CSCI-1200 Data Structures — Spring 2015
Lab 14 Garbage Collection & Smart Pointers

Checkpoint 0

If you haven't done so already, please complete your Digital Measures course evaluation for Data Structures (your honest & anonymous feedback is very important!). Have the webpage receipt saying "completed" open in your browser to receive credit for Checkpoint 1.

Checkpoint 1

For the first checkpoint, download, compile, and run these files:

http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/14_smart_memory/stop_and_copy.h
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/14_smart_memory/stop_and_copy.cpp
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/14_smart_memory/main_stop_and_copy.cpp

In Lecture 24, we stepped through the Stop and Copy garbage collection algorithm on a small example. Examine the output of the program to see a computer simulation of this same example. Verify that the program behaves as we predicted in lecture.

Continuing with the same example, 3 more nodes have been added and the garbage collector must be run again. Draw a "box and pointer" diagram (the more usual human-friendly version of the interconnected node structure with few or no crossing pointer arrows) of the memory accessible from the root pointer after these 3 nodes are added and work through the Stop and Copy algorithm for this example on paper. When you are finished, uncomment the simulation and check your work.

To complete this checkpoint: Show one of the TAs your box and pointer diagram and the scratch paper where you worked through the example (*and* your Digital Measures Course Evaluation "completed" receipt).

Checkpoint 2

The theme for this checkpoint are the helium filled balloons for the Macy's Thanksgiving Day parade. These balloons are held in place by one or more ropes held by people on the ground. Alternately, balloons may be connected to other balloons that are held by people! People can swap which balloon they are holding on to, but if everyone holding on to the ropes for a particular balloon lets go, we will have a *big* problem! Download, compile, and run these files. Use Dr. Memory or Valgrind to inspect the initial code for memory errors and/or memory leaks.

http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/14_smart_memory/ds_smart_pointers.h
http://www.cs.rpi.edu/academics/courses/spring15/csci1200/labs/14_smart_memory/main_smart_pointers.cpp

Carefully examine the example allocations in the main function of the provided code. Draw simple pictures to help keep track of which objects have been allocated with new, and which variables currently point to each dynamically allocated object.

To fix the memory leaks in this program, you will need to add explicit deallocation for the non-smart pointer examples (marked CHECKPOINT 2A and 2B). For comparison, the code includes simple examples of smart pointers for these objects as well! When we know that just one person will hold on to a Balloon at a time (one owner) we can use a `dsAutoPtr` (see also the STL `auto_ptr` or STL `unique_ptr`, depending on your version of `g++/clang/STL`). When multiple people might hold ropes to the same Balloon, we should use a `dsSharedPtr` (see also STL `shared_ptr` or Boost `shared_ptr`). A shared pointer uses reference counting! When the last person disconnects from a Balloon using a shared pointer, the Balloon is automatically deleted.

Re-compile & re-run with the memory debugger to confirm you have fixed the simple leaks. For the final piece of this checkpoint (marked CHECKPOINT 2C), you must also re-rewrite the interconnected balloon

example to use shared pointers. You will need to modify the `Balloon` class to use `dsSharedPointer` as well.

To complete this checkpoint: Explain to your TA the code you needed to add and/or modify to correct the memory leaks in the provided code. Show your TA the result of the memory debugger on your finished implementation.

Checkpoint 3

For the last checkpoint, let's consider cyclic balloon structures. A simple example is included at the bottom of `main_smart_pointer.cpp`. Draw a diagram of this structure on paper. The provided code has memory leaks for this example. We could try to re-rewrite this example to use the shared smart pointer. However, a reference counting smart pointer will still have a problem on this cyclic example. Why?

Instead, let's write a helper function to *explicitly* deallocate a general cyclic structure of Balloon objects. (We will not use smart pointers for this checkpoint). You should switch back to the original `Balloon` class (if you modified it for Checkpoint 2). You will first need to collect all nodes that are reachable from the provided argument. Make sure that you do not enter an infinite loop when tracing through the structure! You may find `std::set` helpful. Once you have identified all of the nodes that are accessible from the input argument, you can call delete on each Node. Write additional test cases to confirm that your code is debugged.

To complete this checkpoint and the entire lab: Show the TA your debugged implementation and the memory debugger output to confirm that your code has no memory errors or leaks.