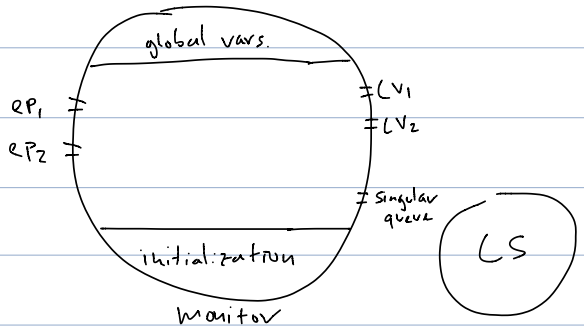
ADT - Abstract Data Type

1)

2) Has entry points

monitor



bounded-buffer problem (multiple producers, multiple consumers)

```
monitor Bounded-Buffer
char buffer[n];
int nextin=0, nextout=0, full_cnt=0;
condition notempty, notfull;
deposit(char c){
    if(full_cnt==n) notfull.wait;
    buffer[nextin]=c;
    nextin=(nextin+1) %n;
    full_cnt++;
    notempty.signal;
}
```

```
cobegin
  P1 // P2 // P3 // C1 // C2
coend

P1: .....
    deposit(c);
    .....

C1: .....
    remove(c);
    .....
```

```
remove(char c){
```

```
    if( full_cnt == 0) notempty.wait;
      c = buffer[nextout];
    nextout = (nextout+1) % n;
     full_cnt --;
    notfull.signal;
}
```

```
monitor readers-writers {
  int read_cnt = 0, writing = 0;
  condition OK_to_read, OK_to_write;
    start_read() {
      if (writing)
        OK_to_read.wait;
      read_cnt ++;
      OK_to_read.signal;
    }
  finish_read(){
    read_cnt --;
    if(read_cnt == 0)
       ok_to_write.signal;
  }
  start_write(){
    if((read_cnt != 0) || writing)
      Ok_to_write.wait;
      writing = 1;
```

```
    }
    finish_write(){
      writing = 0;
      if(!empty(Ok_to_read))
        Ok_to_read.signal;
      else
        Ok_to_write.signal;
    }
}
```

reader:

```
start_read();
file access;
finish_read();
```

---

(1) entry points mutual exclusion

```
bP(mutex);
```

—

```
body of f;
```

—

```
if(next_cnt>0)
    V(next);   <- semaphore
else
    bV(mutex);
```

(2) Condition variables

```
x_count ++;
if(next_count>0)
    V(next);
else
    bV(mutex);
P(x_sem);
x_count--;
```

(3) Signaler queue

```
if(x_count>0) {
    next_count ++;
    v(x_sem);
    p(next);
    next_count--;
}
```

producer-consumer problem in C

```
pthread_cond_wait
cond_t empty, fall;
mutex_t mutex;
void *producer(void *arg) {
    item_t i;
while(1) {
```

```
        pthread_mutex_lock (&mutex);
        while(count==MAX)
            pthread_cond_wait(&empty, &mutex);
        put (i);
        pthread_cond_signal (&full);
        pthread_mutex_unlock(&mutex);
    }
}
void *consumer(void *arg){
    item_t i;
    while(1){
        pthread_mutex_lock(&mutex);
        while(count==0)
            pthread_cond_wait(&full,&mutex);
        int tmp=get();
        pthread_cond_signal(&empty);
        pthread_mutex_unlock(&mutex);
    }
}
int buffer[MAX];
int fill=0;
int use=0;
int count=0;
void put(int value) {
    buffer[fill]=value;
```

```
int get(){
    int tmp=buffer[use];
```

fill = (fill+1) % MAX;	use = (use+1) % MAX;
count++;	count--;
}	return tmp;
	}

MAX = 1

consumer(C1, C2, C3) → producer(P1, P2, P3, P4, P5, P6, P1, P2, P3)

→ consumer2(C1, C2, C4, C5, C6)

→ consumer1(C4)

reusable resources

P1: —
open(F1, w);
open(F2, w);
—

P2: —
open(F2, w);
open(F1, w);

Consumable resources