CSCI 140 PA 11 Submission

Due Date: 11/16/2021 Late (date and time):_____

Name(s): Nero Li

Header file for both exercise

AVLEntry.h:

```
#ifndef AVLENTRY_H
#define AVLENTRY_H
// Updated Fall 21

template <typename E>
class AVLEntry : public E {                          // an AVL entry
private:
    int ht;                                            // node height

protected:
    typedef typename E::Key K;                    // key type
    typedef typename E::Value V;                  // value type
    int height() const { return ht; }       // get height
    void setHeight(int h) { ht = h; }       // set height

public:
    AVLEntry(const K& k = K(), const V& v = V())  // constructor
        : E(k, v), ht(0) { }

    int getHeight() const { return ht; }
    template <typename F>
    friend class AVLTree;                          // allow AVLTree access
};
#endif
```

AVLTree1.h:

```
    #ifndef AVLTREE_H
    #define AVLTREE_H
    // Updated Fall 21
    // Should work reasonably well.
    // Report bugs and possible fixes for extra credit.

    #include <list>
    #include "AVLEntry.h"
    #include "bst1.h"

    template <typename E>                                          // an AVL
tree
```

```
    class AVLTree : public SearchTree< AVLEntry<E> > {
    public:
        typedef AVLEntry<E> AVLEntry;                 // an entry
        typedef typename SearchTree<AVLEntry>::Iterator Iterator; // an
iterator

    protected:
        typedef typename AVLEntry::Key K;              // a key
        typedef typename AVLEntry::Value V;            // a value
        typedef SearchTree<AVLEntry> ST;               // a search tree
        typedef typename ST::TPos TPos;                    // a tree
position
        typedef typename ST::BinaryTree::PositionList PositionList;

    public:
        AVLTree() : ST() { }                          // constructor

        Iterator insert(const K& k, const V& x)   // insert (k,x)
        {
            TPos v = ST::inserter(k, x);                     // insert in
base tree
            setHeight(v);                                 // compute its
height
            rebalance(v);                                 // rebalance if
needed
            return Iterator(v);
        }

        void erase(const K& k)                     // remove key k entry
        {
            TPos v = ST::finder(k, ST::root());         // find in base
tree
            if (Iterator(v) == ST::end())               // not found?
                throw NonexistentElement("Erase of nonexistent");
            TPos w = ST::eraser(v);                                //
remove it
            rebalance(w);                                        //
rebalance if needed
        }

        void erase(Iterator p) {                          // remove entry
at p
            ST::eraser(p.v);
        }

        int countDepth(TPos v)
        {
            if (v.isRoot())
            {
                return 0;
            }
            else
```

```
            {
                return 1 + countDepth(v.parent());
            }
        }

        void draw()
        {
            PositionList pl;
            int maxHeight;
            pl = ST::getTree().positions();
            maxHeight = height(ST::root());
            while (!pl.empty())
            {
                if ((*pl.front()).key())
                {
                    for (int i = 1; i < countDepth(pl.front()); ++i)
                    {
                        cout << "    ";
                    }
                    cout << (*pl.front()).key() << endl;
                }
                pl.pop_front();
            }
        }

    protected:
        int height(TPos v) const                          // node height
utility
        {
            return (v.isExternal() ? 0 : (*v).height());
        }

        void setHeight(TPos v)                            // set height
utility
        {
            int hl = height(v.left());
            int hr = height(v.right());
            (*v).setHeight(1 + std::max(hl, hr)); // max of left & right
        }

        bool isBalanced(const TPos& v) const            // is v balanced?
        {
            int bal = height(v.left()) - height(v.right());
            return ((-1 <= bal) && (bal <= 1));
        }

        TPos tallGrandchild(const TPos& z) const  // get tallest
grandchild
        {
            TPos zl = z.left();
            TPos zr = z.right();
```

```
            if (height(zl) >= height(zr))               // left child
taller
                if (height(zl.left()) >= height(zl.right()))
                    return zl.left();
                else
                    return zl.right();
            else                                    // right child taller
                if (height(zr.right()) >= height(zr.left()))
                    return zr.right();
                else
                    return zr.left();

        }

        void rebalance(const TPos& v)                 // rebalance utility
        {
            TPos z = v;
            while (!(z == ST::root())) {                // rebalance up to
root
                z = z.parent();
                setHeight(z);                               // compute new
height
                if (!isBalanced(z)) {                   // restructuring
needed
                    TPos x = tallGrandchild(z);
                    z = ST::restructure(x);                 // trinode
restructure
                    setHeight(z.left());                // update heights
                    setHeight(z.right());
                    setHeight(z);
                }
            }
        }
    };

    #endif
```

