makefile

```
###################################################################
# Program:
#     Lesson 08, Binary Search Tree
#     Brother Helfrich, CS265
# Author:
#     Derek Calkins and David Lambertson
# Summary:
#     This program allows a user to take an array of data
#     sort it using a Binary Search Tree and then send it back
#     to the array.
# Time:
#     bst.h: 8 Hours sortBinary.h: 10 minutes.
#     David:60%   Derek:40%
###################################################################

##################################################################
# The main rule
##################################################################
a.out: lesson08.o
       g++ -g -o a.out lesson08.o
       tar -cf lesson08.tar *.h *.cpp makefile

##################################################################
# The individual components
#      lesson08.o     : the driver program
##################################################################
lesson08.o: bnode.h bst.h lesson08.cpp sortBinary.h
       g++ -g -c lesson08.cpp
```

Commented [HJ1]: Really? 10 minutes. So cool!

bst.h

```
/*******************************
 * Module:
 *     Lesson 08, BST
 *     Brother Helfrich, CS 235
 * Author:
 *     David Lambertson and Derek Calkins
 * Summary:
 *     This program contains the necessary
 *     methods for creating a Binary Search
 *     Tree and an iterator for it.
 ****************************/

#ifndef BST_H
#define BST_H

#include "bnode.h"

template<class T>
class BSTIterator;

/*************************************************
 * This is the class definition for our Binary
 * Search Tree.
 *************************************************/
template <class T>
class BST
{
  public:
  BST() : myRoot()  {}

  BST(BinaryNode<T> * root) : myRoot(root) {}

   ~BST() { clear(); }

   //adds the new data in the appropriate place
```

Commented [HJ2]: Not quite. You need to actually copy the entire tree, not just the root node.

```
   void insert(const T & data);

   //also a friend of the iterator so we can use it's variable
   void remove(BSTIterator<T> spot);

   //checks to see if we have anything in the tree
   bool empty() const { return (myRoot == NULL); }

   //clears all of the nodes in the tree
   void clear() { deleteBinaryTree(myRoot); }

   //return iterator to data if found
   BSTIterator<T> find(const T & data);

   //iterator to the lowest value node
   BSTIterator<T> begin();
   //iterator to NULL
   BSTIterator<T> end(){ return BSTIterator<T>(NULL); }
   //iterator to the highest value node
   BSTIterator<T> rbegin();
   //iterator to NULL
   BSTIterator<T> rend(){ return BSTIterator<T>(NULL); }

  private:
   BinaryNode<T> * myRoot;
   BinaryNode<T> * findForInsert(BinaryNode<T> * & p, const T & data);

};

/************************************************
 * This is the definition for our insert function
 ***********************************************/
template<class T>
void BST<T> :: insert(const T & data)
{
   BinaryNode<T> * pNew = findForInsert(myRoot, data);
   if (myRoot == NULL)
      myRoot = new BinaryNode<T>(data);
   else if (data > pNew->data)
      pNew->addRight(data);
   else
      pNew->addLeft(data);
}

/**************************************
 * This finds the parent Node of which we should add.
 *************************************/
template <class T>
BinaryNode<T> * BST<T> :: findForInsert(BinaryNode<T> * & p, const T & data)
{
   if (p == NULL)
      return p;
   if (p->data > data)
   {
      if (p->pLeft == NULL)
         return p;
      findForInsert(p->pLeft, data);
   }
   else
   {
      if (p->pRight == NULL)
         return p;
      findForInsert(p->pRight, data);
   }
}

/****************************************
 * Begin (returns a pointer to lowest)
 ***************************************/
template <class T>
BSTIterator<T> BST<T> :: begin()
{
   BinaryNode<T> * tmp = myRoot;
   if (!tmp)
      return tmp;
   while (tmp->pLeft)
   {
      tmp = tmp->pLeft;
   }
```

Commented [HJ3]: Could throw!

Commented [HJ4]: Nice touch.

