

Lab6MorseCodeLab

Generated by Doxygen 1.8.8

Thu Nov 16 2017 15:56:08

Contents

1	Specification	2
2	Analysis	3
3	Design	4
4	Testcase1	5
5	Testcase2	7
6	Testcase3	8
7	Testcase4	10
8	Testcase5	11
9	Testcase6	13
10	Testcase7	14
11	Testcase8	15
12	Testcase9	17

13 Testcase10	19
14 Testcase11	20
15 Testcase12	21
16 diagram	22
17 Class Index	23
17.1 Class List	23
18 File Index	24
18.1 File List	24
19 Class Documentation	25
19.1 Morsecode Struct Reference	25
19.1.1 Member Enumeration Documentation	25
19.1.2 Member Data Documentation	25
19.2 Telegraph Class Reference	26
19.2.1 Member Function Documentation	26
19.3 TNODE Struct Reference	30
19.3.1 Constructor & Destructor Documentation	31
19.3.2 Member Data Documentation	31

19.4	TREENODE Struct Reference	32
19.4.1	Member Data Documentation	32
20	File Documentation	34
20.1	lab.dox File Reference	34
20.2	main.cpp File Reference	34
20.2.1	Function Documentation	34
20.3	morse.cpp File Reference	35
20.4	morse.dox File Reference	35
20.5	morse.h File Reference	36
20.6	morseCode.h File Reference	36
20.6.1	Variable Documentation	36
20.7	tree.h File Reference	37
20.7.1	Function Documentation	37
Index		39

1 Specification

In this project, we will simulate a telegraph station that can encode messages from text to Morse code and decode Morse code back to text. The encoding is accomplished by looking up a symbol in a table and copying its Morse code into the output string. The decoding is implemented with the help of a binary "decoding" tree. Morse code for each letter represents a path from the root of the tree to some node: a "dot" means go left, and a "dash" means go right. The node at the end of the path contains the symbol corresponding to the code.

2 Analysis

Inputs: Morse code The user entering letters The user entering morse code User selction of encode and decode

Outputs: Morse code to letters letters to morse code

3 Design

The design is to have html to display our input and output and have the C++ do all of the work. we have a total of 5 functions. `openTree.cpp`: Builds the tree needed for the morse code translations. Sets the root node to null and builds from the tree from that function builds from the Morse table structure from the private class. `encode.cpp`: Converts text into morse code. It finds each symbol in the table (skips it if not found). `decode.cpp`: Converts morse code into text. It follows the path determined from the string parameter and once it reaches the end, it stores the letter in the string named `text` which is returned at the end of the functions. `closeTree.cpp`: Destroys the tree that was built with the `buildTree` function. Uses recursion to destroy each node of the tree.

4 Testcase1

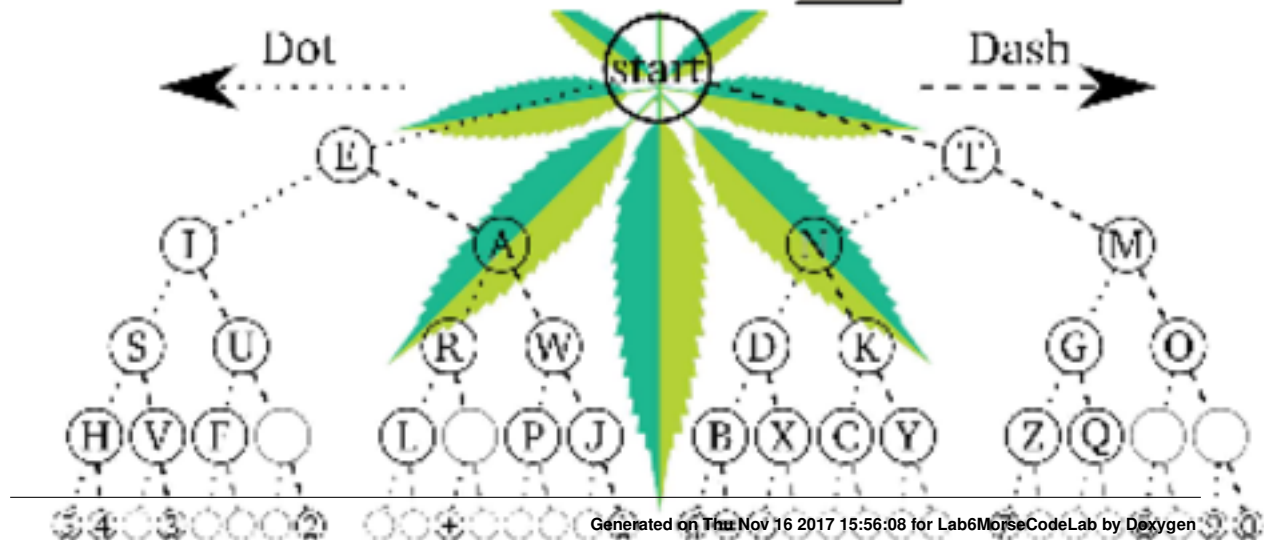
This is our home html page. Let's try to encode the word "trees". After we hit the go button, we should be redirected to a new html page with "trees" translated to morse code.