Exercise 1 -- need to submit source code and I/O
-- check if completely done ✔ ; otherwise, discuss issues below
Source code below:

```
/*  Program: PA_11_exercise_1
    Author: Nero Li
    Class: CSCI 220
    Date: 11/16/2021
    Description:
        Use AVLTree 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 "Entry.h"
#include "AVLEntry.h"
#include "AVLTree1.h"

using namespace std;

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

int main()
{
    AVLTree<AVLEntry<Entry<int, char>>> test;
    AVLTree<AVLEntry<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,2
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:

```
/*  Program: PA_11_exercise_2
    Author: Nero Li
    Class: CSCI 220
    Date: 11/16/2021
    Description:
        You will implement a better population database for California
counties
        using an AVL tree from exercise 1 to store the database records.
Define
        and implement PopBetterMap class that supports standard map
operations
        using county code as a key for each record (no duplicate keys).
Your
        PopBetterMap class uses an AVL 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 AVL 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 <list>
#include "Entry.h"
#include "AVLEntry.h"
#include "AVLTree1.h"

using namespace std;
```

```
class PopBetterMap
{
private:
    struct County
    {
        int pop;
        string county;
    };
    AVLTree<AVLEntry<Entry<int, County>>> countyTree;
    AVLTree<AVLEntry<Entry<int, County>>>::Iterator itr{NULL};
public:
    // constructor accepts file name and construct search tree
    PopBetterMap(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;
        }

        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);
        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";
        }
        else
        {
            cout << "Found data:\n";
            cout << (*itr).key() << "," << (*itr).value().pop << ",\"" <<
(*itr).value().county << "\"" << endl;
            countyTree.erase(code);
            cout << "Data erased...\n";
        }

        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;
    }

    // Draw the tree (key only)
    void draw()
    {
        countyTree.draw();
    }
};
```

```
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. Draw the tree" << endl;
    cout << "6. Exit program" << endl;
    cout << endl;
}

int main()
{
    PopBetterMap 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:
```

```
            p4small.draw();
            break;
        case 6:
            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. Draw the tree
6. 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:
5
6059
    6019
        6001
        6047
            6037
            6055
    6075
```

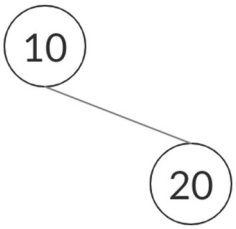
```
        6071
            6065
                6067
            6073
        6097
            6083
            6111
Please input your option:
2
Please input the code:
6037
6037,22851,"Los Angeles, CA"

Please input your option:
2
Please input the code:
6000
Nothing found.

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...

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...

Please input your option:
4
Please input the code:
6999
Nothing found...

Please input your option:
4
Please input the code:
6075
Found data:
6075,2039,"San Francisco, CA"
Data erased...
```

```
Please input your option:
4
Please input the code:
6055
Found data:
6055,225,"Napa, CA"
Data erased...

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
6059
    6019
        6001
        6047
            6037
    6083
        6071
            6066
                6065
                6067
            6073
        6097
            6111
Please input your option:
6

Modified by: Nero Li
```

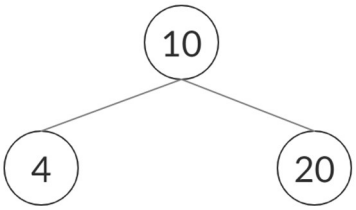
Answer for Question 1:

1. Insert 10

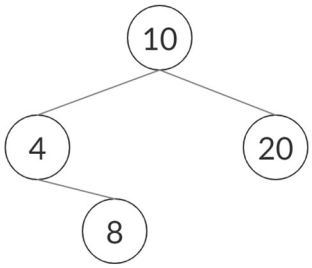
2. Insert 20



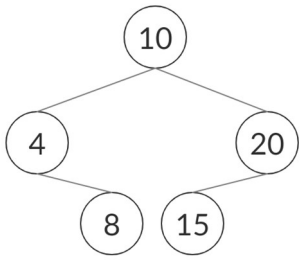
3. Insert 4



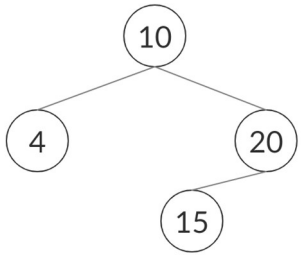
4. Insert 8



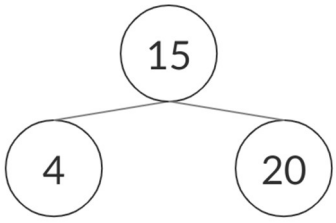
5. Insert 15



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:

For a Splay tree, it is another kind of balanced binary search tree, and it will use splaying to make itself become a balanced binary tree when the function went to the external node. It guarantees the running time for search, insert, and remove be O(logn).

Since we need to reconstruct the tree during the operation, I suggest using the inheritance from an AVL Tree to create our Splay Tree class since the rotate is a good function for us to move a node to its root, which is also the main idea for the

Splay tree. Because of that, creating a new splay(Tpos v) function and using rotate to make sure node v has become the root is a good idea for the class.