Huffman Codes

The source code that follows consists of a class `HuffmanCode` and a simple driver program for it. (It can be downloaded on the book's website — see *Chap10/Huffman*.) The member function `buildDecodingTree()` initializes a tree consisting of a single node and then reads letters and their codes from a code file and constructs the decoding tree. For each letter in the file, it calls function `addToTree()` to follow a path determined by the code of the character, creating nodes as necessary. When the end of the code string is reached, the character is inserted in the last (leaf) node created on this path. The function `decode()` is then called to read a message string of bits from `messageFile` and decode it using the decoding tree. Function `printTree()` is included simply to give an idea of what the tree looks like. It is basically nothing more than an RNL traversal of the tree. It prints the tree "on its side" without the directed arcs and 0/1 labels. We leave it to the reader to draw these in and then rotate the tree 90 degrees so it has its usual orientation.

Huffman Code

//----- Huffman.h -----

```
#include <string>
#include <iostream>
#include <fstream>

#ifndef HUFFMAN
#define HUFFMAN
class Huffman
{
 private:
 /*** Node structure ***/
 class BinNode
 {
  public:
   char data;
   BinNode * left,
           * right;
   // BinNode constructor
   BinNode(char item)
   {
     data = item;
     left = right = 0;
   }
 };

  typedef BinNode * BinNodePointer;

 public:
  /*** Function members ***/
  Huffman();
  /*-------------------------------------------------------------------------------
    Constructor
    Precondition:  None.
    Postcondition: A one-node binary tree with root node pointed to by myRoot
        has been created.
  -----------------------------------------------------------------------------*/

  void buildDecodingTree(ifstream & codeIn);
  /*-------------------------------------------------------------------------------
    Build the Huffman decoding tree.
    Precondition:  ifstream codeIn is open and is connected to a file that contains
        characters and their codes.
    Postcondition: A oneHuffman decoding tree has been created with root node
        pointed to by myRoot.
  -----------------------------------------------------------------------------*/

  void insert(char ch, string code);
  /*-------------------------------------------------------------------------------
    Insert a node for a character in Huffman decoding tree.
    Precondition:  code is the bit string that is the code for ch.
    Postcondition: A node containing ch has been inserted into the Huffman tree
        with root pointed to by myRoot.
  -----------------------------------------------------------------------------*/

  void decode(ifstream & messageIn);
  /*-------------------------------------------------------------------------------
    Build the Huffman decoding tree.
    Precondition:  ifstream messageIn is open and is connected to a file that
        contains the message to be decoded.
    Postcondition: The decoded message has been output to cout.
  -----------------------------------------------------------------------------*/

  void printTree(ostream & out, BinNodePointer root, int indent);
  /*-------------------------------------------------------------------------------
    Recursive function to display a binary tree with root pointed to by root.
    Precondition:  ostream out is open; root points to a binary tree; indent >= 0
        is the amount to indent each level..
    Postcondition: Graphical representation of the binary tree has been output
        to out.
  -----------------------------------------------------------------------------*/

  void displayDecodingTree(ostream & out);
  /*-------------------------------------------------------------------------------
    Display the decoding tree.
    Precondition:  ostream out is open.
    Postcondition: The decoding tree has been output to out (via printTree().)
  -----------------------------------------------------------------------------*/

/*** Data members ***/
 private:
  BinNodePointer myRoot;
};

//--- Definition of constructor
inline Huffman::Huffman()
{
  myRoot = new BinNode('*');
}
```

```
//--- Definition of displayDecodingTree()
inline void Huffman::displayDecodingTree(ostream & out)
{
  printTree(out, myRoot, 0);
}

#endif
```

