Audio Programming

Arrays and Loops C++ Challenges

The challenges for this week revolve around *arrays* and *for loops*. You will find an existing .cpp file for three of the challenges (the fourth requires you to start from scratch):

- *challenge1_correct_the_errors.cpp*
- *challenge2_more_loops.cpp*
- *challenge3_noise.cpp*

Remember, to use these, you will need to do the following:

1. Create a new Xcode or Visual Studio project - giving it an appropriate name like "week2challenges" or similar.
2. Find the .cpp file with the main function in (usually `main.cpp`).
3. Delete all of the text in this original cpp file (including the #include, comments, main function, etc)
4. Open the challenge .cpp file, e.g. `challenge1_correct_the_errors.cpp`
5. Copy and paste everything from that file into the empty cpp file in the project.
6. For each challenge, you can create a new Target in Xcode, each with their own `main.cpp` file. To choose which is being built/run at any given point, you can go to the Product menu ⟶ Scheme, and then select the name of the relevant target.

1 Challenge 1 - correct the errors

There are a number of errors in the code three errors to fix in this code so that it both compiles and runs the appropriate program. The program should output the numbers in the array (3, 5 and 6) using a for loop to step through the array.

The red Xcode/Visual Studio error messages are a useful start to pointing you towards potential errors. As noted above, you will need to paste the code in the *challenge1_correct_the_errors.cpp* file into the main file of a new Xcode or Visual Studio project (replacing everything in the *main.cpp* file).

You will very likely need to look at some of the class examples to check the syntax for things like:

- the definition of the array
- the for loop
- the *cout* function

2 Challenge 2 - more loops

The existing code contains a working for loop that iterates three times. Note that you won't need to use arrays for challenge! You can work directly with the *i* variable inside the loop. Adapt the code so that:

1. it loops 10 times instead of 3 times
2. it prints out the iteration number on each loop (0, 1, 2, 3, 4 ... 9)
3. instead of 0, 1, 2, 3..., it prints the first 10 square numbers, starting from 1: (1, 4, 9, 16 ...)
4. it asks the user how many square numbers it should output, then outputs that amount (e.g. the loop is not always 10, but is customisable).

3 Challenge 3 - noise (in theory)

This example moves us in the direction of using for loops for audio samples. We use the vector library (`#include <vector>`) to set up a vector of floats and set it to be the appropriate size:

```
std::vector<float> samples (numberOfSamples);
```

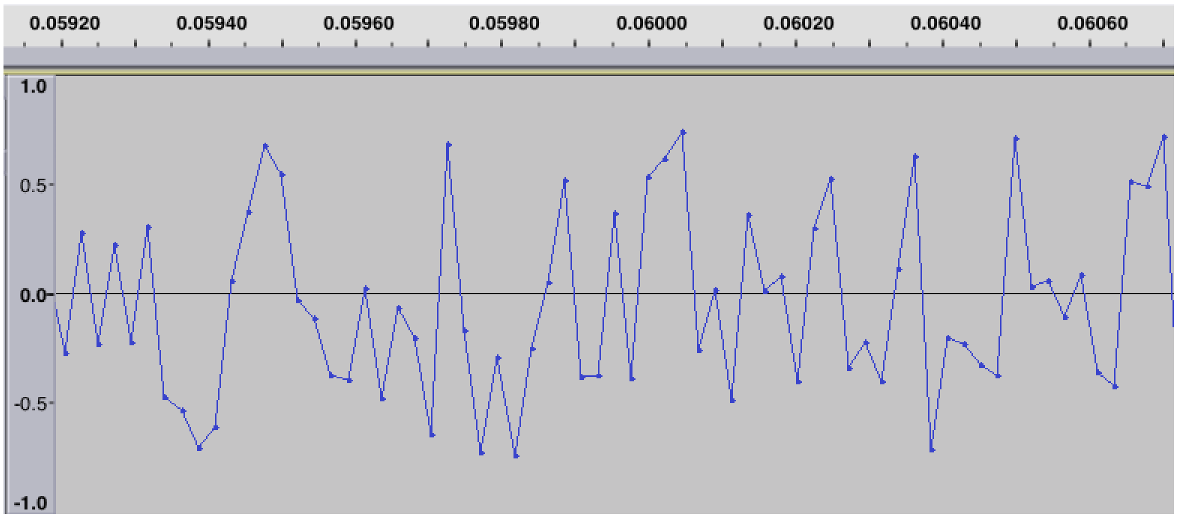

Figure 1: A zoomed in image of a white noise waveform (randomised sample values). We can see around 100 samples here, which is about 2 milliseconds at a sample rate of 44100Hz. Note that the vertical axis is in the range -1 to +1. This is the range for digital audio samples

In this example, we will fill the samples with random values (this is the basis of white noise). Now do the following:

1. Use the numberOfSamples variable to set the vector size to 10 samples.
2. Create a for loop that goes from zero to the number of samples.
3. Assign a random value to each element in the samples vector between 0 and 9999. You can use the rand() function for this, as shown below. This creates a random integer and constrains it to the range 0-9999 (10000 different possible values).

   ```
   rand() % 10000;
   ```

   Note that you'll probably also want to *seed* the rand() function. To do this, you'll need to add an include line at the top (near the iostream include):

   ```
   #include <ctime>
   ```

And then the actual seeding function, srand() inside the main function, initialised with the current time[1] as shown here:

```
srand( time(NULL) );     // seed based on current time
```

4. Add a std::cout to print the value of each element in the vector, with each value on a new line.
5. Adapt the code so that the random number is a floating point number in the range 0 and 1 (hint: don't change the random number itself, but multiply or divide the generated number in order to scale it into the correct range.
6. Adapt this code again so that the random number is in the range -1 to 1 (the range for an audio signal).

4 Challenge 4 - setting up for an actual DSP loop

As the above example shows, before we create any audio in a DSP loop, we need to set up a few variables and settings. In a new Xcode file, create the following:

1. A float variable called `duration` that you can use to set the duration of the audio in seconds.
2. An int variable called `sampleRate` that stores the desired sample rate of the audio (e.g. 44100 is a useful default[2]).
3. Another int variable called `durationInSamples` calculated from the previous two variables.
4. A vector of floats called `samples` the same size as the `durationInSamples` (see the previous challenge for help with this).
5. A for loop that iterates from 0 to `(durationInSamples - 1)`.
6. Use the for loop to set all values of the `samples` array to zero.

This will be our setup for creating actual digital audio in the for loop.

[1]You can read more about the the time function and the ctime library [here]. Interesting if you've ever wondered how many seconds have elapsed since Jan 1st 1970!

[2]for a basic overview of sampling rates, see https://en.wikipedia.org/wiki/Sampling_(signal_processing)