makefile

```
###############################################################
# Program:
#     Lesson 07, Binary Tree
#     Brother Helfrich, CS265
# Author:
#     David Lambertson and Derek Calkins
# Summary:
#     this program allows users to make binary nodes which
#     they can link together to create a binary tree.
# Time:
#     David:60%  Derek: 40%
###############################################################

##############################################################
# The main rule
##############################################################
a.out: lesson07.o huffman.o
       g++ -g -o a.out lesson07.o huffman.o
       tar -cf lesson07.tar *.h *.cpp makefile

##############################################################
# The individual components
#      lesson07.o     : the driver program
#      huffman.o      : the logic for the huffman code program
##############################################################
lesson07.o: bnode.h huffman.h lesson07.cpp
       g++ -g -c lesson07.cpp

huffman.o: bnode.h huffman.h vector.h huffman.cpp
       g++ -g -c huffman.cpp
```

vector.h

Commented [HJ1]: Unchanged I suppose

```
/******************
 * Vector.h
 * This file contains the classes for a Vector
 * and its iterator and the different functions
 * needed for its implementation.
 ******************/

#ifndef VECTOR_H
#define VECTOR_H

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

//because we use this in the Vector class
template<class T>
class VectorIterator;

/********
 * the implementation of the Vector Class
 ********/
template <class T>
class Vector
{
  public:
   // default constructor
  Vector() : myCapacity(0), numItems(0), data(0x00000000) {}

   // non-default contsructor
   Vector(int capacity) throw (const char *);

   // copy constructor
   Vector(const Vector & rhs) throw (const char *) { *this = rhs; }

   // destructor
   ~Vector() { if (myCapacity) delete [] data; }

   // is the container empty
   bool empty() const { return numItems == 0; }

   //return how many items are in the Vector
   int size() const { return numItems; }

   //how big is the Vector
   int capacity() const { return myCapacity; }

   //clears the Vector
   void clear() { numItems = 0; }

   // the push back function, allowing the user to add values to the Vector.
   // Also doubles the size and reallocates when the Vector gets full.
   void push_back(T newValue);

   // the square bracket operator overload
   const T operator [](int item) const;

   //overloaded assignment operator
   Vector<T> & operator = (const Vector<T>& rhs);

   // sets an iterator to the first item of data
   VectorIterator<T> begin() { return VectorIterator<T>(data); }

   // sets an iterator to the last item of data
   VectorIterator<T> end() {return VectorIterator<T>(data +numItems); }

   //private:
   int myCapacity; //how big the Vector is
   int numItems;   //the number of Items in the Vector
   T * data;       // the storage unit of the Vector

};

//implementation of the VectorIterator class
template <class T>
class VectorIterator
{
  public:
   //default constructor
  VectorIterator() : p(0x00000000) {}

   //non-default constructor
  VectorIterator(T * p) : p(p) {}

   //copy contructor
   VectorIterator(const VectorIterator & rhs) { *this = rhs; }

   //assignent operator overloaded
   VectorIterator & operator = (const VectorIterator & rhs)
   {
      this->p = rhs.p;
      return *this;
   }

   //not equal operator
   bool operator != (const VectorIterator & rhs) const
   {
      return rhs.p != this->p;
   }
   //de reference operator
   T & operator * ()
   {
      return *p;
   }

   //increment.
   VectorIterator <T> & operator ++ ()
   {
      p++;
      return *this;
   }
   VectorIterator <T> operator++ (int postfix)
   {
      VectorIterator tmp(*this);
      p++;
      return tmp;
   }

    VectorIterator <T> & operator -- ()
   {
      p--;
      return *this;
   }

   VectorIterator <T> operator-- (int postfix)
   {
      VectorIterator tmp(*this);
      p--;
      return tmp;
   }

   //private:
   T * p;

};



//implementation of the non default constructor
template <class T>
Vector <T> :: Vector (int capacity) throw (const char *)
{
   assert(capacity >= 0);

   if (capacity == 0)
   {
      this->myCapacity = this->numItems = 0;
      this->data = 0x00000000;
      return;
   }

   try
   {
      data = new T[capacity];
   }
   catch (std::bad_alloc)
   {
      throw "ERROR: Unable to allocate a new buffer for Vector";
   }


   this->myCapacity = capacity;
   this->numItems = 0;
}

//implementation of the push back function
template<class T>
void Vector <T> :: push_back(T newValue)
{
   T * newData;
   if (myCapacity == 0)
   {
      myCapacity += 1;
      data = new T[myCapacity];
   }

   if (myCapacity == numItems)
   {
      myCapacity *= 2;

      try
      {
         newData = new T[myCapacity];
      }

      catch(...)
      {
         cout << "Unable to allocate a buffer for Vector";
         myCapacity /= 2;
      }
      int i;
      for (i = 0; i < numItems; i++)
      {
         newData[i] = data[i];
      }
      //newData[i] = '\0';

      delete [] data;
      data = newData;

   }

   data[numItems] = newValue;
   numItems++;
}

//implementation of the square bracket operator
template <class T>
const T Vector<T> :: operator [] (int item) const
{
   return this->data[item];
}

//implementation of the assignment overator
template<class T>
Vector<T> & Vector<T> :: operator = (const Vector<T>& rhs)
{
   assert(rhs.myCapacity >= 0);

   if (rhs.myCapacity == 0)
   {
      this->myCapacity = this->numItems = 0;
      this->data = 0x00000000;
      return *this;
   }

   try
   {
      this->data = new T[rhs.myCapacity];
   }
   catch(std:: bad_alloc)
   {
      throw "ERROR: Unable to allocate buffer";
   }

   //  assert(rhs.numItems >= 0 && rhs.numItems <= rhs.myCapacity);
   this->myCapacity = rhs.myCapacity;
   this->numItems = rhs.numItems;
   for (int i = 0; i < numItems; i++)
   {
      this->data[i] = rhs.data[i];
   }
}

template <class T>
inline ostream & operator << (ostream & out, const Vector <T> rhs)
{
   out << rhs;
   return out;
}

template <class T>
inline istream operator >> (istream & in, Vector <T> rhs)
{
   in >> rhs;
//   return in;
}

#endif //VECTOR_H
```

