5	/2	/1	//
J	/ J	/ L	<i>/</i> U

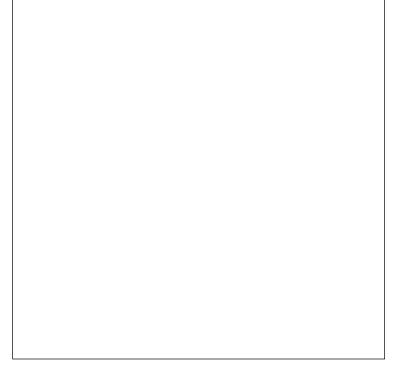
1. Explain why the following code snippet is **not** atomic?

balance = balance + 1

8/5/3/0

2. In the following code snippet what is the expected output of the program? Is the expected output consistent across multiple runs of the program? Explain?

```
int shared;
void * fun(void * args){
    int i;
    for(i=0;i<100;i++){
        shared++;
    return NULL;
}
int main(){
    pthread_t t1,t2;
    pthread_create(&t1, NULL, fun, NULL);
    pthread_create(&t2, NULL, fun, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("shared: %d\n", shared);_
}
```



5/3/1/0 3. In the above code snippet **circle** the critical section. Below, explain describe a critical section.

ı			
ı			
ı			
ı			
ı			
ı			
ı			
ı			
ı			
ı			
ı			
ı			
ı	1		

NAME:			

4. Consider the naïve locking solution used for the thread startup 5/3/1/0 routine from the previous program: Does this provide proper locking?

Why or why not, explain.

```
int shared;
int lock;
void * fun(void * args){
    int i:
    for(i=0;i<100;i++){
        while(lock > 0);//spin
        lock = 1; //set lock
        shared++; //increment
        lock = 0; //unlock
    }
    return NULL;
}
```

5. Explain why using a mutex avoids issues of a lack of atomicity in 5/3/1/0 lock acquisition?

7/5/3/0 6. Which type of locking strategy, coarse or fine, does the following code block use? Is there a possibility of a more efficient locking strategy? Explain.

pthread_mutext_t lock; int avail = MAX_FUNDS; int local_1 = 0; int local 2 = 0; void * fun(void * args){ int v,i; for(i=0; i < 100; i++){

return NULL;

v = random() % 100;

```
pthread_mutext_lock(&lock);
if(avail - v > 0){
    avail -= v;
if(random() % 2){
    local_1 += v;
}else{
    local_2 += v;
}
pthread_mutext_unlock(&lock);
```

NAME:			

10/8/6/3/0

7. Based on the code example from Question 6, fill in locking code to provide a more efficient locking strategy.

```
int avail = MAX_FUNDS;
int local_1 = 0;
int local 2 = 0;
void * fun(void * args){
    int v,i;
    for(i=0; i < 100; i++){
        v = random() % 100;
        if(avail - v > 0){
            avail -= v;
        }
        if(random() % 2){
            local_1 += v;
        }else{
            local_2 += v;
       }
    return NULL:
```

5/3/1/0 8. What is deadlock and provide a small (pseudo-)code example of how deadlock can arrise from coarse grain locking.

•	

5/3/1/0 9. Provide an example of deadlock avoidance when there is a natural ordering of lockable objects.

10/8/6/3/0	10. Provide a detailed description of the problem setup for the dining philosophers problem:
25/23/20/	15/10/5/0
	11. In pseudo code, provie a solution to the dining philosophers problem that avoids deadlock:
10/8/6/3/0	12. Explain your solution and argue that it wll always avoid deadlocks regardless of the number of philosophers.

4 of 4

___/45

NAME: _