15-122: Principles of Imperative Computation

Lab Week 6

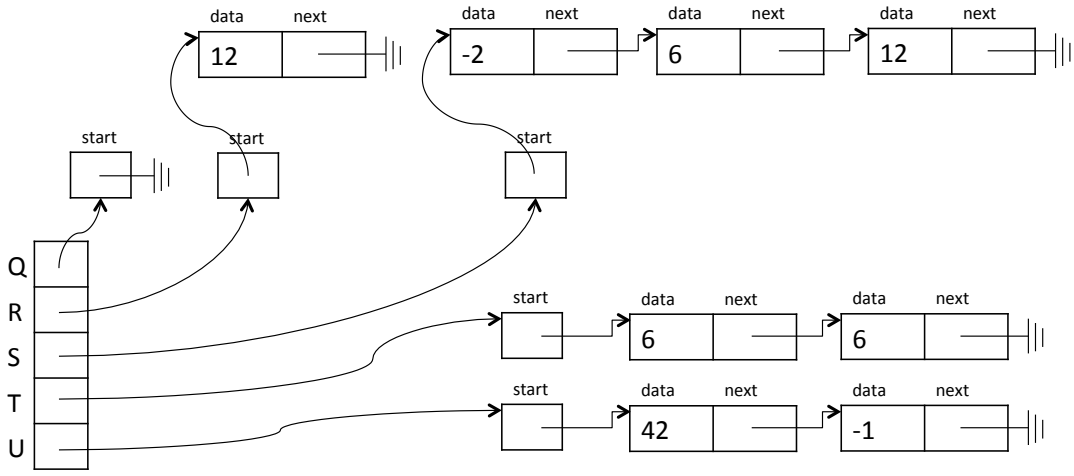
Nivedita Chopra, Rob Simmons

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. Feel free to talk with your neighbors about the problems!

Grading: For partial credit, you must pass all tests for (1.a). For full credit, you must additionally pass all tests for (1.a) and one of (1.b) and (1.c). (Show your TA the autograder output, and they will check you off).

Sorted Linked Lists

Today's lab 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
```

You should add your code to the existing file `sortedlist.c0` in the directory `lab-sorted`. Definitions of the structs and typedefs for `list` and `sortedlist` are in the file `listlib.c0`.

(1.a) Check if a given integer is in a sorted linked list, without modifying the list.

```
bool is_in(sortedlist* L, int n)
//@requires is_sortedlist(L);
```

Testing: 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; @*/;
```

Running `cons(-2, cons(6, cons(12, nil())))` creates the sorted linked list `S` from the example above. You can test your code in Coin like this:

```
% coin -d listlib.c0 sortedlist.c0
```

You'll be submitting the file `sortedlist.c0`, and only that file, to the ungraded Autolab autograder created for this lab. This is the one called "Lab 6 Activity: Sorted Linked Lists" and not the one called "Lab 6." The autograder gives no feedback. Remember that you can review, debug, and test code with your neighbors! Drawing diagrams might also prove useful.

(1.b) Insert an integer into a sorted linked list, while ensuring that the list remains sorted and that every element occurs exactly once. The list should be unchanged if the integer is already in the list.

```
void insert(sortedlist* L, int n)
//@requires is_sortedlist(L);
//@ensures is_sortedlist(L);
//@ensures is_in(L, n);
```

(1.c) Delete an integer from a sorted linked list, while ensuring that the list remains sorted and that every element occurs exactly once. The list should be unchanged if the integer isn't in the list.

```
void delete(sortedlist* L, int n)
//@requires is_sortedlist(L);
//@ensures is_sortedlist(L);
//@ensures !is_in(L, n);
```