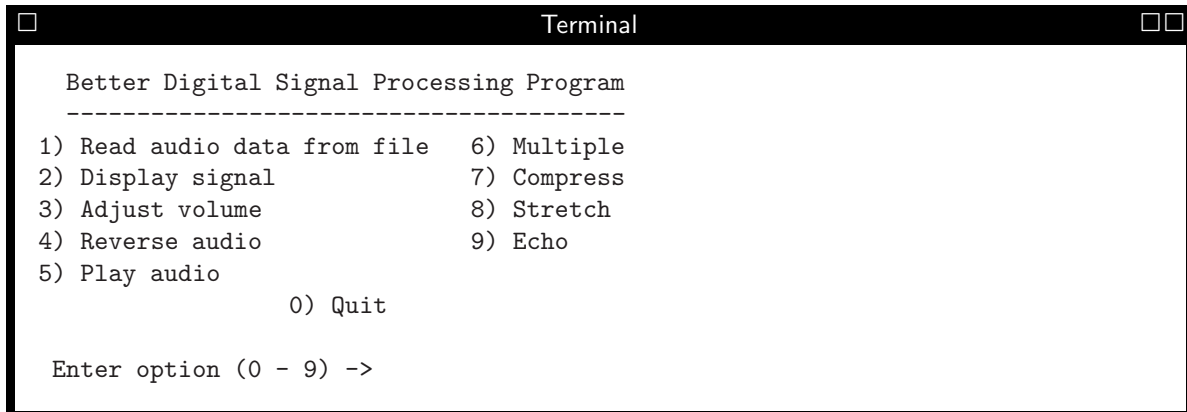


1 Overview

In this lab, you will add three new audio functions to the program you developed in the previous lab. The first six menu options will remain the same. All the options from the previous lab **must** work correctly. You will add option 6) Multiplex, 7) Compress, 8) Stretch, and 9) Echo. Option zero will still cause the program to exit. Your main menu must be updated to include these new options as seen below.



```
Terminal
Better Digital Signal Processing Program
-----
1) Read audio data from file   6) Multiple
2) Display signal              7) Compress
3) Adjust volume               8) Stretch
4) Reverse audio               9) Echo
5) Play audio                  0) Quit

Enter option (0 - 9) ->
```

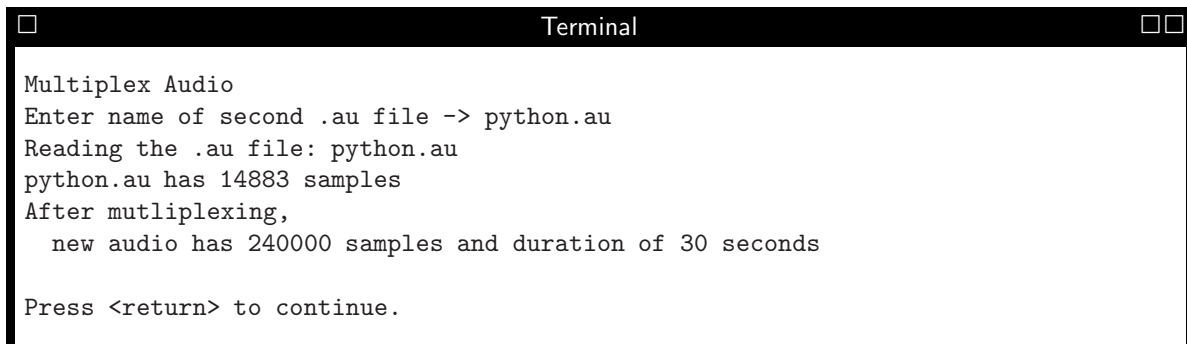
In addition, you **must** divide your program into three files. The file `main.cpp` will only contain the `main` function. The file `signal.h` will contain the declarations (prototypes) of the audio functions (at least one per option), while `signal.cpp` will contain the audio function definitions.

2 New Audio Options

As previously mentioned, for this lab you will add following three functions to your existing audio program. Again, your lab 3 program must work correctly for this assignment.

2.1 Multiplex

Multiplexing is simply adding two audio arrays together element-by-element. The result of multiplexing (layering) is that both audio arrays can be heard simultaneously. The program should have already read an initial audio file before this option can be selected. The function should first prompt the user for a second `au` file and read `au` file samples into another array. Again, no more than 30 seconds of audio should be stored. Once the second audio array has been created, the shorter of the audio arrays will be added to the longer. The number of samples and the duration (in seconds) of the multiplexed audio should be displayed to the user. The user can select option 5 to hear the results. The example below is the result of multiplexing `python.au` with `fat.au` (which was loaded earlier). The number of samples and duration is the same as for the `fat.au` audio.



```
Terminal
Multiplex Audio
Enter name of second .au file -> python.au
Reading the .au file: python.au
python.au has 14883 samples
After mutliplexing,
  new audio has 240000 samples and duration of 30 seconds

Press <return> to continue.
```

2.2 Compress

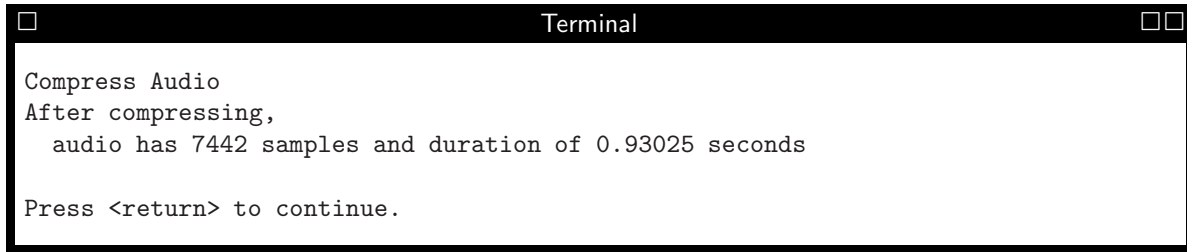
Compress removes every other audio sample stored in the `audioData` array. When the array is subsequently played, it sounds as if it is playing *quickly*. For example, assume the audio consists of 7 samples, where each sample is one byte.

