Morse code translator

**I. Problem:**

Design a simple program to translate the inputted morse code to a readable message. Use the International Morse Code table, the procedural signals are excluded.

Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes, or dits and dahs. International Morse Code encodes the 26 Latin letters a through z, one non-Latin letter, the Arabic numerals, and a small set of punctuation and procedural signals (prosigns). There is no distinction between upper and lower case letters. (Source: <https://en.wikipedia.org/wiki/Morse_code> ).



There are many ways to store the decoded characters for each set of morse signal, among them, there is binary tree. Why ?

- Each character is encoded into a sequence of dots and dashes. This means each individual element of the encoded character only have two statuses: dot or dash. Similarly, the binary tree only has two possibilities in branching: left or right.

- The binary tree is only used as a storage of characters, and the number of these characters is fixed, or if changed in the future, it is still easy to re-organize the tree. Another benefit of this is that the tree as well as its height is constant as the program executing, no inserting, deleting, editing or balancing function needed.

The international morse code table can be organized as a binary tree if we use dots and dashes as the branch between nodes. The combination of branches on the path from root to the node holding a character is the morse encoded version of it. The tree look like this :



The drawback is this method does not prove effective in the reverse way, which mean translate from character to encoded morse code. The characters were not added to the tree by their value. Using the morse code to track a character is easy, but to track the path from begin to a character would require some extra stored information in each node or traversing the whole tree, which is not effective.

**II. Implementation:**

**1. Input:**

First, we must have an input file that stores the value for the tree. I want the file to have the pre-order form, the root is read first, then left child and right child. Each line in the file holds a single character which is the value that will be stored in the tree.

Lines holding values of the leaf nodes have an extra “end” follow the character to mark the end for the recursive call. The algorithm detects a leaf node when a line has more than one character, so “end” can be replaced with anything else, even a space.

The \* character in the file mark value of an empty node in the tree, as \* is nowhere to be found in the morse code table. If this change in the future, it can be fix easily.

The form of the stored input file:



Now we go through form of the input morse code.

I use ‘.’ for dots and ‘-’ for dashes. For example, S is three dots “...” and O is three dashes “---”.

Morse code can be written in different ways. The two most common are shown below:

- Individual letters are separated by a space while individual words are separated by a forward slash (/).

- Individual letters are separated by a forward slash (/) and individual words are separated by a DOUBLE forward slash (//). Sentences in this format are separated by Four slashes (////).

(Source: <https://www.geocaching.com/geocache/GC2T9M6_morse-code?guid=1490daee-7c43-4761-b8d7-be7db2d7b2d5> )

For my program, I use the first way.

Example for an input sentence : - .... .. ... / .. ... / .- -. / . -..- .- -- .--. .-.. . / --- ..-. / .- / -- --- .-. ... . / -.-. --- -.. . / .. -. .--. ..- - / .-.-.-



Note that for now, the input must be on a single line.

**2. Program:**

When run, the program bring up the main menu. Currently there are only two option to either use the program or exit.



To use the translator, user must first choose an input method :



Choosing input from file, the user will be asked for file’s name, then the program will read input from that file. Otherway, the user will be guide to enter input form keyboard.



Afterward, the user is asked for an output method:



Choosing output to file, user will need to enter the output file’s name. Elsewise, the translated message will be display on terminal.



As the user press any key after this, the main menu will come back. This repeat until user choose to exit.

**III. Source code:**

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#include <conio.h>

using namespace std;

struct node

{

char value;

node\* dot;

node\* dash;

};

node\* createNode()

{

node\* newnode = new node;

newnode->value = '\0';

newnode->dot = NULL; // left child

newnode->dash = NULL; // right child

return newnode;

}

void readTree(node\*& curr, ifstream& ifs)

{

// read the input for the tree

if (ifs.eof()) return;

string s;

getline(ifs, s);

curr = createNode();

// empty nodes keep value '\0'

if(s[0] != '\*') curr->value = s[0];

// end of a path

if(s.length() > 1) return;

// read left child

readTree(curr->dot,ifs);

//read right child

readTree(curr->dash, ifs);

}

char singleTranslate(node\* root, string code)

{

// translate a single character

// if it is a slash, return a space character

if (code == "/") return ' ';

int index = 0;

node\* temp = root;

while (index < code.length())

{

// if reach NULL node, break

if (temp == NULL) break;

// if any character in the source morse code is not dot or dash, break

if ((code[index] != '.') && (code[index] != '-'))

{

temp = NULL;

break;

}

// if the character is dot, go to dot subtree and vice versa

if (code[index] == '.') temp = temp->dot;

else temp = temp->dash;

index++;

}

// if the input is violated, return NULL character

if (temp == NULL) return '\0';

return temp->value;

}

string messageTranslate(node\* root, string code)

{

// translate 1 line of morse code

stringstream ss(code);

string character, message = "";

getline(ss, character, ' ');

do

{

char translatedChar = singleTranslate(root, character);

if (translatedChar == '\0') message += "<unable to translate>";

else message.push\_back(translatedChar);

getline(ss, character,' ');

} while (ss);

return message;

}

string terminal\_input()

{

system("CLS");

cin.ignore();

string code;

cout << "Enter morse code. Use \".\" for dots and \"-\" for dashes \nIndividual letters are seperated by a space (\" \").\nIndividual words are seperated by a forward slash(\" / \").\n";

getline(cin, code);

return code;

}

string file\_input()

{

system("CLS");

cin.ignore();

string filename;

cout << "Enter input file name: ";

getline(cin, filename);

ifstream ifs(filename);

if (ifs)

{

string code;

getline(ifs, code);

return code;

}

return "\0";

}

void terminal\_output(string mess)

{

cout << "Translated message: " << mess;

}

void file\_output(string mess)

{

string filename;

cout << "Enter output file name: ";

getline(cin, filename);

ofstream ofs(filename);

ofs << mess;

}

void menu()

{

node\* root = NULL;

ifstream ifs("input.txt");

readTree(root, ifs);

if (ifs)

{

while (1)

{

// choose to use or exit

char choice = '\0';

while ((choice != '2') && (choice != '1'))

{

system("CLS");

cout << "<< Morse code to message translator >>\n";

cout << "Choose your action: \n1. Use translator.\n2. Exit.\n";

cin >> choice;

}

if (choice == '2') break;

string code, mess;

// choose input method

choice = '\0';

while ((choice != '2') && (choice != '1'))

{

system("CLS");

cout << "Choose your input method:\n";

cout << "1. Terminal input.\n2. File input.\n";

cin >> choice;

}

// read input

if (choice == '1') code = terminal\_input();

else code = file\_input();

if (code == "\0")

{

// break if the input file is not found

cout << "Input code file not found!";

}

else

{

// translate input

mess = messageTranslate(root, code);

// choose output method

choice = '\0';

while ((choice != '2') && (choice != '1'))

{

cout << "Choose your output method:\n";

cout << "1. Terminal output.\n2. File output.\n";

cin >> choice;

}

if (choice == '1') terminal\_output(mess);

else file\_output(mess);

}

cout << "\n\nPress any key to continue ...";

\_getch();

}

}

// does not execute without the input stored file

else cout << "Input file is missing!";

}