

# IC210 Fall 2007

## Programming Project 1

### Audio Speed Dial

---

#### **Pre-Coding Analysis: Due by the close of business on Wednesday, 12 September 2007.**

Turn in a flowchart (either neatly hand-drawn or using the RAPTOR tool) for a **Step 3** solution for the program in Project 1 (notice that Step 4 is not required). Make sure that your flowchart clearly shows how you intend to solve the *conceptually difficult parts* of the project. Note that you **DO NOT** have to actually code up the solution as part of the Pre-Coding Analysis.

1. Note that this Pre-Coding Analysis is NOT a routine out-of-class assignment, but rather is part of the project and therefore subject to the Department's policy concerning programming projects.
2. Recall that the Department's policy states that for a programming project "midshipmen may give no assistance whatsoever to any person and may receive no assistance whatsoever from any person other than the midshipman's instructor for the course assigning the project." Please view <http://www.cs.usna.edu/academics/ProgrammingPolicy.pdf> for additional details on this issue.

#### **Executive Summary**

Touch-tone dialing is used to signal a number to connect to the call switching center. It is a trademarked form of dual-tone multi-frequency signaling that is in the voice-frequency band. Every time a key is pressed on a touch tone phone, two distinct frequencies are transmitted from the phone to the call switching center. The switching center then determines the number pressed from the two tones. The consequence of this system is that you do not have to press a key to make a call if you can generate the tones required.



For this project, you will write a program that creates a sound wave (as a .wav file) that plays a telephone number. By holding the telephone up to your computer speakers, playing the file will dial a number on your telephone.

## Due Dates and Honor

The Pre-Coding Analysis will be due by the close of business on Wed Sep 12 2007.

This project will be due by the close of business on Thursday Sep 20, 2007. See the course policy for details about due dates and late penalties.

Again, this is a *Programming Project*, and it is very important that you understand and abide by the Department's policy concerning programming projects. Please view: <http://www.cs.usna.edu/academics/ProgrammingPolicy.pdf>

## Extra Information

### 1. Sound and wave files

What our ears perceives as sound is simply a wave moving through the air that vibrates our eardrums. This wave propagates by the compression and rarefaction of the atmosphere (Figure 1). This change in air density at a single point can be graphed with respect to time (Figure 2) This change in air density over time is the data that describes any sound wave and is encoded in a wave file.