0	52	40	87	30	-10	-37
---	----	----	----	----	-----	-----

After compressing, the contents of the array would be the following.

0	40	30	-37
---	----	----	-----

Your program must display the number of samples and the duration after compressing has taken place. The example below is the result of compressing the `python.au` audio.



```
Terminal
Compress Audio
After compressing,
  audio has 7442 samples and duration of 0.93025 seconds
Press <return> to continue.
```

2.3 Stretch

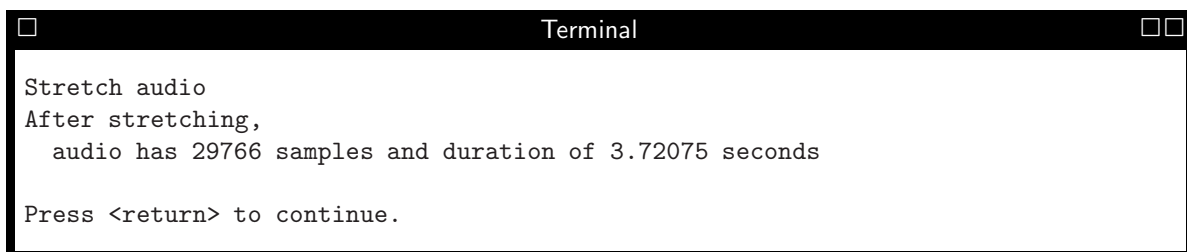
Duplicates every audio sample stored in the `audioData` array (doubling the length of the array). When the array is subsequently played, it sounds as if it is playing *slowly*. For example, assume the audio consists of 7 samples, where each sample is one byte.

0	52	40	87	30	-10	-37
---	----	----	----	----	-----	-----

After stretching, the contents of the array would be the following.

0	0	52	52	40	40	87	87	30	30	-10	-10	-37	-37
---	---	----	----	----	----	----	----	----	----	-----	-----	-----	-----

Remember, **your array can only store 30 seconds of audio**; therefore, if stretching will cause the duration to be longer than 30 seconds, then only the first 30 seconds of audio will be stored. Your program must display the number of samples and the duration after stretching has taken place. The example below is the result of stretching the `python.au` audio.



```
Terminal
Stretch audio
After stretching,
  audio has 29766 samples and duration of 3.72075 seconds
Press <return> to continue.
```

2.4 Echo

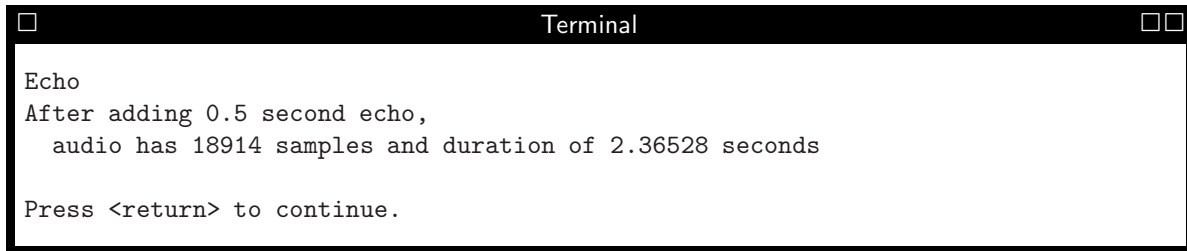
Add the elements of the `audioData` array to itself by an *offset* index. When the array is subsequently played, it sounds as if it has an *echo*. For example, assume the audio consists of 7 samples, where each sample is one byte.

0	52	40	87	30	-10	-37
---	----	----	----	----	-----	-----

If the offset is 3 samples, the contents of the array would be the following.

0	52	40	139	70	-77	-7	-10	-37
---	----	----	-----	----	-----	----	-----	-----

Remember, **your array can only store 30 seconds of audio**; therefore, if adding an echo will cause the duration to be longer than 30 seconds, then only the first 30 seconds of audio will be stored. Your program must ask the user to input the echo delay (offset) **in seconds, which can be a decimal number**. Once the echo has been added display the number of samples and the duration. The example below is the result of adding a 0.5 echo to the `python.au` audio.

A terminal window titled "Terminal" with standard window controls (minimize, maximize, close) in the top right corner. The text inside the terminal reads: "Echo", "After adding 0.5 second echo,", "audio has 18914 samples and duration of 2.36528 seconds", and "Press <return> to continue." The text is left-aligned and uses a monospaced font.

```
Terminal
Echo
After adding 0.5 second echo,
  audio has 18914 samples and duration of 2.36528 seconds
Press <return> to continue.
```

3 Programming Points

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

- Turn-in (print-outs and electronically) the C++ source file for this program.
- The program must be broken into 3 files
 - `main.cpp` Contains the main function and a function to display the menu and read the option.
 - `signal.h` Contains the signal function prototypes.
 - `signal.cpp` Contains the signal function definitions. At least, one function per option.
- The main function should only contain variable declarations, function calls, selection structure and a repetition structure.
- Error check for input (values within bounds).
- Must adhere to documentation style and standards.