CSCI-1200 Data Structures — Fall 2019
Lab 6 — 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 `lab6` and download these files into that folder:

http://www.cs.rpi.edu/academics/courses/fall19/csci1200/labs/06_list_implementation/dslist.h
http://www.cs.rpi.edu/academics/courses/fall19/csci1200/labs/06_list_implementation/lab6.cpp

Checkpoint 1 — *estimate: 20-30 minutes*

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 `lab6.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. 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. Also finish the implementation of the `push_front`, `pop_front`, and `pop_back` functions.

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

Checkpoint 2 — *estimate: 30-50 minutes*

This checkpoint is a "pair programming" exercise.
https://en.wikipedia.org/wiki/Pair_programming
Find a partner who has finished Checkpoint 1 and team up with them.
Ask a TA or mentor if you haven't found a partner yet.

Download the test cases for Checkpoint 2 from Submitty:
https://submitty.cs.rpi.edu/f19/csci1200/course_materials

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 two main options. One involves adding a *dummy node* at the end of the linked list chain (the node won't store a real value). The other involves adding a member variable to the iterator (the variable stores a pointer to the entire dslist manager object). Discuss the details with your partner and draw pictures of both methods. *If you and your partner get stuck on the design, please raise your hand and ask a TA or mentor for help.*

Now tackle the implementation of both of these two options using pair programming. First, student B should close their laptop and look over the shoulder of Student A. Student A does the typing, but Student B should suggest what to modify and add to the code to implement the first option (you may start with either option). Two sets of eyes on the code are more likely to catch typos even before the code is compiled. Once the option is tested and debugged (*or after working for about 25 minutes, whichever is first*), switch roles. Student A should close their laptop and look over the shoulder of Student B, while Student B types up the solution for the other option – again with verbal help from Student A. *After approximately 50 total minutes on this checkpoint, even if you're not completely finished, put your name in the queue for checkoff.*

To complete this checkpoint, show a TA or mentor your diagrams and implementations of the two options. Also discuss the strengths and weaknesses of the pair programming methodology.

Checkpoint 3

estimate: 20-30 minutes

For the remainder of lab time, work on Homework 5 and ask your TA and mentors lots of questions!

Work on your student test cases. Yes! You can and should write test cases before your code is finished. Specifically how will you test your copy constructor and assignment operator? Make sure you've got the syntax correct. What are the challenges in implementing these functions and what possible flaws might occur with a buggy solution?

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.