Dining Philosophers with Semaphores

**dp_sem1.h:**

```c
const int NumPhil = 5;
enum Activity = {Thinking, Hungry, Eating};
Activity State[NumPhil] = {Thinking, . . . , Thinking};
semaphore Mutex = 1, // for mutual exclusion
    S[NumPhil] = {0, 0, 0, 0, 0};
void Philosopher (int i);
void TakeChops (int i);
void ReleaseChops (int i);
void Test (int k);
```

**dp_sem1.cpp:**

```c
#include "dp_sem1.h"

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

void Philosopher (int i) {
    do {
        ThinkABit();
        TakeChops (i);
        EatAWhile();
        ReleaseChops (i);
    } while (1);
}
```
void TakeChops(int i) {
    Mutex.wait(); // wait if another Philosopher is accessing chopsticks
    State[i] = Hungry; // Philosopher i is now Hungry
    Test(i); // attempt to acquire chops
    Mutex.signal(); // release mutual exclusion
    S[i].wait(); // wait if chops were not acquired
}

void ReleaseChops(int i) {
    Mutex.wait(); // wait if another Philosopher is accessing chopsticks
    State[i] = Thinking; // Philosopher i is Thinking
    Test((i - 1) % NumPhil); // see if neighbors want to eat
    Test((i + 1) % NumPhil);
    Mutex.signal(); // release mutual exclusion
}

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