CSCI 140 PA 10 Submission

Due Date: 11/09/2021 Late (date and time):_____

Name(s): Nero Li

Exercise 1 -- need to submit source code and I/O

-- check if completely done ✔ ; otherwise, discuss issues below

Source code below:

```
/*  Program: PA_10_exercise_1
    Author: Nero Li
    Class: CSCI 220
    Date: 11/09/2021
    Description:
        Use SearchTree class in C++ book (modified by me and provided
here)
        and set up a test driver to perform some operations such as
insert,
        erase, and find. Perform the operations in question 1 below (steps
        1 to 7) and then search for 15, 30, and 8. Print the BST as the
final
        step. Assume that key is an integer and value is a string such as
a
        name (come up with your own names).
    I certify that the code below is my own work.

      Exception(s): N/A

*/
#include <iostream>
#include "bst.h"
#include "BinaryTree.h"
#include "Entry.h"
#include "RuntimeExceptions.h"

using namespace std;

void findKey(int key,SearchTree<Entry<int, char>> test,
SearchTree<Entry<int, char>>::Iterator itr)
{
    itr = test.find(key);
    if (!(itr == test.end()))
    {
        cout << (*itr).key() << ": " << (*itr).value() << endl;
    }
}
```

```
int main()
{
    SearchTree<Entry<int, char>> test;
    SearchTree<Entry<int, char>>::Iterator itr{NULL};

    test.insert(10, 'a');
    test.insert(20, 'b');
    test.insert(4, 'c');
    test.insert(8, 'd');
    test.insert(15, 'e');
    test.erase(8);
    test.erase(10);

    findKey(15, test, itr);
    findKey(30, test, itr);
    findKey(8, test, itr);

    itr = test.begin();
    while (!(itr == test.end()))
    {
        cout << (*itr).key() << ' ';
        ++itr;
    }
    cout << endl;

    cout << "Modified by: Nero Li\n";
    return 0;
}
```

