

PSWC

Jackfruit Problem

CWave

a basic Synthesizer made using C.

Prema P Kotur | PES1UG24CS343

Pranav Chandrasekar | PES1UG24CS332

Pranav Prabhakar | PES1UG24AM463

Prateek Pawar | PES1UG24CS598

What is CWave?

- CWave is a simple synthesizer, made in C. It lets you play and record sounds using your computer keyboard or a MIDI device.
- When we set out to make CWave, we planned on implementing the following features:
 - Different waveforms (sine, square, saw...)
 - Adding effects like volume, octave-changing
 - Connecting a keyboard.
 - A feature to record a session.

Includes and Definitions

`#include <stdio.h>` – lets you use input/output functions like `printf()`.

`#include <stdlib.h>` – for general functions like memory allocation, random numbers, etc.

`#include <math.h>` – gives you math functions like `sin()`, `cos()`, etc.

`#include <Windows.h>` – used to interact with Windows stuff (like windows, messages, etc).

`#include <mmsystem.h>` – used for multimedia things like MIDI and sound.

`#include "portaudio.h"` – includes the PortAudio library, which helps with playing/recording audio.

`#pragma comment(lib, "winmm.lib")` – tells the compiler to link with the `winmm.lib` file, which is needed for Windows multimedia functions.


Data Structures

-> typedef enum { WAVE_SINE, WAVE_SQUARE, WAVE_SAW } Waveform;

This just lists the 3 kinds of sound waves the synth can make:

- **Sine** – smooth & clean
- **Square** – choppy & buzzy
- **Saw** – edgy & harsh

This describes a single note:

- **frequency** – how high or low the pitch is 
- **phase** – keeps track of where the wave is at (used to shape the sound)
- **active** – is the note currently being played? (1 = yes, 0 = no)

```
typedef struct {  
    float frequency;  
    float phase;  
    int active;  
} Note;
```

Synth structure for defining waveforms (kinda important)

This is the main control center for the synth.

- **notes** – a list of all notes it can play
- **volume** – how loud it is 🔊
- **waveform** – the type of wave it's using (from earlier!)
- **octaveShift** – moves notes up/down in pitch
- **recording** – is it recording right now? (yes/no)
- **recordBuffer** – where the recorded sound gets stored
- **recordPos** – how far into the recording we are
- **recordMaxSamples** – how much we can record total

```
typedef struct {  
    Note notes[MAX_NOTES];  
    float volume;  
    Waveform waveform;  
    int octaveShift;  
    int recording;  
    float *recordBuffer;  
    size_t recordPos;  
    size_t recordMaxSamples;  
} SynthData;
```

Note Handling

The function `midiNoteToFreq` turns a **MIDI note number** (like 60 for Middle C) into a real sound **frequency in Hz** (like 261.63 Hz).

It uses **equal temperament tuning**, which is the standard way instruments are tuned—every semitone is spaced evenly.

`note_on(freq)` – starts playing a note at that frequency
Sets that note's active = 1

`note_off(freq)` – stops playing it
Sets active = 0

They just switch a note **on or off** depending on what frequency you tell them. Super helpful for controlling which notes are playing in your synth.



```
float midiNoteToFreq(int midiNote);
```

```
void note_on(float freq);  
void note_off(float freq);
```

MIDI Callback

MIDI callback function is a **special function** that gets called **automatically** whenever a **MIDI message** comes in (like pressing a key on a MIDI keyboard).

What does it do?

- Checks the message to see if it's a **Note On** or **Note Off**
- Then it calls:
 -  `note_on()` – if the note should start playing
 -  `note_off()` – if the note should stop playing

```
void CALLBACK midi_callback(...)
```

Basically, this function listens for MIDI input and tells your synth what to do in real time.

Waveform Generator

The Waveform Generator function creates **a single audio sample** (tiny piece of sound) for a specific moment in time, based on:

- **frequency** – how high/low the sound is (pitch)
- **time** – when in the sound you are
- **waveform** – the shape of the sound wave

Waveforms it can generate:

- **Sine**: $\sin(2\pi ft)$
- **Square**: sign of the sine wave.
- **Sawtooth**: ramp function.

```
float generateWaveSample(float frequency, float time, Waveform waveform);
```

Recording to WAV

This function finds the next free filename like:

- recording1.wav
- recording2.wav
- recording3.wav

```
void writeWavFile(const char *filename, float *data, size_t numSamples);
```

...and so on.

It makes sure it doesn't overwrite the previous recordings!

This function takes the recorded sound (in floating point format) and saves it as a **WAV file** with:

- **16-bit audio**
- **Mono (one channel)**
- **Standard WAV format** using RIFF headers and chunks (how WAV files are structured)

PortAudio Callback

For each audio frame:

- Sum all active notes' waveforms.
- Normalize by the number of active notes.
- Multiply by volume.
- Write to stereo output buffer.
- Optionally store in recording buffer.

```
void writeWavFile(const char *filename, float *data, size_t numSamples);
```

Keyboard Interaction

Monitors key states using `GetAsyncKeyState()`:

- Plays notes based on keys A–K and W–U.
- Adjusts volume (+/-).
- Changes waveform (Z, X, C).
- Shifts octaves (↑, ↓).
- Toggles recording (R).

```
void updateNotesFromKeyboard(SynthData *data)
```

Key Helper Function

What It Does:

- Takes a **keyboard key code** (vkey)
- Looks up which **musical note** it maps to (like C, D, E, etc.)
- Applies the **octave shift** (so you can move notes up/down an octave)
- Returns the final **frequency in Hz** (like 440.0 for A4)

```
float getFrequencyForKey(int vkey, int octaveShift)
```

So if you press a key like **A**, and your synth is set to **Octave 4**, this function might return 440.0 Hz.

If you shift it up one octave, it gives you 880.0 Hz instead.

It's how your computer keyboard becomes a piano !!

Main Function - int main()

What It Does:

1. **Starts things up:**
 - Initializes **PortAudio** (for sound output)
 - Sets up **MIDI input** (to listen to your keyboard/MIDI controller)
2. **Begins audio playback:**
 - Starts the **audio stream**
 - Enters a loop to keep the synth running
3. **Main loop:**
 - **Watches for keyboard input** (like computer keys)
 - If you press **ESC**, it stops the loop and shuts everything down
4. **When you're done:**
 - Closes MIDI + audio connections
 - Shuts down PortAudio
 - Saves any **recording** you were making to a WAV file

Function / Concept

Explanation

`Pa_Initialize()` / `Pa_Terminate()`

Initialize or close PortAudio system.

`Pa_OpenDefaultStream()`

Opens audio stream for output.

`Pa_StartStream()` / `Pa_StopStream()`

Start or stop audio processing.

`audioCallback()`

Called repeatedly to generate sound.

`generateWaveSample()`

Converts frequency and time into a waveform sample.

`GetAsyncKeyState()`

Windows function to check if a key is pressed.

`midiInOpen()` / `midiInStart()`

Opens and starts receiving data from a MIDI device.

`MIM_DATA`

MIDI input message type (data packet).

`note_on()` / `note_off()`

Starts or stops a note.

`writeWavFile()`

Manually constructs and writes a WAV file.

`malloc()` / `free()`

Allocates and releases memory for recording buffer.