```
//============================================================
// Professional.cpp
// This file contains exercepts hilighting my commenting
// style.
//============================================================
//============================================================
// SET CLASS
//============================================================

//============================================================
//operator ==
//returns a boolean value indicating if the two sets are equal
//============================================================
template <class Element>
bool    Set<Element>::operator==        (const Set<Element> & s) const
{
        if (length != s.length)
                return false;

        else
        {
                Node<Element> * ptr = head;

                while (ptr != NULL)
                {
                        if (!s.contains(ptr->data))
                                return false;
                        ptr = ptr->next;
                }

                return true;
        }
}

//============================================================
//operator <=
//returns a boolean value indicating if the set is a subset
//of another set
//============================================================
template <class Element>
bool    Set<Element>::operator<=        (const Set<Element> & s) const
{
        if (s.length == 0)
                return true;                                    //empty set is always
                                                                        //a subset
        Node<Element> * ptr = head;

        while (ptr != NULL)
        {
                if (!s.contains(ptr->data))
                        return false;

                ptr = ptr->next;
        }

        return true;
}

//============================================================
//operator +
//returns the union of two sets
//============================================================
template <class Element>
```

```
Set<Element>&   Set<Element>::operator+ (const Set<Element> & s) const
{
        Set<Element> *s1 = new Set();
        Node<Element> * ptr = head;

        while (ptr != NULL)                             //inserts elements from
        {                                                               //first set
                s1->insert(ptr->data);
                ptr = ptr->next;
        }

        Node<Element> * qtr = s.head;
        while (qtr != NULL)                             //inserts elements from
        {                                                               //second set
                s1->insert(qtr->data);
                qtr=qtr->next;
        }

        return *s1;

}
//=============================================================
// RBT CLASS
//=============================================================

//=============================================================
//insert
//inserts item into RBT
//Pre-Condition:
//Post-Condition:
//Parameters: KeyType *k - pointer to item to be inserted
//Return Value: void
//=============================================================
template <class KeyType>
void            RBT<KeyType>::insert    (KeyType *k)
{
        Node<KeyType> *parent = nil;
        Node<KeyType> *current = root;
        Node<KeyType> *newNode = new Node<KeyType>(k, nil);
        //newNode->left = nil;
        //newNode->right = nil;
        while (current != nil)          //find where to insert newNode
        {
                parent = current;
                if (*newNode->data > *current->data)
                {
                        current = current->right;
                }
                else
                {
                        current = current->left;
                }
        }
        if (current == nil)                     //newNode inserted at root
        {
                root = newNode;
        }
        if (current == current->p->left)        //newNode should be
        {                                                                               //inserted to
left
                parent->left = newNode;
        }
        else
```

```
        {
                parent->right = newNode;                //inserted to right
        }

        RBTFix(newNode);                //fix black heights and balance tree


}
//=============================================================
//LeftRotate
//rotates the node left
//Pre-Condition:
//Post-Condition:
//Parameters: Node *current - pointer to node to be rotated
//Return Value: void
//=============================================================
template <class KeyType>
void            RBT<KeyType>::LeftRotate(Node<KeyType> *z)
{
        Node<KeyType> y = z->right;     //sets y to z's right child
        z->right = y->left;             //sets z's right subtree to y's
        if (y->left != nil)                             //left subtree
        {
                y->left->p = z;         //sets y's left subtree parent to z
        }
        y->p = z->p;                            //sets y's parent to z's parent
        if (z->p == nil)                                                //case 1: z is root
        {
                root = y;
        }
        else if (z == z->p->left)               //case 2: z is left child
        {
                z->p->left = y;
        }
        else                                                    //case 3: z is right child
        {
                z->p->right = y;
        }
        y->left = z;                                                    //put z on y's left
        z->p = y;
}
//=============================================================
//RightRotate
//rotates node right
//Pre-Condition:
//Post-Condition:
//Parameters:  Node *current - pointer to node to be rotated
//Return Value: void
//=============================================================
template <class KeyType>
void            RBT<KeyType>::RightRotate(Node<KeyType> *z)
{
        Node<KeyType> y = z->left;              //sets y to z's left child
        z->left = y->right;             //sets y's right subtree to
        if (y->right != nil)            //z's left subtree
        {
                y->right->p = z;
        }
        y->p = z->p;
        if (z->p == nil)                                //case 1: z is root
        {
                root = y;
        }
```

```
        else if (z == z->p->left)               //case 2: z is left child
        {
                z->p->left = y;
        }
