Dining Philosophers with a Monitor

**dp_mon.h:**

```cpp
enum Activity = {Thinking, Hungry, Eating};

typedef monitor {
private:
    const int NumPhil = 5; // number of philosophers
    Activity State[NumPhil] = {Thinking, Thinking, ..., Thinking};
    condition Self[NumPhil]; // for waiting philosophers
public:
    void PickUp(int i);
    void PutDown(int i);
private:
    void Test(int k);
} dp_mon;
```

**dp_mon.cpp:**

```cpp
#include "dp_mon.h"

void PickUp(int i) {
    State[i] = Hungry; // Philosopher i is now Hungry
    Test(i); // see if chopsticks are available
    if (State[i] == Eating) // if Philosopher i isn't Eating
        Self[i].wait(); // then he/she must wait
}

void PutDown(int i) {
    State[i] = Thinking; // Philosopher i is now Thinking
    Test((i - 1) % NumPhil); // see if right and left neighbors
    Test((i + 1) % NumPhil); // want to eat
}

void Test (int k) { // Philosopher k may eat if:
    if (((State[(k - 1) % NumPhil] != Eating) && // right neighbor isn't eating
         (State[k] == Hungry) && // Philosopher k is Hungry
         (State[(k + 1) % NumPhil] != Eating)) { // left neighbor isn't eating
        State[k] = Eating; // Philosopher k may now eat
        Self[k].signal(); // wake her/him up if necessary
    }
}
```
dp_main.cpp:

#include "dp_mon.h" // monitor definition

dp_mon dp; // declare a monitor

void Philosopher(int i);

void main() {
    concurrent{ // spawn off concurrent Philosophers
        Philosopher(0); // spawn off concurrent Philosophers
        Philosopher(1);
        Philosopher(2);
        Philosopher(3);
        Philosopher(4);
    }
}

void Philosopher(int i) {
    do {
        ThinkABit( );
        dp.PickUp(i); // acquire chopsticks
        EatAWhile( );
        dp.PutDown(i); // release chopsticks
    } while (1);
}