 <http://localhost/cs124/lab6/morse.html>

MorseCode Lab

Decode:

Encode:



5 Testcase2

Nice it worked. Now let's copy the output from our morse code and validate it using our decode function.



6 Testcase3

After copying pasting into our decode text field. We should get an abc translation when we click the go button.

← → ↻ <http://localhost/cs124/lab6/morse.html>

MorseCode Lab

Decode:

Encode:

Generated on Thu Nov 16 2017 15:56:08 for Lab6MorseCodeLab by Doxygen

7 Testcase4

Great! Our encode function works and our decode function work since our inputs match the outputs.



8 Testcase5

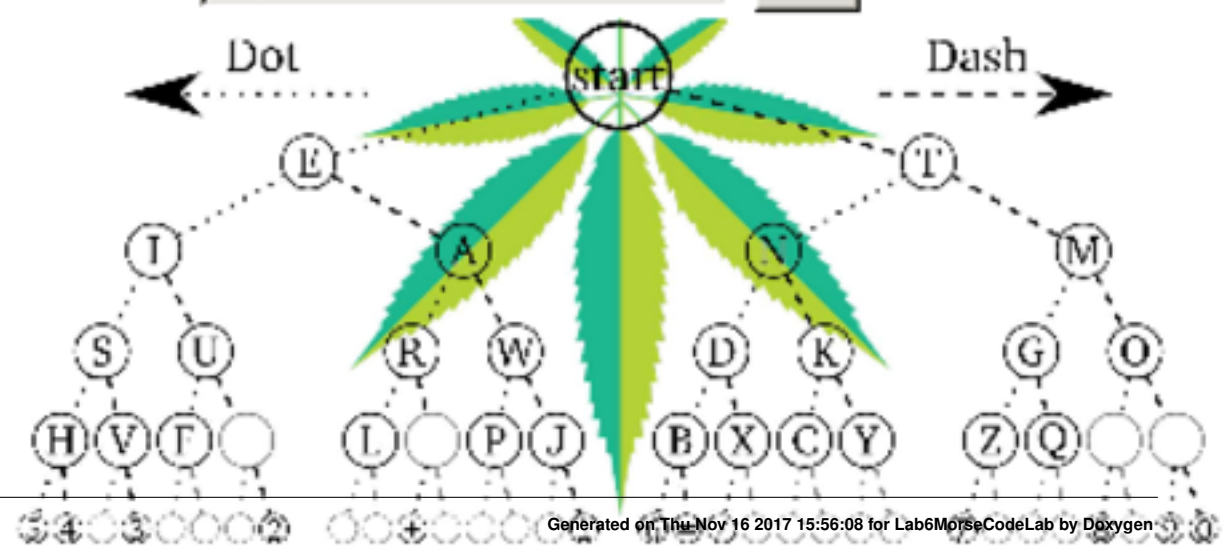
Now let's try another test. Let's encode "Topham". We should receive the morsecode translation of it.