huffman.h

```
/************************************************************************
 * Module:
 *    Lesson 07, Huffman
 *    Brother Helfrich, CS 235
 * Author:
```

Commented [HJ2]: Unchanged.

```
 *    Br. Helfrich
 * Summary:
 *    This program will implement the huffman() function
 ***********************************************************************/

#ifndef HUFFMAN_H
#define HUFFMAN_H

#include "pair.h"
#include "bnode.h"

void huffman();

#endif // HUFFMAN_h
```

pair.h

Commented [HJ3]: Unchanged.

```
/***********************************************************************
 * Module:
 *    Lesson 07, Pair
 *    Brother Helfrich, CS 235
 * Author:
 *    Br. Helfrich
 * Summary:
 *    This program will implement a pair: two values
 ***********************************************************************/

#ifndef PAIR_H
#define PAIR_H

#include <iostream>  // for ISTREAM and OSTREAM

/**********************************************
 * PAIR
 * This class couples together a pair of values, which may be of
 * different types (T1 and T2). The individual values can be
 * accessed through its public members first and second.
 *
 * Additionally, when comparing two pairs, only T1 is compared. This
 * is a key in a name-value pair.
 ***********************************************/
template <class T1, class T2>
class Pair
{
public:
   // constructors
   Pair() {}
   Pair(const T1 & first, const T2 & second) : first(first), second(second) {}
   Pair(const Pair <T1, T2> & rhs) : first(rhs.first), second(rhs.second) {}

   // copy the values
   Pair <T1, T2> & operator = (const Pair <T1, T2> & rhs)
   {
      first  = rhs.first;
      second = rhs.second;
      return *this;
   }

   // constant fetchers
   const T1 & getFirst()  const { return first;  }
   const T2 & getSecond() const { return second; }

   // compare Pairs.  Only first will be compared!
   bool operator >  (const Pair & rhs) const { return first >  rhs.first; }
   bool operator >= (const Pair & rhs) const { return first >= rhs.first; }
   bool operator <  (const Pair & rhs) const { return first <  rhs.first; }
   bool operator <= (const Pair & rhs) const { return first <= rhs.first; }
   bool operator == (const Pair & rhs) const { return first == rhs.first; }
   bool operator != (const Pair & rhs) const { return first != rhs.first; }

   // these are public.  We cannot validate!
   T1 first;
   T2 second;
};

/*****************************************************
 * PAIR INSERTION
 * Display a pair for debug purposes
 *****************************************************/
template <class T1, class T2>
inline std::ostream & operator << (std::ostream & out, const Pair <T1, T2> & rhs)
{
   out << '(' << rhs.first << ", " << rhs.second << ')';
   return out;
}

/*****************************************************
 * PAIR EXTRACTION
 * input a pair
 *****************************************************/
template <class T1, class T2>
inline std::istream & operator >> (std::istream & in, Pair <T1, T2> & rhs)
{
   in >> rhs.first >> rhs.second;
   return in;
}

#endif // PAIR_H
```