        else                                    //case 3: z is right child
        {
                z->p->right = y;
        }
        y->right = z;                           //put z on y's right
        z->p = y;
}
//=============================================================
//RBTFix
//"fixes" RBT to uphold RBT properties after an insert
//Pre-Condition:
//Post-Condition:
//Parameters: Node *current - pointer to node to be rotated
//Return Value: void
//=============================================================
template <class KeyType>
void            RBT<KeyType>::RBTFix    (Node<KeyType> *current)
{
        while (current->p->color == "red")
        {
                if(current->p == current->p->p->left)//parent is l child
                {
                        Node<KeyType> *uncle = current->p->p->right;
                        if (uncle->color == "red")      //if uncle red can just
                        {                                                       //change color
 of parent
                                current->p->color = "black";//and uncle to fix
                                uncle->color = "black";                 //black height
                                current->p->p->color = "red";
                                current = current->p->p;
                        }
                        else
                        {
                                if (current == current->p->right)
                                {
                                        current = current->p;   //makes current a
                                        LeftRotate(current);    //left child
                                }
                                current->p->color = "black";
                                current->p->p->color = "red";//fixes colors
                                RightRotate(current->p->p);
                        }
                }
                else                    //parent is right child
                {
                        Node<KeyType> *uncle = current->p->p->left;
                        if (uncle->color == "red")
                        {                                                       //switches col
or
                                current->p->color = "black";//of uncle
                                uncle->color = "black"; //and parent to fix
                                current->p->p->color = "red";//black height
                                current = current->p->p;
                        }
                        else
                        {
                                if (current == current->p->left)
                                {
                                        current = current->p;
```

```
                                        RightRotate(current);//makes current a right
                                    }                                                   //chil
d
                                    current->p->color = "black";
                                    current->p->p->color = "red";//fixes colors
                                    LeftRotate(current->p->p);

                                }
                            }
                }
        root->color = "black";          //fixes root color to black
}


//=========================================================
// SET TEST CASES
//=========================================================


//=========================================================
// tests default constructor
//=========================================================

void test1 (void)
{
        Set<int> s1;
        string str = s1.toString();
        assert(str=="{}");

}

//=========================================================
// tests insert
//=========================================================

void test2 (void)
{
        Set<int> s1;
        s1.insert(1);
        s1.insert(4);
        s1.insert(89);
        s1.insert(3);
        string str = s1.toString();
        assert(str=="{1, 4, 89, 3}");
}

//=========================================================
// tests copy constructor
//=========================================================

void test3 (void)
{
        Set<int> s1;
        s1.insert(4);
        s1.insert(44);
        s1.insert(55);
        Set<int> s2(s1);
        string str = s2.toString();

        assert(str == "{4, 44, 55}");
}

//=========================================================
```

```
// tests remove
//============================================================

void test4 (void)
{
        Set<char> s1;
        s1.insert('s');
        s1.insert('e');
        s1.insert('a');
        s1.insert('g');
        s1.insert('z');

        s1.remove('a');

        string str = s1.toString();
        assert(str == "{s, e, g, z}");
}

//============================================================
// tests cardinality
//============================================================

void test5 (void)
{
        Set<int> s1;
        s1.insert(4);
        s1.insert(8);
        s1.insert(12);
        s1.insert(16);
        int length = s1.cardinality();
        assert (length == 4);
}

//============================================================
// tests ==
//============================================================
void test6 (void)
{
        Set<char> s1;
        Set<char> s2;
        for (char letter = 'a';letter<='z';letter++)
        {
                s1.insert(letter);
                s2.insert(letter);
        }

        if (s1==s2)
                return;

        else
                cout << "Test 6 failed." << endl;
}
//============================================================
// tests contains
//============================================================

void test7 (void)
{
        Set<int> s1;
        for (int i = 0; i < 8; i++)
                s1.insert(i);
        if (s1.contains(3))
                return;
```

```
        else
                cout << "Test 7 failed" << endl;
}


//=============================================================
// tests union
//=============================================================

void test8 (void)
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 6; i<11; i++)
                s2.insert(i);
        Set<int> s3;
        s3.insert(5);
        s3.insert(3);
        s3.insert(6);
        s3 = s1+s2;
        string str = s3.toString();
        assert(str == "{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}");
}