Commented [HJ5]: Perfectly done.

```
   BSTIterator<T> it = tmp;
   return it;
}

/****************************************
 * Reverse Begin (returns a pointer to highest)
 ****************************************/
template <class T>
BSTIterator<T> BST<T> :: rbegin()
{
   BinaryNode<T> * tmp = myRoot;
   if (!tmp)
      return tmp;
   while (tmp->pRight)
   {
      tmp = tmp->pRight;
   }
   BSTIterator<T> it = tmp;
   return it;
}

/****************************************
 * FIND
 * Finds if we have the data and returns
 * a pointer to that node
 ****************************************/
template <class T>
BSTIterator<T> BST<T> :: find(const T & data)
{
   for(BSTIterator<T> it = begin(); it != end(); ++it)
   {
      if (*it == data )
      {
         return it;
         break;
      }
   }

   return end();
}

/***************************************
 * REMOVE
 * removes node at location given.
 ***************************************/
template <class T>
void BST<T> :: remove(BSTIterator<T> p)
{
   //if the node is not within the BST
   if (p == end())
      return;

   //if I don't have any children
   if(!p.spot->pLeft && !p.spot->pRight)
   {
      if(p.spot->amIRight())
         p.spot->pParent->pRight = NULL;
      else
         p.spot->pParent->pLeft = NULL;
      //after setting parent to NULL, delete node
      delete p.spot;
   }
   //if I have two children
   else if(p.spot->pLeft && p.spot->pRight)
   {
      //create iterator to point to successor
      BSTIterator<T> it = p;
      //move new iterator to point to successor
      ++it;
      //copy data from successor to node to overwrite
      p.spot->data = it.spot->data;
      //since we know that the successor will have one
      //child or no children, pass back successor node
      //to be able to delete
      remove(it);
   }
   //I have a child
   else
   {
```

Commented [HJ6]: Good!

Commented [HJ7]: Great comments.

```
      //do I have a left child?
      if(p.spot->pLeft)
      {
         p.spot->pLeft->pParent = p.spot->pParent;
         if(p.spot->amIRight())
            p.spot->pParent->pRight = p.spot->pLeft;
         else
            p.spot->pParent->pLeft = p.spot->pLeft;
      }
      //I must have a right child
      else
      {
         p.spot->pRight->pParent = p.spot->pParent;
         if(p.spot->amIRight())
            p.spot->pParent->pRight = p.spot->pRight;
         else
            p.spot->pParent->pLeft = p.spot->pRight;
      }
      //after changing the pointers delete node
      delete p.spot;
   }
}

/************************************************
 * This is the class definition for the Binary
 * Search Tree Iterator.
 ***********************************************/
template <class T>
class BSTIterator
{
  public:
   //default constructor
  BSTIterator() : spot() {}

   //non-default constructor
   BSTIterator(BinaryNode<T> * p)
   {
      if (p == NULL)
         spot = NULL;
      else
         spot = p;
   }

   //assignment operator
   BSTIterator<T> operator =(const BSTIterator<T> & rhs)
   {
      this->spot = rhs.spot;
      return *this;
   }

   //are they equal?
   bool operator ==(const BSTIterator<T> & rhs)
   { return (this->spot == rhs.spot); }

   //are they not equal?
   bool operator !=(const BSTIterator<T> & rhs)
   { return (this->spot != rhs.spot); }

   //dereference operator
   T & operator *() { return spot->data; }

   //overloaded operators
   BSTIterator<T> operator ++();
   BSTIterator<T> operator --();

  private:
   BinaryNode<T> * spot;

   //friend so we can access the iterator
   template <class U>
   friend void BST<U> :: remove(BSTIterator<U> p);
};

/*********************************************
 * Increment Operator
 * Goes to the successor
 ********************************************/
template <class T>
BSTIterator<T> BSTIterator<T> :: operator ++()
```

```
{
   // has right child
   if (spot->pRight != NULL)
   {
      spot = spot->pRight;
      while (spot->pLeft)
         spot = spot->pLeft;
   }

   // has no right child
   else
   {
      while (spot->amIRight())
         spot = spot->pParent;
      if (spot->pParent != NULL)
         spot = spot->pParent;
      else
         spot = NULL;
   }
   return *this;
}

/**********************************************
 * Decrement Operator
 * Goes to the predecessor
 **********************************************/
template <class T>
BSTIterator<T> BSTIterator<T> :: operator --()
{
   // has left child
   if (spot->pLeft != NULL)
   {
      spot = spot->pLeft;
      while (spot->pRight)
         spot = spot->pRight;
   }

   // has no left child
   else
   {
      while (spot->amILeft())
         spot = spot->pParent;
      if (spot->pParent != NULL)
         spot = spot->pParent;
      else
         spot = NULL;
   }
   return *this;
}


#endif // BST_H
```

