
CSE 331
Software Design & Implementation

Autumn 2023
Section 10 – Final Review

Administrivia

- HW9
 - Due tomorrow @11pm
- Final
 - **Tuesday, 12/12, MGH 389**
 - Exam A: **2:30 – 4:20**
 - Exam B: **4:30 – 6:20**
 - Please arrive a couple minutes early
 - No notecards, all needed definitions will be included
- Final review session
 - **Monday, 12/11, 7-8:30pm**
 - **CSE1** (Allen), across breakout rooms
 - Bring questions related to practice exams or general concepts

Course Evals!!

- Please fill them out!
- We appreciate the feedback (TAs and Kevin both)
 - We will actually read them, so any suggestions will be considered!
- **If 50% of responses are completed, we will give everyone an additional day to complete HW9!!** 🎉🎉
 - New on time deadline would be Saturday, 12/9
 - Deadline with late day would be Sunday, 12/10

Final topics

- **All topics covered by midterm are fair game**
(Remember, midterm was largely final practice)
 - Reasoning about Recursion
 - Reasoning about Loops
 - Writing Methods
 - Testing
- **New topics that may be included:**
 - Writing the code of a for loop, given the loop idea and invariant.
 - Writing or proving correct the methods of classes that implement mutable ADTs

ADT

- **MutableIntCursor ADT** represents a list of integers with the ability to insert new characters at the “cursor index” within the list.
 - cursor index can be moved forward or backward
- **LineCountingCursor** implements MutableIntCursor by:
 - using the abstract state (an index and a list of values) as its concrete state
 - + records the number of newline characters (so class can easily, quickly determine the number of lines in the text)
- **Reminder:** familiar functions on last page of WS!

Problem 1b

Pre: $\text{this.numNewlines}_0 = \text{count}(\text{this.values}_0, \text{newline})$ }

Explain, in English, why the facts listed in **Pre** will be true when the function is called:

Problem 1c

Post: $\text{this.index} = \text{this.index}_0 + 1$ and $\text{this.values} = \text{concat}(P, \text{cons}(m, S))$
and $\text{this.numNewlines} = \text{count}(\text{this.values}, \text{newline})$
where $(P, S) = \text{split}(\text{this.index}_0, \text{this.values}_0)$ }

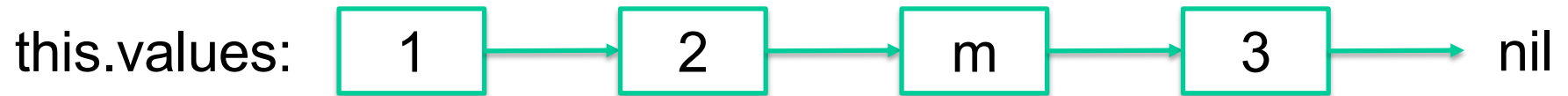
Explain, in English, why the facts listed in **Post** need to be true when the function completes in order for insert to be complete:

Problem 2

- Fill in the missing parts of the method so it is correct with the *given invariant*
- **Loop idea:**
 - skip past elements in `this.values` until we reach one that equals the given number or we hit the end
- **Invariant:**
 - `this.values` is split up between skipped and rest, with skipped being the front part in reverse order
 - no element of skipped is equal to the number `m`
- Do not write any other loops or call any other methods. The only list functions that should be needed are `cons` and `len`

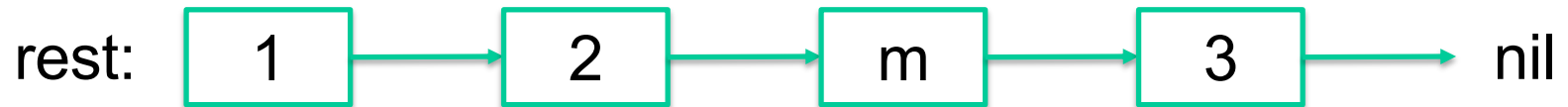
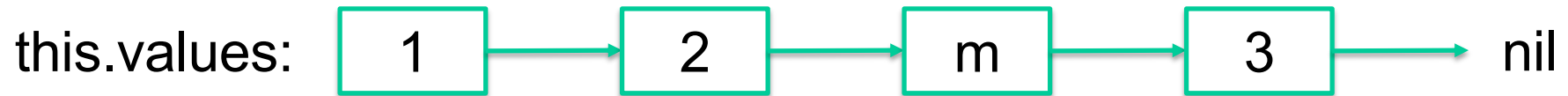
Problem 2

```
// Inv: this.values = concat(rev(skipped), rest) and  
//      contains(m, skipped) = false
```



