# 15-122: Principles of Imperative Computation

## Lab 6: Testing, Testing, 1 2 2

James Wu, Rob Simmons

**Collaboration:** In lab, we encourage collaboration and discussion as you work through the problems. These activities, like recitation, are meant to get you to review what we've learned, look at problems from a different perspective and allow you to ask questions about topics you don't understand. We encourage discussing problems with your neighbors as you work through this lab!

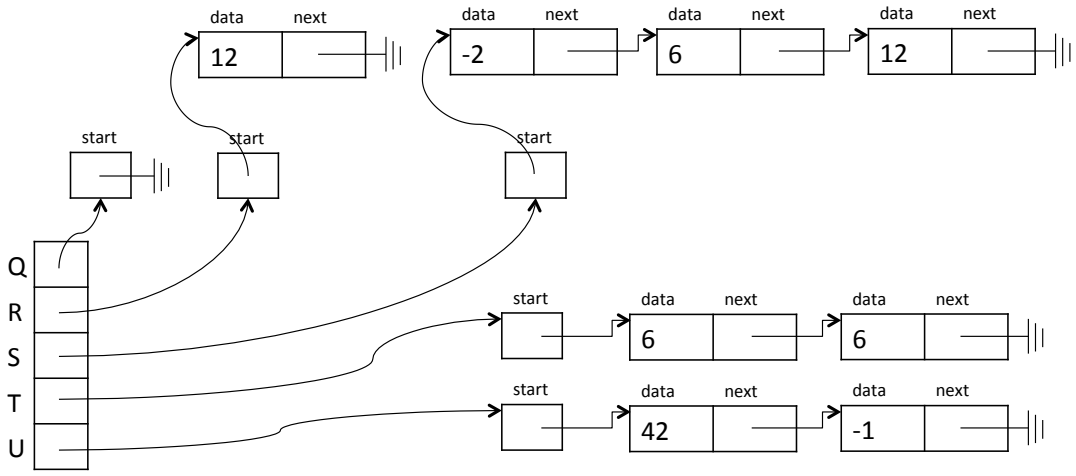
**Grading:** For two points, you must figure out the problems in at least the first three broken implementations of sortedlist. For three points, fix all broken implementations.

## Introduction

James's cloud-based motorcycle repair company C0 on Wheels is in trouble! In order to prepare for large number of clients and their repairs on their motorcycles, he implemented a new data structure to keep track of id numbers in sorted order. Unfortunately, all of his implementations seem to be having weird problems! He's hired you to figure out the problems with his code. The future of this SaaS(Scooter-repair as a Service) company is in your hands!

## Sorted Linked Lists

James's data structure involves sorted linked lists of unique integers. This is an invariant that should be maintained throughout the lab – all linked lists must be sorted and **must not contain duplicates**. Another thing that's different from the linked lists that you've seen in lecture and on homework is that there is no "dummy node" at the end of the list. The end of the linked list is reached when the `next` pointer on a node is NULL.




In the illustration above, `Q` is a sorted linked list containing no numbers, `R` contains just $12$, and `S` contains $-2$, $6$, and $12$. Neither `T` nor `U` is a valid sorted linked list (that is, `is_sortedlist(T)` and `is_sortedlist(U)` will both return `false`).

**Setup:** Copy the lab code from our public directory to your private directory:

```
% cd private/15122
% cp -R /afs/andrew/course/15/122/misc/lab-sorted .
% cd lab-sorted
```

**(2.a)** We would like you to write exhaustive test cases for sortedlist, to catch the bugs in the broken implementations.

The `listlib.c0` file contains the following specification functions and helper functions, which may be useful while testing:

```
bool is_segment(list* start, list* end);
bool is_sortedlist(sortedlist* L);
sortedlist* nil()                          /*@ensures \result != NULL; @*/;
sortedlist* cons(int i, sortedlist* S)     /*@requires S != NULL; @*/;
string to_string(sortedlist* S)            /*@requires S != NULL; @*/;
```

For example, `cons(-2, cons(6, cons(12, nil())))` creates the sorted linked list `S` from the example above.

Write your tests in the `sortedlist-test.c0` file in the directory `lab-sorted`. Inside the folder, you'll find a few bad implementations of sortedlist, named sortedlist-bad1.c0, sortedlist-bad2.c0, etc. To get credit for this part, compile your code with the commands:

```
% cc0 -x listlib.c0 sortedlist-bad1.c0 sortedlist-test.c0
% cc0 -x listlib.c0 sortedlist-bad2.c0 sortedlist-test.c0
% cc0 -x listlib.c0 sortedlist-bad3.c0 sortedlist-test.c0
% cc0 -x listlib.c0 sortedlist-bad4.c0 sortedlist-test.c0
% cc0 -x listlib.c0 sortedlist-bad5.c0 sortedlist-test.c0
```

Your code should indicate a problem for each of the bad implementations. Figure out what's wrong with each version of the code. Tell your TA the bugs in the code for credit!

Some hints:

- To get the most out of this lab, don't spend a long time reading the bad implementations! Some of the bugs are quite subtle, and we also want to be teaching you to write good tests.

- This might be obvious, but be thorough with your edge cases! Make sure the linked list behaves exactly as specifies.

- Some tests cause null pointer dereferences. Some tests cause contract failures. Others cause test failures. Debug each one separately.

- Some bugs "cancel" each other out and make the lists appear to work correctly and not fail any contracts. The later versions of sortedlist may have multiple errors!