CS 216 Wednesday Programming Assignment: Morse Deliverable 2.

At this pint you should have completed Deliverable 1 meaning that you have:

- MEMsg, defined and implemented in MEMsg.h and MEMsg.cc
    - string MEMsg::toEnglish(const string& morse) as at least a stub function
    - string MEMsg::toMorse(const string& english) as at least a stub function
    - a driver.cc that
        - reads from the keyboard, stopping on end of input (^d)
        - handles *all* IO
        - can distinguish an English message from a Morse message
            - Assuming Morse messages consist of: . (dot), - (dash), space (between codes representing letters) and # (between the last letter code of one message word and the first letter code of the next message word.)
    - a basic makefile
- Additionally You've at least thought about an appropriate data structure to support the translation between Morse codes and English characters

Your task is as follows:
Complete the program so that rather then replying "English" to a Morse input message the program provides the Morse translation, and instead of replying "Morse" to a Morse input message the program provides the English translation. If you did not compelet deliverable #1 see the end of this document.

Requirements for Deliverable 2:

1. Deliverable 2 must be an extension of deliverable 1. You cannot, for example, change the prototype of toEnglish(). The same 3 files must exist, though you could in principle create others. (There is no need for more files.)
2. Use space between Morse codes representing letters and # between words in a Morse message.
3. The English to Morse method is <u>string toMorse(const string& english)</u>, where english is the English message and the return value is the Morse string.
    a. english will have whitespace between words. In the Morse encoding of the English a single space will appear between Morse character codes, and a tab (#) will replace the whitespace between two English words. If there are English characters in the message we do not support then Morse should include this special word: …….. (8 dots in one word) like this (with > and error):

```
( X1 > 1 )→ -.--.     -..- .----     ........     .----     -.--.-
              (          X    1          >           1          )
```

4. The Morse to English method is <u>string toEnglish(const string& morse)</u>, where morse is the Morse encoding and the return value is the English representation.

a. morse will have space between character codes and #s between words. In the English method that results there is no space between characters but there a single space between words. (A period, question mark etc is considered part of the word that it follows for these purposes.) If you find a code word in a Morse message that is not known then the English output should include *<error> </error>* surrounding the bad code word in the resulting test. Like this (------- not being a Morse code word):

```
.- -… -.-.  ---------  -…  → ABC<error>--------</error>D
 A  B   C                D
```

5. MEMsg must encapsulate the data to make the translation *by using two* STL **map**s for the data structure that supports the translation. One map is from English char to Morse string, the other from Morse string to English char.)[i]

6. Is completed by deliverable 1

7. All I/O is in main. Not even error messages are sent from MEMsg methods. When a message is read the *only* output is the translation of the message. The translation should appear on the line immediately below the input.

8. main() should continue to work as described in deliverable 1.

9. You must provide a tcsh script called mytests (not mytests.csh, just mytests) that tests your code.

10. You must provide a makefile with a default target of runme that builds your code to an executable called runme and uses the –Wall and –g switches.

11. The assignment is due to be submitted via the portal on Wednesday 2/23 by 11:59:59 pm[ii].

Submission of Deliverable #2

By 11:59:59pm Wednesday 2/23 submit the following:

- All three of the required source files as well as any other source files
- The makefile and mytests scripts (requirements #9 and #10 above)
  - Any data files needed by your mytests scripts
- All files must be submitted in *yourlastname*MorseD2.zip
  - Wrong file name will be a letter grade off.
- The zip file must unzip to a directory (folder) called *yourlastName*MorseD2.
  - Unzipping the files directly, rather than a folder, will be 1.5 letter grades off.
  - Unzipping to a misnamed directory will be ½ letter grade off.

For those who had trouble with deliverable #1

Someone who submitted a program on time that failed to compile will receive 30/80 on deliverable #1. (This value was chosen to be a sever penalty, but "non-fatal" in the sense that if this happens to you once then it should preclude you obtaining an A in the course.) Therefore I will give those who failed to complete deliverable #1 until 10:45am Monday 1/21 to submit for 30/80, assuming the submission looks like a legitimate effort to me. As I leave for class on Monday 1/21 I will post a solution to deliverable #1. Anyone in the class is welcome to use my solution rather than their own as the basis for deliverable #2. It's not as good as having your own good solution, but it is better than not having anything to work from.

Endnotes

[i] A single map will not do. Consider the Enlish message ".—is Morse for A". Then '.' must be translated to .-.-.-, the Morse for '.'. However, Morse message ". .. . .. ---" requires '.' to be translated to 'E'. Both . and – must be treated differently when going from English to Morse and vice versa, so two maps are needed unless we want an awkward workaround. We do not want an awkward workaround.

[ii] This is a short time, but this is a tiny increment if deliverable #1 is complete. You really only have to finish the two methods and call them from main.