# Project 2 - Generating Digital Audio

In this project, you will create a C program that generates a sine wave—a fundamental audio signal—and saves it as raw PCM (Pulse Code Modulation) data in a file. The output will be a `.raw` file containing uncompressed audio samples, which can later be imported into audio software (e.g., Audacity) as PCM data for playback. This project simulates how digital audio is created and stored, offering hands-on experience with signal generation and file I/O in C.

## Objectives

- Generate a sine wave with customizable frequency, amplitude, and duration.
- Encode the sine wave as PCM data using a specified sample rate and bit depth.
- Write the PCM data to a file in a raw format.
- Explore key C programming concepts through practical application.

## Project Features

1. User Input: Accept parameters like frequency (e.g., 440 Hz for an A note), duration (in seconds), and amplitude (volume level).
2. Sine Wave Generation: Use the `sin()` function from the math library to compute sine wave samples.
3. PCM Encoding: Convert the sine wave into 16-bit PCM samples (signed integers).
4. File Output: Save the PCM data to a `.raw` file, which can be played back with proper audio tools.
5. Basic Error Handling: Check for file creation issues and invalid user inputs.

## Implementation Steps

1. Setup: Include necessary headers (`stdio.h`, `stdlib.h`, `math.h`) and define constants (e.g., sample rate = 44,100 Hz, bit depth = 16).
2. Input Handling: Use `scanf()` to get frequency, duration, and amplitude from the user.
3. Sine Wave Calculation: Compute samples using the formula `sample = amplitude * sin(2 * PI * frequency * t / sample_rate)`, where `t` is the sample index.
4. PCM Conversion: Scale the floating-point sine values (-1.0 to 1.0) to 16-bit integers (-32,768 to 32,767).
5. File Writing: Open a file in binary mode (`fopen("output.raw", "wb")`) and write the samples using `fwrite()`.
6. Cleanup: Close the file and handle any errors.

### Sample Output

- A 440 Hz sine wave for 5 seconds at 44.1 kHz sample rate will produce a 441,000-byte file (44,100 samples/sec × 5 sec × 2 bytes/sample for 16-bit mono).
- Import the `.raw` file into Audacity (File > Import > Raw Data, 44.1 kHz, 16-bit PCM, mono) to hear the tone.

## C Language Features Learners Can Explore

This project provides a practical context to master several core C language concepts. Here's what learners can focus on:

1. Basic I/O Operations
    - Feature: `printf()` and `scanf()` for user interaction.
    - Learning Outcome: Understand formatted input/output, including how to read and validate user inputs (e.g., ensuring frequency is positive).
2. Mathematical Functions
    - Feature: Use of `math.h` library and `sin()` function.
    - Learning Outcome: Link the math library (`-lm` flag during compilation) and apply trigonometric functions to real-world problems like signal generation.
3. Data Types and Casting
    - Feature: Conversion between `double` (sine wave values) and `int16_t` (PCM samples).
    - Learning Outcome: Learn about type casting, integer overflow, and the importance of choosing appropriate data types (e.g., `int16_t` from `stdint.h` for 16-bit audio).
4. Loops and Iteration
    - Feature: `for` loop to generate samples over time.
    - Learning Outcome: Practice loop control and iteration to process large datasets (e.g., 44,100 samples per second).
5. File Handling
    - Feature: `fopen()`, `fwrite()`, `fclose()` for binary file operations.
    - Learning Outcome: Master binary file I/O, understand the difference between text and binary modes, and handle file pointers.
6. Pointers and Memory Management
    - Feature: Use pointers with `fwrite()` to write sample data.
    - Learning Outcome: Gain familiarity with pointers, memory addressing, and passing data to functions efficiently.
7. Error Handling
    - Feature: Check return values of `fopen()` and `fwrite()` for NULL or failure.

- Learning Outcome: Implement basic error checking and use conditional statements (`if`) to manage program flow.

8. Constants and Preprocessor Directives
   - Feature: `#define` for constants like `SAMPLE_RATE` and `PI`.
   - Learning Outcome: Learn to use the preprocessor to make code more readable and maintainable.
9. Arrays (Optional Extension)
   - Feature: Store samples in an array before writing to file.
   - Learning Outcome: Practice dynamic memory allocation (`malloc()`, `free()`) and array manipulation.
10. Modularity (Optional Extension)
    - Feature: Split code into functions (e.g., `generate_sine()`, `write_pcm()`).
    - Learning Outcome: Understand function declarations, prototypes, and passing parameters by value or reference.

## Compile and Run

- compile:`gcc main.c -lm`
- run: `./a.out`
- play: Import in Audacity and play.

## Learning Extensions

- Stereo Output: Modify the program to write two channels (left and right).
- Multiple Frequencies: Generate a chord by summing multiple sine waves.
- WAV Header: Add a WAV file header to make the output playable without importing as raw data.