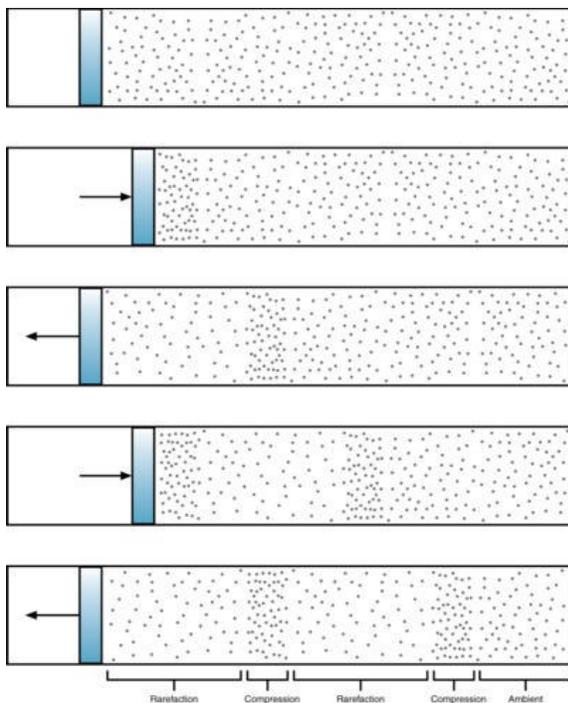


Figure 1

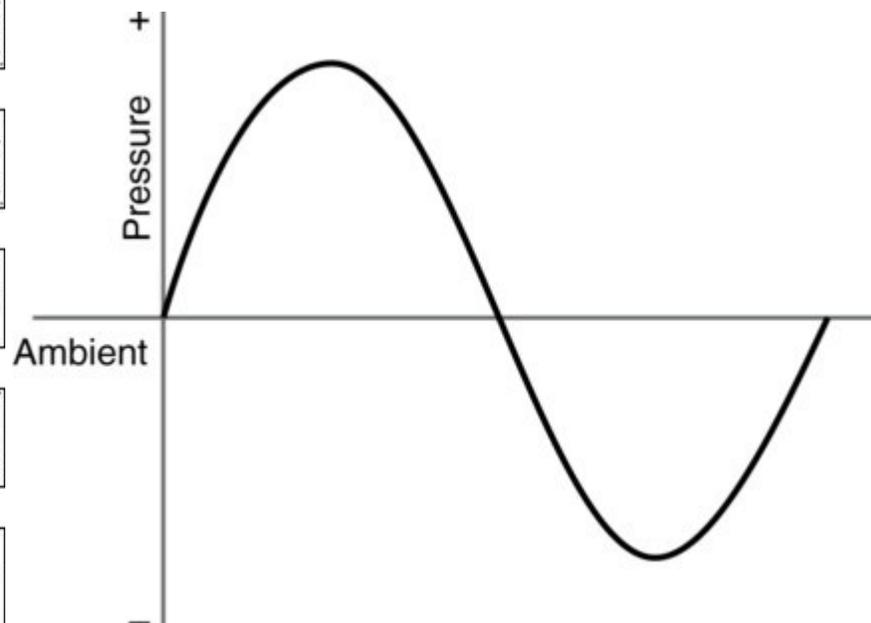


Figure 2

A wave file is data consisting of two parts: a header chunk that describes the amount of information that it contains and a data chunk that contains numbers that correspond to sound amplitudes (or air pressure) over time. This data chunk is exactly what is encoded on a compact disc. The header portion is not required for CDs because the encoding options are fixed. To understand the files you will be creating, you need to understand the options for encoding a wave file.

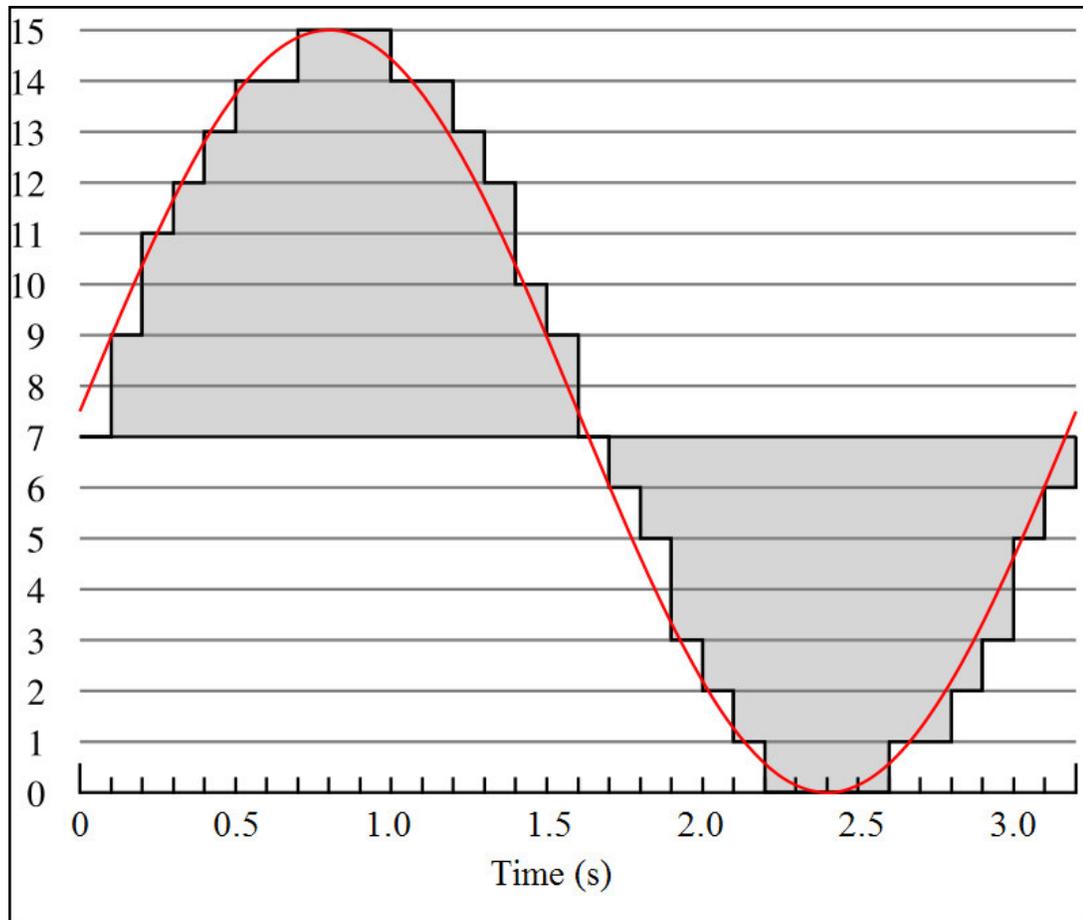
The header chunk contains the following pieces of information (among others):