← → ↻ <http://localhost/cs124/lab6/morse.html>

MorseCode Lab

Decode:

Encode:



The diagram illustrates the Morse code tree structure. At the top is a 'start' node. A dashed arrow labeled 'Dot' points left, and a dashed arrow labeled 'Dash' points right. The tree branches out from the 'start' node into three main paths: 'Dot', 'Dash', and a central path. The 'Dot' path leads to 'E', which branches into 'I' and 'A'. 'I' further branches into 'S' and 'U', which then lead to 'H', 'V', 'F', and an empty node. 'A' branches into 'R' and 'W', which lead to 'L', an empty node, 'P', and 'J'. The central path leads to 'N', which branches into 'D' and 'K', leading to 'B', 'X', 'C', and 'Y'. The 'Dash' path leads to 'T', which branches into 'M' and an empty node, which then lead to 'G', 'O', 'Z', 'Q', and two empty nodes. At the bottom, there are two rows of small circles representing the Morse code sequence for 'Topham'.

Generated on Thu Nov 16 2017 15:56:08 for Lab6MorseCodeLab by Doxygen

9 Testcase6

Awesome! let's copy and paste this morsecode in to our decode function to validate it.



10 Testcase7



We should see "TOPHAM" when we submit it.

11 Testcase8

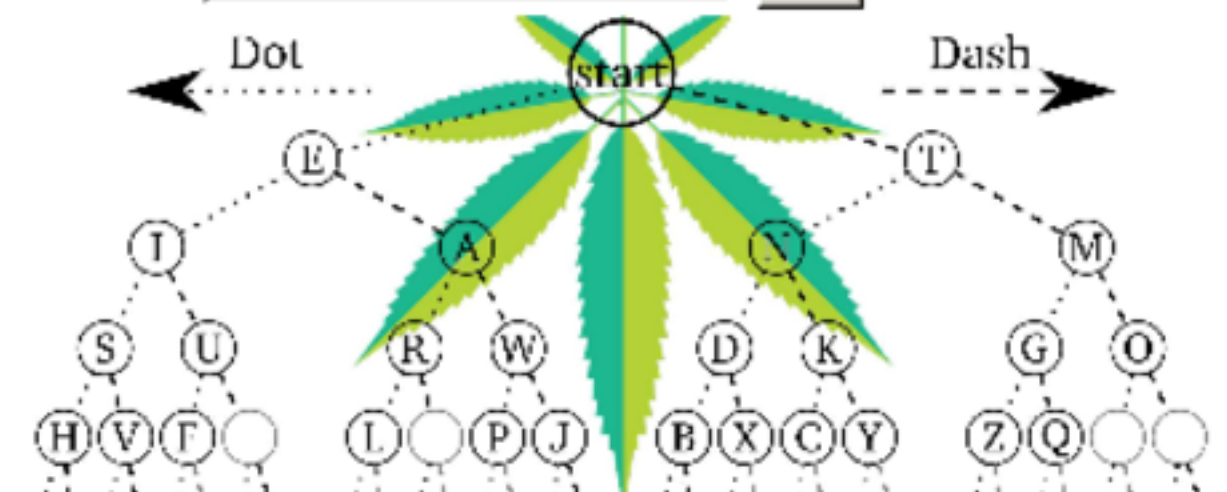
Beautiful. Let's try one more test case to make sure it works.

← → ↻ <http://localhost/cs124/lab6/morse.html>

MorseCode Lab

Decode:

Encode:



The diagram illustrates the Morse code tree structure. The root node is labeled "start". From "start", a dotted line (representing a dot) leads to the letter 'E', and a dashed line (representing a dash) leads to the letter 'T'. The tree branches out from these letters to represent all 26 letters of the alphabet. The letters are arranged in a circular pattern around the "start" node, with each letter's path highlighted by a green leaf. The letters are: E, T, M, O, Q, Z, G, N, K, Y, C, X, B, D, W, J, P, L, R, A, U, F, V, H, S, I. The letters are arranged in a circular pattern around the "start" node, with each letter's path highlighted by a green leaf. The letters are: E, T, M, O, Q, Z, G, N, K, Y, C, X, B, D, W, J, P, L, R, A, U, F, V, H, S, I.