bnode.h

```
/***********************************************
 * 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;
```

Commented [HJ4]: Good.

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

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

Commented [HJ5]: Do not make a copy of T.

```
   /*********************************
    * 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);
```

Commented [HJ6]: Good!

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

};

/*********************************
 * 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;
}
```

Commented [HJ7]: perfect

```
/***************************************
 * 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);
}
```

Commented [HJ8]: nice touch. Different than the design, but a nice adaptation.

```
/*****************************************
 * second definition of addLeft
 ****************************************/
template <class T>
void BinaryNode<T> :: addLeft(BinaryNode<T> * left)
{
   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)
{
   this->pRight = right;
   right->pParent = this;
}
```

Commented [HJ9]: What is pRight != NULL?

```
/*********************************************
 * function definition of deleteBinaryTree allowing us
 * to delete a binary tree we have created.
 ******************************************/
template <class T>
void deleteBinaryTree(BinaryNode<T> * root)
{
   if (root->pLeft == NULL && root->pRight == NULL)
   {
      delete root;
   }
   else if (root->pLeft != NULL)
   {
      deleteBinaryTree(root->pLeft);
   }
   else
      deleteBinaryTree(root->pRight);
}
```

Commented [HJ10]: No! You must always delete root, not just if root is a leaf!

```
#endif //BNODE_H
```

huffman.cpp

```
/***********************************************************************
 * Module:
 *    Lesson 07, Huffman
 *    Brother Helfrich, CS 235
 * Author:
 *    Derek Calkins
 * Summary:
 *    This program will implement the huffman() function
 ************************************************************************/

#include "huffman.h"       // for HUFFMAN() prototype
#include <fstream>
#include <iostream>
#include "vector.h"
using namespace std;

/*******************************************
 * HUFFMAN
 * Driver program to exercise the huffman generation code
 *******************************************/

/**************************
 * Gets the filename
 **************************/
string getFilename(string fileName)
{
   cout << "Enter the filename containing the value frequencies: ";
   cin  >> fileName;
   return fileName;
}
```

Commented [HJ11]: This is not the best. Since fileName comes in as a by-value parameter, no data is leaving through it. Therefore, you should either pass it by-reference (and return by-reference) or this should be a local variable.

Commented [HJ12]: I presume this is a debug routine so you can see what is going on each iteration?

Commented [HJ13]: Should be a const.

```
/***************************************
 * To display the results of the code
 ***************************************/
void display(BinaryNode <Pair<float, string> > * pHead, int num)
{
   //to hold the codes to display
   Vector <string> data;

   int numItems = num;
   //loop through every node in the tree and add a 0 if we go the the left
   //or a 1 if we go to the right
   for (int i = 0; i < numItems; i++)
   {
      string chars; //to combine both the character and huffman code
      if(pHead->pLeft != NULL)
      {
         pHead = pHead->pLeft;
         chars = (pHead->data.second + pHead->pParent->data.second + "0");
         data.push_back(chars); //adds character and code to vector

