explanation.md

Bowling Score Calculator: Code Breakdown

The following is a break down of the code step-by-step. Imagine you're learning how to play a video game, but instead of buttons, you're working with code. Here's what each part does and why it matters:

1. Including Necessary Libraries

```
#include <iostream>
#include <iomanip>
```

Explanation:

- `#include <iostream>:` This allows the program to use features for input and output operations.
- `#include <iomanip>:` This lets you format the output, such as adjusting spacing.

2. Setting Constants

```
const int MAX_FRAMES = 10;
const int MAX_THROWS = 21;
```

- **`MAX_FRAMES`**: The number of frames in a standard game of bowling (10 frames).
- **`MAX_THROWS`**: The maximum number of throws in a perfect game (21 throws).

3. Function to Calculate Scores

```
void calculateScores(int throws[], int frameScores[], int currentThrow) {
    int frameIndex = 0;
    for (int i = 0; i < MAX_FRAMES; i++) {
        if (throws[frameIndex] == 10) { // Strike
            frameScores[i] = 10 + throws[frameIndex + 1] + throws[frameIndex + 2];
            frameIndex++;
        } else if (throws[frameIndex] + throws[frameIndex + 1] == 10) { // Spare
            frameScores[i] = 10 + throws[frameIndex + 2];
            frameIndex += 2;
        } else { // Open frame
            frameScores[i] = throws[frameIndex] + throws[frameIndex + 1];
            frameIndex += 2;
        }
        if (i > 0) {
            frameScores[i] += frameScores[i - 1]; // Add previous frame's score to the current frame
        }
    }
}
```

Explanation:

- **`Strike`**: If the first throw is 10, the score for the frame is 10 plus the next two throws.
- **`Spare`**: If the total of two throws is 10, the score is 10 plus the next throw.
- **`Open Frame`**: Simply add up the two throws.
- Keeps a running total of scores for each frame.

4. Function to Print Frame Headers

```
void printFramesHeader() {
    std::cout << "Frame#: ";
    for (int i = 0; i < MAX_FRAMES; i++) {
        std::cout << std::setw(5) << i + 1; // Print frame numbers with padding
    }
    std::cout << std::endl;
}
```

Explanation:

- Prints the numbers of the frames at the top of the output to indicate which numbers correspond to which frames.

5. Function to Print Throws

```
void printThrows(int throws[], int currentThrow) {
    std::cout << "Throws: ";
    for (int i = 0; i < currentThrow; i++) {
        std::cout << std::setw(2) << throws[i] << " "; // Print each throw with padding
    }
    std::cout << std::endl;
}
```

Explanation:

- Displays all the throws made so far, helping track how many pins have been knocked down.

6. Function to Print Scores

```
void printScores(int frameScores[], int currentFrame) {
    std::cout << "Scores: ";
    for (int i = 0; i < currentFrame; i++) {
        std::cout << std::setw(5) << frameScores[i]; // Print each frame score with padding
    }
    std::cout << std::endl;
}
```

Explanation:

- Shows the cumulative scores for each frame, indicating performance frame-by-frame.

7. The Main Function

```
int main() {
    int throws[MAX_THROWS] = {0}; // Store the pins knocked down in each throw
    int frameScores[MAX_FRAMES] = {0}; // Store the cumulative scores for each frame
    int currentThrow = 0; // Index to track the current throw

    // Initial print of frames header, throws, and scores
    printFramesHeader();
    printThrows(throws, currentThrow);
    printScores(frameScores, 0);
```

```
    // Loop through each frame
    for (int frame = 0; frame < MAX_FRAMES; frame++) {
        std::cout << "\nTurn Frame " << frame + 1 << std::endl;
        std::cout << "1st Throw: ";
        std::cin >> throws[currentThrow]; // Input for the first throw of the frame

        if (throws[currentThrow] == 10) { // Strike
            if (frame < 9) {
                currentThrow++; // Move to the next throw
            } else { // 10th frame
                std::cout << "2nd Throw: ";
                std::cin >> throws[currentThrow + 1];
                std::cout << "3rd Throw: ";
                std::cin >> throws[currentThrow + 2];
                currentThrow += 3; // Move to the next set of throws
            }
        } else { // Not a strike
            std::cout << "2nd Throw: ";
            std::cin >> throws[currentThrow + 1]; // Input for the second throw of the frame
            currentThrow += 2; // Move to the next set of throws

            if (frame == 9 && throws[currentThrow - 2] + throws[currentThrow - 1] == 10) { // 10th frame spare
                std::cout << "3rd Throw: ";
                std::cin >> throws[currentThrow]; // Input for the third throw if spare
                currentThrow++;
            }
        }

        // Calculate scores after each frame
        calculateScores(throws, frameScores, currentThrow);

        // Print the updated frames header, throws, and scores
        printFramesHeader();
        printThrows(throws, currentThrow);
        printScores(frameScores, frame + 1);
    }

    return 0;
}
```

Explanation:

- **`Setup`**: Initializes arrays for throws and frame scores, and sets up for user input.
- **`Loop Through Frames`**: For each frame:
  - Asks for the first throw. Handles the next throws if it's a strike or not.
  - If not a strike, asks for the second throw, and if in the 10th frame and it's a spare, asks for a third throw.
  - Updates scores and prints the current state.

Key Points to Remember

- **Arrays**: Used to store and manage lists of throws and scores.
- **Loops**: Repeat actions for each frame.
- **Conditionals**: Decide what to do based on the result of each throw.

This breakdown should help you understand how the code works, step by step!