1. Atomic Update with Journaling

We are trying to transfer $5 from Alice to Bob. The application uses the extra blocks as a journal. The disk thus goes through the following series of states:

Time	**Block 0: Alice**	**Block 1: Bob**	**Block 2: extra**	**Block 3: extra**	**Block 4: extra**
1	12	3	0	0	0
2	12	3	**7**	0	0
3	12	3	7	**8**	0
4	12	3	7	8	**1**
5	**7**	3	7	8	1
6	7	**8**	7	8	1
7	7	8	7	8	**0**

If we crash, the following recovery function (pseudo code) runs:

```
void recovery() {
     if (read(block-4) == 1) {
          A = read(block-2)
          B = read(block-3)
          write(A to block-0)
          write(B to block-1)
          write(0 to block-4)
     }
}
```

Show the resulting state after running recovery() for each time in the above table:

Time	**Block 0: Alice**	**Block 1: Bob**	**Block 2: extra**	**Block 3: extra**	**Block 4: extra**
1					
2					
3					
4					
5					
6					
7					

2. Atomic Update with Copy-On-Write

One disadvantage with journaling is that new data is written twice (or the old data must be backed up). Design a new algorithm with fewer I/O that only writes new data once.

Hints:
- with read_alice(), etc., you don't always have to store the latest data in same block
- a solution that doesn't require a recovery method is possible (and desirable)

```
void update_accounts(int alice_cash, int bob_cash) {







}

int read_alice() {







}

int read_bob() {







}
```