makefile

```
####################################################################
# Program:
#     Week 08, Binary Tree
#     Brother Ercanbrack, CS235
# Author:
#     Yurii Vasiuk
# Summary:
#     Binary Tree Node and Huffman code
# Time:
#     20 hours
####################################################################

####################################################################
# The main rule
####################################################################
a.out: week08.o huffman.o
       g++ -o a.out week08.o huffman.o -g
       tar -cf week08.tar *.h *.cpp makefile

####################################################################
# The individual components
#      week08.o     : the driver program
#      huffman.o      : the logic for the huffman code program
####################################################################
week08.o: bnode.h huffman.h week08.cpp
       g++ -c week08.cpp -g

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

bnode.h

```
/***********************************************************************
* Header:
*    bNode
* Summary:
*    Node will be a part of BinaryTree class
* Author
*    Yura Vasiuk
************************************************************************/

#ifndef BNODE_H
#define BNODE_H

#include <iostream>
using namespace std;

/*************************************************
* NODE
* A class to be used in LinkedList
*************************************************/
template <typename T>
class BinaryNode
{
public:
   BinaryNode<T> * pLeft;
   BinaryNode<T> * pRight;
   BinaryNode<T> * pParent;
   T data;

   // default and non-default constructors
   BinaryNode() : pLeft(NULL), pRight(NULL), pParent(NULL) {}
   BinaryNode(T data)
   {
      this->data = data; this->pLeft = NULL; this->pRight = NULL; this->pParent = NULL;
   }
```

```
   // add left and right
   BinaryNode<T> * addLeft(BinaryNode<T> *p) throw (const char *);
   BinaryNode<T> * addLeft(T t) throw (const char *);
   BinaryNode<T> * addRight(BinaryNode<T> *p) throw (const char *);
   BinaryNode<T> * addRight(T t) throw (const char *);

   int size() const;
};

/************************************************
* SIZE
* Calculate and return the size of the tree
************************************************/
template <typename T>
int BinaryNode<T>::size() const
{
   return 1 + (this->pLeft == NULL ? 0 : this->pLeft->size()) +
      (this->pRight == NULL ? 0 : pRight->size());
}

/************************************************
* ADDLEFT
* Add a node to the left of the current one (two functions, overload)
************************************************/
template <typename T>
BinaryNode<T> * BinaryNode<T> ::addLeft(BinaryNode<T> *p) throw (const char *)
{
   if (p != NULL)
   {
      p->pParent = this;
   }
   this->pLeft = p;

   return this;
}

template <typename T>
BinaryNode<T> * BinaryNode<T>::addLeft(T t) throw (const char *)
{
   BinaryNode<T> *p;
   try
   {
      p = new BinaryNode<T>(t);
   }
   catch (std::bad_alloc)
   {
      throw "ERROR: Unable to allocate a node";
   }
   addLeft(p);

   return this;
}

/************************************************
* ADDRIGHT
* Add a node to the right of the current one (two functions, overload)
************************************************/
template <typename T>
BinaryNode<T> * BinaryNode<T>::addRight(BinaryNode<T> * p) throw (const char *)
{
   if (p != NULL)
   {
      p->pParent = this;
   }
   this->pRight = p;

   return this;
}

template <typename T>
BinaryNode<T> * BinaryNode<T>::addRight(T t) throw (const char *)
{
   BinaryNode<T> *p;
   try
   {
      p = new BinaryNode<T>(t);
   }
   catch (std::bad_alloc)
```

```
   {
      throw "ERROR: Unable to allocate a node";
   }
   addRight(p);

   return this;
}

/************************************************
* DELETE
* Delete all nodes
************************************************/
template <typename T>
void deleteBinaryTree(BinaryNode<T> * & p)
{
   if (p->pLeft != NULL)
      deleteBinaryTree(* & p->pLeft);
   if (p->pRight != NULL)
      deleteBinaryTree(* & p->pRight);
   delete p;
}

