Project 1: Computational Cluster Building

Objective:
Develop a computational cluster to manage and process jobs based on their dependencies and the availability of computational resources.

Instructions:
1. Cluster Creation (Basic):
   - Set up at least two virtual machines on your system.
   - Designate one machine as the 'master' node responsible for resource allocation and job initiation.
   - The remaining machines will function as 'worker' nodes.
   - Create the capability to submit jobs to the cluster. These jobs can be basic programs for demonstration.
   - The master node will receive a list of jobs with varying runtimes. It should allocate jobs to idle worker nodes and retrieve the results. If the master node is idle, it can also function as a worker node.
   - Provide proof of the cluster's functionality using test cases, logs, and screenshots.

2. Job Dependencies (Advanced):
   - Extend the basic functionality to support job dependencies. Some jobs can operate on a 'first come, first serve' basis, while others might require the completion of certain tasks before starting.
   - Develop a format for job submission that includes a description of dependencies.
   - Ensure the master node launches jobs based on worker node availability and job dependencies.
   - Demonstrate the cluster's enhanced functionality with comprehensive test cases and logs.
   - Detect and handle cyclic dependencies, which are deemed invalid.
   - Implement mechanisms to gracefully manage failed or stuck jobs.

Project 2: Trading Strategy Simulation

Objective:
Design a system to simulate trading strategies using aggressive orders. Evaluate the strategies using a real-time Level 3 order book.

Instructions:

1. System Framework:
   - Utilize a Level 3 order book that updates in real-time with new orders, cancellations, or executions.
   - Each update to the order book should trigger a callback function. Similarly, a callback (`onSendOrder`) should be activated each time a strategy sends an order.
   - Your primary simulation logic should operate within these callbacks. Extend the system as needed for market data or trading strategy considerations.

2. Callback Functions:
   - For C++:

```cpp
void onOrderAdd(OrderBook& orderBook, const OrderInfo& orderInfo);
void onOrderCancel(OrderBook& orderBook, const OrderInfo& orderInfo);
void onOrderExecution(OrderBook& orderBook, const OrderInfo& orderInfo, const ExecutionInfo& executionInfo);
void onSendOrder(OrderBook& orderBook, const OrderInfo& orderInfo);
```

   - For Golang:

```go
func onOrderAdd(orderBook OrderBook, orderInfo OrderInfo) {}
func onOrderCancel(orderBook OrderBook, orderInfo OrderInfo) {}
func onOrderExecution(orderBook OrderBook, orderInfo OrderInfo, executionInfo ExecutionInfo) {}
func onSendOrder(orderBook OrderBook, orderInfo OrderInfo) {}
```

3. Data Structures:
   - `OrderBook`, `OrderInfo`, and `ExecutionInfo` are structured datasets containing relevant data. In each callback, the order book will have been updated with the latest order operations.

4. Submission:
   - Your submission should not necessarily be a fully compilable program, but it must be syntactically correct.
   - Use comments judiciously to explain your logic and design choices.
   - Ensure your code reflects not only programming competency but also a deep understanding of the simulation's precision.

- Submit your finalized code in either C++ or Golang.