Problem 2

```
// Inv: this.values = concat(rev(skipped), rest) and  
//      contains(m, skipped) = false
```

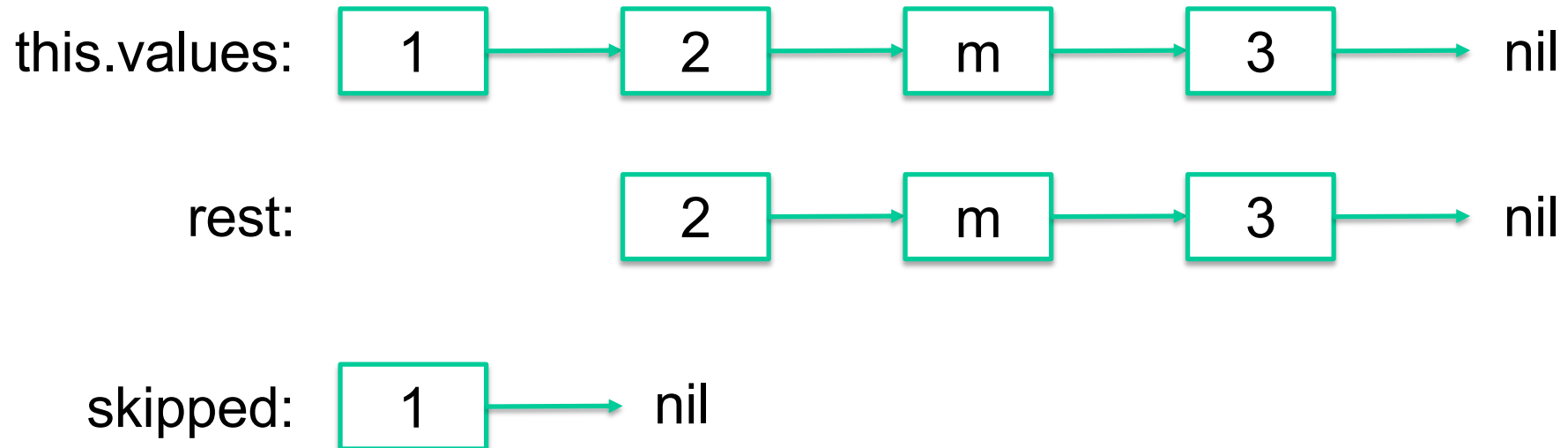


skipped: nil

Easiest way to satisfy the invariant

Problem 2

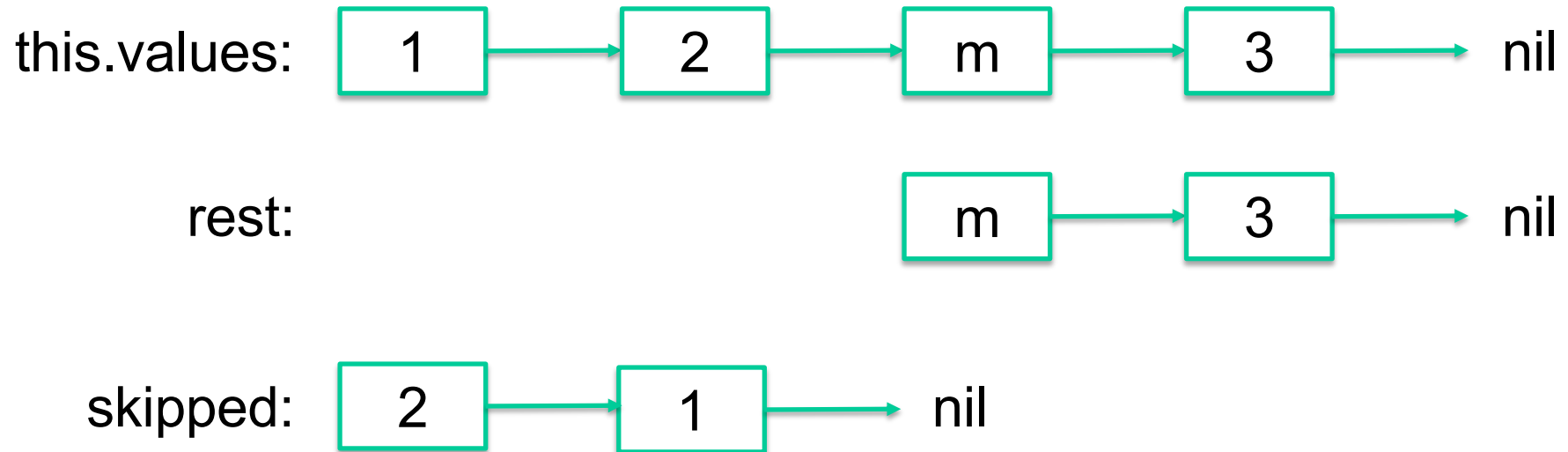
```
// Inv: this.values = concat(rev(skipped), rest) and  
//      contains(m, skipped) = false
```



