CSCI-1200 Data Structures — Spring 2018
Lab 7 — List Implementation

This lab gives you practice in working with our implementation of the `dslist` class that mimics the STL `list` class. Create a directory/folder named `lab7` and download these files into that folder:

http://www.cs.rpi.edu/academics/courses/spring18/csci1200/labs/07_list_implementation/dslist.h

http://www.cs.rpi.edu/academics/courses/spring18/csci1200/labs/07_list_implementation/lab7.cpp

Checkpoint 1

The implementation of the `dslist` class is incomplete. In particular, the class is missing the `destroy_list` private member function that is used by the destructor and the `clear` member function. The provided test case in `lab7.cpp` works "fine", so what's the problem?

Before we fix the problem, let's use Dr. Memory and/or Valgrind to look at the details more carefully. You should use the memory debugging tools *both* on your local machine *and* by submitting the files to the homework server (we have set up a practice space for Lab 7). Study the memory debugger output carefully. The output should match your understanding of the problems caused by the missing `destroy_list` implementation. Ask a TA if you have any questions.

Now write and debug the `destroy_list` function and then re-run the memory debugger (both locally and on the submission server) to show that the memory problems have been fixed.

To complete this checkpoint, show a TA the implementation and memory debugger output before and after writing `destroy_list`.

Checkpoint 2

One subtle difference between the STL `list` implementation and our version of the `dslist` class is the behavior of the iterator that represents the end of the list (the value returned by `end()`). In STL you may decrement the end iterator. For example, you can print the contents of a list in reverse order:

```
std::list<int>::iterator itr = my_lst.end();
while (itr != my_lst.begin()) {
  itr--;
  cout << *itr;
}
```

The syntax is admittedly rather awkward, that's why we might typically prefer to use a reverse iterator to do this task. How does the `dslist` class behave on a corresponding test case? Try it out. How could you fix the implementation so that it more closely matches the behavior of the STL version? There are a couple different options... *If you can't come up with one quickly, please raise your hand and ask a TA.* Make the necessary changes to the implementation and test out your solution.

To complete this checkpoint, describe to a TA how you changed the implementation to allow the end iterator to be decremented.

Checkpoint 3

For the remainder of lab time, work on Homework 5 and ask your TA and mentors lots of questions! Show your TA or a mentor your 3 or more test cases (handdrawn "box & pointer" diagrams) for the `Separate` function and discuss your proposed algorithm and resulting output of those functions. What is the cost of the operation in terms of # of unlinks, # of links, and length of track that cars must be dragged? Use Dr. Memory or Valgrind on your local machine to debug the memory usage of your program (both memory errors and memory leaks).

(∼10 minutes before the end of lab:) To complete this checkpoint and the entire lab, show your TA or mentor your progress on the homework.