🚀 Real-World Project: Event System Using Callbacks

We'll create a **simple event-driven system** using callbacks, where different event handlers (functions) can be registered and called dynamically.

🎯 Project Overview

1. We will create an **`Event` class** that allows **registering callbacks**.
2. Users can register different **event handlers** (callback functions).
3. When an event is triggered, all registered callbacks will execute.

1️⃣ Code: Event System with Callbacks

cpp
CopyEdit
```cpp
#include <iostream>
#include <vector>
#include <functional>  // For std::function
using namespace std;

// Event class to store and call callback functions
class Event {
private:
    vector<function<void()>> callbacks;  // List of callback functions

public:
    // Method to add a callback function
    void addListener(function<void()> callback) {
        callbacks.push_back(callback);
    }

    // Method to trigger the event and call all registered callbacks
    void trigger() {
        cout << "Event triggered! Calling registered functions...\n";
        for (auto& callback : callbacks) {
            callback();  // Execute each registered function
        }
    }
};

// Sample event handlers
void onEvent1() {
    cout << "Handler 1: Event handled!\n";
}

void onEvent2() {
    cout << "Handler 2: Another event response!\n";
}

int main() {
    Event myEvent;  // Create an event

    // Register event handlers
    myEvent.addListener(onEvent1);
    myEvent.addListener(onEvent2);

    // Trigger the event
    myEvent.trigger();

    return 0;
}
```

2️⃣🔍 Explanation

- **`Event` Class:**
  - Stores a list of **callback functions** in a `vector<std::function<void()>>`.
  - `addListener(function<void()>)`: **Registers a function** to be called when the event triggers.
  - `trigger()`: Calls all registered functions.
- **Event Handlers (`onEvent1` and `onEvent2`)**
  - Simple functions that print a message when executed.
- **In `main()`**
  - We **register event handlers** (`onEvent1` and `onEvent2`).
  - When `trigger()` is called, both functions execute.

3️⃣ ✅ Output

vbnet
CopyEdit
```
Event triggered! Calling registered functions...
Handler 1: Event handled!
Handler 2: Another event response!
```

4️⃣ 🎯 Why Use Callbacks in Events?

1. **Dynamic Execution** → Call different functions **without modifying core logic**.
2. **Decoupling** → The event class **doesn't know** which functions will be executed.
3. **Scalability** → We can **register multiple event handlers** dynamically.

5️⃣ 🛠️ Bonus: Using Lambda Functions

We can also register **lambda functions** as callbacks dynamically.

cpp
CopyEdit
```
int main() {
    Event myEvent;

    myEvent.addListener([]() { cout << "Lambda function executed!\n"; });
    myEvent.addListener([]() { cout << "Another lambda responding!\n"; });

    myEvent.trigger();

    return 0;
}
```

✅ **Output:**

bash
CopyEdit
```
Event triggered! Calling registered functions...
Lambda function executed!
Another lambda responding!
```

🔹 Summary

- We built a **simple event system** using callbacks.
- **Callbacks enable flexible and modular event handling**.
- We used **function pointers**, `std::function`, and **lambdas**.