For our warmup exercise, let’s follow on the theme from Lab 5, and do one more version of reverse. Let’s reverse a home-made singly-linked chain of Node objects.

Pull out some paper. Following the conventions from lecture, draw a picture of a “homemade” singly-linked list that stores the values 1, 2, 3, and 4. Make a variable on the stack named my_list of type Node* that points to the first node in the chain (the node storing the value 1). The 4 node objects should be separate blobs of memory dynamically-allocated on the heap.

Now, modify this diagram to reverse the list – you can do this by only changing pointers! You should use the existing node objects. Don’t copy the entire diagram or make any new nodes. You should not change the values inside of any node – don’t swap values like we did for Checkpoint 1.

Then, write pseudo-code to reverse this list, just changing the pointers as you diagrammed above. You may use helper variables (of type Node*) but no other data structures or variables. Remember that when we directly manipulate homemade linked lists we don’t use iterators.

Finally, download this starter code:

http://www.cs.rpi.edu/academics/courses/fall21/csci1200/labs/06_list_implementation/checkpoint1.cpp

Complete the reverse function using your diagram and pseudocode as a guide. Test and debug the code. Add a few additional test cases to the main function to ensure your code works with an empty list, and lists with one or two values. Also add a test or two of a node chain with something other than ints.

If you have time, write 2 versions of this function, one version should be iterative (using a for or while loop) and one version should be recursive.

To complete this checkpoint, show a TA or mentor your diagram and your debugged function(s) to reverse a homemade singly-linked list.

Checkpoints 2 & 3 give you practice in working with our implementation of the dslist class that mimics the STL list class. Download these files:

http://www.cs.rpi.edu/academics/courses/fall21/csci1200/labs/06_list_implementation/dslist.h
http://www.cs.rpi.edu/academics/courses/fall21/csci1200/labs/06_list_implementation/checkpoint2.cpp

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 3

Checkpoint 3 will be available at the start of Wednesday’s lab. This checkpoint will be a “pair programming” exercise.

https://en.wikipedia.org/wiki/Pair_programming