      }
      else if(pHead->pRight != NULL)
      {
         pHead = pHead->pRight;
         chars = (pHead->data.second + pHead->pParent->data.second + "1");
         data.push_back(chars); //adds character and code to vector
      }
      else //if we don't find a right or left node go back up the tree
      {
         pHead = pHead->pParent; //go back to parent
         if(pHead->pRight != NULL)
         {
            pHead->pLeft = NULL; //delete node we were just at
         }
         else if(pHead->pLeft != NULL)
         {
            pHead->pRight = NULL; //delete node we were just at
         }
         else
         {
            pHead->pLeft = NULL; //delete both nodes
            pHead->pRight = NULL;
         }
      }
   }

   //supposed to be for displaying the data
   for(int j = 0; j < numItems; j++)
   {
      cout << data[j] << endl;
   }
   //cout << pHead->pLeft->data.second << endl;

}

/******************************
 * Reads in the data from the file.
 *****************************/
```

Commented [HJ14]: Const by-reference please so you don’t copy fileName

```
void readFile(string fileName)
{
   //tries to open file
   ifstream fin(fileName.c_str());
   if(fin.fail())
   {
      cout << "Failed to open file.\n";
      return;
   }

   //create a vector of pairs to hold the data
   Vector <Pair<float, string> > data;
   int numItems = 0; //for the number of items in the vector
   int num = 0;      //for the number of items in the tree
   Pair<float, string> frequency; //create a pair to hold the frequency

```

Commented [HJ15]: This is not quite right. What happens when the last valid item you read is .first and there is an EOF right after?

```
   fin >> frequency.second;
   fin >> frequency.first;
   while(!fin.eof())
   {
      data.push_back(frequency);
      numItems++;
      fin >> frequency.second;
      fin >> frequency.first;
   }

/******************************
 *need to make a new tree here
 * which will allow us to save our nodes,
 * create our paths and then display them again.
 *****************************/
```

Commented [HJ16]: This part should be a separate function. You are both reading and converting, thereby loosing cohesion

```

   //two nodes to be able to hold two subtrees
   BinaryNode<Pair<float, string> > * pHead;
   BinaryNode<Pair<float, string> > * pNew;
   do
   {
      //three iterators to find where we are, and the two least nodes
      VectorIterator<Pair<float, string> > it = data.begin();
      VectorIterator<Pair<float, string> > pLeast2 = it;
      VectorIterator<Pair<float, string> > pLeast1 = it;

      //two pairs to hold the data of least 1 and least 2
      Pair<float, string> least1 = *it;
      Pair<float, string> least2 = *it;

      //finds the least two digits
      for (int i = 0; i < numItems; i++, it++)
      {
         if (*it < least1)
         {
            least2 = least1;
            least1 = *it;
            pLeast1 = pLeast2;
            pLeast2 = it;
         }
         else if (*it == least1)
         {
            Pair<float, string> temp = *it;
```

Commented [HJ17]: Good.

```
      if (temp.second != " ")
      {
         least2 = least1;
         least1 = *it;
         pLeast1 = pLeast2;
         pLeast2 = it;
      }
   }
}

--it; //goes back to the last pLeast2

//swap if they are equal to get right order
if (least1 == least2)
{
   Pair<float, string> tmp5 = least2;
   least2 = least1;
   least1 = tmp5;
}

Pair<float, string> end = *it; // so we can move the last node
Pair<float, string> hold;
float total = least1.first + least2.first;
hold.first = total;            // the root node having the sum of the
hold.second = " ";             // of the frequencies
*pLeast1 = hold;
*pLeast2 = end;

//creates the subtree for us
BinaryNode<Pair<float, string> > * parent =
   new BinaryNode<Pair<float, string> >(hold);

bool change = false; //see if one of the subtrees in the least
//adds the appropriate node to the left
if (least1.second == " " && least1 == pHead->data)
{
   parent->addLeft(pHead);
   num++;
   change = true;
}
else if(least1.second == " " && least1 == pNew->data)
{
   num++;
   parent->addLeft(pNew);
   change = true;
}
else
{
   num++;
   parent->addLeft(least1);
   change = false;
}



bool change2 = false; //to see if one of the subtrees was the least

//adds the appropriate node to the right
if (least2.second == " " && least2 == pHead->data)
{
   parent->addRight(pHead);
   num++;
   change = true;
}
else if(least2.second == " " && least2 == pNew->data)
{
   parent->addRight(pNew);
   change = true;
   num++;
}
else
{
   parent->addRight(least2);
   change2 = false;
   num++;
}

