Problem 1

Part b

```
monitor bank {
    int balance = 1000000;
    cond delay;

    proc deposit( int amount ) {
        balance += amount;

        # Determine if I'm waking someone up
        if( amount(delay) <= balance) {
            signal(delay);
        }
    } # deposit

    proc withdraw( int amount ) {
        while ( amount > balance ) {
            wait( delay );
        }

        balance -= amount;
        return amount;
    } # withdraw
} # bank monitor
```

Part c

```
monitor bank {
    int balance = 1000000;
    queue withdrawAmounts;
    cond delay;

    proc deposit( int amount ) {
        balance += amount;

        # Determine if I'm waking someone up
        int nextWithdraw = withdrAwamounts.head();
        if( nextWithdraw <= balance ) {
            signal(delay);
        }
    } # deposit
```

```
    proc withdraw( int amount ) {
        withdrawAmounts.push(amount);
        while ( amount > balance ) {
            wait( delay );
        }

        withdrawAmounts.pop();

        balance -= amount;
        return amount;
    } # withdraw
} # bank monitor
```

Problem 2

Our implementation of this will make use of a list of queues as our buffer. Each time the producer produces an item, it will be added to the queues for all consumers waiting. This way, when a consumer finishes processing an item, it can go fetch the next item to process in its queue. This allows each consumer to handle each item in its own time.

```
monitor producerConsumer {
    buffer = queue[n]
    itemCondition = cond[n]

    def produceItem(item) {
        for i in range(0, n):
            queue = buffer[i]
            queue.add(item)
            signal(itemCondition[i])
    }

    def consumeItem(producerId) {
        while empty(buffer[producerId]):
            wait(itemCondition);

        item = buffer[producerId].pop()

        # Consume item
    }
}
```

For this implementation, consumers will call consumeItem, which will wait until an item has been produced and then consume it. Producers should call produ-

ceItem, which will distribute the produced item to all consumers via a buffer of queues.

Problem 3

For this solution, our monitor will have three separate functions. The first function will handle calls to A, the second function will handle calls to B, and the last function will handle calls to either printer. Each function will rely on a separate condition variable.

```
monitor printer {
    boolean aAvailable = true, bAvailable = true;
    cond printerCondition;

    procedure printA() {
        while( ! aAvailable ) {
            wait(printerCondition);
        }

        aAvailable = false;

        // Print

        aAvailable = true;
        signal_all(printerCondition);
    }

    procedure printB() {
        while( ! bAvailable ) {
            wait(printerCondition);
        }

        aAvailable = false;

        // Print

        bAvailable = true;
        signal_all(printerCondition);
    }

    procedure printEither() {
        while( !aAvailable and !bAvailable ) {
            wait(printerCondition);
        }
```

```
        boolean usingA = false, usingB = false;

        if( aAvailable ) {
            aAvailable = false;
            usingA = true;
        } else if( bAvailable ) {
            bAvailable = false;
            usingB = true;
        }

        // Print

        if( usingA ) {
            aAvailable = true;
        } else if( usingB ) {
            bAvailable = true;
        }

        signal_all(printerCondition);
    }
}
```