INSTRUCTIONS:

- **This is just a lab activity.** **Not for a grade.** Complete it as it is good practice for your future quizzes and exams.

- Download the code distribution available at:

  http://www.cs.umd.edu/class/fall2021/cmsc132-030X/projects/zipFiles/lab/LLlab.zip

- Import the above distribution as you usually import a project.
- **Immediately after importing the code, verify you can submit (check the submit server submission).**
- There are no public tests, just 10 release tests with 5 tokens per 24 hours.
- Work by yourself to see if you can figure it out.

Specifications

For this lab, you will implement methods for the class **LinkedList.** The code for the **SampleDriver** and **Student** class is given to you. You will find the classes in the **sysImplementation** package. A description of each method is provided below. Regarding the code you need to implement:

- You don't need to add comments to your code, but you must have good style (variable names, indentation, etc.)
- At this point you may want to look at the classes you will find in **sysImplementation**. There is a shell for each method you need to implement. You can also see the fields associated with each class.
- During the implementation of the above classe(s), you should **not add** any more fields (instance and static member variables) or any other methods to the class.
- You can assume parameters are valid unless we indicate otherwise (e.g., if the parameter is null throw …).
- You must provide an implementation for every method you need to implement, otherwise, your code will not work in the submit server. If you don't know what to do for a method, leave the body empty for void methods, and for a method returning a value, return any value that makes the code compile.

LinkedList Class Specification

The **LinkedList<T extends Comparable<T>>** class represents a generic class that can be used to make a linked list of type **T**. All the code needed is already in the file **and should not be altered in any way**. You only need to code up the two methods below:

1. **replaceLess** – You can decide what code goes in **replaceLess**, but you must not do any recursive work and **no looping** allowed (i.e. that means no for loop, enhanced for loop, while loop, or do while loop). Also, you **cannot call any method** other than at some point making **ONE** call to **replaceLessAux**. After **replaceLess** completes, all values "less than" the parameter **T** in the linked list should be replaced with the largest value in the list. A reference assignment is sufficient when replacing the data of a node with the largest value. If more than one largest value exits in the list, use the first one encountered when traversing from the head to do the replacement described above. If **replaceLess** is called using an empty list, just return.

2. **replaceLessAux** – This private auxiliary method called once by **replaceLess** does the actual recursive work to modify the list as described above. You can decide the type of the return value of **replaceLessAux.** You can also decide the type and number of parameters of **replaceLessAux,** but no more than 3 parameters maximum. Again, the code in **replaceLessAux** must be recursive **without any looping construct**s: no while, do while, for, or enhanced for loop. **If you use loops or have a non-recursive solution, you did not do it correctly.** The **only method** **replaceLessAux** can call is itself and compareTo of type T.

Student Class Specification

Nothing to code here. Do not modify it, otherwise, you will not pass release / secret tests.

Driver

```
public class SampleDriver {

	public static void main(String[] args) {

		LinkedList<Integer> intList = new LinkedList<Integer>();
		LinkedList<String> strList = new LinkedList<String>();
		LinkedList<Student> studentList = new LinkedList<Student>();

		String answer = "";

		answer += "==================intList Output======================\n";
		intList.add(5).add(8).add(14).add(16).add(7).add(21).add(17);
		answer+= "Before: "+ intList + "\n";
		intList.replaceLess(15);
		answer+= " After: "+ intList + "\n";


		answer += "==================strList Output======================\n";
		strList.add("dog").add("deer").add("bird").add("turtle").add("fish").add("bear").add("fox");
		answer+= "Before: "+ strList + "\n";
		strList.replaceLess("eel");
		answer+= " After: "+ strList + "\n";

		answer += "==================strList Output======================\n";
		studentList.add(new Student("Joe", 351)).add(new Student("Tom", 100)).add(new Student("Kate", 51));
		studentList.add(new Student("David", 75)).add(new Student("Joe", 580)).add(new Student("Kevin", 85));
		studentList.add(new Student("Lisa", 175));
		answer+= "Before: "+ studentList + "\n";
		studentList.replaceLess(new Student("Amy", 100));
		answer+= " After: "+ studentList + "\n";

		System.out.println(answer);
	}

}
```

Output

```
==================intList Output======================
Before: " 17 21 7 16 14 8 5 "
 After: " 17 21 21 16 21 21 21 "
==================strList Output======================
Before: " fox bear fish turtle bird deer dog "
 After: " fox turtle fish turtle turtle turtle turtle "
==================strList Output======================
Before: " Name: Lisa, Id: 175 Name: Kevin, Id: 85 Name: Joe, Id: 580 Name: David, Id: 75 Name: Kate, Id: 51 Name: Tom, Id: 100 Name: Joe, Id: 351 "
 After: " Name: Lisa, Id: 175 Name: Joe, Id: 580 Name: Joe, Id: 580 Name: Joe, Id: 580 Name: Joe, Id: 580 Name: Tom, Id: 100 Name: Joe, Id: 351 "
```