sample rate: samples per second

bits per channel: 8 or 16

number of channels: mono or stereo

Channels should be self-explanatory. Sample rate and bits per channel are necessary because computers can only handle discrete numbers. In the real world, waves are continuous but computers need to approximate to represent continuous numbers. This is shown below.



In order to represent a continuous wave, the computer will connect a series of points that are close to the original waveform. The red, smooth line represents the original waveform. The black, jagged line represents the waveform played by the computer. This example shows 4 bits per channel ( $2^4=16$  levels) and a sample taken every 0.1 seconds. This means the sample rate is 10 Hz ( $=1/0.1$  seconds).

## 2. Useful data

A basic sine wave is described by the following formula:

$$y(t) = \text{Amplitude} * \sin(2 * \pi * \text{frequency(Hz)} * \text{time(sec)})$$

The tone for a single telephone digit is created by **adding** together **two sine waves** at different frequencies. For instance, a '1' is made by adding sine waves of frequency 697 Hz and 1209 Hz. The table below shows the other combinations.

| Hz  | 1209     | 1336     | 1477     |
|-----|----------|----------|----------|
| 697 | <b>1</b> | <b>2</b> | <b>3</b> |
| 770 | <b>4</b> | <b>5</b> | <b>6</b> |
| 852 | <b>7</b> | <b>8</b> | <b>9</b> |
| 941 | <b>*</b> | <b>0</b> | <b>#</b> |

### Touchtone Frequencies

We will use mono (1 channel) 8-bit encoding at a sample rate of 8000 Hz.

## 3. Useful code

A Visual Studio "project" with some needed code is provided: [starter.zip](#). Download this and unzip it somewhere (your X: drive would be ideal – if you haven't already, set up access to that from Bancroft; see the course home page for help). Then from Visual Studio, choose "Open Workspace" and browse to and open "Project 1.dsw."

A file project1.cpp is provided – **this is the only file you should change and submit!** (i.e. do not modify other .cpp or .h files). Along with the standard stuff, this file has some starter code for you. At the top, to provide access to the code that will generate sound files for you, is this:

```
#include "Wave.h"
```

Then in main() is the following code. Start your code after this:

```
WaveFile out; //creates WaveFile object named out
string fileName = "frequency.wav"; //creates variable to hold wave file name
out.OpenWrite(fileName.c_str()); //links object out to file fileName
out.SetupFormat(8000, 8, 1); //sampleRate = 8000 Hz
//bitsPerSample = 8, channels = mono
```

Whenever you have a number  $y$  that you want to write to the wave file use the statement:

```
out.WriteSample(y);
```

$y$  must be a continuous number between -1 and 1. **You also must output  $y$  to the screen.** This will aid your debugging, but you will only see the last few samples on the screen. See the sample output below for an example.

Note: if you make a statement such as

```
out.WriteSample(1)
```

you may receive a compile error regarding “ambiguous overloaded function.” This will make sense later – for now, instead always use a variable (such as *y*) instead of a constant (such as the literal 1) when you use `WriteSample`.