//----- Huffman.cpp -----

```
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

#include "Huffman.h"

//--- Definition of buildDecodingTree()
void Huffman::buildDecodingTree(ifstream & codeIn)
{
  char ch;           // a character
  string code;       // its code
  for (;;)
  {
    codeIn >> ch >> code;
    if ( codeIn.eof() ) return;
    insert(ch, code);
  }
}

//--- Definition of insert()
void Huffman::insert(char ch, string code)
{
  Huffman::BinNodePointer p = myRoot;   // pointer to move down the tree
  for(int i = 0; i < code.length(); i++)
  {
    switch (code[i])
    {
      case '0' :            // descend left
        if (p->left == 0)  // create node along path
          p->left = new Huffman::BinNode('*');
        p = p->left;
        break;
      case '1' :            // descend right
        if (p->right == 0) // create node along path
          p->right = new Huffman::BinNode('*');
        p = p->right;
        break;
      default:
        cerr << "*** Illegal character in code ***\n";
        exit(1);
    }
  }
  p->data = ch;
}
```

```
//--- Definition of decode()
void Huffman::decode(ifstream & messageIn)
{
  char bit;                   // next message bit
  Huffman::BinNodePointer p; // pointer to trace path in decoding tree
  for(;;)
  {
    p = myRoot;
    while (p->left != 0 || p->right != 0)
    {
      messageIn >> bit;
      if ( messageIn.eof() ) return;
      cout << bit;
      if (bit == '0')
        p = p->left;
      else if (bit == '1')
        p = p->right;
      else
        cerr << "Illegal bit: " << bit << " -- ignored\n";
    }
    cout << "--" << p->data << endl;
  }
}
//--- Definition of printTree()
void Huffman::printTree(ostream & out, Huffman::BinNodePointer root,
                        int indent)
{
  if (root != 0)
  {
    printTree(out, root->right, indent + 8);
    out << setw(indent) << " " << root->data << endl;
    printTree(out, root->left, indent + 8);
  }
}
```

//----- Driver Program -----

```
#include <iostream>
#include <fstream>
using namespace std;

#include "Huffman.h"

int main()
{
  char filename[32];
  cout << "Enter name of code file: ";
  cin >> filename;
  ifstream codestream(filename);
  if (!codestream.is_open())
  {
    cout << "Cannot open code file.\n";
    exit(1);
  }

  Huffman h;
  h.buildDecodingTree(codestream);
  cout << "Here is the Huffman decoding tree:\n";
  h.displayDecodingTree(cout);
  cout << endl;

  cout << "\nName of message file: ";
  cin >> filename;
  ifstream message(filename);
  if (!message.is_open())
  {
    cout << "Cannot open message file.\n";
    exit(1);
  }
  h.decode(message);
}
```

Listing of CodeFile:

```
A 1101
B 001101
C 01100
D 0010
E 101
F 111100
G 001110
H 0100
I 1000
J 11111100
K 11111101
L 01111
M 01101
N 1100
O 1110
P 111101
Q 111111100
R 1001
S 0101
T 000
U 01110
V 001100
W 001111
X 111111101
Y 111110
Z 11111111
```

Listing of MessageFile:

```
0000100101100110100010
01100111011010000101
110110011010110011100110110001100001110
```

Sample Execution:

```
Enter name of code file: CodeFile
Here is the Huffman decoding tree:
                                                                Z
                                                        *
                                                                        X
                                                                *
                                                                        Q
                                                *
                                                                K
                                                        *
                                                                J
                                        *
                                                Y
                                *
                                                P
                                        *
                                                F
                        *
                                O
                *
                                A
                        *
                                N
        *
                        E
                *
                                R
                        *
                                I
 *
                                        L
                                *
                                        U
                        *
                                        M
                                *
                                        C
                *
                                S
                        *
                                H
        *
                                                W
                                        *
                                                G
                                *
                                                B
                                        *
                                                V
                        *
                                D
                *
                        T
```

```
Name of message file: MessageFile
000--T
0100--H
101--E
1001--R
101--E
0010--D
01100--C
1110--O
1101--A
000--T
0101--S
1101--A
1001--R
101--E
01100--C
1110--O
01101--M
1000--I
1100--N
001110--G
```