Explanation of the Arduino Code for Beethoven's Seventh Symphony Performance

Overall Structure

Both programs (`Accompaniment.ino` and `Melody.ino`) use digital output pins to play musical notes. They share a common design pattern:

1. **Note definitions**: Store musical sequences in PROGMEM (flash memory) to conserve RAM.
2. **Setup**: Initialize pins and serial communication.
3. **Loop**: Play notes sequentially with precise timing using `millis()` .

Accompaniment.ino: Chord-Based Harmony

Plays chords (dual notes) and rests to form harmonic accompaniment.

Key Components

1. **Note Structure**:

```
struct Note {
  int8_t pin1;      // First pin (-1 = rest)
  int8_t pin2;      // Second pin (-1 = rest)
  uint16_t duration; // Duration in ms
};
```

2. **Durations**:
   - `EIGHTH = 125ms` , `DOTTED_QUARTER = 375ms`
3. **Pin Definitions**:
   - Assigns musical notes (e.g., `pin_DObasso = 2` for low C#).
4. **Melody Sequence**:
   - Stored in `PROGMEM` to save RAM.
   - Example chord: `{pin_MI, pin_LA, EIGHTH}` plays E and A simultaneously.

Workflow

1. **Setup**:
   - Initializes all pins referenced in the melody as outputs.

```
if (currentNote.pin1 != REST) pinMode(pin1, OUTPUT);
```

2. **Loop** (State Machine Logic):
   - **Start Note**:
     - Load next note from PROGMEM.
     - Turn off previous pins, activate new pins.

```
digitalWrite(activePin1, HIGH); // Activate new note
```

   - **Sustain Note**: Wait for the duration without blocking.
   - **End Note**: Turn off pins after duration elapses.
   - **Loop**: Reset to start after the last note.

Melody.ino: Single-Note Melody

Plays a monophonic melody line with rests.

Key Differences from Accompaniment

1. **Simpler Note Structure**:

```
struct Nota {
  int8_t pin;     // Single pin (-1 = rest)
  uint16_t durata; // Duration
};
```

2. **Durations**:
   - Based on `beat = 250ms` (e.g., `quarter_note = 1000ms`).
3. **Initialization**:
   - Explicitly turns off all pins at startup:

```
{2,1}, {3,1}, ... // Set pins LOW briefly
```

Workflow

1. **Setup**:
   - Configures melody pins as outputs.
2. **Loop**:
   - Similar non-blocking state machine:

    - Load next note → Activate pin → Wait → Deactivate → Repeat.
- Handles rests by skipping pin activation:

```
if (currentPin != REST) digitalWrite(pin, HIGH);
```

Critical Technical Details

1. **PROGMEM Usage**:
    - Data stored in flash with `PROGMEM`.
    - Accessed via `memcpy_P()` to avoid RAM exhaustion.
2. **Non-Blocking Timing**:
    - Uses `millis()` for precise timing without `delay()`.
    - Maintains state between loop iterations via `static` variables.
3. **Rest Handling**:
    - `REST = -1` skips pin operations while maintaining duration.
4. **Resource Management**:
    - Pins are initialized only once in `setup()`.
    - PROGMEM ensures large melodies fit on memory-constrained devices.

Synchronization Consideration

- Both sketches run independently. For synchronized playback:
    1. Upload both to separate Arduinos.
    2. Power them simultaneously.
    3. Ensure tempo constants match (e.g., `beat` aligns with `EIGHTH`).

Summary

- **Accompaniment**: Dual-note chords, complex timing.
- **Melody**: Single-note line, simpler structure.
- **Shared Core**:
    - PROGMEM storage
    - Non-blocking state machines
    - Precise timing via `millis()`
    - Rest support via `REST = -1`

This structure allows efficient, reliable playback of complex musical pieces on resource-limited hardware.