

Specifications for MorseTree Class

Note: this is **not** a template class, though it does use the Tnode structure template from the textbook.

Attributes

There is only one attribute of this class, a single **Tnode<char>** pointer variable that points to the root of the constructed codetree.

Operations

Explicit-value constructor - builds the codetree from an input stream connected to a codefile.

- Analysis

Objects	Type	Kind	Movement	Name
input stream connected to codefile	istream	variable	received & passed back	codeFile

The codefile will consist of an integer n , followed by n lines of the form *letter code* where *code* is the Morse encoding for *letter*. The constructor is to read the codefile and build the codetree. For example, the codefile for the codetree above would be:

```

7
a .-
e .
g --.
m --
n -.
r .-.
t -

```

The basic algorithm is to read each letter (as a character) and its code (as a string) and use the code to traverse the tree (starting with the root) as one would do when decoding. If an interior node does not exist, create it and continue. When the end of the code is reached, you are at the node that stores the letter.

Copy constructor - creates a copy of the original MorseTree using the CopyTree function

- Analysis

Objects	Type	Kind	Movement	Name
original tree	MorseTree	variable	received	original

CopyTree - **private** recursive helper function to copy a tree (page 529 of textbook) that returns the root of the copy.

- Analysis

Objects	Type	Kind	Movement	Name
root of tree to copy	Tnode<char>*	variable	received	root
root of copy	Tnode<char>*	variable	returned	copyRoot

Destructor - deletes MorseTree nodes using DeleteTree function

- Analysis - no objects

DeleteTree - **private** recursive helper function to delete tree nodes (page 530 of textbook)

- Analysis

Objects	Type	Kind	Movement	Name
root of tree to delete	Tnode<char>*	variable	received	root

operator= - assignment operator, make this MorseTree the same as the original MorseTree using both DeleteTree and CopyTree, returns a reference to this object

- Analysis

Objects	Type	Kind	Movement	Name
original tree	MorseTree	variable	received	original
this tree	MorseTree &	variable	returned	*this

WriteTree - writes out the codetree “sideways” to an output stream using WriteTreeHelper function

- Analysis

Objects	Type	Kind	Movement	Name
output stream	ostream	variable	received & passed back	out

WriteTreeHelper - **private** recursive helper function to write out tree to an output stream using RNL traversal (right, visit node, left). Each value should be written on a separate line with indent number of spaces before it. In order for the tree to "look right" on the screen, an extra newline should be output after each step when the indent number is 16 or less.

- Analysis

Objects	Type	Kind	Movement	Name
output stream	ostream	variable	received & passed back	out
root of tree to write	Tnode<char>*	variable	received	root
number of spaces to indent	int	variable	received	indent

By using a RNL traversal with indenting, the tree will be written “sideways” with the root on the left side of the output. Each recursive call to WriteTreeHelper should increase the indenting by 8 spaces. For example, the codetree above would print out as:

m
g
t
n
*
a
r
e

Encode - encodes message from an input stream, writing results to an output stream, using CharToCode

- Analysis

Objects	Type	Kind	Movement	Name
input stream connected to message file	istream	variable	received & passed back	messageFile
output stream	ostream	variable	received & passed back	out

The basic algorithm for this function is to repeatedly read a character (**including spaces and newlines**) from the input stream, encode it using CharToCode, then write the encoding to the output stream. You may assume that the message file contains only the characters in the codetree and ignore any that do not encode properly. However, you may want to display an error message to **cerr** if no encoding is found for a read character stating that the character is being ignored.

CharToCode - **private** recursive helper function to find an encoding from the codetree.

- Analysis

Objects	Type	Kind	Movement	Name
character to encode	char	variable	received	ch
node of tree being traversed	Tnode<char>*	variable	received	t
path constructed so far	string	variable	received	pathSoFar
encoding for character	string	variable	returned	code

The basic algorithm for this function is to build a path (in pathSoFar) as the function traverses the codetree. For each node traversed, if the node value is the character being encoded, return the path. Otherwise, recursively try the paths going down the left and right subtrees by appending '.' or '-' respectively to pathSoFar. If the traversal reaches an empty tree, the character to encode is not in this path from the root, and return the empty string.

Decode - decodes message from an input stream, writing results to an output stream

- Analysis

Objects	Type	Kind	Movement	Name
input stream connected to encoded message file	istream	variable	received & passed back	messageFile
output stream	ostream	variable	received & passed back	out

The basic algorithm for this function is to read in a character code and use it to traverse the codetree starting at the root. When the end of the character code is reached, write out the letter in the node, then start over again with the next character code. In addition to character codes, there may be "codes" of '|' and '||' representing a space and a newline, respectively, that are to be decoded to those characters. You may assume that the encoded message file contains only valid input, and ignore any other inputs. However, you may want to display an error message to **cerr** if invalid characters or character codes are found in the input stream stating they are being ignored.

Assignment

(20 points) Write the implementation of the MorseTree class. The Tnode struct definition and the MorseTree class definition should be put in header file **morse.h** with suitable compilation guards (the Tnode struct first, then the MorseTree class). The implementations of the MorseTree class functions should be put in source file **morse.cpp**. This project **must be implemented using a binary tree with linked nodes**. Projects that do not so will be returned for resubmission with late penalties.

(10 points) Write a (main) program **coder.cpp** that has one command-line argument, the name of a codefile. The program should construct a MorseTree object using the given codefile, then display the constructed codetree to the screen. Then the program should enter a menu driven loop that asks the user if they want to encode, decode or quit. For encode and decode, the program should ask for an input file name and an output filename, then do the appropriate action reading in from the input file and sending the output to the output file.

Implementation note: If you want to reuse an input stream in your coder program (i.e., you close one file and use the open member function to attach a different file to the input stream object), please note that the stream must be reset using the clear member function after it has been closed (and before the next open). I.e.,

```
infile.close();
infile.clear();
```

You can also explicitly construct a new input stream object every time through the coder loop by putting the declaration for infile inside the loop body. E.g.,

```
while (choice != 'Q')
{
    :
    cin >> inputFileNames;
    ifstream infile (inputFileNames.c_str());
    :
```

Two codefiles will be given in **/home/hwang/cs215/project7** on cserver. The file **samplecodefile.dat** will be the partial Morse code used in the examples above, thus suitable for preliminary testing of your class. The file **codefile.dat** will contain the full Morse code for the lowercase Latin alphabet. You will have to write your own sample input files. Note that when Encode and Decode are working, you should be able to send the output of Encode into Decode and get back your original unencoded message (and vice versa). You can use the diff command to see if two files are the same.

You must submit a makefile named **Makefile.project7** that creates executable name **coder** for your project. Submissions without working makefiles will be assessed up to a 3-point penalty as indicated in the syllabus. It should conform to the examples given in the handout Very Basic make and demonstrated in class.

REMINDER: Your project must compile for it to be graded. Submissions that do not compile will be returned for resubmission and assessed a late penalty. Submissions that do not substantially work also will be returned for resubmission and assessed a late penalty.

Follow the guidelines in the [C++ Programming Style Guideline](#) handout. As stated in the syllabus, part of the grade on a programming project depends on how well you adhere to the guidelines. The grader will look at your code listing and grade it according to the guidelines. s.

What to submit

Electronically submit a tarfile containing **Makefile.project7**, **morse.h**, **morse.cpp**, and **coder.cpp** as explained in the handout [Submission Instructions for CS 215](#). Turn in a hardcopy of your **Makefile.project7**, **morse.h**, **morse.cpp**, and **coder.cpp**. The submission system will compile and test the MorseTree implementation against a test driver program. It will compile the project main program, but will not test it.