# Computer Science 327
# Project 1, Part b
# C Programming Project
# Digital Sound

## The Assignment (Part b)

1. First, create a header file named "gensnd.h" that contains the prototypes for functions defined below as you create and test them. You will also need to define a structure and typedef in this file that represents a sound sample. The definition of the typedef statement is:

   ```
   typedef struct sound_t sound;
   ```

   where you will define the sound_t structure tag that contains a member of type "float *" is a pointer to a float array named "samples" that are the samples in the sound. Also including in this structure definition is an int named "length" that is the number of the samples in the array, and a float named "rate" that is the sample rate for the sequence of samples.

2. Modify your gensnd.c file from Part a to include a function named "gensine2" with the following prototype:

   ```
   sound * gensine2( float hertz, float sample_rate, float duration);
   ```

   Code this function so that it acts just like gensine except that it returns a pointer to the structure defined above. The size of the returned array in the structure must be exactly that of the number of samples generated in this function. In other words, use malloc.

3. Modify the gensnd.c file from Part a to include a function named "genDTMF" with the following prototype:

   ```
   sound * genDTMF2(char key, float sample_rate, float duration);
   ```

   Code this function so that it creates the correct dual tones (similar to part a) except that it is at the specified sample rate and duration given in the parameters. In addition, the dual tones are returned the structure defined above and the array in the structure that must be

exactly the size as the number of samples generated by the function. Also, this function must be implemented so that it uses (calls) gensine2 to create the sound samples. Remember to free any unused memory that is allocated.

Modify the gensnd.c file from Part a to include at function named "genSilence" with the following prototype:

```
sound * genSilence(float sample_rate, float duration);
```

Code this function so that it outputs samples of value 0 at the specified sample rate and duration. In addition, the samples are returned in the structure defined above and the array in the structure must be exactly the same size as the number of samples generated by the function. In other words, use malloc.

4. Modify your gensnd.c file from Part a to include an "outputSound" function with the following prototype.

   ```
   int outputSound( sound *s, FILE *f);
   ```

   where "s" is the sound to output to the file, and "f " is a file descriptor to the output file. The return value is 0 if it succeeded in writing the file, and 1 if any error occurs. The function should output the sound sample to the specified file similar to Part a.

5. Create a file named "main1b.c" that contains a main function that utilizes the command line parameters as follows. The first parameter is a string that is the phone number to dial. The second parameter is a filename to output the sound samples to.

   The program must have the first command line parameter present. If it is not, or if it does not represent a valid set of digits, then the program prints an error message to stderr. Note that there is no prescribed lower or upper limit on the number of digits that may be given in the first argument, only that they be valid phone key digits/letters (upper or lowercase).

   The second command line parameter is a file name that may or may not be present. If it is present, then the program uses this file for output of the sound. If it is not present, then the program outputs the sound to stdout.

   If there are no command line parameters present then the program simply outputs a usage message on how to run the program.

   Examples of valid command lines to run the program are:

   ```
   dtmf 5551212 output.wav
   ```

```
dtmf 5551212

dtmf 5aa553* output2.wav

dtmf
```

Note that the above examples use an executable file named “dtmf” This is the program that will be specified in the Makefile specified below.

6. Update the Makefile to include a default target of “all” The intent of this target is to build all the files in Part a and Part b. Create a target named “parta” that will build all the files in Part a of the assignment. Likewise, create a target named “partb” that will build all the files in Part b. Finally update the “clean” target if necessary.