout << "Test3: county\_ranges\n";  
   myTree.county\_ranges("Bbbb", "K");  
   cout << "End Test3\n";

   cout << "Test4: del\_name\n";  
   myTree.del\_name("miami-dade");  
   myTree.del\_name("Lee");  
   myTree.print\_tree();  
   cout << "End Test4\n";

   cout << "Test5: sorted\_info\n";  
   myTree.sorted\_info();  
   cout << "End Test5\n";

   return 0;

}