

## 1 Digital Signals

The process of converting an analog signal to a digital signal is known as digitization. An analog signal (such as your voice or music) is sampled over regular intervals of time. These samples, called Pulse Code Modulation (PCM) samples, are then put together and processed to create the final digital signal. As a result, the digital representation of the analog signal is a list of discrete values. The digitization process is illustrated in figure 1. For this assignment audio will be sampled at 8000 samples per second, where

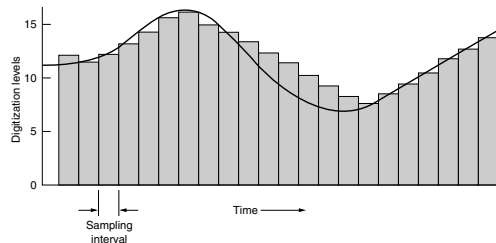


Figure 1: An analog signal (the continuous wave) is sampled at regular intervals to create a digital signal.

each sample is represented with 16 bits (digital sample can range from -32768 to +32767). Higher sampling rates and a higher number of bits per sample would result in a *better* signal. The digital data can then be stored in a file then converted back to an analog signal for listening. Different file formats are available (for example `.wav` and `.mp3`); however this assignment will use `.au` files.

## 2 Program

Your program will perform various tasks on audio data, upon user's request. The program should first **clear the screen** and display a menu. The menu has the following choices; read audio data from file (1), display signal (2), adjust volume (3), reverse audio (4), play audio (5) and quit (0). The user is allowed to select only options 1 or 0 at the beginning (since options 2 through 5 require the file to be loaded first). After the file has been loaded at least once, the user is allowed to select any option. If the user selects an invalid option re-prompt the user until a valid option is entered. When any valid option is entered **the screen should be cleared** and the appropriate task (options 1 through 5) performed. After a task is done, pause and return to the main menu. This will repeat until the user enters 0 to quit at the main menu.

```
Terminal
-----
Digital Signal Processing Program
-----
1) Read audio data from file   4) Reverse audio
2) Display signal             5) Play audio
3) Adjust volume              0) Quit

Enter option (0 - 5) - >
```

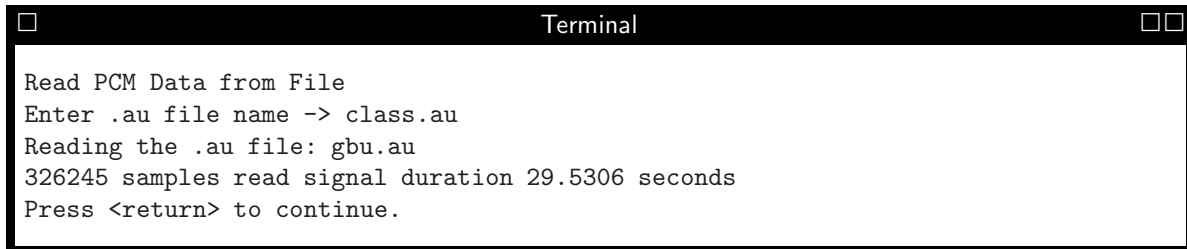
## 3 Option 1: Read PCM Data from File

Option one should first prompt the user for an `.au` file name. Once a valid file name is entered, read **at most** 30 seconds of audio from the file. As mentioned earlier, audio samples are 16 bit numbers; therefore represent each sample with a `short`. Reading audio data from an `.au` file is beyond this course, so a sound

library (`csound.o` or `lsound.o`) is available for this assignment. In the library, the following function will read PCM samples from an `.au` file

```
bool readAuFile(string fileName, short samples[], int MaxSize, int& numSamples);
```

