15-122: Principles of Imperative Computation, Fall 2014
Lab 6: Linked Lists

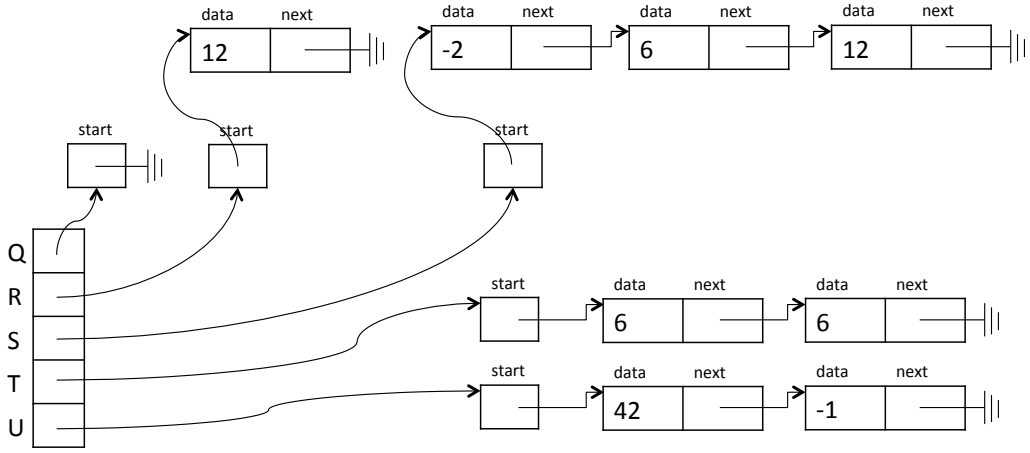
Nivedita Chopra (`niveditc@andrew`) and Rob Simmons (`rjsimmon@cs`)

Monday, October 6, 2014

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 if you get confused or stumped!

1 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 should be sorted at all times. Another thing that is different from the linked lists that you have seen in lecture and on the 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` are valid sorted linked lists (that is, `is_sortedlist(T)` and `is_sortedlist(U)` will both return `false`).

To get started, use `wget` to download these two files into a new directory. You'll be adding code to `sorted-lists.c0` and submitting just that file to Autolab.

```
http://www.cs.cmu.edu/~rjsimmon/15122-f14/rec/lab6lib.c0
http://www.cs.cmu.edu/~rjsimmon/15122-f14/rec/sorted-lists.c0
```

In addition to the definitions of the structs and typedefs for `list` and `sortedlist`, the `lab6lib.c0` file contains the following specification functions and helper functions, which may be useful when you test your code:

```
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 on the last page. You can test your code in Coin like this:

```
% coin -d lab6lib.c0 sorted-lists.c0
```

For full credit on today's lab, a TA should record that Autolab has passed you on both Task 1 and either Task 2 or Task 3. The autograder gives no feedback, but you can review, debug, and test code with your neighbors!

Exercise 1. Check if a given integer is in a sorted linked list.

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

Exercise 2. 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);
```

Exercise 3. 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);
```