midway through dining philosophers problem

issue w/ current soln: deadlock

to introduced "permit" or "permissions slip"

assign 2 permission slips, or 4?

4: as high as possible w/o introducing deadlock

numForks - 1

not allowed to do the job of the thread manager & decide how threads are scheduled

unless you "know more" than the OS

```
int main(-->) {
    size_t permits kNumForks - 1;
    mutex forks[kNumForks], permitsLock;
    thread philosophers[kNumPhilosophers];



grantPermission(size_t& permits, mutex& permitsLock){
    permitsLock.lock();
    permits ++;
    permitsLock.unlock();
}

waitForPermission()
    //keep on looping until permit is available
    // "willing to keep on polling permits until I'm sure its not zero
    while (true) {
        permitsLock.lock();
        if (permits > 0) break;
        permitsLock.unlock();
        sleep_for(10); // 10 millisecond nap! allow other threads to use lock
    }
    permits--;
    permitsLock.unlock();
```

we'd want something like sleep_until(permits > 0); ...

problem... this relies on "busy waiting"

also nothing special about 10 ms... we're making up numbers

want to be as sensible about CPU mgmt as possible

Condition Variables

condition_variable_any cv;

designed to monitor the truth/falsity of a condition you need in order to proceed

```
class condition_variable_any {
    private:
        //blissfully unaware
    public:
        void wait(mutex& m);
        void notify_one(); //if there are multiple threads embedded in a wait call, pick 1 at random & let it proceed
        void notify_all();
```

template < typename Pred > void wait(mutex& m, Pred pred);

alternatively: cv.wait(m, [&permits]() { return permits > 0; });

raw:

```
waitForPermission(-->) {
    m.lock();
```

```
    while (permits == 0) {
        cv.wait(m); // don't want to go to sleep holding a lock! wait unlocks it for you
    }               // CPU intelligently removes lock when thread swapped off CPU - looks to the thread like they always have permission
    permits ++;
    m.unlock();
}

grantPermission (...) {
    m.lock();
    permits ++;
    if(permits == 1) cv.notify_all();   // why not notify one? could have gone w/ just 2 permission slips...
    m.unlock();                         // think of them like push notifications to your phone
}
```

`lock_guard<mutex> lg(m);`
- used in place of `m.lock();` - useful for more complex threading examples
  - no way to leave the function w/o unlocking
- don't have to explicitly unlock

Class `semaphore` ← a class for the wait for / grant permission functionality above

```
public:
    semaphore(int count);   // number of available resources
    void wait();            // ← decrements count
    void signal();          // ← increments & notifies all
private:
    int count;
    mutex m;
    condition_variable_any cv;
```

↳ a wait & notify mechanism

↳ a generic "resource counter" class

* do minimal amount of work needed to remove deadlock
* try to lock down resources for as narrow a time window as possible