CS 61B Spring 2026

Testing, IntLists, SLLists

Discussion 02: February 2, 2026

1 Interweave

Implement `interweave`, which takes in an IntList `lst` and an integer `k`, and *destructively* interweaves `lst` into `k IntList`s, stored in an array of `IntList`s. Here, destructively means that instead of creating new `IntList` instances, you should focus on modifying the pointers in the existing `IntList lst`.

Specifically, we require:

- It is the **same** length as the other lists. You may assume the `IntList` is evenly divisible.
- The first element in `lst` is put in the first index of the array of `IntList`s. The second element is put in the second index. This goes on until the array is traversed, and then we wrap around to put elements in the first index of the array.
- Its ordering is consistent with the ordering of `lst`, i.e. items in earlier in `lst` must **precede** items that are later.

For instance, if `lst` contains the elements [6, 5, 4, 3, 2, 1], and `k` = 2, then the method should return an array of `IntList`, [6, 4, 2] at index 0, and [5, 3, 1] at index 1.

In the beginning, we reversed the `IntList lst` destructively, because it's usually easier to build `IntList` backwards.

Hint: The elements in the array should track the head of the small `IntList` that they are building.

```
public static IntList[] interweave(IntList lst, int k) {
  IntList[] array = new IntList[k];
  int index = k - 1;
  IntList L = reverse(lst); // Assume reverse is implemented correctly

  while (_____) {

    IntList prevAtIndex = _____;

    IntList next = _____;

    _____;

    _____;

    L = _____;

    index -= 1;

    if (_____) {

      _____;

    }
  }
  return array;
}
```

2 Skippify

We have the following **IntList** class, as defined in lecture and lab, with an added **skippify** function. Suppose that we define two **IntList**s as follows.

```
IntList A = IntList.list(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
IntList B = IntList.list(9, 8, 7, 6, 5, 4, 3, 2, 1);
```

Fill in the method **skippify** such that the result of calling **skippify** on A and B are as below:

- After calling **A.skippify()**, A: (1, 3, 6, 10)
- After calling **B.skippify()**, B: (9, 7, 4)

```
public class IntList {
    public int first;
    public IntList rest;

    @Override
    public boolean equals(Object o) { ... }
    public static IntList list(int... args) { ... }

    public void skippify() {
        IntList p = this;
        int n = 1;
        while (p != null) {

            IntList next = ____________________________;

            for (_____________________________________) {

                if (_____________________________________) {

                    _____________________________________
                }

                _____________________________________
            }

            _____________________________________

            _____________________________________

            _____________________________________
        }
    }
}
```

3 Gridify

(a) Consider a circular sentinel implementation of an **SLList** of **Node**s. For the first **rows** * **cols** **Node**s, place the item of each **Node** into a 2D **rows** × **cols** array in row-major order. Elements are sequentially added filling up an entire row before moving onto the next row.

For example, if the **SLList** contains elements $5 \to 3 \to 7 \to 2 \to 8$ and **rows** $= 2$ and **cols** $= 3$, calling **gridify** on it should return this grid.

5	3	7
2	8	0

If the SLList contains fewer elements than the capacity of the 2D array, the remaining array elements should be 0; if it contains more elements, ignore the extra elements.

```
public class SLList {
  Node sentinel;

  public SLList() {
    this.sentinel = new Node();
  }

  private static class Node {
    int item;
    Node next;
  }

  public int[][] gridify(int rows, int cols) {

    int[][] grid = _____;

    _____;

    return grid;
  }

  private void gridifyHelper(int[][] grid, Node curr, int numFilled) {

    if (_____) {

      return;
    }


    int row = _____;

    int col = _____;


    grid[row][col] = _____;

    _____;

  }
}
```