//if both were subtrees point both to parent
if (change && change2)
```

```
      {
         pHead = parent;
         pNew = parent;
      }
      //alternate between which node will change parent
      else if (numItems%2)
      {
         pHead = parent;
      }
      else
      {
         pNew = parent;
      }

      //decrement number of items in the vector
      numItems--;
   }
   while (numItems-1);

   cout << pNew << endl; //displays the tree

   fin.close();

   //display(pNew, num); //should display the data
}

void huffman()
{
   string fileName;
   fileName = getFilename(fileName);
   readFile(fileName);

   return;
}
```

lesson07.cpp

```
/************************************************************************
* Program:
*    Lesson 07, Binary Trees
*    Brother Helfrich, CS 235
* Author:
*    Br. Helfrich
* Summary:
*    This is a driver program to exercise the BinaryNode 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>      //
#include "bnode.h"     // your BinaryNode class should be in bnode.h
#include "huffman.h"     // for huffman()
using namespace std;


// prototypes for our four test functions
void testSimple();
void testAdd();
void testDisplay();
void testMerge();

// 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 testDisplay()
#define TEST4   // for testMerge()

/**********************************************************************
 * 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 << "\t0. To generate Huffman codes\n";
   cout << "\t1. Just create and destroy a BinaryNode\n";
   cout << "\t2. The above plus add a few nodes to create a Binary Tree\n";
   cout << "\t3. The above plus display the contents of a Binary Tree\n";
   cout << "\t4. The above plus merge Binary Trees\n";

   // select
   int choice;
   cout << "> ";
   cin  >> choice;
   switch (choice)
   {
      case 0:
         huffman();
         break;
      case 1:
         testSimple();
         cout << "Test 1 complete\n";
         break;
      case 2:
         testAdd();
         cout << "Test 2 complete\n";
         break;
      case 3:
         testDisplay();
         cout << "Test 3 complete\n";
         break;
      case 4:
         testMerge();
         cout << "Test 4 complete\n";
         break;
      default:
         cout << "Unrecognized command, exiting...\n";
   }

   return 0;
}

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

   // Test2: double Stack with non-default constructor
   cout << "Create a double BinaryNode using the non-default constructor\n";
   BinaryNode <double> *pTree = new BinaryNode <double>(3.14159);
   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 Tree with the default constructor\n";
   BinaryNode <int> * pTree = new BinaryNode <int> (1);

   // add 2 to the left and 6 to the right
   pTree->addLeft(2);
   pTree->addRight(3);

   // add 1 and 3 off the left node
   pTree->pLeft->addLeft(4);
   pTree->pLeft->addRight(5);

   // add 5 and 7 to the right node
   pTree->pRight->addLeft(6);
   pTree->pRight->addRight(7);

   // now display the results:
   cout << "\tRoot......... " << pTree->data                 << endl;
   cout << "\tLeft......... " << pTree->pLeft->data          << endl;
   cout << "\tRight........ " << pTree->pRight->data         << endl;
   cout << "\tLeft-Left.... " << pTree->pLeft->pLeft->data   << endl;
   cout << "\tLeft-Right... " << pTree->pLeft->pRight->data  << endl;
   cout << "\tRight-Left... " << pTree->pRight->pLeft->data  << endl;
   cout << "\tRight-Right.. " << pTree->pRight->pRight->data << endl;

   // finally, delete everything
   deleteBinaryTree(pTree);
   cout << "\tTree deleted\n";
#endif // TEST2
}

/*******************************************
 * TEST Display
 * We will build a binary tree and display the
 * results on the screen
 ******************************************/
void testDisplay()
{
#ifdef TEST3
   // create
   cout << "Create a string Binary Node with the default constructor\n";
   BinaryNode <string> *pTree = NULL;

   // prompt for seven words
   cout << "\tEnter seven words\n";
   string words[7];
   for (int i = 0; i < 7; i++)
   {
      cout << "\t> ";
      cin  >> words[i];
   }