Input/output below:

```
15: e
4 15 20
Modified by: Nero Li
```

Exercise 2 (with extra credit) -- need to submit source code and I/O

-- check if completely done ✔; otherwise, discuss issues below

Source code below:
bst.h:

```
    #ifndef BST_H
    #define BST_H
    // Modified for CSCI 220 Fall 15
    // Updated Fall 21

    #include "BinaryTree.h"
    #include "RuntimeExceptions.h"

    template <typename E>
    class SearchTree {
      // a binary search tree
```

```
    public:
      // public types
        typedef typename E::Key K;                                  // a
key
        typedef typename E::Value V;                            // a value
        class Iterator;
      // an iterator/position

        SearchTree(): T(), n(0)
        { T.addRoot(); T.expandExternal(T.root()); }    // create the
super root

      int size() const {                                            //
number of entries
            return n;
      };

      int trace()
      {
            return traceCount;
      }

      bool empty() const {                                          // is
the tree empty?
            return size() == 0;
      }

        Iterator find(const K& k)
      {
            traceCount = 0;
            TPos v = finder(k, root());                             //
search from virtual root
            if (!v.isExternal()) return Iterator(v);    // found it
            else return end();                                      //
didn't find it
        }

        Iterator insert(const K& k, const V& x)               // insert
(k,x)
        {
            traceCount = 0;
            TPos v = inserter(k, x);
            return Iterator(v);
      }

        void erase(const K& k) //throw(NonexistentElement) {
      {
            traceCount = 0;
            TPos v = finder(k, root());                             //
search from virtual root
            if (v.isExternal())
      // not found?
```

```
              throw NonexistentElement("Erase of nonexistent");
            eraser(v);
      // remove it
      }

        void erase(Iterator& p)                                        //
remove entry at p
        {
            traceCount = 0;
            eraser(p.v);
      }

        Iterator begin() {
            TPos v = root();                                           //
start at virtual root
            while (!v.isExternal()) v = v.left();       // find leftmost
node
            return Iterator(v.parent());
      }

        Iterator end()
      // iterator to end entry
        { return Iterator(T.root()); }                                 //
return the super root

      protected:
      // local utilities
        typedef BinaryTree<E> BinaryTree;                      // linked
binary tree
        typedef typename BinaryTree::Position TPos;            // position
in the tree

        TPos root() const { return T.root().left(); }   // left child of
super root

        TPos finder(const K& k, TPos v){
            //TPos finder(const K & k, TPos & v) {
            ++traceCount;
            if (v.isExternal())
                  return v;                // key not found
            if (k < (*v).key())
                  return finder(k, v.left());                          //
search left subtree
            else if ((*v).key() < k)
                  return finder(k, v.right());                 // search
right subtree
            else
                  return v;
      // found it here
        }
      /* this version alows duplicates
        TPos inserter(const K& k, const V& x) {
```

```
            TPos v = finder(k, root());                                 //
search from virtual root
            while (!v.isExternal())                                     // key
already exists?
              v = finder(k, v.right());                                 //
look further
            T.expandExternal(v);                                        // add
new internal node
            (*v).setKey(k); (*v).setValue(x);                 // set entry
    // operator -> is not overloaded
    //      v->setKey(k); v->setValue(x);                     // set entry
            n++;
      // one more entry
            return v;
      // return insert position
      }
      */

      // no duplicates -- modified by T. Vo
      TPos inserter(const K& k, const V& x) {
            TPos v = finder(k, root());                                 //
search from virtual root
            if (!v.isExternal())                                        // key
already exists?
                  (*v).setValue(x);                                     //
replace value
            else
            {
                  T.expandExternal(v);                                  // add
new internal node
                  (*v).setKey(k); (*v).setValue(x);           // set entry
                  n++;
      // one more entry
            }
            return v;
      // return insert position
      }

      TPos eraser(TPos& v) {
            TPos w;
            if (v.left().isExternal()) w = v.left();    // remove from
left
            else if (v.right().isExternal())
                  w = v.right();
      // remove from right
            else {
      // both internal?
              w = v.right();                                            // go
to right subtree
              do { w = w.left(); } while (!w.isExternal());   // get
leftmost node
              TPos u = w.parent();
```

```
              (*v).setKey((*u).key());
              (*v).setValue((*u).value());                    // copy w's
parent to v
            }
            n--;
      // one less entry
            return T.removeAboveExternal(w);                  // remove w
and parent
        }
    // not needed here
     //   TPos restructure(const TPos& v);                    //
restructure
            // throw(BoundaryViolation);

    private:
      // member data
        BinaryTree T;
      // the binary tree
        int n;
      // number of entries
      int traceCount;
      // number of Nodes that went through

    public:
        // ...insert Iterator class declaration here
        class Iterator {                                  // an
iterator/position
        private:
          TPos v;
      // which entry
        public:
          Iterator(const TPos& vv) : v(vv) { }                //
constructor

          const E& operator*() const { return *v; }           // get entry
(read only)

          E& operator*() { return *v; }                             // get
entry (read/write)

          bool operator==(const Iterator& p) const            // are
iterators equal?
        { return v == p.v; }

        Iterator& operator++( ){
            TPos w = v.right();
            if (!w.isExternal()) {                                //
have right subtree?
              do { v = w; w = w.left(); }                     // move down
left chain
              while (!w.isExternal());
            }
```

```
            else {
              w = v.parent();                                    // get
parent
              while (v == w.right())                            // move up
right chain
                  { v = w; w = w.parent(); }
              v = w;
      // and first link to left
            }
            return *this;
        }

          friend class SearchTree;                               //
give search tree access
        };
      };
    #endif
```

