

# CS1112 Spring 2011 Project 6 Part 1    due Thursday 5/5 at 11pm

(Part 2 will appear in a separate document. Both parts have the same submission deadline.)

You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, “you” below refers to “your group.” You may discuss background issues and general strategies with others, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not ok for you to see or hear another student’s code and it is certainly not ok to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on your own, please seek help from the course staff.

## Objectives

Completing this project will solidify your understanding of acoustic data generation and manipulation (Part 1) and recursion (Part 2). You will also practice problem decomposition and testing.

## Listen! I said . . .    --- . . .

That’s “SOS” in Morse Code. You will write a program to turn a string (1-d array of characters) into Morse Code. Download the data file `morseTable.txt` to see the conversion from character to Morse Code. Each character is represented as a sequence of dots (say dit) and/or dashes (say dah). We include only English capital letters, the digits 0 to 9, and three punctuation marks (period, comma, and question mark). Here are the first four lines of the data file:

```
A  .-
B  -...
C  -.-.
D  -..
```

You will convert a string into Morse Code both in print and in sound. In print, the string ‘A Dab’ is

```
.- / -... .- -...
```

Notice that words are separated by the slash character and the letters in a word are separated by a space character. Lower case letters are treated as upper case letters. If unrecognized characters appear in the original string, in Morse Code an exclamation mark is shown for each unrecognized character, e.g., the string ‘what 20 in the world#\$\$%??’ is

```
..- . . . . .- - / ..- - - - - / .. -. / - . . . . / .- - - - .- .- . - . ! ! ! ..- . .- . .
```

Notice that a space follows an exclamation mark; indeed a space follows every “Morse string” corresponding to a character. Unrecognized characters has no sound effect. Therefore the Morse Code *sound* for the two strings ‘what 20 in the world#\$\$%??’ and ‘what 20 in the world??’ is the same (including the durations of silence).

## Di-di-dit-dah-dah-dah-di-di-dit

That’s the sound of “SOS” as Morse Code. The actual sound doesn’t matter; the timing is key. We will call the duration of the sound corresponding to the dot (dit) one “dit unit.” Here are the timings:

dit (the dot)	1 dit unit
dah (the dash)	3 dit units
silence between dits, dahs within one Morse string for a character	1 dit unit
silence between characters in a word	3 dit units
silence between words	7 dit units

Any unrecognized character (shown as ‘!’ in print Morse Code) does not produce any sound (or silence). The speed of Morse Code transmission is measured in “words per minute,” i.e., how many times the word ‘PARIS’ can be transmitted per minute (an alternative standard measures how many times the word ‘CODEX’ can be transmitted in a minute). The word ‘PARIS’ takes 50 dit units to transmit (including the intra word silence). Therefore if the Morse Code speed is specified to be 20 words per minute, then each dit unit is  $60/(50 \times 20)$  seconds.

## Morse Code Generator

In this final project, you will do most of the problem decomposition yourself, unlike in previous projects. We specify here two functions that you should write as part of your solution and the format of the output to be produced, but you will do the detailed program planning yourself. Take the time to plan *before* writing code! Taking the time to plan first will end up saving you time on the project.

The main function is `genMorseCode`:

```
function [ditdah, y] = genMorseCode(englishStr, wpm, omega)
% Generate and play the Morse Code for the string in englishStr with a speed of wpm
% words-per-minute and the sound signal frequency omega.
% ditdah is the Morse Code in print, a 1-d array of characters.
% y is the Morse Code in sound signal data, a column vector.
% If englishStr has length greater than 20, write a plain text file morseCode.txt
% to store the string in ditdah.
```

You will implement a function `genSoundData` which you will call in `genMorseCode`:

```
function [dit, dah, ditSilence]= genSoundData(wpm,omega,Fs)
% Generate sound signal data for one dit, one dah, and one dit unit of silence based
% on wpm words-per-minute, frequency omega, and Fs samples/second of sound signals.
% omega is the frequency for the sinusoid sin(2*pi*omega*t), where t is time, for
% generating sound signals.
% dit, dah, and ditSilence are column vectors of sound signals. ditSilence contains zeros.
% Generate the sound data only, do not play the sound within this function.
```

## Some details and hints

- You must implement functions `genMorseCode` and `genSoundData` as specified. You may (and should) implement *subfunctions* in `genMorseCode` to decompose the problem. Use good programming style as you design your program.
- You can choose the number of samples per second in generating the sound signals. See *Insight* §13.2 `MakeShowPlay.m` for example.
- In `genMorseCode` build the entire sound vector of the Morse Code corresponding to `englishStr` before playing the sound.
- The output file `morseCode.txt` stores the print version of the Morse Code corresponding to `englishStr`. Each line of the file has no more than 75 characters including spaces. Break a line only between words, i.e., break a line after ‘/’ (slash followed by a space). (This means that all but the final line in the file ends with the space character. The final line ends with the space character as well if you always append a space after a character in `englishStr` when building the Morse Code string; this is up to you.) Don’t just put every word on a new line—fill every line as close to 75 characters as possible (without going over). For example, the string ‘Hello space creatures, take me to your leader.’ would be stored as Morse Code in the output file `morseCode.txt` like this:

```
.... . -.-. .-. --- / ... .--. .- -. . /
-.-. .-. .- -. .-. .-. .-. --- -- / - .- -. . / -- . / - --- /
-.- -- .- .-. / .-. .- -. .-. .-. .-
```

Submit your files `genMorseCode.m` and `genSoundData.m` in CMS.