sortBinary.h

```
/***********************************************************************
 * Module:
 *    Lesson 08, Sort Binary
 *    Brother Helfrich, CS 235
 * Author:
 *    David Lambertson
 * Summary:
 *    This program will implement the Binary Tree Sort
 ************************************************************************/

#ifndef SORT_BINARY_H
#define SORT_BINARY_H

#include "bst.h"

/*****************************************************
 * SORT BINARY
 * Perform the binary tree sort
 ****************************************************/
template <class T>
void sortBinary(T array[], int size)
{
   BST<T> tree;
```

```
   for (int i = 0; i < size; i++) //creates the tree
      tree.insert(array[i]);

   int i = 0;//save it back into the array
   for (BSTIterator<T> it = tree.begin(); it != tree.end(); ++it, i++)
      array[i] = *it;
}


#endif // SORT_BINARY_H
```

Commented [HJ8]: Flawless.

bnode.h

Commented [HJ9]: Unchanged, I presume

```
/***********************************************
 * Program:
 *     Lesson 07, Binary Tree
 *     Brother Helfrich, CS265
 * Author:
 *     David Lambertson
 * Summary:
 *     This file holds the definition of the binary node
 *     used to create a binary tree.
 * Time:
 *    this part of the program took me around 5 hours.
 ***********************************************/
#ifndef BNODE_H
#define BNODE_H


#include <iostream>
#include <cassert>

/****************************************************
 * This is the class that holds our Binary Node Definition.
 * It allows us to create Binary Nodes which are used for the tree.
 ***************************************************/
template <class T>
class BinaryNode
{
  public:
   T data;
   BinaryNode<T> * pLeft;
   BinaryNode<T> * pRight;
   BinaryNode<T> * pParent;

   //Default Constructor
  BinaryNode() :pLeft(NULL), pRight(NULL), pParent(NULL) {}

   //Non-Default Constructor
  BinaryNode(T data) : data(data), pLeft(NULL), pRight(NULL),pParent(NULL) {}

   /********************************
    * These are our two add Left functions.
    * One takes data and the other takes a Node
    *********************************/
   void addLeft(const T & data);
   void addLeft(BinaryNode<T> * pNew);

   /********************************
    * Similar to our add Lefts, just for right.
    ********************************/
   void addRight(const T & data);
   void addRight(BinaryNode<T> * pNew);

   /****************************************
    *This checks to see if I am the Right child.
    ************************************/
   bool amIRight() const
   { return ((this->pParent) && (this->pParent->pRight == this)); }

   /************************************
    * This checks if I am the Left child.
    ************************************/
   bool amILeft() const
   { return ((this->pParent) && (this->pParent->pLeft == this)); }

};
```

```
/*********************************
 * overloaded insertion operator allows us to display.
 *******************************/
template <class T>
std::ostream& operator <<(std::ostream& out,  const BinaryNode<T> * tmp)
{
   if (tmp == NULL)
      return out;
   return out << tmp->pLeft << tmp->data << ' ' << tmp->pRight;
}

/*************************************
 * Function definition of our first addLeft
 ************************************/
template <class T>
void BinaryNode<T> :: addLeft(const T & data)
{
   if (this->pLeft == NULL)
   {
      BinaryNode<T> * left = new  BinaryNode<T>;
      left->data = data;
      this->pLeft = left;
      left->pParent = this;
   }
   else
      this->pLeft->addLeft(data);
}

/***************************************
 * second definition of addLeft
 **************************************/
template <class T>
void BinaryNode<T> :: addLeft(BinaryNode<T> * left)
{
   if (this->pLeft != NULL)
      this->pLeft->addLeft(left);
   else
   {
      this->pLeft = left;
      left->pParent = this;
   }
}

/**********************************
 * first definition of addRight
 *******************************/
template <class T>
void BinaryNode<T> :: addRight(const T & data)
{

   if (pRight == NULL)
   {
      BinaryNode<T> * right = new BinaryNode<T>;
      right->data = data;
      this->pRight = right;
      right->pParent = this;
   }
   else
      this->pRight->addRight(data);
}

/***************************************
 * Second definition of addRight
 ***********************************/
template <class T>
void BinaryNode<T> :: addRight(BinaryNode<T> * right)
{
   if (this->pRight != NULL)
      this->pRight->addRight(right);
   else
   {
      this->pRight = right;
      right->pParent = this;
   }
}

/*******************************************
 * function definition of deleteBinaryTree allowing us
 * to delete a binary tree we have created.
```

```
 *****************************************/
template <class T>
void deleteBinaryTree(BinaryNode<T> * root)
{
   if (root == NULL)
      return;
   deleteBinaryTree(root->pLeft);
   deleteBinaryTree(root->pRight);
   delete root;
}


#endif //BNODE_H
```