Also, the constant `PI` has been provided for you. You can use it just like a variable, e.g.

```
area = 2 * PI * radius * radius;
```

## Details

The project is divided up into four steps. You are strongly encouraged to solve each step completely - including thorough testing - before you move on to the next step. Also, remember that your program’s source code must comply with the [Required Style Guide](#) in order to maximize the grade you receive on this project. Compile your program often so you don’t end up with a snake’s nest of errors at your first compile, and occasionally save a backup copy of your program’s source code in case your sponsor’s dog eats your X drive.

### **Step 1: Create a waveform from a user frequency and write it to a wave file**

Write a program that reads in a frequency from a user and creates a wave file named `frequency.wav` with the corresponding waveform. This waveform should have an amplitude = 0.5 and a length of 1 second. Sample output is shown below.

If your sample output looks reasonable, try playing the sound by double-clicking on the `.wav` file (it will be in the same directory as your `.cpp` file). This will run Windows Media Player or a similar player. You should be able to hear a 800 Hz tone.

Key tip #1: If your program doesn’t stop running, click on the output window, then press Ctrl-C to kill it.

Key tip #2: You **MUST** close Windows Media Player after you play your sound. If not, your program **will crash** when you try to run it next!!!!

Key tip #3: Your output may not exactly match the sample output shown below. How can you tell though if it is truly outputting a 800 Hz wave? (think about the period of the wave)

Key tip #4: Until you get to step 4, you should need no more than 40 lines of code (at most). Otherwise, you’re probably not on the right track.

Key tip #5: Save a **complete copy** of each step when it works before you go on to the next step.

Key tip #6: If you’re not sure why your program doesn’t work, add additional “cout” statements to test the values of variables at different points in your program (or just to see how far your program is getting).

```
C:\ "C:\Documents and Settings\hottenst.CSDEPARTMENT\My Documents\IC210\WaveTest\Debu... - □ X
Welcome to LT Hottenstein's Project #1
Enter desired frequency in Hertz: 800_
```

```
C:\ "C:\Documents and Settings\hottenst.CSDEPARTMENT\My Documents\IC210\WaveTest\Debu... - □ X
-0.47487
-0.476178
-0.295603
-0.00211756
0.292177
0.474869
0.476179
0.295604
0.00211888
-0.292176
-0.474869
-0.476179
-0.295605
-0.00212021
0.292174
0.474869
0.476179
0.295606
0.00212154
-0.292173
-0.474868
-0.47618
-0.295607
Created frequency.wav
Press any key to continue_
```

**Step 2: Create a waveform from a user frequency and amplitude and write it to a wave file**

Write a program that reads in a frequency and amplitude from the user and creates a wave file named amplitude.wav with the corresponding waveform. See sample output for expected values. Check that the amplitude provided by the user is between 0 and 0.5. **If it is outside this range, set it to 0.5.**

```
C:\ "C:\Documents and Settings\hottenst.CSDEPARTMENT\My Documents\IC210\WaveTest\Debu... - □ X
Welcome to LT Hottenstein's Project #1
Enter desired frequency in Hertz: 800
Enter desired amplitude <0-0.5>: 1.5_
```

```
C:\ "C:\Documents and Settings\hottenst.CSDEPARTMENT\My Documents\IC210\WaveTest\Debu... - □ X
-0.47487
-0.476178
-0.295603
-0.00211756
0.292177
0.474869
0.476179
0.295604
0.00211888
-0.292176
-0.474869
-0.476179
-0.295605
-0.00212021
0.292174
0.474869
0.476179
0.295606
0.00212154
-0.292173
-0.474868
-0.47618
-0.295607
Created amplitude.wav
Press any key to continue
```

### Step 3: Create a wave file that dials a single button

Write a program that creates a wave file named star.wav that dials the star key. It should still get the user input for the amplitude, and still be one second in length. Check that the amplitude is between 0 and 0.5. If it is outside this range, set it to 0.5. It should sound like your phone when the star key is pressed. See the sample output for expected values.

```
C:\ "C:\Documents and Settings\hottenst.CSDEPARTMENT\My Documents\IC210\WaveTest\Debu...
0.40882
-0.180366
-0.535851
-0.471959
-0.138679
0.154109
0.211985
0.0922518
0.000887907
0.0651191
0.200534
0.197673
-0.052654
-0.409873
-0.565317
-0.305497
0.265851
0.755279
0.764605
0.218051
-0.538943
-0.970332
-0.747072
Created star.wav
Press any key to continue.
```

