in C, pointers and arrays are equiv

```
type x * x;

x[i] = * (x + i)
```

when adding to pointer, there is implicit multiplication by sizeof type

last question on hw 1 is new

more synchronization
last time: mutex, semaphore, monitor
today: higher level properties

mutex, semaphore, monitor have equal power
prove by construction

construct sem w/ 2 mutexes
- use mutex to guard state of queue
  mutex m, q
  int count

operations:

```java
// wait:
  m.lock
  tmp = --count
  m.unlock
  if (tmp < 0)
    q.lock()
Signal:
  m.lock
  if (count < 0)
    q.signal
  count ++
  m.unlock
```
Constructing a semaphore as a monitor

```c
mun {
    int count
    cond C
    wait () {
        if --count < 0
            wait (C)
    }
    signal (X)
    count ++
    signal C
}
```

Constructing monitor with semaphore

- implicit mutex of monitor (exactly like calling method synchronized in Java)
- sem m = sem(1)
- cond C1, C2, ... → C1.q = sem(0), ...
- int C1::count = 0
- F() → m.lock()
- F()
- m.unlock()
- C.wait → C.count++
- m.unlock
- c.q, wait
- m lock
- C.count --

Count is count of how many threads are waiting
Thu 1/15/2009  CS6712

Construct monitor w/ semaphore calls

c.signal \rightarrow \begin{cases} 
\text{if } c\text{.count }\geq 0 \\
\text{c.q, signal(c)} 
\end{cases}

- Synchronization mechanism in Java is essentially a monitor

\underline{Construct java sync w/ monitor}

- implicit mutex \rightarrow java sync
  - condition c_1, c_2 \ldots \rightarrow self
- POSIX threads

\underline{Construct POSIX threads w/ monitor}

- implicit mutex \rightarrow explicit m
- cond c_1, c_2 \ldots \rightarrow pthread_cond_\text{wait}(c_1, c_2, \ldots)
- F() \rightarrow pthread_mutex_lock(m)
- F() \rightarrow \text{unlock}(m)
- c.wait \rightarrow cond_wait(c, m)
- c.signal \rightarrow cond_signal(c)
last part of Ch 6:

transactional memory (optimistic concurrency vs pessimistic)
-
have hw detect conflict and rollback if necessary

benefits of transactional mem:
-
comes for free w/ modern cache hierarchy
Thu 11/15/2007  CSG 712

Data structure consistency

Failure -> Concurrency

Application: File systems and databases

Locking: Synchronization

Ordering: Soft updates (BSD) — allows OS to buffer blocks to be written to disk but allows some ordering to preserve consistency

Journaling: Ext3, XFS, ...

Write transactions for file systems

Dec 2008

Group of writes that need to finish together

Flash makes this easy — mapping table

- Writing in log structure
- Old data is still accessible