/************************************************
* OPERATOR<<
* Display the content of the passed Linked List
************************************************/
template <typename T>
ostream & operator<<(ostream & out, BinaryNode<T> * pRhs)
{
   if (pRhs != NULL)
   {
      out << pRhs->pLeft;
      out << pRhs->data << " ";
      out << pRhs->pRight;
   }

   return out;
}


#endif // BNODE_H
```

huffman.h

```
/**********************************************************************
 * Module:
 *    Week 08, Huffman
 *    Brother Helfrich, CS 235
 * Author:
 *    Br. Helfrich
 * Summary:
 *    This program will implement the huffman() function
 **********************************************************************/

#ifndef HUFFMAN_H
#define HUFFMAN_H

#include "bnode.h"
#include "pair.h"
#include <iostream>    // for streams
#include <fstream>     // for files
#include <string>      // for strings
#include <deque>       // for deque (in making the tree)
#include <vector>      // for vector (in coding the tree)
#include <algorithm>   // for sort()
using namespace std;

void huffman(string fileName);

#endif // HUFFMAN_h
```

pair.h

```
/**********************************************************************
 * Module:
 *    Week 08, 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 compairing 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() : first(), second() {}
   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
```

huffman.cpp

```
/**************************************************************************
 * Module:
 *    Week 08, Huffman
 *    Brother Helfrich, CS 235
 * Author:
 *    Yurii Vasiuk
 * Summary:
 *    This program will implement the huffman() function
 **************************************************************************/
#include "huffman.h"       // for HUFFMAN() prototype
using namespace std;

// the prototipes
BinaryNode<Pair<string, float> > * buildHuffmanTree(deque<Pair<string, float> > & pairs);
string & makeHuffmanCode(BinaryNode<Pair<string, float> > * & root, string & code, string & table);

/***************************************
 * HUFFMAN
 * Driver program to exercise the huffman generation code
 ***************************************/
void huffman(string fileName)
{
   // the container for reading data from the file
   Pair<string, float> thePair = Pair<string, float>();
   // the container to store the pairs
   deque<Pair<string, float> > pairs;

   // read from the file
  // "/home/vas14001/CS235_Spring2016/week08_Node_BinaryTree/huffman1.txt"
  // the name of the file is hardcoded because the program does not compile with the fileName ???
   ifstream fin(fileName.c_str());
   if (fin.fail())
   {
      cout << "Could not open the file " << fileName << endl;
      return;
   }
   while (fin >> thePair)
   {
      pairs.push_back(thePair);
   }
   // finish with the file reading
   fin.close();

   // sort the deque of pairs by the frequency
   sort(pairs.begin(), pairs.end());

   // build the tree
   BinaryNode<Pair<string, float> > *theHuffmanTree = new BinaryNode<Pair<string, float> >();
   theHuffmanTree = buildHuffmanTree(pairs);

   // make the Huffman code
   // the containers for 1 string of code and the whole table of code strings
   string code = "";
   string table;
   makeHuffmanCode(theHuffmanTree, code, table);

   // display the Huffman code
   cout << table;

    return;
}

// BUILD THE TREE
BinaryNode<Pair<string, float> > * buildHuffmanTree(deque<Pair<string, float> > & pairs)
{
   /**********************************************************************
   * the pairs are in the sorted deque
   * build the tree
   **********************************************************************/

   // the initial root of the tree
   BinaryNode<Pair<string, float> > *root = new BinaryNode<Pair<string, float> >();
   // this will hold the root temporary
   BinaryNode<Pair<string, float> > *temp = new BinaryNode<Pair<string, float> >();
   // the iterator pointing to the pairs in the array
   deque<Pair<string, float> >::iterator it = pairs.begin();
```

Commented [ES1]: Needs to be an array of strings. So that each