Dot → ←

Dash → →

start

E T M O Q Z G N K Y C X B D W J P L R A U F V H S I


Generated on Thu Nov 16 2017 15:56:08 for Lab6MorseCodeLab by Doxygen

12 Testcase9

Here we try our favorite word "banana".




13 Testcase10



MorseCode Lab

Decode:

Encode:



Generated on Thu Nov 16 2017 15:56:08 for Lab6MorseCodeLab by Doxygen

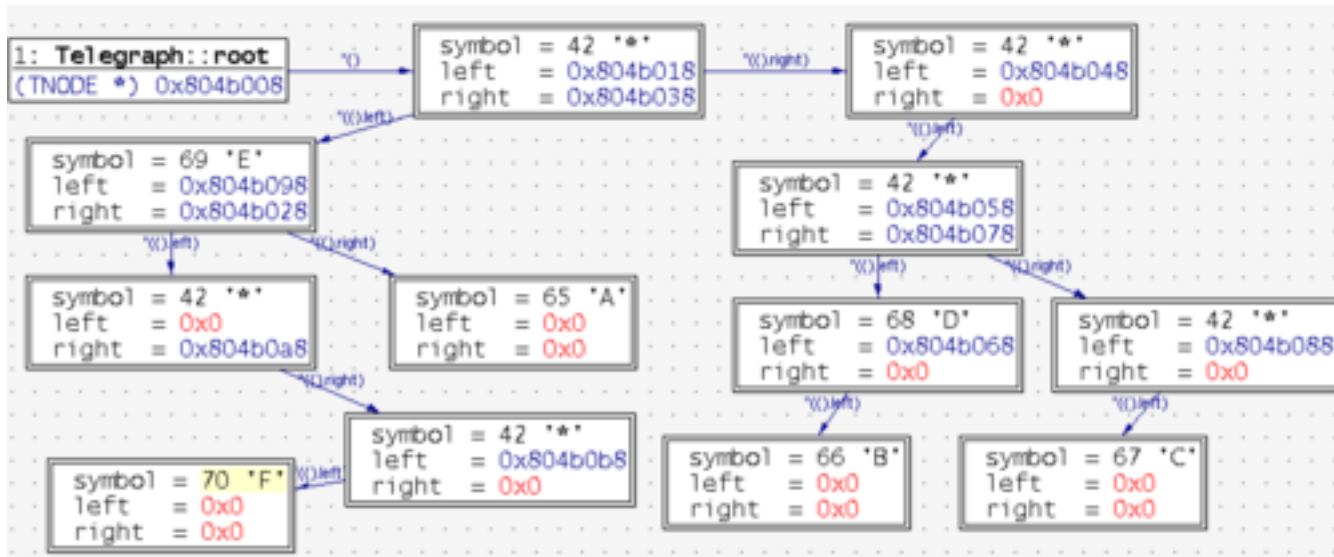
14 Testcase11

Great test case 3 works too!

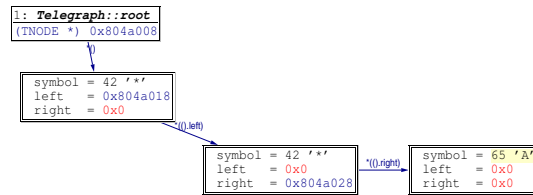


15 Testcase12

DDD Tree Example



16 diagram



17 Class Index

17.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Morsecode	25
Telegraph	26
TNODE	30
TREENODE	32

18 File Index

18.1 File List

Here is a list of all files with brief descriptions:

main.cpp	34
morse.cpp	35
morse.h	36
morseCode.h	36
tree.h	37

19 Class Documentation

19.1 Morsecode Struct Reference

```
#include <morse.h>
```

Public Types

- enum { `N` = 7 }

Public Attributes

- char `symbol`
- char `code` [`N`]

19.1.1 Member Enumeration Documentation

19.1.1.1 anonymous enum

Enumerator

`N`

```
7 {N=7};
```

19.1.2 Member Data Documentation

19.1.2.1 char Morsecode::code[N]

19.1.2.2 char Morsecode::symbol

The documentation for this struct was generated from the following file:

- [morse.h](#)

19.2 Telegraph Class Reference

```
#include <morse.h>
```

Public Member Functions

- void [Encode](#) (char text[], char morse[])
- void [Decode](#) (char morse[], char text[])

Static Public Member Functions

- static void [open](#) ()
open creates the morse code table
- static void [close](#) ()
- static void [destroyTree](#) ()

19.2.1 Member Function Documentation

19.2.1.1 void Telegraph::close() [static]

```
68 {  
69     destroyTree(root);  
70     root = 0;  
71 }
```

19.2.1.2 void Telegraph::Decode (char *morse*[], char *text*[])

```
101 {  
102     char *dd;  
103     TNODE *node;  
104     node = root;  
105     //char *t;  
106     // For each char in the encoded message (can be  
107     // a dot, a dash, or a space):  
108     for (dd = morse; *dd; dd++) {  
109         if(*dd == '.')  
110             node = node->left;  
111         else if(*dd == '-')  
112             node = node->right;  
113         else {  
114             *text++ = node->symbol;  
115             node = root;  
116         }  
117     }  
118     *text = '\0';  
119 }
```

19.2.1.3 static void Telegraph::destroyTree () [static]

19.2.1.4 void Telegraph::Encode (char text[], char morse[])

```

75 {
76     int i;
77     char c, *t, *dd; // t points to text;
78     // dd points to a string of dots and dashes.
79     for (t = text; *t; t++) {
80         c = toupper(*t);
81         // If space, add a space to the morse string:
82         if (c == ' ') {
83             *morse++ = ' ';
84             continue;
85         }
86         // Find this symbol in the MORSECODE table;
87         // skip this symbol if not found:
88         for (i = 0; table[i].symbol; i++)
89             if (table[i].symbol == c) break;
90         if (!table[i].symbol) continue;
91         // Copy its code into the morse string:
92         dd = table[i].code;
93         while (*dd) *morse++ = *dd++;
94         // Add one space to separate letters:
95         *morse++ = ' ';
96     }
97     *morse = '\0';
98 }

```

19.2.1.5 void Telegraph::open () [static]

open creates the morse code table

```

22 {

```

```
23     char* dd;
24     Telegraph::root = new TNODE;
25     TNODE* node; TNODE* nextnode;
26     for (int i = 0 ; i < N; i++)
27     {
28         node = root;
29         for(dd = table[i].code; *dd ; dd++)
30         { // loops through each char of code
31             if(*dd == '.')
32             {
33                 nextnode = node->left;
34                 if(not nextnode)
35                 {
36                     nextnode = new TNODE;
37                     node->left = nextnode;
38                 }
39             }
40             else if(*dd == '-')
41             {
42                 nextnode = node->right;
43                 if(not nextnode)
44                 {
45                     nextnode = new TNODE;
46                     node->right = nextnode;
47                 }
48             }
49             else std::cerr << "unknown morse code" << std::endl;
50             node = nextnode;
51         } // not dash, not dot, therefore
52         // it must be null, so assign symbol
53         node->symbol = table[i].symbol;
54     }
```



```
55 }
```

The documentation for this class was generated from the following files:

- [morse.h](#)
- [morse.cpp](#)

19.3 TNODE Struct Reference

```
#include <morse.h>
```

Collaboration diagram for TNODE:



Public Member Functions

- [TNODE](#) ()

Public Attributes

- char [symbol](#)
- [TNODE](#) * [left](#)
- [TNODE](#) * [right](#)

19.3.1 Constructor & Destructor Documentation

19.3.1.1 TNODE::TNODE() [inline]

```
17         { // Constructor
18             symbol = '*';
19             left = 0;
20             right = 0;
21         }
```

19.3.2 Member Data Documentation

19.3.2.1 TNODE* TNODE::left

19.3.2.2 TNODE* TNODE::right

19.3.2.3 char TNODE::symbol

The documentation for this struct was generated from the following file:

- [morse.h](#)

19.4 TREENODE Struct Reference

```
#include <tree.h>
```

Collaboration diagram for TREENODE:



Public Attributes

- `SOMETYPE` [data](#)
- [TREENODE](#) * [left](#)
- [TREENODE](#) * [right](#)

19.4.1 Member Data Documentation

19.4.1.1 `SOMETYPE TREENODE::data`

19.4.1.2 `TREENODE* TREENODE::left`

19.4.1.3 TREENODE* TREENODE::right

The documentation for this struct was generated from the following file:

- [tree.h](#)

20 File Documentation

20.1 lab.dox File Reference

20.2 main.cpp File Reference

```
#include "morse.h"
```

Functions

- `int main()`

20.2.1 Function Documentation

20.2.1.1 `int main()`

```
5 {
6     Telegraph::open();
7     Telegraph t;
8     std::string str = getenv("QUERY_STRING");
9     std::string strOG = str; std::string strInput = str;
10    strOG.erase(1,30); // extract first char
11    strInput.erase(0,2); // extract the string from the text field
12    if ( strOG == "d") { // Split string where the +'s are instead of converting to spaces
13        for (int i = 0; i < strInput.size(); i++) // Convert all +'s to spaces
14            if(strInput[i] == '+')
15                strInput[i] = ' ';
```

```
16         strInput += " ..";
17         //std::cout << strInput << std::endl;
18         char text[strInput.length() + 1];
19         strcpy(text, strInput.c_str());
20         char eMorse[600];
21         t.Decode(text, eMorse);
22         std::cout << "<html><h2>ABC:</h2><h3>" << eMorse << "</h3></html>" << std::endl;
23     }
24     else if ( strOG == "e") {
25         char morse[strInput.length() + 1];
26         strcpy(morse, strInput.c_str());
27         char dText[600];
28         t.Encode(morse, dText);
29         std::cout << "<html><h2>Morsecode:</h2><h3>" << dText << "</h3></html>" << std::endl;
30     }
31     Telegraph::close();
32 }
```

20.3 morse.cpp File Reference

```
#include "morse.h"
```

20.4 morse.dox File Reference


```

{ 'E', "." },      { 'F', ".-." },      { 'G', "--." },      { 'H', "..."},
{ 'I', ".." },     { 'J', ".---" },     { 'K', "-.-" },      { 'L', ".-.."},
{ 'M', "--" },     { 'N', "-." },      { 'O', "---" },      { 'P', "--." },
{ 'Q', "--.-" },   { 'R', ".-." },     { 'S', "..."},      { 'T', "-"},
{ 'U', ".-.-" },   { 'V', "...-"},     { 'W', "-.-"},       { 'X', "-.-.-"},
{ 'Y', "-.-.-"},   { 'Z', "---.."},
{ '0', "-----" }, { '1', ".-----" }, { '2', ".-.-.-"},    { '3', "...-"},
{ '4', "....-" },   { '5', "....."},    { '6', "-....."},    { '7', "--...."},
{ '8', "---.."},   { '9', "-----"},
{ '.', ".-.-.-"}, { ',', "--.-.-"}, { '?', ".-.-.-"},
{ '\0', "" }
}

```

20.7 tree.h File Reference

Classes

- struct [TREENODE](#)

Functions

- void [Destroy](#) ([TREENODE](#) *root)
- int [Compare](#) (const SOMETYPE data1, const SOMETYPE data2)
- void [Copy](#) (const SOMETYPE data1, SOMETYPE data2)

20.7.1 Function Documentation

20.7.1.1 int Compare (const SOMETYPE *data1*, const SOMETYPE *data2*)

20.7.1.2 void Copy (const SOMETYPE *data1*, SOMETYPE *data2*)

20.7.1.3 void Destroy (**TREENODE** * *root*)

Index

close

Telegraph, [26](#)

code

Morsecode, [25](#)

Decode

Telegraph, [27](#)

Encode

Telegraph, [27](#)

Morsecode, [25](#)

code, [25](#)

N, [25](#)

symbol, [26](#)

N

Morsecode, [25](#)

open

Telegraph, [28](#)

symbol

Morsecode, [26](#)

Telegraph, [26](#)

close, [26](#)

Decode, [27](#)

Encode, [27](#)

open, [28](#)