lesson08.cpp

```
/***********************************************************************
* Program:
*    Lesson 08, Binary Search Trees and the Binary Sort
*    Brother Helfrich, CS 235
* Author:
*    Br. Helfrich
* Summary:
*    This is a driver program to exercise the BST class.  When you
*    submit your program, this should not be changed in any way.  That being
*    said, you may need to modify this once or twice to get it to work.
************************************************************************/

#include <iostream>      // for CIN and COUT
#include <string>        // for STRING
#include "bst.h"         // for BST class which should be in bst.h
#include "sortBinary.h"  // for sortBinary()
using namespace std;


// prototypes for our four test functions
void testSimple();
void testAdd();
void testIterate();
void testDelete();
void testSort();

// To get your program to compile, you might need to comment out a few
// of these. The idea is to help you avoid too many compile errors at once.
// I suggest first commenting out all of these tests, then try to use only
// TEST1.  Then, when TEST1 works, try TEST2 and so on.
#define TEST1   // for testSimple()
#define TEST2   // for testAdd()
#define TEST3   // for testIterate()
#define TEST4   // for testDelete()

/**********************************************************************
 * MAIN
 * This is just a simple menu to launch a collection of tests
 ***********************************************************************/
int main()
{
   // menu
   cout << "Select the test you want to run:\n";
   cout << "\t1. Just create and destroy a BST\n";
   cout << "\t2. The above plus add a few nodes\n";
   cout << "\t3. The above plus display the contents of a BST\n";
   cout << "\t4. The above plus find and delete nodes from a BST\n";
   cout << "\ta. To test the binarySort() function\n";

   // select
   char choice;
   cout << "> ";
   cin  >> choice;
   switch (choice)
   {
      case 'a':
         testSort();
         break;
      case '1':
         testSimple();
         cout << "Test 1 complete\n";
         break;
```

```
      case '2':
         testAdd();
         cout << "Test 2 complete\n";
         break;
      case '3':
         testIterate();
         cout << "Test 3 complete\n";
         break;
      case '4':
         testDelete();
         cout << "Test 4 complete\n";
         break;
      default:
         cout << "Unrecognized command, exiting...\n";
   }

   return 0;
}

/********************************************
 * TEST SIMPLE
 * Very simple test for a BST: create and destroy
 ********************************************/
void testSimple()
{
#ifdef TEST1
   // Test1: a bool BST
   cout << "Create a bool Binary Search Tree using the default constructor\n";
   BST <bool> tree;

   // Test2: double BST
   cout << "Create a double Binary Search Tree\n";
   BST <double> * pTree = new BST <double>;
   delete pTree;
#endif //TEST1
}

/********************************************
 * TEST ADD
 * Add a few nodes together to create a tree, then
 * destroy it when done
 *******************************************/
void testAdd()
{
#ifdef TEST2
   // create
   cout << "Create an integer Binary Search Tree\n";
   BST <int> tree;

   tree.insert(8);     //               8
   tree.insert(4);     //          +----+----+
   tree.insert(12);    //          4         12
   tree.insert(2);     //       +--+--+   +--+--+
   tree.insert(6);     //       2     6   9     13
   tree.insert(9);     //     +-+   +-+   +-+
   tree.insert(13);    //     0     5       11
   tree.insert(0);
   tree.insert(5);
   tree.insert(11);

   cout << "\tTree deleted\n";
#endif // TEST2
}

/********************************************
 * TEST ITERATE
 * We will build a binary tree and display the
 * results on the screen
 ********************************************/
void testIterate()
{
#ifdef TEST3
   cout.setf(ios::fixed | ios::showpoint);
   cout.precision(1);

   //
   // An empty tree
   //
   try
```

```
{
   cout << "Create an empty bool BST\n";
   BST <bool> tree;
   BSTIterator <bool> it;
   cout << "\tEmpty tree\n";

   // display the contents
   cout << "\tContents: ";
   for (it = tree.begin(); it != tree.end(); ++it)
      cout << *it << " ";
   cout << endl;

   // tree deleted
   cout << "\tTree deleted\n";
}
