CSCI-1200 Data Structures — Fall 2009
Lab 7 — List Implementation

Introduction

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/fall09/ds/labs/07_list_implementation/dslist.h
http://www.cs.rpi.edu/academics/courses/fall09/ds/labs/07_list_implementation/code.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 `test_dslist.cpp` works "fine", so what's the problem?

Before we fix the problem, let's use the memory debugger Valgrind (`http://valgrind.org`) to look at the details more carefully. Valgrind is available for free on Linux (but unfortunately not Windows!) If you don't have Linux, or if you don't have Valgrind installed yet, don't fear. We've setup a homework submission site for this lab to let you run Valgrind remotely. Zip up the 2 original files just like you would a homework assignment (in a directory named "`lab7`", etc.). Submit the assignment and look at the Valgrind output. The output should match your understanding of the problems caused by the missing `destroy_list` implementation. Ask a TA if you have any questions. To run Valgrind on your own linux machine, first compile the executable (e.g., to `a.out`), and then type this:

```
valgrind  --leak-check=yes --show-reachable=yes  a.out
```

Now write and debug the `destroy_list` function and then re-run Valgrind to show that the memory problems have been fixed. Note that `cout` statements may cause additional messages about memory that don't directly refer to your code. It has to do with advanced memory allocation techniques that are used by STL. You may ignore those messages.

To complete this checkpoint, show a TA the implementation and Valgrind output.

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:

```
std::list<int>::iterator itr2 = b.end();
while (itr2 != b.begin()) {
  itr--;
  cout << *itr2;
}
```

The syntax is admittedly rather awkward, that's why we would typically rely on reverse iterators 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.