exercise_2.cpp:

```
/*  Program: PA_10_exercise_2
    Author: Nero Li
    Class: CSCI 220
    Date: 11/09/2021
    Description:
        You will implement a simple population database for California
counties
        using a simple search tree from exercise 1 to store the database
records.
        Define and implement PopMap class that supports standard map
operations
        using county code as a key for each record (no duplicate keys).
Your
        PopMap class uses binary search tree to store population records.
Download
        the data file p4small.txt, containing a list of a few population
records
        – county code, population in million, and county with state
abbreviation
        (3 fields separated by commas). Build the search tree from the
records of
        the input data file by inserting one record at a time to the tree.

    I certify that the code below is my own work.

      Exception(s): N/A

*/
#include <iostream>
#include <fstream>
#include <string>
#include "bst.h"
```

```
#include "BinaryTree.h"
#include "Entry.h"
#include "RuntimeExceptions.h"

using namespace std;

class PopMap
{
private:
    struct County
    {
        int pop;
        string county;
    };
    SearchTree<Entry<int,County>> countyTree;
    SearchTree<Entry<int,County>>::Iterator itr{NULL};
public:
    // constructor accepts file name and construct search tree
    PopMap(string filename)
    {
        ifstream fin;
        string countyData;

        fin.open(filename, ios::binary);
        while (!fin.eof())
        {
            County newData;
            int code{-1};
            bool gotKey{false};
            newData.pop = -1;
            newData.county = "";

            getline(fin, countyData);
            for (int i = 0; i < countyData.size(); ++i)
            {

                if (countyData[i] == ',')
                {
                    gotKey = true;
                }
                else if (countyData[i] >= '0' && countyData[i] <= '9')
                {
                    if (gotKey)
                    {
                        if (newData.pop == -1)
                        {
                            newData.pop = countyData[i] - '0';
                        }
                        else
                        {
                            newData.pop *= 10;
                            newData.pop += countyData[i] - '0';
```

```
                    }
                }
                else
                {
                    if (code == -1)
                    {
                        code = countyData[i] - '0';
                    }
                    else
                    {
                        code *= 10;
                        code += countyData[i] - '0';
                    }
                }
            }
            else if (countyData[i] == '\"')
            {
                ++i;
                while (countyData[i] != '\"')
                {
                    newData.county += countyData[i];
                    ++i;
                }
            }
        }
        countyTree.insert(code, newData);
    }
    countyTree.erase(-1);
}

// print appropriate message and data if found
void find(int code)
{
    itr = countyTree.find(code);
    if (itr == countyTree.end())
    {
        cout << "Nothing found.\n";
    }
    else
    {
        cout << (*itr).key() << "," << (*itr).value().pop << ",\"" <<
(*itr).value().county << "\"" << endl;
    }

    numberOfNodesExamined("search", countyTree.trace());
    cout << endl;
}

// print appropriate message and insert node if not found
// replace data if found
void insert(int code, int pop, string county)
{
```

```
        County newData;
        newData.county = county;
        newData.pop = pop;

        itr = countyTree.find(code);
        if (itr == countyTree.end())
        {
            cout << "Inserting a new data...\n";
        }
        else
        {
            cout << "Replacing exist data...\n";
        }

        countyTree.insert(code, newData);
        numberOfNodesExamined("insert", countyTree.trace());
        cout << endl;
    }

    // print appropriate message and erase node if found
    void erase(int code)
    {
        itr = countyTree.find(code);
        if (itr == countyTree.end())
        {
            cout << "Nothing found...\n";
            numberOfNodesExamined("erase", countyTree.trace());
        }
        else
        {
            cout << "Found data:\n";
            cout << (*itr).key() << "," << (*itr).value().pop << ",\"" <<
(*itr).value().county << "\"" << endl;
            countyTree.erase(code);
            cout << "Data erased...\n";
            numberOfNodesExamined("erase", countyTree.trace());
        }

        cout << endl;
    }

    // print one record per line using an in-order traversal
    void print()
    {
        itr = countyTree.begin();
        while (!(itr == countyTree.end()))
        {
            cout << (*itr).key() << "," << (*itr).value().pop << ",\"" <<
(*itr).value().county << "\"" << endl;
            ++itr;
        }
        cout << endl;
```

```
    }
protected:
    void numberOfNodesExamined(string op, int num)
    {
        cout << "Number of nodes examined for " << op << ": " << num <<
endl;
    }
};

void menu()
{
    cout << "   Operating Menu\n" << endl;
    cout << "1. List all records" << endl;
    cout << "2. Search for record" << endl;
    cout << "3. Insert new record" << endl;
    cout << "4. Delete a record" << endl;
    cout << "5. Exit program" << endl;
    cout << endl;
}

int main()
{
    PopMap p4small("p4small.txt");
    int choice;
    int code;
    int pop;
    string county;
    bool exitCode{true};

    menu();
    while (exitCode)
    {
        cout << "Please input your option: \n";
        cin >> choice;
        switch (choice)
        {
        case 1:
            p4small.print();
            break;
        case 2:
            cout << "Please input the code: \n";
            cin >> code;
            p4small.find(code);
            break;
        case 3:
            cout << "Please input the code: \n";
            cin >> code;
            cout << "Please input the population: \n";
            cin >> pop;
            cout << "Please input the county data: \n";
            getline(cin, county);
            getline(cin, county);
```

```
            p4small.insert(code, pop, county);
            break;
        case 4:
            cout << "Please input the code: \n";
            cin >> code;
            p4small.erase(code);
            break;
        case 5:
            exitCode = false;
            cout << endl;
            break;
        default:
            break;
        }
    }

    cout << "Modified by: Nero Li\n";
    return 0;
}
```