catch (const char * s)
{
   cout << "Thrown exception: " << s << endl;
}

//
// A tree with three nodes
//
try
{
   cout << "Create an double BST\n";
   BST <double> tree;
   BSTIterator <double> it;

   // fill the tree
   cout << "\tFill the BST with: 2.2  1.1  3.3  \n";
   tree.insert(2.2);     //          2.2
   tree.insert(1.1);     //     +-----+-----+
   tree.insert(3.3);     //    1.1         3.3

   // display the contents forward
   cout << "\tContents forward:  ";
   for (it = tree.begin(); it != tree.end(); ++it)
      cout << *it << "  ";
   cout << endl;

   // display the contents backwards
   cout << "\tContents backward: ";
   for (it = tree.rbegin(); it != tree.rend(); --it)
      cout << *it << "  ";
   cout << endl;

   // tree deleted
   cout << "\tTree deleted\n";
   }
   catch (const char * s)
   {
      cout << "Thrown exception: " << s << endl;
   }


//
// a non-trivial tree
//
try
{
   cout << "Create a string BST\n";
   BST <string> tree;
   BSTIterator <string> it;

   // fill the tree
   cout << "\tFill the BST with: f  c  i  b  e  g  j  a  d  h  \n";
   tree.insert(string("f"));     //                f
   tree.insert(string("c"));     //          +----+----+
   tree.insert(string("i"));     //          c         i
   tree.insert(string("b"));     //       +--+--+   +--+--+
   tree.insert(string("e"));     //       b     e   g     j
   tree.insert(string("g"));     //     +-+    +-+   +-+
   tree.insert(string("j"));     //     a      d       h
   tree.insert(string("a"));
   tree.insert(string("d"));
   tree.insert(string("h"));

   // display the contents forward
```

```
      cout << "\tContents forward:  ";
      for (it = tree.begin(); it != tree.end(); ++it)
         cout << *it << "  ";
      cout << endl;

      // display the contents backwards
      cout << "\tContents backward: ";
      for (it = tree.rbegin(); it != tree.rend(); --it)
         cout << *it << "  ";
      cout << endl;

      // tree deleted
      cout << "\tTree deleted\n";
   }
   catch (const char * s)
   {
      cout << "Thrown exception: " << s << endl;
   }
#endif // TEST3
}

/********************************************
 * TEST DELETE
 * Insert a few items into a tree, then delete a few items
 *******************************************/
void testDelete()
{
#ifdef TEST4
   try
   {
      cout << "Create a char BST\n";
      BST <char> tree;

      // Fill the tree
      cout << "\tFill the tree with: G F A E C B D J H I O M K L N P\n";
      tree.insert('G');
      tree.insert('F'); //                       G
      tree.insert('A'); //          +------------+------------+
      tree.insert('E'); //          F                         J
      tree.insert('C'); //   +------+                  +------+------+
      tree.insert('B'); //   A                         H             O
      tree.insert('D'); //   +---+                     +---+     +---+---+
      tree.insert('J'); //       E                         I     M       P
      tree.insert('H'); //    +--+                            +--+--+
      tree.insert('I'); //    C                               K     N
      tree.insert('O'); //  +-+-+                             +-+
      tree.insert('M'); //  B   D                               L
      tree.insert('K');
      tree.insert('L');
      tree.insert('N');
      tree.insert('P');

      // display the tree
      cout << "\tContents without removal: ";
      BSTIterator <char> it;
      for (it = tree.begin(); it != tree.end(); ++it)
         cout << *it << ' ';
      cout << endl;

      //
      // Remove node D: leaf
      //

      cout << "Remove a leaf node\n";

      // find node 'D' and remove it
      it = tree.find('D');
      if (it == tree.end())
         cout << "\tNode not found!\n";
      else
         cout << "\tNode '" << *it << "' found\n";
      if (it != tree.end())
         tree.remove(it);

      // display the tree again
      cout << "\tContents after 'D' was removed: ";
      for (it = tree.begin(); it != tree.end(); ++it)
         cout << *it << ' ';
      cout << endl;
```

```
      // look for node 'D' again
      it = tree.find('D');
      if (it == tree.end())
         cout << "\tNode not found!\n";
      else
         cout << "\tNode '" << *it << "' found\n";

      //
      // Remove node E: one child (left)