```
   // building the tree
   // make left leaf
   BinaryNode<Pair<string, float> > *leftLeaf = new BinaryNode<Pair<string, float> >(*it);
   root->addLeft(leftLeaf);
   // keep making the root and the right leaves
   while (pairs.size() > 2)
   {
      // for the access to the first two pairs
      it = pairs.begin();
      // assine the values of the tree root node
      root->data.second = (it->getSecond()) + ((++it)->getSecond());
      root->data.first = " ";
      // make the right leaf
      BinaryNode<Pair<string, float> > *rightLeaf =
         new BinaryNode<Pair<string, float> >(*it);
      root->addRight(rightLeaf);
      // make the new root pointing left to the current one
      temp = root;
      root = new BinaryNode<Pair<string, float> >(temp->data);
      root->addLeft(temp);
      // done with the tree

      // take care of the deque
      pairs.pop_front();
      pairs.pop_front();
      pairs.push_front(root->data);
   }

   // handle the last pair in the deque
   // iterator to the last pair in the deque
   it = pairs.begin();
   it++;
   // make the right leaf
   BinaryNode<Pair<string, float> > *rightLeaf =
      new BinaryNode<Pair<string, float> >(*it);
   root->addRight(rightLeaf);
   // the tree is built now ------------------------------------------
   return root;
}

// MAKE CODE BASED UPON THE TREE
string & makeHuffmanCode(BinaryNode<Pair<string, float> > * & root, string & code, string & table)
{
   if (root->pLeft == NULL && root->pRight == NULL)
      table = table + root->data.getFirst() + " = " + code + "\n";
   else
   {
      makeHuffmanCode(root->pLeft, code += "0", table);
      makeHuffmanCode(root->pRight, code += "1", table);
   }

   return table;
}
```

Commented [ES2]: You have lots of problems here.
You don't always add together the smallest two frequencies.

You aren't building the tree correctly!

Commented [ES3]: You must sort the array after every push() to be sure you always have the smallest two frequencies at the front of the deque.

Commented [ES4]: This needs to be an array of strings not a string.

You are supposed to store the code into the table array. This table array may be a two dimensional array or something where you would use the character to compute an index into the table so you can find the code in the table based upon what character it is.

week08.cpp

```
/**********************************************************************
* Program:
*    Week 08, 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>      // for STRING
#include <cassert>     // for ASSERT
#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 << "\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";
   cout << "\ta. To generate Huffman codes\n";

   // select
   char choice;
   cout << "> ";
   cin  >> choice;
   switch (choice)
   {
      case 'a':
      {
         // get the filename
         string fileName;
         cout << "Enter the filename containing the value frequencies.\n";
         cout << "Enter \"quit\" when done.\n";

         cout << "> ";
         cin  >> fileName;

         while (fileName != "quit")
         {
            huffman(fileName);
            cout << "> ";
            cin  >> fileName;
         }
         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
   try
   {
      // Test1: a bool Stack with defeault constructor
      cout << "Create a bool BinaryNode using the default constructor\n";
      BinaryNode <bool> tree;
      cout << "\tSize:    " << tree.size() << endl;

      // 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);
      cout << "\tSize:    " << pTree->size() << endl;
      delete pTree;
   }
   catch (const char * error)
   {
      cout << error << endl;
   }
#endif //TEST1
}

/***************************************
 * TEST ADD
 * Add a few nodes together to create a tree, then
 * destroy it when done
 ***************************************/
void testAdd()
{
#ifdef TEST2
   try
   {
      // create
      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 << "The elements in the binary tree:\n";
      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;
      cout << "\tSize: "         << pTree->size()               << endl;

      // double-check the parents
      assert(pTree->pLeft->pParent  ==  pTree);
      assert(pTree->pRight->pParent ==  pTree);
      assert(pTree->pLeft ->pLeft ->pParent  ==  pTree->pLeft);
      assert(pTree->pLeft ->pRight->pParent  ==  pTree->pLeft);
      assert(pTree->pRight->pLeft ->pParent  ==  pTree->pRight);
      assert(pTree->pRight->pRight->pParent  ==  pTree->pRight);
      assert(pTree->pLeft ->pLeft ->pParent->pParent  ==  pTree);
      assert(pTree->pLeft ->pRight->pParent->pParent  ==  pTree);
      assert(pTree->pRight->pLeft ->pParent->pParent  ==  pTree);
      assert(pTree->pRight->pRight->pParent->pParent  ==  pTree);
      cout << "All the parent nodes are correctly set\n";