Input/output below:

```
   Operating Menu

1. List all records
2. Search for record
3. Insert new record
4. Delete a record
5. Exit program

Please input your option:
1
6001,3648,"Alameda, CA"
6019,1242,"Fresno, CA"
6037,22851,"Los Angeles, CA"
6047,341,"Merced, CA"
6055,225,"Napa, CA"
6059,6214,"Orange, CA"
6065,1784,"Riverside, CA"
6067,1809,"Sacramento, CA"
6071,1920,"San Bernardino, CA"
6073,5351,"San Diego, CA"
6075,2039,"San Francisco, CA"
6083,721,"Santa Barbara, CA"
6097,655,"Sonoma, CA"
6111,1130,"Ventura, CA"

Please input your option:
2
Please input the code:
6037
6037,22851,"Los Angeles, CA"
```

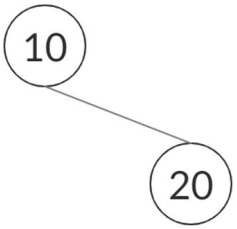
```
Number of nodes examined for search: 5

Please input your option:
2
Please input the code:
6000
Nothing found.
Number of nodes examined for search: 5

Please input your option:
3
Please input the code:
6066
Please input the population:
1
Please input the county data:
New County, CA
Inserting a new data...
Number of nodes examined for insert: 5

Please input your option:
3
Please input the code:
6065
Please input the population:
2000
Please input the county data:
Riverside, CA
Replacing exist data...
Number of nodes examined for insert: 3

Please input your option:
4
Please input the code:
6999
Nothing found...
Number of nodes examined for erase: 6

Please input your option:
4
Please input the code:
6075
Found data:
6075,2039,"San Francisco, CA"
Data erased...
Number of nodes examined for erase: 2

Please input your option:
4
Please input the code:
6055
Found data:
```

```
6055,225,"Napa, CA"
Data erased...
Number of nodes examined for erase: 5

Please input your option:
1
6001,3648,"Alameda, CA"
6019,1242,"Fresno, CA"
6037,22851,"Los Angeles, CA"
6047,341,"Merced, CA"
6059,6214,"Orange, CA"
6065,2000,"Riverside, CA"
6066,1,"New County, CA"
6067,1809,"Sacramento, CA"
6071,1920,"San Bernardino, CA"
6073,5351,"San Diego, CA"
6083,721,"Santa Barbara, CA"
6097,655,"Sonoma, CA"
6111,1130,"Ventura, CA"

Please input your option:
5

Modified by: Nero Li
```