      //

      cout << "Remove a one-child node\n";

      // look for node 'E' and remove it
      it = tree.find('E');
      if (it == tree.end())
         cout << "\tNode not found!\n";
      else
         cout << "\tNode '" << *it << "' found\n";
      if (it != tree.end())
         tree.remove(it);

      // display the tree again
      cout << "\tContents after 'E' was removed: ";
      for (it = tree.begin(); it != tree.end(); ++it)
         cout << *it << ' ';
      cout << endl;

      //
      // Remove node J: two children where 'K' is inorder successor
      //

      cout << "Remove a two-child node\n";

      // look for node 'J' and remove it
      it = tree.find('J');
      if (it == tree.end())
         cout << "\tNode not found!\n";
      else
         cout << "\tNode '" << *it << "' found\n";
      if (it != tree.end())
         tree.remove(it);

      // display the tree again
      cout << "\tContents after 'J' was removed: ";
      for (it = tree.begin(); it != tree.end(); ++it)
         cout << *it << ' ';
      cout << endl;

      //
      // Remove node G: the root
      //

      cout << "Remove the root\n";

      // look for node 'G' and remove it
      it = tree.find('G');
      if (it == tree.end())
         cout << "\tNode not found!\n";
      else
         cout << "\tNode '" << *it << "' found\n";
      if (it != tree.end())
         tree.remove(it);

      // display the tree again
      cout << "\tContents after 'G' was removed: ";
      for (it = tree.begin(); it != tree.end(); ++it)
         cout << *it << ' ';
      cout << endl;


      cout << "\tTree deleted\n";
   }
   catch (const char * s)
   {
      cout << "Thrown exception: " << s << endl;
   }
#endif // TEST4
```

```
}

/***************************************
 * TEST SORT
 * Sort three things using the binary sort
 ***************************************/
void testSort()
{
   cout.setf(ios::fixed | ios::showpoint);
   cout.precision(1);

   //
   // Test a small set of strings
   //

   cout << "Four string objects\n";

   // before
   string array1[4] =
   {
      string("Beta"), string("Alpha"), string("Epsilon"), string("Delta")
   };
   int size1 = sizeof(array1) / sizeof(array1[0]);
   cout << "\tBefore: " << array1[0];
   for (int i = 1; i < size1; i++)
      cout << ", " << array1[i];
   cout << endl;

   // sort
   sortBinary(array1, size1);

   // after
   cout << "\tAfter:  " << array1[0];
   for (int i = 1; i < size1; i++)
      cout << ", " << array1[i];
   cout << endl;

   //
   // Test a medium set of floats
   //

   cout << "Twenty one-decimal numbers\n";

   // before
   float array2[20] =
   {
      5.1, 2.4, 8.2, 2.7, 4.7, 1.8, 9.9, 3.4, 5.0, 1.0,
      4.4, 3.4, 8.3, 2.9, 1.7, 7.9, 9.5, 9.3, 3.6, 2.9
   };
   int size2 = sizeof(array2) / sizeof(array2[0]);
   cout << "\tBefore:\t" << array2[0];
   for (int i = 1; i < size2; i++)
      cout << (i % 10 == 0 ? ",\n\t\t" : ", ")
           << array2[i];
   cout << endl;

   // sort
   sortBinary(array2, size2);

   // after
   cout << "\tAfter:\t" << array2[0];
   for (int i = 1; i < size2; i++)
      cout << (i % 10 == 0 ? ",\n\t\t" : ", ")
           << array2[i];
   cout << endl;

   //
   // Test a large set of integers
   //

   cout << "One hundred three-digit numbers\n";

   // before
   int array3[100] =
   {
      889, 192, 528, 675, 154, 746, 562, 482, 448, 842, 929, 330, 615, 225,
      785, 577, 606, 426, 311, 867, 773, 775, 190, 414, 155, 771, 499, 337,
      298, 242, 656, 188, 334, 184, 815, 388, 831, 429, 823, 331, 323, 752,
      613, 838, 877, 398, 415, 535, 776, 679, 455, 602, 454, 545, 916, 561,
```

```
      369, 467, 851, 567, 609, 507, 707, 844, 643, 522, 284, 526, 903, 107,
      809, 227, 759, 474, 965, 689, 825, 433, 224, 601, 112, 631, 255, 518,
      177, 224, 131, 446, 591, 882, 913, 201, 441, 673, 997, 137, 195, 281,
      563, 151,
   };
   int size3 = sizeof(array3) / sizeof(array3[0]);
   cout << "\tBefore:\t" << array3[0];
   for (int i = 1; i < size3; i++)
      cout << (i % 10 == 0 ? ",\n\t\t" : ", ")
           << array3[i];
   cout << endl;

   // sort
   sortBinary(array3, size3);

   // after
   cout << "\tAfter:\t" << array3[0];
   for (int i = 1; i < size3; i++)
      cout << (i % 10 == 0 ? ",\n\t\t" : ", ")
           << array3[i];
   cout << endl;
}
```