   // put the seven words in the tree
   pTree =                 new BinaryNode <string> (words[3]);
   pTree->addLeft(         new BinaryNode <string> (words[1]));
   pTree->addRight(        new BinaryNode <string> (words[5]));
   pTree->pLeft->addLeft(  new BinaryNode <string> (words[0]));
   pTree->pLeft->addRight( new BinaryNode <string> (words[2]));
   pTree->pRight->addLeft( new BinaryNode <string> (words[4]));
   pTree->pRight->addRight(new BinaryNode <string> (words[6]));

   //display(pTree);

   // display the results
   cout << pTree << endl;

   // delete the tree
   deleteBinaryTree(pTree);
   cout << "\tTree deleted\n";
#endif // TEST3
}

/*******************************************
 * TEST MERGE
 * Create three binary trees and merge them
 *******************************************/
void testMerge()
{
#ifdef TEST4
   // create three trees
   cout << "Create three 3-node binary trees\n";
   BinaryNode <char> * pLower  = new BinaryNode <char> ('b');
   BinaryNode <char> * pMiddle = new BinaryNode <char> ('m');
   BinaryNode <char> * pUpper  = new BinaryNode <char> ('y');
   pLower->addLeft  ('a');
   pLower->addRight ('c');
   pMiddle->addLeft ('l');
   pMiddle->addRight('n');
   pUpper->addLeft  ('x');
   pUpper->addRight ('z');

   // add Lower to the left of Middle, and Upper to the right of Middle
   pMiddle->pLeft->addLeft(pLower);
   pMiddle->pRight->addRight(pUpper);

   // display the results
   cout << pMiddle << endl;

   // delete the tree
   deleteBinaryTree(pMiddle);
   cout << "\tTree deleted\n";
#endif // TEST4
}
```

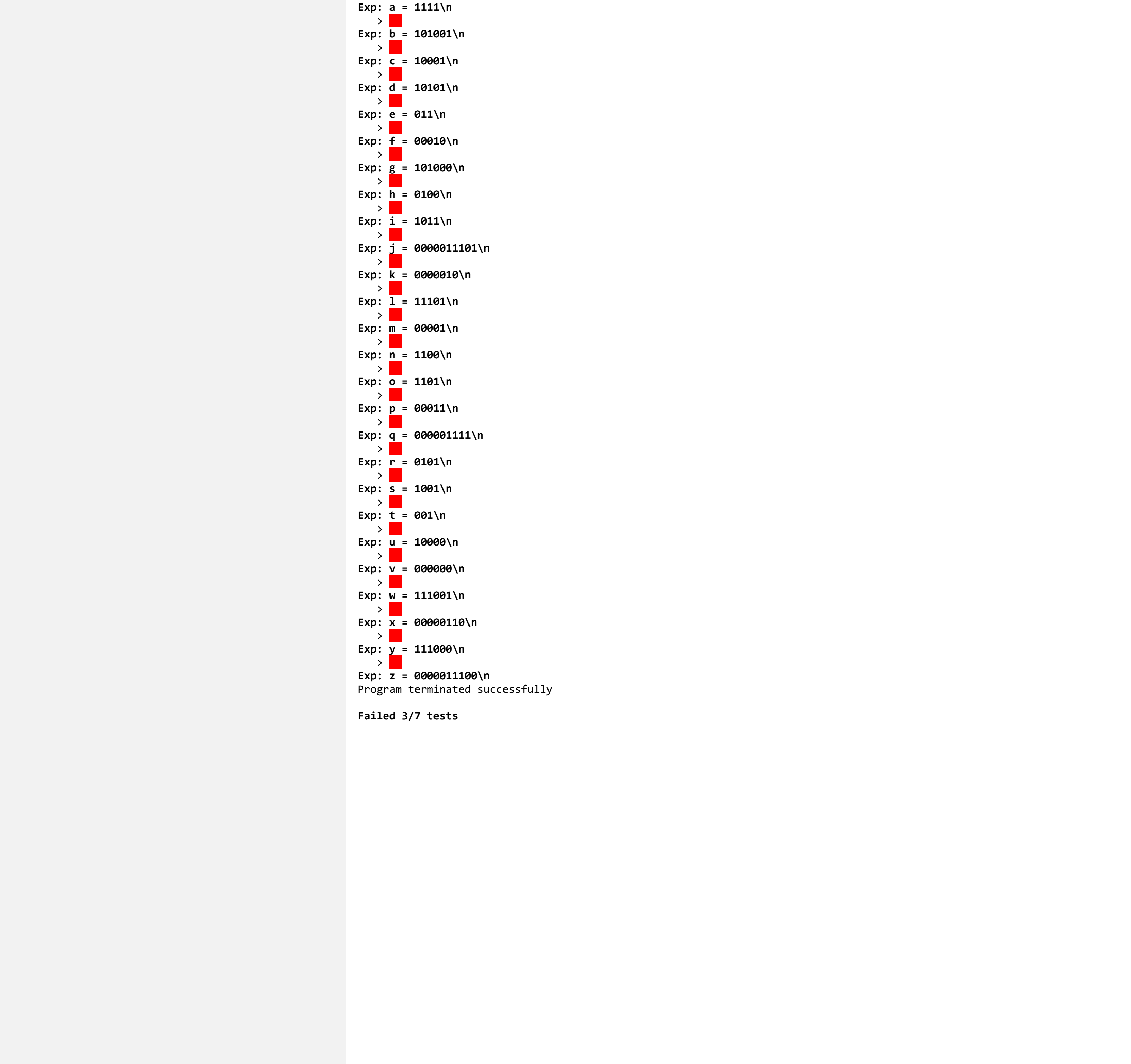
Test Bed Results