//=============================================================
//tests copy to an empty list
//=============================================================

void test9 (void)
{
        Set<int> s1;
        for (int i = 0; i < 6; i++)
                s1.insert(i);
        Set<int> s2(s1);
        string str = s2.toString();
        assert(str == "{0, 1, 2, 3, 4, 5}");
}

//=============================================================
//tests operator =
//=============================================================

void test10     (void)
{
        Set<char> s1;
        for (char a = 'a'; a<='z'; a++)
                s1.insert(a);
        Set<char> s2;
        s2 = s1;
        string str = s2.toString();
        assert(str == "{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x
, y, z}");
}


//=============================================================
//tests operator &
//=============================================================

void test11     (void)
{
```

```
        Set<char> s1;
        for (char a = 'a'; a<='e'; a++)
                s1.insert(a);
        Set<char> s2;
        for (char f = 'c'; f<= 'j'; f++)
                s2.insert(f);
        Set<char> s3;
        s3 = s1 & s2;
        string str = s3.toString();
        assert(str == "{c, d, e}");
}

//============================================================
//tests remove with item not in list
// TERMINAL
//============================================================

void test12     (void)
{
        Set<char> s1;
        for (char a = 'a'; a < 'r'; a++)
                s1.insert(a);
        s1.remove('z');


}


//============================================================
//tests operator -
//============================================================

void test13     (void)
{
        Set<int> s1;
        Set<int> s2;
        for (int i = 0; i <= 10; i++)
                s1.insert(i);
        for (int i = 0; i <=10; i+=2)
                s2.insert(i);
        Set<int> s3;
        s3 = s1-s2;
        string str = s3.toString();
        assert(str == "{1, 3, 5, 7, 9}");
}

//============================================================
//tests operator = with items in set previously
//============================================================

void test14     (void)
{
        Set<int> s1;
        for (int i = 0; i < 10; i++)
                s1.insert(i);
        Set<int> s2;
        for (int j = 0; j < 3; j++)
                s2.insert(j);
        s2 = s1;
        string str = s2.toString();
        assert(str=="{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}");
}
```

```
//=============================================================
//tests operator<=
//=============================================================
void test15     (void)
{
        Set<int> s1;
        for (int i = 0; i < 10; i++)
                s1.insert(i);
        Set<int> s2;
        for (int j = 0; j < 3; j++)
                s2.insert(j);
        if (s2 <= s1)
                return;
        else
                cout << "test 15 failed" << endl;
}

//=============================================================
//tests insert with existing item
//TERMINAL
//=============================================================

void test16     (void)
{
        Set<int> s1;
        for (int i = 0; i < 10; i++)
                s1.insert(i);
        for (int j = 5; j < 15; j++)
                s1.insert(j);
        string str = s1.toString();
        assert(str=="{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}");
}

//=============================================================
//tests <= with empty set
//=============================================================

void test17     (void)
{
        Set<int> s1;
        for (int i = 0; i < 10; i++)
                s1.insert(i);
        Set<int> s2;
        if (s1 <= s2)
                return;
        else
                cout << "test 17 failed" << endl;
}

//=============================================================
//tests intersection between two different sets
//=============================================================

void test18()
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 6; i<11; i++)
                s2.insert(i);
        Set<int> s3;
        s3 = s1&s2;
```

```
        string str = s3.toString();
        assert(str == "{}");
}

//=============================================================
//tests difference between two different sets
//=============================================================

void test19()
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 6; i<11; i++)
                s2.insert(i);
        Set<int> s3;
        s3 = s1-s2;
        string str = s3.toString();
        assert(str == "{0, 1, 2, 3, 4, 5}");
}

//=============================================================
//tests cardinality of empty set
//=============================================================

void test20     (void)
{
        Set<char> s1;
        assert(s1.cardinality() == 0);
}

//=============================================================
//tests difference operator with a bigger second set
//=============================================================

void test21     (void)
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 0; i<11; i++)
                s2.insert(i);
        Set<int> s3 = s1 - s2;
        string str = s3.toString();
        assert(str == "{}");
}

//=============================================================
//tests == with non equal sets
//=============================================================

void test22     (void)
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        Set<int> s2;
        for (int i = 6; i<11; i++)
                s2.insert(i);
        if (s1 == s2)
                cout << "Test 22 failed" << endl;
```

```
}

//============================================================
//tests string set
//============================================================

void test23 (void)
{
        Set<string> s1;
        s1.insert("Emma");
        s1.insert("Eliza");
        s1.insert("Evelyn");
        assert(s1.toString()=="{Emma, Eliza, Evelyn}");
}

//============================================================
// tests contains when item not in set
//============================================================

void test24 (void)
{
        Set<int> s1;
        for (int i = 0; i<6; i++)
                s1.insert(i);
        bool cont = s1.contains(6);
        assert(cont==0);
}
```