Test Bed Results

```
cs235d.out:

Started program
   > Select the test you want to run:
   >     1. Just create and destroy a BST
   >     2. The above plus add a few nodes
   >     3. The above plus display the contents of a BST
   >     4. The above plus find and delete nodes from a BST
   >     a. To test the binarySort() function
   > > 1
   > Create a bool Binary Search Tree using the default constructor
   > Create a double Binary Search Tree
   > Test 1 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >     1. Just create and destroy a BST
   >     2. The above plus add a few nodes
   >     3. The above plus display the contents of a BST
   >     4. The above plus find and delete nodes from a BST
   >     a. To test the binarySort() function
   > > 2
   > Create an integer Binary Search Tree
   >     Tree deleted
   > Test 2 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >     1. Just create and destroy a BST
   >     2. The above plus add a few nodes
   >     3. The above plus display the contents of a BST
   >     4. The above plus find and delete nodes from a BST
   >     a. To test the binarySort() function
   > > 3
   > Create an empty bool BST
   >     Empty tree
   >     Contents:
   >     Tree deleted
   > Create an double BST
   >     Fill the BST with: 2.2  1.1  3.3
   >     Contents forward:  1.1  2.2  3.3
   >     Contents backward: 3.3  2.2  1.1
   >     Tree deleted
   > Create a string BST
   >     Fill the BST with: f  c  i  b  e  g  j  a  d  h
   >     Contents forward:  a  b  c  d  e  f  g  h  i  j
   >     Contents backward: j  i  h  g  f  e  d  c  b  a
   >     Tree deleted
   > Test 3 complete
Program terminated successfully
```

```
Started program
   > Select the test you want to run:
   >    1. Just create and destroy a BST
   >    2. The above plus add a few nodes
   >    3. The above plus display the contents of a BST
   >    4. The above plus find and delete nodes from a BST
   >    a. To test the binarySort() function
   > > 4
   > Create a char BST
   >    Fill the tree with: G F A E C B D J H I O M K L N P
   >    Contents without removal: A B C D E F G H I J K L M N O P
   > Remove a leaf node
   >    Node 'D' found
   >    Contents after 'D' was removed: A B C E F G H I J K L M N O P
   >    Node not found!
   > Remove a one-child node
   >    Node 'E' found
   >    Contents after 'E' was removed: A B C F G H I J K L M N O P
   > Remove a two-child node
   >    Node 'J' found
   >    Contents after 'J' was removed: A B C F G H I K L M N O P
   > Remove the root
   >    Node 'G' found
   >    Contents after 'G' was removed: A B C F H I K L M N O P
   >    Tree deleted
   > Test 4 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a BST
   >    2. The above plus add a few nodes
   >    3. The above plus display the contents of a BST
   >    4. The above plus find and delete nodes from a BST
   >    a. To test the binarySort() function
   > > a
   > Four string objects
   >    Before: Beta, Alpha, Epsilon, Delta
   >    After:  Alpha, Beta, Delta, Epsilon
   > Twenty one-decimal numbers
   >    Before:   5.1, 2.4, 8.2, 2.7, 4.7, 1.8, 9.9, 3.4, 5.0, 1.0,
   >       4.4, 3.4, 8.3, 2.9, 1.7, 7.9, 9.5, 9.3, 3.6, 2.9
   >    After:   1.0, 1.7, 1.8, 2.4, 2.7, 2.9, 2.9, 3.4, 3.4, 3.6,
   >       4.4, 4.7, 5.0, 5.1, 7.9, 8.2, 8.3, 9.3, 9.5, 9.9
   > One hundred three-digit numbers
   >    Before:   889, 192, 528, 675, 154, 746, 562, 482, 448, 842,
   >       929, 330, 615, 225, 785, 577, 606, 426, 311, 867,
   >       773, 775, 190, 414, 155, 771, 499, 337, 298, 242,
   >       656, 188, 334, 184, 815, 388, 831, 429, 823, 331,
   >       323, 752, 613, 838, 877, 398, 415, 535, 776, 679,
   >       455, 602, 454, 545, 916, 561, 369, 467, 851, 567,
   >       609, 507, 707, 844, 643, 522, 284, 526, 903, 107,
   >       809, 227, 759, 474, 965, 689, 825, 433, 224, 601,
   >       112, 631, 255, 518, 177, 224, 131, 446, 591, 882,
   >       913, 201, 441, 673, 997, 137, 195, 281, 563, 151
   >    After:   107, 112, 131, 137, 151, 154, 155, 177, 184, 188,
   >       190, 192, 195, 201, 224, 224, 225, 227, 242, 255,
   >       281, 284, 298, 311, 323, 330, 331, 334, 337, 369,
   >       388, 398, 414, 415, 426, 429, 433, 441, 446, 448,
   >       454, 455, 467, 474, 482, 499, 507, 518, 522, 526,
   >       528, 535, 545, 561, 562, 563, 567, 577, 591, 601,
   >       602, 606, 609, 613, 615, 631, 643, 656, 673, 675,
   >       679, 689, 707, 746, 752, 759, 771, 773, 775, 776,
   >       785, 809, 815, 823, 825, 831, 838, 842, 844, 851,
   >       867, 877, 882, 889, 903, 913, 916, 929, 965, 997
Program terminated successfully

No Errors
```