While `rest.hd != m` (need to check `rest != nil` first),
remove and append `rest.hd` to `skipped`
(`cons` adds to front which reverses the list which matches
the invariant)

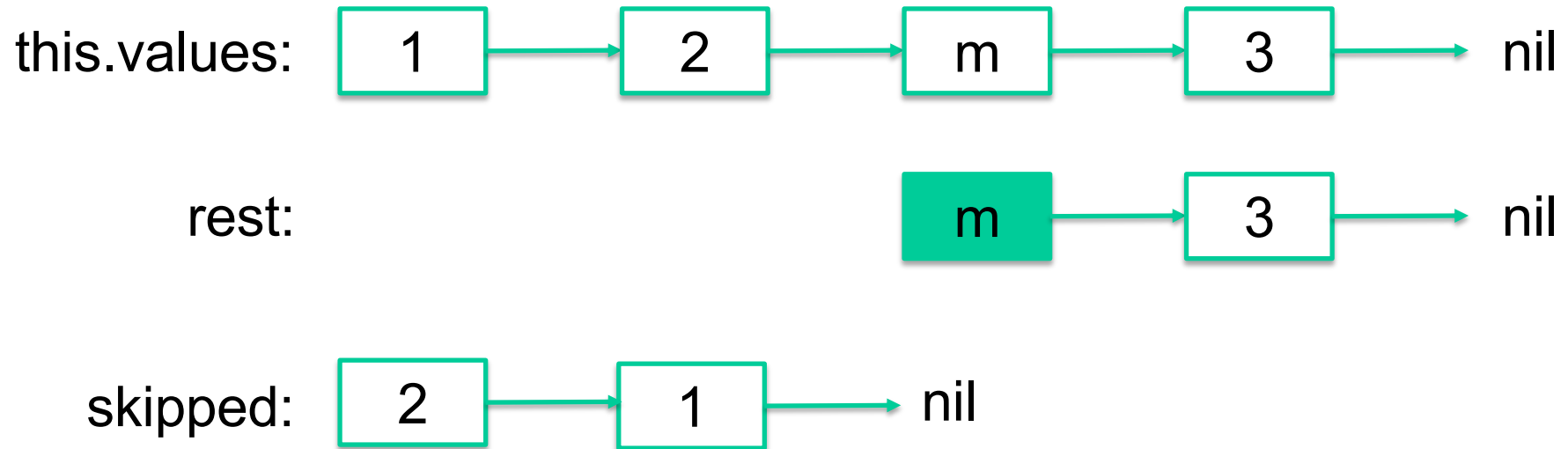
Problem 2

```
// Inv: this.values = concat(rev(skipped), rest) and  
//      contains(m, skipped) = false
```



Problem 2

```
// Inv: this.values = concat(rev(skipped), rest) and  
//      contains(m, skipped) = false
```



When we exit the loop

- If rest = nil then we didn't find m
- Otherwise, Index of m is the length of the skipped list

Problem 2

```
// Move the index to the first occurrence of m in values.
moveToFirst = (m: number): void => {
  let skipped: List<number> = _____;
  let rest: List<number> = _____;

  // Inv: this.values = concat(rev(skipped), rest) and
  //       contains(m, skipped) = false
  while (_____ ) {

  }

  if (rest === nil) {
    throw new Error('did not find ${x}');
  } else {
    this.index = _____;
  }
};
```