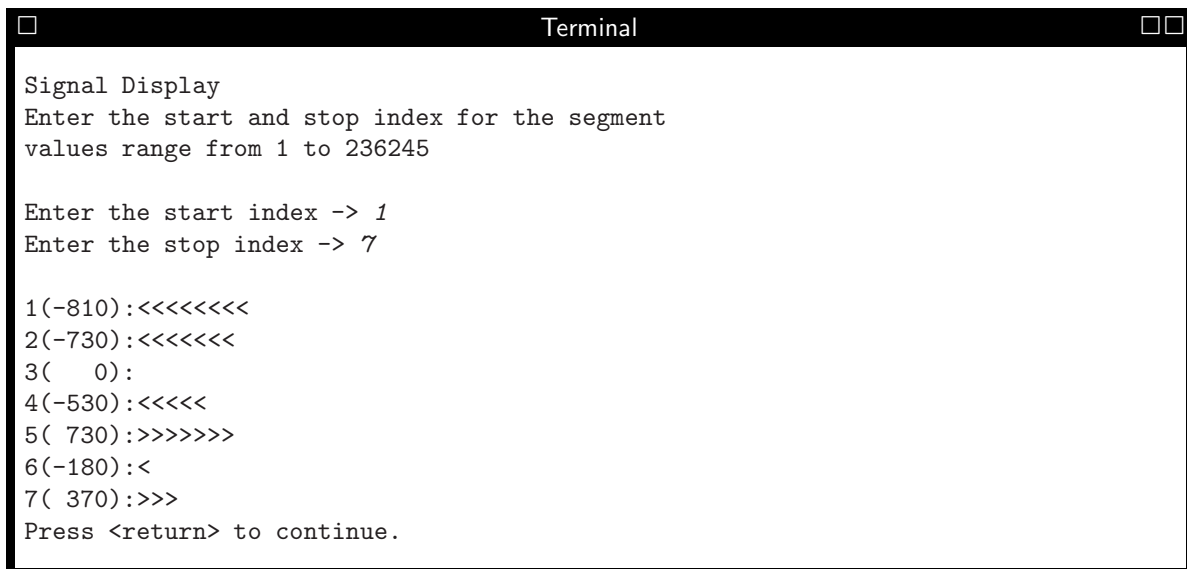
Where `fileName` is the `.au` file name, `samples` is an array of PCM samples, `MaxSize` is the maximum number of elements in `samples` (physical size), and `numSamples` is the number of PCM samples read from the `.au` file. When `readAuFile` is called, the function will open `fileName` and attempt to read `MaxSize` samples, storing the values in `samples`. The function will return the actual number of samples read in `numSamples`. If an error occurs, the function will provide an error message and return `false`, otherwise it will return `true`. When reading the file is done, display the number of samples read, the signal duration in seconds, and pause.



```
Terminal
Read PCM Data from File
Enter .au file name -> class.au
Reading the .au file: gbu.au
326245 samples read signal duration 29.5306 seconds
Press <return> to continue.
```

## 4 Option 2: Display Signal

Option two will display a segment of the signal. First inform the user of the signal range (number of samples). Next prompt for the first and last index of the segment to be displayed (**index must start with 1**). If either value is invalid, re-prompt until a valid value is entered. If the start index is greater than the stop index, interchange the two values. Display the signal to the user from the start index to the stop index. For each sample display; the index number, the sample value inside parentheses and some number of `>` or `<` (depending on the sign) which represents the sample value (one symbol for each multiple of one hundred).



```
Terminal
Signal Display
Enter the start and stop index for the segment
values range from 1 to 236245

Enter the start index -> 1
Enter the stop index -> 7

1(-810):<<<<<<<<
2(-730):<<<<<<<<
3(  0):
4(-530):<<<<<
5( 730):>>>>>>>>
6(-180):<
7( 370):>>>
Press <return> to continue.
```

## 5 Option 3: Adjust Volume

Prompt the user for a value to adjust the audio volume. Values will be floating point numbers between 0 and 10. If the user enters an illegal value, print an error message and reprompt. To adjust the volume of the signal multiply each element by the input number. As a result, any value less than 1.0 decreases the volume while values above 1.0 increases the volume. After the volume has been adjusted, inform the user that the volume has been increased or decreased and pause the program.