```
cs235d.out:

Started program
   > Select the test you want to run:
   >    0. To generate Huffman codes
   >    1. Just create and destroy a BinaryNode
   >    2. The above plus add a few nodes to create a Binary Tree
   >    3. The above plus display the contents of a Binary Tree
   >    4. The above plus merge Binary Trees
   > > 1
   > Create a bool BinaryNode using the default constructor
   > Create a double BinaryNode using the non-default constructor
   > Test 1 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    0. To generate Huffman codes
   >    1. Just create and destroy a BinaryNode
   >    2. The above plus add a few nodes to create a Binary Tree
   >    3. The above plus display the contents of a Binary Tree
   >    4. The above plus merge Binary Trees
   > > 2
   > Create an integer Binary Tree with the default constructor
   >    Root......... 1
   >    Left......... 2
   >    Right........ 3
   >    Left-Left.... 4
   >    Left-Right... 5
   >    Right-Left... 6
   >    Right-Right.. 7
   >    Tree deleted
   > Test 2 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    0. To generate Huffman codes
   >    1. Just create and destroy a BinaryNode
   >    2. The above plus add a few nodes to create a Binary Tree
   >    3. The above plus display the contents of a Binary Tree
   >    4. The above plus merge Binary Trees
   > > 3
   > Create a string Binary Node with the default constructor
   >    Enter seven words
   >    > one
   >    > two
   >    > three
   >    > four
   >    > five
   >    > six
   >    > seven
   > one two three four five six seven
   >    Tree deleted
   > Test 3 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    0. To generate Huffman codes
   >    1. Just create and destroy a BinaryNode
   >    2. The above plus add a few nodes to create a Binary Tree
   >    3. The above plus display the contents of a Binary Tree
   >    4. The above plus merge Binary Trees
```

```
    > > 4
    > Create three 3-node binary trees
    > a b c l m n x y z
    >    Tree deleted
    > Test 4 complete
Program terminated successfully

Started program
    > Select the test you want to run:
    >    0. To generate Huffman codes
    >    1. Just create and destroy a BinaryNode
    >    2. The above plus add a few nodes to create a Binary Tree
    >    3. The above plus display the contents of a Binary Tree
    >    4. The above plus merge Binary Trees
    > > 0
    > Enter the filename containing the value frequencies: /home/cs235/lesson07/huffman1.txt
```

Commented [HJ18]: Remove these please.

