CS 216 Wednesday Programming Assignment.

This assignment is an initially small development project that will count in the Wednesday portion of the class. There may be further, related assignments, and those may count in the Monday-Friday portion or the Wednesday. This assignment itself may evolve a bit before it is due so pay attention. **Read this document before you jump in – you need to know what part is due 2/18**

Your task is as follows:

Write a class MEMsg that provides methods to

1. Convert a message represented by a std::string of English text to a string of . (dot) and – (dash) representing the Morse code for the characters in the message string. The result returned is also a string.
2. Convert a message represented as a string of dot and dash representing Morse code as a string of characters representing English text.

Requirements:

1. Use the International Morse code described at http://en.wikipedia.org/wiki/Morse_code. Look down the page to section labeled **Letters, numbers, punctuation**. Use that chart. (It is reproduced at the end of this document.)

2. Your initial solution should have these 3 source files. (More are OK):
   a. MEMsg.h: the header file defining the class MEMsg
   b. MEMsg.cc: implementing the non-inline methods of MEMsg etc.
   c. driver.cc: holding main(), built explicitly to test MEMsg.

3. The English to Morse method is string toMorse(const string& english), 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 (\t) 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 string toEnglish(const string& morse), where morse is the Morse encoding and the return value is the English representation.
   a. morse will have space between character codes and tabs 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. For full credit you must use an STL Container described here: http://www.cplusplus.com/reference/stl/
   Before the class MEMsg line in MEMsg.h you should have a (class header) comment that explains clearly and concisely why you chose the container you did. *I chose vector because I know it* (or because you do not know the others) is wholly inadequate and will earn you points off.

6. toEnglish and toMorse may both be implemented as static methods or both as instance methods. The containerthat holds the translation may be a static or an instance member.
   Before the class MEMsg line in MEMsg.h you should have a (class header) comment that explains clearly and concisely why you chose to use a static or an instance approach for the methods and the member. "I do not know about static methods/members" and "I am more comfortable with instance methods/members" are unacceptable explanations that will earn a deduction. If you do not know about static versus instance methods find out – you have resources. I am one such resource.

7. All I/O is in main. Not even error messages are sent from MEMsg methods.

8. main() should read single line messages from the keyboard and print their translation to the screen immediately. It should read until end of input (^D) is received. Please note messages will include whitespace. Your program should decide whether the message is English to be converted to Morse or Morse to be converted to English without asking the user to indicate which in any explicit way. (The message should tell you implicitly.)

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. Expect this assignment to be extended, possibly before the immediate due date, so code as if you will revisit the code.

12. **<u>The first deliverable is due to be submitted</u>** via the portal by 1pm Friday 2/18 as detailed below. I will announce the next step later.

Submission of Deliverable #1

By 1pm Friday February 18 submit the following:

- All three of the required source files (satisfy requirement #2)
  - Any other source files needed to build your program
- The makefile and mytests scripts (requirements #9 and #10 above)
  - Any data files needed by your mytests scripts

Requirements for Deliverable #1

1. Standard Header comment and class header comment in MEMsg.h
2. Your code must:
   a. Read input from the user as described in requirement #8 above
   b. Report to the screen that the message input was Morse or English by outputting *Morse* or *English* on a line by itself
   c. Include the explanation of your decisions as described in requirements #5 and #6
   d. The MEMsg class must have at least a stub implementation of the required methods – they have to be declared and defined, they do not have to work.
   e. Confine all I/O to main (as requirement #7)

Short Version:

Translate between Morse Code and English, using spaces to separate English and space and tab to separate Morse. There is a required class, two required translation methods, a data requirement and a container requirement, and so on. No translation has to be successful.

The Morse Code we will use

Letters, numbers, punctuation

Character	Code	Character	Code	Character	Code	Character	Code	Character	Code	Character	Code
A (info)	· —	J (info)	· — — —	S (info)	· · ·	1 (info)	· — — — —	Period [.]	· — · — · —	Colon [:]	— — — · · ·
B (info)	— · · ·	K (info)	— · —	T (info)	—	2 (info)	· · — — —	Comma [,]	— — · · — —	Semicolon [;]	— · — · — ·
C (info)	— · — ·	L (info)	· — · ·	U (info)	· · —	3 (info)	· · · — —	Question mark [?]	· · — — · ·	Double dash [=]	— · · · —
D (info)	— · ·	M (info)	— —	V (info)	· · · —	4 (info)	· · · · —	Apostrophe [']	· — — — — ·	Plus [+]	· — · — ·
E (info)	·	N (info)	— ·	W (info)	· — —	5 (info)	· · · · ·	Exclamation mark [!]	— · — · — —	Hyphen, Minus [-]	— · · · · —
F (info)	· · — ·	O (info)	— — —	X (info)	— · · —	6 (info)	— · · · ·	Slash [/], Fraction bar	— · · — ·	Underscore [_]	· · — — · —
G (info)	— — ·	P (info)	· — — ·	Y (info)	— · — —	7 (info)	— — · · ·	Parenthesis open [(]	— · — — ·	Quotation mark ["]	· — · · — ·
H (info)	· · · ·	Q (info)	— — · —	Z (info)	— — · ·	8 (info)	— — — · ·	Parenthesis closed [)]	— · — — · —	Dollar sign [$]	· · · — · · —
I (info)	· ·	R (info)	· — ·	0 (info)	— — — — —	9 (info)	— — — — ·	Ampersand [&], Wait	· — · · ·	At sign [@]	· — — · — ·