Homework 4: Smart pointers

April 9, 2018

1 Introduction

The goal of this assignment to understand the trade-offs between unique pointers, reference counted pointers and manual memory management. `unique.cpp` implements a singly linked list using unique pointers. Notice, that with unique pointers, we can have at most one reference to a node, therefore in order to iterate through list, we need to work with the stack locations, which contain unique pointers. With shared pointers or manual memory management, we can keep multiple references and hence the implementation would be simpler.

In this homework, you need to re-implement unique.cpp, using shared pointers and raw pointers (manual memory management). To compile the code run, `g++ -std=c++14 -O3 unique.cpp -o unique`.
Please don't forget to use **-O3** parameter during compilation.

The unique.cpp takes number of nodes in the linked list as a parameter. The unique.cpp first creates the linked list and then randomly deletes 5% nodes. The `std::move()` construct is used for ownership transfer and access to a moved object is illegal. Similarly, `node = NULL` statement towards the end of the main routine deletes the entire linked list. Notice, in raw pointer implementation you need to manually delete the list at this point. In addition to that, you also need to delete the objects in `delete_node` routine manually.

When you run this program, it prints times in milliseconds during the creation, deletion, and cleanup of the list. Renaming `unique_pointer` with `shared_pointer` would suffice for shared pointers implementation, but for this homework, you have to change the implementation to use multiple references (similar to raw pointer implementation).

2 Turn in:

- Implement the linked list using raw pointers and shared pointers.
- Run all the three variants with 1000000 nodes. Report the run-times of all the variants.

- It is possible that the smart pointers (unique and shared) implementations give a segmentation fault for a large number of nodes. Try to scale the input nodes to get the segmentation fault. Debug the cause of segmentation fault.
- If possible, suggest the workaround for the segmentation fault.
- Report the run-times of all the three variants for the number of nodes which works without the segmentation fault (if you get the segmentation fault for 1000000 nodes and don't know the workaround).

3 How to submit

Submit a zip folder of your raw and shared pointers implementations, and a pdf report of run-times and workaround for segmentation fault.