Tutorial 5 - Smart Pointers

Make sure you download 'Tutorial_5.tar.gz' from the Assignment tab on Vula.

In this session we are going to learn about smart pointers. As you know by now, managing dynamic memory can be seriously painful. Smart pointers, introduced in C++ 11, are the solution to this problem. Smart pointers allow us to utilize dynamic memory without worrying about the allocating and releasing of said memory.

In this tutorial we are going to:

1. use smart pointers to construct an acyclic directional sparce graph.
2. use catch.hpp to unit test our code to demonstrate how smart pointers work.

NOTE: - Lines in the code that should be commented or uncommented have been clearly marked for each of the 3 tasks.

Write your answers to a text file called `answers.txt`. This will be a file that you submit(along with your code) to Vula.

Part 1: A Memory Leak

1. In the first implementation we used raw pointers without an appropriate destructor to simulate a memory leak. Ensure the lines marked with " ————————ENABLE ONLY FOR TASK 1…" are uncommented.
2. Compile the program and run it.
3. You should see that the program is leaking memory. Now scroll down to the test cases and look at what was done: an automatic Graph variable has been instantiated inside an inner scope. As soon as that scope closes Graph (along with all its resources) should be destroyed according to the RAII paradigm. Since we didn't implement the destructor, the Graph object is leaking memory!

Your task it to implement a destructor for the Graph class such that it passes the simple **Insertion** unit test (Depending on how you implement the destructor, your may code may even pass all of the unit tests).

Part 2: Shared Pointers

Disable the lines that are not used in Task 2 and enable all of the lines that are.

Compile and run the program again.

- Q1. As you'll notice, our program now passes 2 out of the 3 unit tests (despite disabling the destructor code we created in Part 1). Why is this?
- Q2. Why couldn't we use unique pointers here? Why can we use shared pointers and how does their implementation, under the hood, allow us to do this?

Part 3: Weak Pointers

Now disable the lines not being used in this task. Enable the lines for task 3. You should see all the test cases pass.

- Q3. Peruse the changes we've made to the Node class. What have we done differently here? Why was it necessary (explain this in terms of the lifetime of the managed Node objects).
- Once you are done, submit your code and answer file to Vula
- **(Optional):** Take a look at part 4

Part 4: (Optional) Task

Implement the big 6 for the `Node` and `Graph` class. Additionally, write appropraite unit test cases for each function you implement.

NOTE: - Don't just copy accross the pointers. Allocate new Nodes with duplicate data (ie. make a deep copy) - Keep the code from part 3 uncommented. You should be working with smart pointers when implementing these functions.