Problem X (A: 3 points, B: 3 points, C: 4 points) What comes between?

Write a procedure called `between?` which takes three numbers as arguments, and returns true if and only if the second argument is between and not equal to the first and the third:

```
(between? 5 6 7) → #t
(between? 7 6 5) → #t
```

Part A: Write `between?` without using `if` or `cond`.

> Answers were all over the map here! A common solution, and one we like best:
>
> ```
> (define (between? bound1 mid bound2)
>     (or (and (< bound1 mid) (> bound2 mid))
>         (and (< bound2 mid) (> bound1 mid))))
> ```
>
> A lot of you did not notice that both `(between? 5 6 7)` and `(between? 7 6 5)` returned true! 2 points were deducted if you only handled one of these possibilities. 1 point was taken off if you had some issues with equal numbers.

Part B: Write `between?` without using `and` or `or`.

> ```
> (define (between? bound1 mid bound2)
>     (cond ((< bound1 mid) (> bound2 mid))
>           ((< bound2 mid) (> bound1 mid))
>           (else #f)))
> ```
>
> This part has the same grading standards as *Part A*. 1 point was taken off if the `else` case was not included, or, basically, not handling the case of equal numbers. 1 point was taken off if the code was overly complicated and unnecessarily long. You received either zero points or 0.5 points if you used `and` or `or` in this part when you were not allowed to.
>
> A note on coding style (no points taken off):
> A number of you wrote your `cond` clauses like this:
> `(cond ((< bound1 mid) (if (> bound2 mid) #t #f)) …)`
> The use of `if` is unnecessary and redundant here. Instead, the clause could have been shortened to `((< bound1 mid) (> bound2 mid))`
>
> Consider this: `(define (bar x y) (if (< x y) #t #f))`. The use of `if` is repetitive because it basically says to return true if `(< x y)` evaluates to true, and return false, if `(< x y)` evaluates to false. A better way to define `bar` would be `(define (bar x y) (< x y))`, so `bar` returns directly whatever the expression `(< x y)` evaluates to. Though both versions return the same results, style should certainly be a part of your consideration in coding.

Part C: Write a suite of test cases for `between?`. Make sure you test the possible sets of parameters exhaustively as possible, in order to test different ways the code could be written.

Also, make sure you describe what the result of the call should be!

> The set of test cases is meant to test other people's code, not your own. This means that you cannot assume anything about how they coded this procedure. Thus the test cases must be as

exhaustive as possible to handle the different ways other students could have coded this same problem.

One point was given to the two cases that can make `between?` return true.

```
        (between? 5 6 7) → #t
        (between? 7 6 5) → #t
```

Two points were assigned to the handling of out of bounds cases. This means that `mid` is outside the range of `bound1` to `bound2`, where `bound1` and `bound2` are not equal to each other. In order to receive two points, you needed a test case with a `mid` that is greater than both `bound1` and `bound2`, and another test case with a `mid` that is less than both `bound1` and `bound2`.

```
        (between? 5 9 7) → #f
        (between? 7 9 5) → #f
        (between? 5 1 7) → #f
        (between? 7 1 5) → #f
```

One point was allocated to the following set of test cases. This group of tests check for equal numbers among the three arguments. You needed to check for at least two of these possibilities.

```
        (between? 7 6 6), (between? 6 6 7) → #f
        (between? 7 7 6), (between? 6 7 7) → #f
        (between? 5 8 5), (between? 5 3 5) → #f
        (between? 2 2 2) → #f
```