In this lab we are going to be revisiting the Young tableaux that were the subject of the labs in week 4.

1	4	5	10	11
2	6	8		
3	9	12		
7				

The key features of a Young tableau are as follows:

- it consists of cells which are filled with integers, and arranged in left-justified rows,
- no row is longer than a preceding row,
- from left to right in any row, and down any column the integers are increasing,
- the set of integers used is $\{1, 2, \ldots, n\}$ where n is the number of cells.

In week 4 we represented tableaux using `int[][]`. A disadvantage of that approach is that arrays are static data structures and so it was not convenient to represent tableaux that changed, or to make structural modifications to a tableau. In this lab we'll use a different linked data structure that provides us with the dynamic behaviour required for such operations.

At first glance you might think that items could be added to a tableau using something like `addValue(x, y, value)` where x and y are the indexes of where you want the *value* to go. However, your mathematician friend comes back and says "That's not really how I want to add values to tableaux." She goes on to explain that a tableau is usually built up from a sequence of positive integers (frequently a permutation) by a process called *bumping*. This works as follows: to add a new value, v, to a tableau we first look in the first row of the tableau. If v can be added to the end of the row (because it's at least as big as anything in the row), then we do so and are done. Otherwise, we find the first element of the row, say w, that is bigger than v. We replace the cell containing w by one containing v ("v bumps w out of the row") and then proceed to try and insert w into the second row by the same means. We always add to the end of a row if we can (in which case we're done), or bump something into the next row. If absolutely necessary, we can add an extra row containing the final bumped element.

Problem description

Your task is to complete a `Tableau` class which implements the bumping algorithm described above. There will be no need for a global check like the `isTableau` method of week 4 because you will ensure that all the methods that actually change the structure of a tableau never create a structure that is not a tableau (e.g., by having cells in non-increasing order, or by having later rows longer than earlier ones). You can assume that the numbers given as input to your program will be a valid set of integers and don't need to perform any checking.

The basic idea is that a `Tableau` will be built up of objects of type `Cell` (`Cell` will be an inner class of `Tableau`). A `Cell` contains an `int` which is its value, together with references to four other `Cell` objects, its left, right, upper, and lower neighbours in the tableau (some of these might be `null`). Ensuring consistency among these references (if `a.left` is a cell `b` (not `null`) then `b.right` has to be `a` etc.) is one of the key things to take care of – and the way to do this is to make sure that all modifications of neighbour references are handled by methods that maintain the consistency.

Task (2%)

A skeleton (but compilable) version of the `Tableau` class is provided in the directory `/home/cshome/coursework/241/pickup/07` and also in the Files tab in Microsoft Teams . You need to complete the methods (testing as you go of course) that remain to be implemented.

`addToRow(Cell curr, int value)` This method takes a `cell` as a starting point and follows *right* pointing links until it finds a value which is greater than the given value or until the right pointing link is `null`. If it finds a bigger value then it replaces it with the given value and returns the previous value. If it comes to the end of the row it adds a new cell with the given value and returns `null`.

`addValue(Integer value)` This method takes care of the case where the tableau is empty, you must complete the implementation by adding the other cases. You can call the `addToRow` method to add the value to the first row. If `addToRow` returns `null` there is nothing more to do. If it returns a value then that value must be inserted into the row below. If the row below is empty then a new cell should be added as the only item in that row, otherwise just call `addToRow` again to add the returned value to the row below.

Marking

Check that your program works correctly, and then use the command `241-check` to make sure it passes all of our tests. If all is well then you can submit using `241-submit` as usual.

Reflection and extension

- Remember that in the conjugate of a tableau, rows become columns, and columns become rows. How could you convert a tableau into its own conjugate? Some care is necessary . . .
- If you were to add a *delete* operation to your tableau, how could you 'fix up' the tableau after performing a deletion so that the first three properties are maintained?