      // move some nodes around
      BinaryNode <int> * pSix   = pTree->pRight->pLeft;
      BinaryNode <int> * pSeven = pTree->pRight->pRight;
      pTree->pRight->addRight(pSix);
      pTree->pRight->addLeft(pSeven);
      assert(pTree->pRight->pRight->data == 6);
```

```
      assert(pTree->pRight->pLeft->data == 7);
      cout << "Was able to move the '6' and '7' nodes\n";

      // delete the left half of the tree
      BinaryNode <int> * pTemp = pTree->pLeft;
      pTree->addLeft((BinaryNode <int> *)NULL);
      assert(pTree->pLeft == NULL);
      deleteBinaryTree(pTemp);
      cout << "Size after deleting half the nodes: " << pTree->size() << endl;

      // finally, delete everything else
      deleteBinaryTree(pTree);
      cout << "Was able to delete the rest of the binary tree\n";
   }
   catch (const char * error)
   {
      cout << error << endl;
   }
#endif // TEST2
}

/*****************************************
 * TEST Display
 * We will build a binary tree and display the
 * results on the screen
 *****************************************/
void testDisplay()
{
#ifdef TEST3
   try
   {
      // create
      BinaryNode <string> *pTree = NULL;

      // prompt for seven words
      string word;
      cout << "Enter seven words\n";
      cout << "\tRoot node:         ";
      cin  >> word;
      pTree = new BinaryNode <string> (word);

      cout << "\tLeft child:        ";
      cin  >> word;
      pTree->addLeft(new BinaryNode <string> (word));

      cout << "\tRight child:       ";
      cin  >> word;
      pTree->addRight(new BinaryNode <string> (word));

      cout << "\tLeft-Left child:   ";
      cin  >> word;
      pTree->pLeft->addLeft(new BinaryNode <string> (word));

      cout << "\tLeft-Right child:  ";
      cin  >> word;
      pTree->pLeft->addRight(new BinaryNode <string> (word));

      cout << "\tRight-Left child:  ";
      cin  >> word;
      pTree->pRight->addLeft(new BinaryNode <string> (word));

      cout << "\tRight-Right child: ";
      cin  >> word;
      pTree->pRight->addRight(new BinaryNode <string> (word));

      // when we are adding nothing, we should just return
      pTree->pLeft->pLeft->addLeft(NULL);
      pTree->pRight->pRight->addRight(NULL);

      // display the results
      cout << "Completed tree: { " << pTree << "}\n";

      // delete the tree
      deleteBinaryTree(pTree);
   }
   catch (const char * error)
   {
      cout << error << endl;
   }
```

```
#endif // TEST3
}

/****************************************
 * TEST MERGE
 * Create three binary trees and merge them
 ****************************************/
void testMerge()
{
#ifdef TEST4
   try
   {
      // create the middle tree
      BinaryNode <char> * pMiddle = new BinaryNode <char> ('m');
      pMiddle->addLeft ('l');
      pMiddle->addRight('n');
      cout << "Middle tree: { " << pMiddle << "}"
           << " size = " << pMiddle->size() << endl;

      // create lower tree
      BinaryNode <char> * pLower  = new BinaryNode <char> ('b');
      pLower->addLeft  ('a');
      pLower->addRight ('c');
      cout << "Lower tree: { " << pLower << "}"
           << " size = " << pLower->size() << endl;


      // create upper tree
      BinaryNode <char> * pUpper  = new BinaryNode <char> ('y');
      pUpper->addLeft  ('x');
      pUpper->addRight ('z');
      cout << "Upper tree: { " << pUpper << "}"
           << " size = " << pUpper->size() << endl;


      // add Lower to the left of Middle, and Upper to the right of Middle
      pMiddle->pLeft->addLeft(pLower);
      pMiddle->pRight->addRight(pUpper);
      cout << "Merged tree: { " << pMiddle << "}"
           << " size = " << pMiddle->size() << endl;