Grading Criteria

Criteria	Exceptional 100%	Good 90%	Acceptable 70%	Developing 50%	Missing 0%	Weight	Score
BST interface	The interfaces are perfectly specified with respect to const, pass-by-reference, etc.	lesson08.cpp compiles without modification	All the methods in BST match the problem definition	BST has many of the same interfaces as the problem definition	The public methods and variables in the BST class do not resemble the problem definition	10	
BST implementation	Passes all four BST testBed tests	Passes three testBed tests	Passes two testBed tests	Passes one testBed test	Program fails to compile or does not pass any testBed tests	20	
BSTIterator	Solution works, is elegant, and efficient	Both forward and reverse iterators work	Works in some limited cases	Elements of the solution are present	No attempt was made to iterate through the BST	30	
Binary Search	Code is elegant and efficient	Passes the Binary Search testBed tests	The code essentially works but with minor defects	Elements of the solution are present	The Binary Search problem was not attempted	10	
Code Quality	There is no obvious room for improvement	All the principles of encapsulation and modularization are honored	One function is written in a "backwards" way or could be improved	Two or more functions appears "thrown together."	The code appears to be written without any obvious forethought	20	
Style	Great variable names, no errors, great comments	No obvious style errors	A few minor style errors: non-standard spacing, poor variable names, missing comments, etc.	Overly generic variable names, misleading comments, or other gross style errors	No knowledge of the BYU-I code style guidelines were demonstrated	10	
Total							100

Commented [HJ10]: Fantastic work, as usual.