Problem 3

- Fill `removeNextLine` so it removes all the text on the next line: text between the first and second newline characters *after* the cursor index
 - remove second newline, but leave cursor index in place
 - If there are no newlines after cursor, then do nothing
 - If there is only one newline after cursor, remove all text after it
- method of `LineCountingCursor`, so you can access `this.index` and `this.values`
- Can use any Familiar List Functions from final page and assume they've been translated to TS
- Hint: `split-at` function from HW5 may be useful, assume the TS translation of it is called `splitAt`

Problem 3

```
// Removes the line of text after the one containing the cursor index  
removeNextLine = (): void => {
```



Index

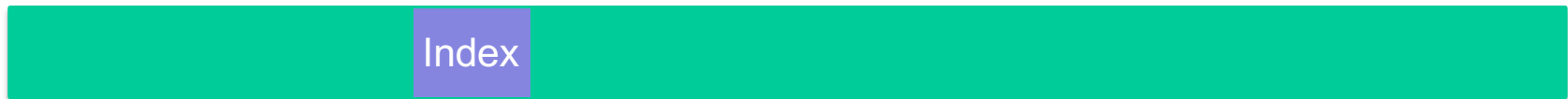
Problem 3

```
// Removes the line of text after the one containing the cursor index  
removeNextLine = (): void => {
```

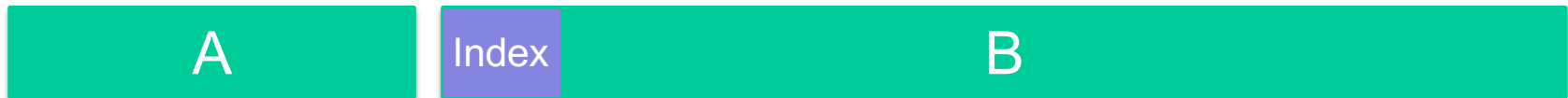


Problem 3

```
// Removes the line of text after the one containing the cursor index  
removeNextLine = (): void => {
```



[A, B] = split(index, values)



[C, D] = splitAt(B, newline)

No \n after cursor



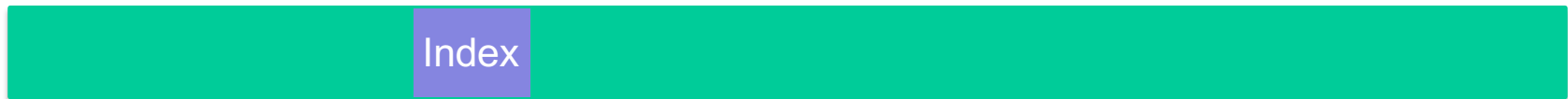
OR

\n after cursor

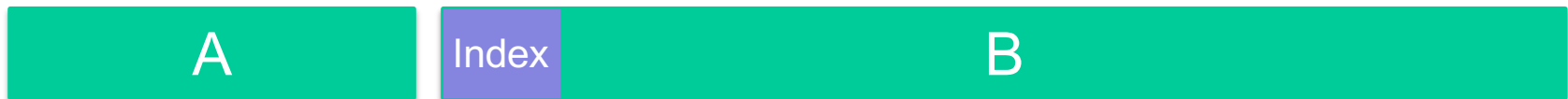


Problem 3

```
// Removes the line of text after the one containing the cursor index  
removeNextLine = (): void => {
```



[A, B] = split(index, values)

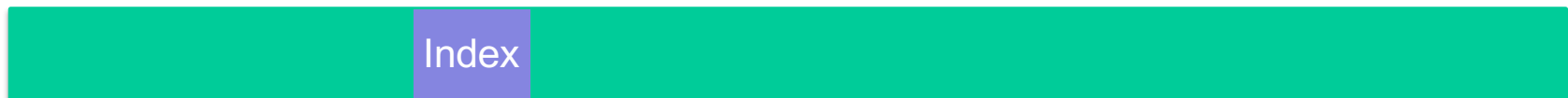


[C, D] = splitAt(B, newline)