      // delete the tree
      deleteBinaryTree(pMiddle);
   }
   catch (const char * error)
   {
      cout << error << endl;
   }
#endif // TEST4
}
```

Test Bed Results

```
a.out:

------------------------------------------------------------
Starting Test 1

   > Select the test you want to run:
   >    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
   >    a. To generate Huffman codes
   > > 1
Empty tree of size one
   > Create a bool BinaryNode using the default constructor
   >    Size:    1
A singleton tree of size one
   > Create a double BinaryNode using the non-default constructor
   >    Size:    1
   > Test 1 complete

Test 1 passed.
------------------------------------------------------------

------------------------------------------------------------
```

```
Starting Test 2

   > Select the test you want to run:
   >    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
   >    a. To generate Huffman codes
   > > 2

Create an integer Binary Tree with the non-default constructor.
Next we will add six items to make a tree
   > The elements in the binary tree:
   >    Root......... 1
   >    Left......... 2
   >    Right........ 3
   >    Left-Left.... 4
   >    Left-Right... 5
   >    Right-Left... 6
   >    Right-Right.. 7
   >    Size: 7

Test to make sure pParent is set up correctly
   > All the parent nodes are correctly set

Check to see if we can move nodes around
   > Was able to move the '6' and '7' nodes

Check to see if deleteBinaryTree() works with partial trees
   > Size after deleting half the nodes: 4

Check to see if deleteBinaryTree() can delete the rest of the tree.
If this fails, it probably means that the first call to deleteBinaryTree()
left the tree in an invalid state
   > Was able to delete the rest of the binary tree
   > Test 2 complete

Test 2 passed.
------------------------------------------------------------

------------------------------------------------------------
Starting Test 3

   > Select the test you want to run:
   >    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
   >    a. To generate Huffman codes
   > > 3
Create a string Binary Node with the default constructor\n
   > Enter seven words

Test to see if you can add a node to a NULL root
   >    Root node:         four

The next two cases were exercised in Test 2
   >    Left child:        two
   >    Right child:       six

These four also have been exercised in Test 2
   >    Left-Left child:   one
   >    Left-Right child:  three
   >    Right-Left child:  five
   >    Right-Right child: seven

Two tests here.
1. See if we can add NULL nodes
2. See if the insertion operator was correctly written
   > Completed tree: { one two three four five six seven }
   > Test 3 complete

Test 3 passed.
------------------------------------------------------------

------------------------------------------------------------
Starting Test 4

   > Select the test you want to run:
```

```
   >    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
   >    a. To generate Huffman codes
   > > 4

Create the middle tree, nothing fancy
   > Middle tree: { l m n } size = 3

Create the lower tree
   > Lower tree: { a b c } size = 3

Create the upper tree
   > Upper tree: { x y z } size = 3

The merged tree.
We are putting { a b c } under the l of { l m n }
We are putting { x y z } under the n of { l m n }
   > Merged tree: { a b c l m n x y z } size = 9
   > Test 4 complete

Test 4 passed.
---------------------------------------------------------------

---------------------------------------------------------------
Starting Test 5

   > Select the test you want to run:
   >    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
   >    a. To generate Huffman codes
   > > a
   > Enter the filename containing the value frequencies.
   > Enter "quit" when done.

The example from the assignment
               1- A
               |
          1----+
          |    |
          |    0- D
     1----+
     |    |    1- C
     |    |    |
 ----+    0----+
     |         |
     |         0- B
     0-E
   > > /home/cs235/week08/huffman1.txt
   > A = 0000\n
Exp: A = 111\n
   > B = 00001\n
Exp: B = 100\n
   > C = 000011\n
Exp: C = 101\n
   > D = 0000111\n
Exp: D = 110\n
   > E = 00001111\n
Exp: E = 0\n

This one is quite a bit larger
                    1-main
               1----+
               |    0-true   1-case
               |             |
          1----+        1----+    1-false
          |    |        |    |    |
          |    |        |    0----+    1-static
          |    0--------+         |    |
     1----+              |         0----+
     |    |              0-struct       |
     |    |    1-while                  0-goto
     |    0----+
 ----+         0-int    1-for     1-switch
     |                   |         |
     |    1-------------+    1----+
```