```
    > (0.45, E) (1,  ) (0.1, B) (0.2,  ) (0.1, C) (0.55,  ) (0.15, D) (0.35,  ) (0.2, A) \n
Exp: A = 111\n
    > 
Exp: B = 100\n
    > 
Exp: C = 101\n
    > 
Exp: D = 110\n
    > 
Exp: E = 0\n
Program terminated successfully

Started program
    > Select the test you want to run:
    >    0. To generate Huffman codes
    >    1. Just create and destroy a BinaryNode
    >    2. The above plus add a few nodes to create a Binary Tree
    >    3. The above plus display the contents of a Binary Tree
    >    4. The above plus merge Binary Trees
    > > 0
    > Enter the filename containing the value frequencies: /home/cs235/lesson07/huffman2.txt
    > (41,  ) (162,  ) (39,  ) (121,  ) (36,  ) (82,  ) (21, for) (46,  ) (25, while) \n
Exp: case = 11011\n
    > 
Exp: class = 001\n
    > 
Exp: do = 01010\n
    > 
Exp: else = 0100\n
    > 
Exp: false = 110101\n
    > 
Exp: for = 011\n
    > 
Exp: goto = 1101000\n
    > 
Exp: if = 000\n
    > 
Exp: int = 100\n
    > 
Exp: main = 1111\n
    > 
Exp: static = 1101001\n
    > 
Exp: struct = 1100\n
    > 
Exp: switch = 01011\n
    > 
Exp: true = 1110\n
    > 
Exp: while = 101\n
Program terminated successfully

Started program
    > Select the test you want to run:
    >    0. To generate Huffman codes
    >    1. Just create and destroy a BinaryNode
    >    2. The above plus add a few nodes to create a Binary Tree
    >    3. The above plus display the contents of a Binary Tree
    >    4. The above plus merge Binary Trees
    > > 0
    > Enter the filename containing the value frequencies: /home/cs235/lesson07/huffman3.txt
    > (3142,  ) (7413,  ) (1778,  ) (4271,  ) (1231, e) (2493,  ) (1262,  ) \n
```



```
Exp: a = 1111\n
   > 🟥
Exp: b = 101001\n
   > 🟥
Exp: c = 10001\n
   > 🟥
Exp: d = 10101\n
   > 🟥
Exp: e = 011\n
   > 🟥
Exp: f = 00010\n
   > 🟥
Exp: g = 101000\n
   > 🟥
Exp: h = 0100\n
   > 🟥
Exp: i = 1011\n
   > 🟥
Exp: j = 0000011101\n
   > 🟥
Exp: k = 0000010\n
   > 🟥
Exp: l = 11101\n
   > 🟥
Exp: m = 00001\n
   > 🟥
Exp: n = 1100\n
   > 🟥
Exp: o = 1101\n
   > 🟥
Exp: p = 00011\n
   > 🟥
Exp: q = 000001111\n
   > 🟥
Exp: r = 0101\n
   > 🟥
Exp: s = 1001\n
   > 🟥
Exp: t = 001\n
   > 🟥
Exp: u = 10000\n
   > 🟥
Exp: v = 000000\n
   > 🟥
Exp: w = 111001\n
   > 🟥
Exp: x = 00000110\n
   > 🟥
Exp: y = 111000\n
   > 🟥
Exp: z = 0000011100\n
Program terminated successfully

Failed 3/7 tests
```

Grading Criteria

Criteria	Exceptional	Good	Acceptable	Developing	Missing	Weight	Score
	100%	90%	70%	50%	0%		
BinaryNode interface	The interfaces are perfectly specified with respect to const, pass-by-reference, etc.	lesson07.cpp compiles without modification	All of the methods in BinaryNode match the problem definition	BinaryNode has many of the same interfaces as the problem definition	The public methods and variables in the BinaryNode class do not resemble the problem definition	20	
BinaryNode Implementation	Passes all four BinaryNode 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	10	
Huffman Code	The code is elegant and efficient	Passes the Huffman Code testBed test	The code essentially works but with minor defects	Elements of the solution are present	The Huffman Code problem was not attempted	40	-16
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	-6
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							78

Commented [HJ19]: Good work, but not quite finished.