### C++ Monitor Class Design

```
Monitor
Semaphore gate
Semaphore urgent
int urgentcount
class Condition
void enter()
void leave()
void wait(Condition &c) = 0
void signal(Condition &c) = 0

class Condition
public:// Constructor
Condition();
protected:// Data members
int count;
Semaphore sem;
friend class Monitor;
friend class Mediator;
friend class Gladiator;
}; // Condition

protected:// Condition operations
virtual void wait(Condition &c) = 0;
virtual void signal(Condition &c) = 0;
bool queue(Condition &c) const;
}; // Monitor
```

### Monitor Base Class Declaration

```
class Monitor {
public:// Constructor
Monitor();

protected:// Entry/exit operations
void enter();
void leave();

protected:// Data members
Semaphore gate;
Semaphore urgent;
int urgentcount;

protected:// Nested condition class
class Condition {
public:// Constructor
Condition();

protected:// Data members
int count;
Semaphore sem;
friend class Monitor;
friend class Mediator;
friend class Gladiator;
}; // Condition

protected:// Condition operations
virtual void wait(Condition &c) = 0;
virtual void signal(Condition &c) = 0;
bool queue(Condition &c) const;
}; // Monitor
```

---

The implementor must put calls to these functions at the beginning and end of every monitor routine that they write.

The data needed to implement the monitor semantics. These values are accessed by the monitor routines `enter` and `leave` as well as the `wait` and `signal` routines.

This is a nested class because conditions have no meaning outside of a monitor.

Friend declarations are needed so that `wait` and `signal` (defined in `Mediator` and `Gladiator`) and `queue` (defined in `Monitor`) have access to the condition’s data members.

Wait and signal are pure virtual because their implementations differ in the different monitors.
**Mediator Class Declaration**

```cpp
// Mediator class

class Mediator: public Monitor {
public: // Constructor
    Mediator();

protected: // Condition operations
    virtual void wait( Condition &c );
    virtual void signal( Condition &c );
}; // Mediator
```

**Gladiator Class Declaration**

```cpp
// Gladiator class

class Gladiator: public Monitor {
public: // Constructor
    Gladiator();

protected: // Condition operations
    virtual void wait( Condition &c );
    virtual void signal( Condition &c );
}; // Gladiator
```
Base Class Implementation

Monitor::Monitor:
    gate( 1 ),
    urgent( 0 ),
    urgentcount( 0 ){
}

void Monitor::enter()
    gate.P();

void Monitor::leave()
    if( urgentcount > 0 ){
        urgentcount -= 1;
        urgent.V();
    } else {
        gate.V();
    }

Monitor::Condition::Condition:
    count( 0 ),
    sem( 0 ){  
}

bool Monitor::queue( Condition &c ) const {
    return c.count > 0;
}
Mediator Implementation

Mediator::Mediator():
    Monitor();

void Mediator::wait( Condition &c ){
    c.count += 1;
    if( urgentcount > 0 ){
        urgentcount -= 1;
        urgent.V();
    }
    else {
        gate.V();
    }
    c.sem.P();
}

void Mediator::signal( Condition &c ){
    if( c.count > 0 ){
        c.count -= 1;
        urgentcount += 1;
        c.sem.V();
        urgent.P();
    }
}

Gladiator Implementation

Gladiator::Gladiator():
    Monitor();

void Gladiator::wait( Condition &c ){
    c.count += 1;
    if( urgentcount > 0 ){
        urgentcount -= 1;
        urgent.V();
    }
    else {
        gate.V();
    }
    c.sem.P();
    urgent.P();
}

void Gladiator::signal( Condition &c ){
    if( c.count > 0 ){
        c.count -= 1;
        urgentcount += 1;
        c.sem.V();
    }
}