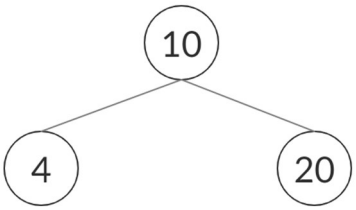
Answer for Question 1:

1. Insert 10

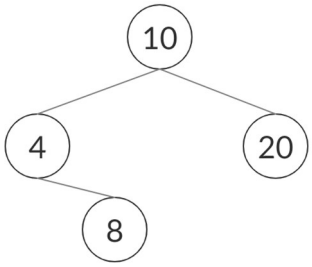
2. Insert 20



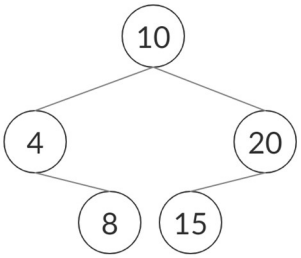
3. Insert 4



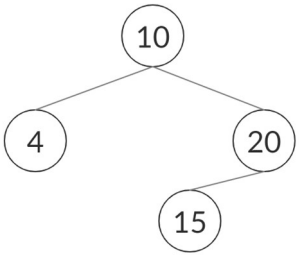
4. Insert 8



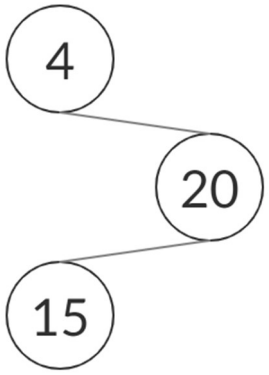
5. Insert 15



6. Erase 8



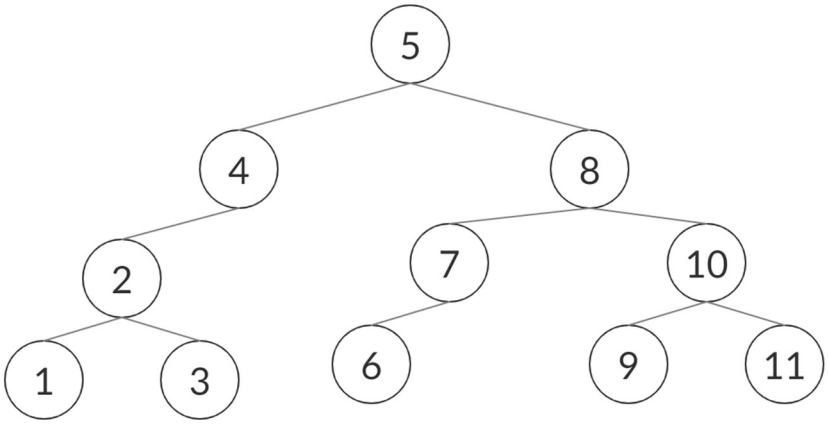
7. Erase 10



The tree that has shown in the final step is also the result tree after operations.

Answer for Question 2:

Here is an example Binary Search Tree:



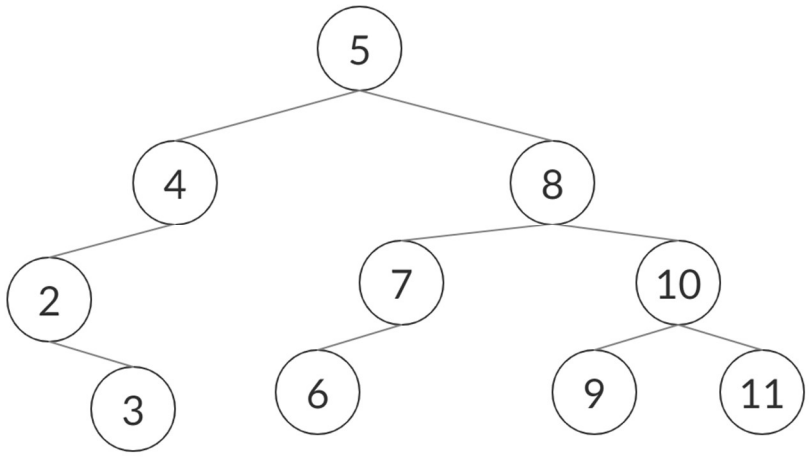
The in-order traversal for this tree will be:

1 2 3 4 5 6 7 8 9 10 11

For removing a node in a BST:

- If the node is an external node, directly remove that node.

  For example, after we remove key 1 in the binary tree shown above, it will look like:

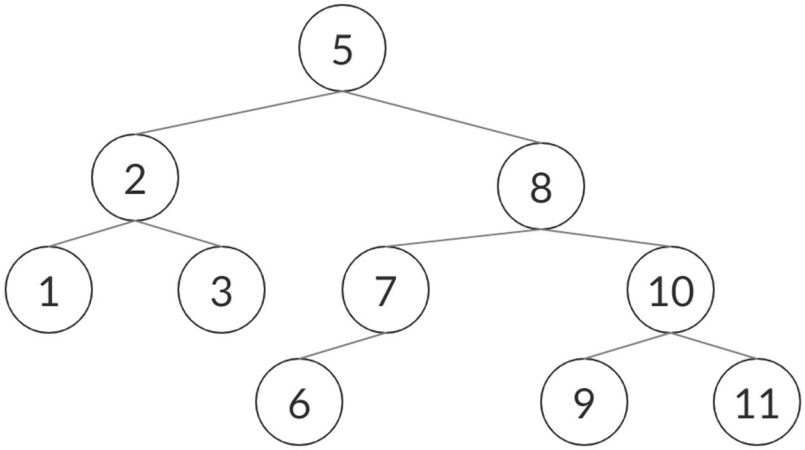


The in-order traversal for this tree will be:

2 3 4 5 6 7 8 9 10 11

- If the node has only left child or right child, use that child to replace the node that we plan to remove.

  For example, after we remove key 4 in the binary tree shown above, it will look like:

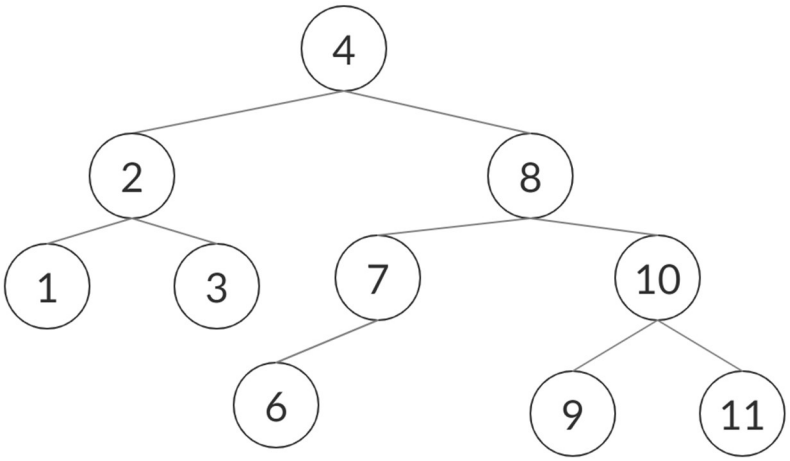


The in-order traversal for this tree will be:

1 2 3 5 6 7 8 9 10 11

- If the node has both left child and right child, use the left child to replace the node that we plan to remove.

  For example, after we remove key 5 in the binary tree shown above, it will look like:



The in-order traversal for this tree will be:

1 2 3 4 6 7 8 9 10 11