## 6 Option 4: Reverse Audio

Option 4 will reverse PCM samples in your array. For example if the array contents were

<code>samples[0]</code>	<code>samples[1]</code>	<code>samples[2]</code>	<code>samples[3]</code>
10.0	32.0	-5.0	17.0

After this option, the array would be

<code>samples[0]</code>	<code>samples[1]</code>	<code>samples[2]</code>	<code>samples[3]</code>
17.0	-5.0	32.0	10.0

Since the array containing the signal is large, **reversing must be accomplished without creating an additional array!** After the reverse is complete, tell the user the total number of samples in the signal and the signal duration (these values should not change due to this option), then pause the program. Note, playing the signal (option 5) after this option will cause the audio to be played in reverse. You can now answer the important question: *Is there a hidden message in the Fat Albert theme song?*

## 7 Option 5: Play Audio Signal

Option 5 will play the signal stored in your array. Similar to reading, playing PCM audio data is beyond this course, so the sound library (`csound.o`) (or `lsound.o` if you are using Linux) contains the following function that will play PCM samples.

```
bool playPCMsamples(short samples[], int numSamples);
```

Where `samples` is an array of PCM samples and `numSamples` is the number of samples in the array. When playing the audio signal is done, display the number of samples and the audio signal duration in seconds, then pause the program.

## 8 Audio Files, Updates, Audio Functions, and Compilation

To complete this lab you will need to copy a few `.au` audio files, the header file `sound.h`, and the object file `csound.o` (or `lsound.o` if you are using Linux) to your working directory. These files are located at the course web-site.

### 8.1 Audio Files

This program will read and process data from `.au` files, which is an audio file format developed by Sun. The sound files located at the course web-site have different lengths (audio durations).

### 8.2 Sound Functions

The functions `readAuFile` and `playPCMsamples` are required for this assignment. To use these functions you must copy `sound.h` and `csound.o` (or `lsound.o` if you are using Linux) located at the course web-site to your directory. The `sound.h` file contains the function declarations, while the file `csound.o` is the object code. You will `#include` the file `sound.h` and `compile` (actually just link) the `csound.o` file.

### 8.3 Compilation

To compile your program you must have copied the `sound.h` and `csound.o` (`lsound.o` for Linux) files in your directory. The `sound.h` must be included in your `main.cpp` file. In addition, you must link the `csound.o` object file with your program. The following compile command will accomplish this.

```
g++ -o lab3.exe main.cpp csound.o
```

Where `lab3.exe` is the final executable name, `main.cpp` contains your functions, and `csound.o` contains the object code for reading and playing audio signals.

## 9 Example Main Function

Since the main program is restricted in what it may contain, the code below demonstrates one possible design. The design consists of only function calls, selection and loop structures.

```
int main()
{
    // partial variable declaration list
    int option;           // users menu choice
    bool loaded = false; // set to true if the file has been read

    // display the menu and read option
    displayMenu(option, loaded);

    // repeat as long as the user does not enter 0
    while(option != 0)
    {
        // determine which option was selected
        // and call the appropriate function
        if(option == 1)
            readFile(...);
        else if(option == 2 && loaded)
            displaySignal(...);
        else if(option == 3 && loaded)
            adjustVolume(...);
        ...

        // finished the option, display the main menu again
        // and read the users next option (check if valid)
        displayMenu(option, loaded);
    }
}
```

Notice the overall structure of the main function. It contains only function calls, one loop, and a selection structure. From these components the execution of the program can be easily seen. Each function performs one task (a menu option) and returns. The object `option` stores the user's menu choice. The object `loaded` is used to signal when the file has been loaded. It is used to prevent options 2 - 5 from being selected before the file is read. This variable should be changed to `true` when the file has been loaded.

## 10 Programming Points

You **must** adhere to all of the following points to receive credit for this program.

1. Turn-in (print-outs and electronically) the C++ source file for this program
2. Your program will consist of 1 (rather large) file called `main.cpp`
3. The main function should only contain variable declarations, function calls, selection structure and a repetition structure.
4. Error check for input (values within bounds).
5. Can only declare one array, which can store no more than 30 seconds of audio.
6. Must adhere to documentation style and standards.