#### Step 4: Create a wave file that dials a five digit extension

Write a program that creates a single wave file named `mystery.wav` that dials the following mystery number: 36816 (you could, of course, just dial this on your phone, but resist the temptation. Instead, wait for the satisfaction of getting your project to work! You'll know if it was the right number). You should test it by attempting to dial with your computer speakers. If it does not work, turn up the volume on the speakers! If it still doesn't work, the waveform is incorrect. Use the frequencies given above to make a dual tone waveform for each number (**0.25 seconds for each**) separating each number by 0.1 seconds of silence. It should still get the user input for the amplitude. Check that the amplitude is between 0 and 0.5. If it is outside this range, set it to 0.5.

Tip #1: This step will probably involve duplicating very similar code for each of the 5 digits (later we'll see much better ways to handle this!). So, you'll want to cut and paste – but make sure you've got everything set up well for one digit before you start, and think about how to minimize the number of changes you need for each of the 5 sections.

Tip #2: Don't forget to save a complete copy of Step 3 before you start this!

#### What to submit and how it will be graded

You will only submit a solution to **one** of the four steps above. You will submit your solution for the highest numbered step *that actually works!* For example, if you had a working solution to Step 3, but only a partially completed solution to Step 4, you would submit your Step 3 solution.

The maximum points for a Step 0 solution 5 pts, for a Step 1 solution, 30, for a Step 2 solution, 55, for a step 3 solution, 80, and for a Step 4 solution, 100 pts. How points are assigned for a particular solution is at your instructor's discretion, however, program structure, documentation, variable names, etc. will be considered along with program correctness. Important points:

- **If your program does not compile as submitted**, you will receive a zero.
- If your program does not give correct results, penalties will be substantial. If you can't get a working Step k+1 solution, submit your Step k solution instead.
- Your program must read input and write output in the exact format specified in this project description.
- Your program's source code must comply with the Required Style Guide in order to maximize the grade you receive on this project

There will be both a paper and an electronic part to your submissions. The paper submission can be handed to your instructor in class, slid under their office door, or put in their mailbox. For the purposes of any late penalties, your project is not considered submitted until your instructor receives BOTH the electronic and paper portions of your submission.

**Electronic submission:** Unless otherwise specified by your instructor, electronic submission should be a single email message with title "**IC210 Project Submission**". Include only your project1.cpp file for the appropriate one part that you are submitting (the other files should not be changed). If you have any questions, send a separate message with a different subject. Be certain to include the file!

Here's what to submit (follow instructions for **one** of the following):

**Step 0 Pre-coding Analysis:** (5 pts). See description at start of project.

**Step 1 submission:** (max 30 pts)

**Paper:** Submit a printout of your source code and a screen capture showing your program being run on input **800**. You will also submit a signed copy of the **project cover page** (see IC210 home page) and your graded Pre-Coding Analysis from Step 0.

**Electronic:** See above.

**Step 2 submission:** (max 55 pts)

**Paper:** Submit a printout of your source code and a screen capture showing your program being run on input **800** and **1.5**. You will also submit a signed copy of the **project cover page** (see IC210 home page) and your graded Pre-Coding Analysis from Step 0.

**Electronic:** See above.

**Step 3 submission:** (max 80 pts)

**Paper:** Submit a printout of your source code and a screen capture showing your program being run on input **1.5**. You will also submit a signed copy of the **project cover page** (see IC210 home page) and your graded Pre-Coding Analysis from Step 0.

**Electronic:** See above.

**Step 4 submission:** (max 100 pts)

**Paper:** Submit a printout of your source code, a screen capture showing your program being run on input **1.5**. You will also submit a signed copy of the **project cover page** (see IC210 home page) and your graded Pre-Coding Analysis from Step 0.

**Electronic:** See above.

**Extra Credit: (Maximum of 10 points)** Make your program capable of dialing any five digit extension. Get user input and include additional screen captures showing your program handling the full variety of incorrect/invalid input that your program is capable of handling.