No \n after cursor

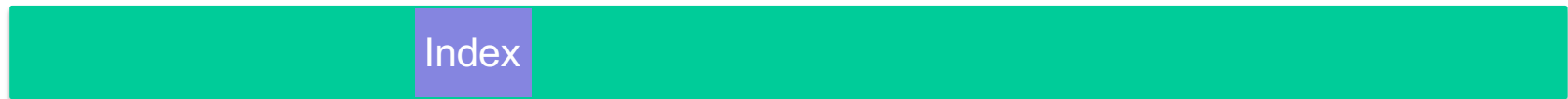


No change:

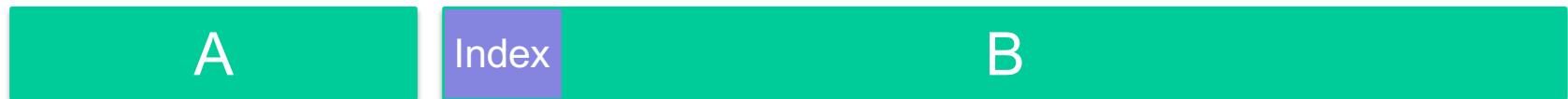


Problem 3

```
// Removes the line of text after the one containing the cursor index  
removeNextLine = (): void => {
```



[A, B] = split(index, values)



[C, D] = splitAt(B, newline)



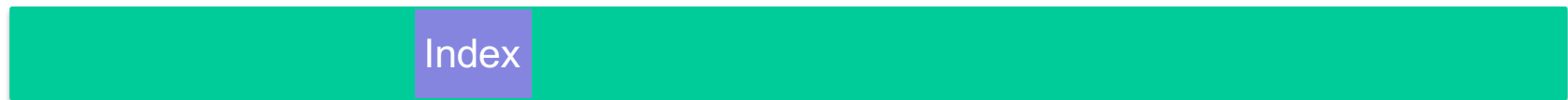
[E, F] = splitAt(D.tl, newline)

No second \n
OR
Second \n

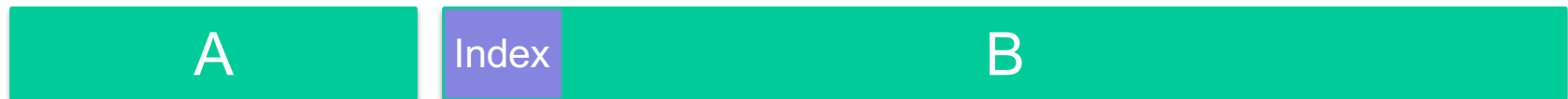


Problem 3

```
// Removes the line of text after the one containing the cursor index  
removeNextLine = (): void => {
```



[A, B] = split(index, values)



[C, D] = splitAt(B, newline)



[E, F] = splitAt(D.tl, newline)

No second \n

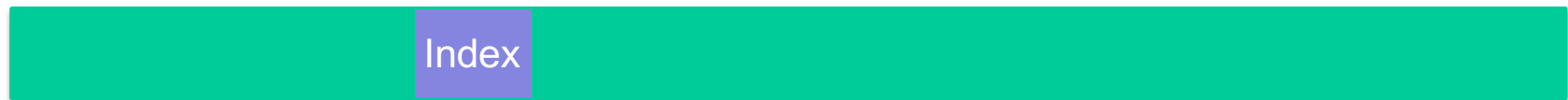


Remove everything after \n

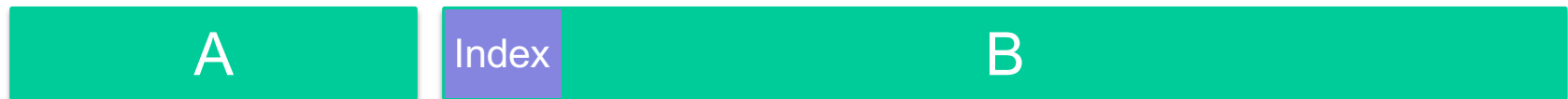


Problem 3

```
// Removes the line of text after the one containing the cursor index  
removeNextLine = (): void => {
```



$[A, B] = \text{split}(\text{index}, \text{values})$



$[C, D] = \text{splitAt}(B, \text{newline})$



$[E, F] = \text{splitAt}(D.\text{tl}, \text{newline})$



Remove next line:



Problem 3

```
// Removes the line of text after the one containing the cursor index
```

```
removeNextLine = (): void => {
```

```
};
```