```
   > > /home/cs235/week08/huffman2.txt
   > case = 00000000000000\n
Exp: case = 11011\n
   > class = 0000000000000001\n
Exp: class = 001\n
   > do = 000000000000011\n
Exp: do = 01010\n
   > else = 00000000000000111\n
Exp: else = 0100\n
   > false = 000000000000001111\n
Exp: false = 110101\n
   > for = 0000000000000011111\n
Exp: for = 011\n
   > goto = 00000000000000111111\n
Exp: goto = 1101000\n
   > if = 000000000000001111111\n
Exp: if = 000\n
   > int = 0000000000000011111111\n
Exp: int = 100\n
   > main = 00000000000000111111111\n
Exp: main = 1111\n
   > static = 000000000000001111111111\n
Exp: static = 1101001\n
   > struct = 000000000000001111111111\n
Exp: struct = 1100\n
   > switch = 0000000000000011111111111\n
Exp: switch = 01011\n
   > true = 0000000000000011111111111111\n
Exp: true = 1110\n
   > while = 00000000000000111111111111111\n
Exp: while = 101\n
```

I have seen this done by hand once by a student; it was huge!

```
   > > /home/cs235/week08/huffman3.txt
   > a = 00000000000000000000000000\n
Exp: a = 1111\n
   > b = 00000000000000000000000001\n
Exp: b = 101001\n
   > c = 00000000000000000000000011\n
Exp: c = 10001\n
   > d = 00000000000000000000000111\n
Exp: d = 10101\n
   > e = 00000000000000000000000001111\n
Exp: e = 011\n
   > f = 00000000000000000000000011111\n
Exp: f = 00010\n
   > g = 00000000000000000000000000111111\n
Exp: g = 101000\n
   > h = 00000000000000000000000000011111111\n
Exp: h = 0100\n
   > i = 00000000000000000000000000011111111\n
Exp: i = 1011\n
   > j = 00000000000000000000000000111111111\n
Exp: j = 0000011101\n
   > k = 00000000000000000000000000011111111111\n
Exp: k = 0000010\n
   > l = 00000000000000000000000000011111111111\n
Exp: l = 11101\n
   > m = 00000000000000000000000000111111111111\n
Exp: m = 00001\n
   > n = 00000000000000000000000000111111111111\n
Exp: n = 1100\n
   > o = 00000000000000000000000000111111111111111\n
Exp: o = 1101\n
   > p = 00000000000000000000000000111111111111111\n
Exp: p = 00011\n
   > q = 00000000000000000000000000111111111111111\n
Exp: q = 000001111\n
   > r = 00000000000000000000000000111111111111111\n
Exp: r = 0101\n
   > s = 00000000000000000000000000111111111111111111\n
Exp: s = 1001\n
   > t = 00000000000000000000000000111111111111111111\n
Exp: t = 001\n
```

```
   > u = 00000000000000000000000000011111111111111111111\n
Exp: u = 10000\n
   > v = 00000000000000000000000000011111111111111111111\n
Exp: v = 000000\n
   > w = 00000000000000000000000000011111111111111111111\n
Exp: w = 111001\n
   > x = 00000000000000000000000000011111111111111111111\n
Exp: x = 00000110\n
   > y = 00000000000000000000000000011111111111111111111\n
Exp: y = 111000\n
   > z = 00000000000000000000000000011111111111111111111\n
Exp: z = 0000011100\n
   > > quit

Test 5 failed.
------------------------------------------------------------


============================================================
Failed 1/5 tests.
============================================================
```

Grading Criteria

Criteria	Exceptional 100%	Good 90%	Acceptable 70%	Developing 50%	Missing 0%	Weight	Score
BinaryNode interface	The interfaces are perfectly specified with respect to const, pass-by-reference, etc.	week08.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	
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	Huffman Codes are not correct -15 pts						85/100

vas14001@byui.edu