Lab 6

Week of 2/8

Partnering Up

- Each partner does the **same** provided lab (*node* and *sequence* classes)
  - **Not** working together, both partners need to **individually** complete and demo the lab
- Still need to write test cases and report for partner like normal

Node (linked list toolkit)

- One node of a linked list
  - Data and pointer to next node
- This class is meant to make your sequence easier
- Take a second to look at, and understand, ALL of the functions of this class
  - A lot of the functions written in node can be (and should be) used in your sequence class implementation
- Private variables
  - value_type data_field; // double being stored with each node
  - node *link_field; // pointer to the next node in the linked list
- Write and test node before you write sequence

Sequence Class (using linked list)

- Sequence of nodes
- Forward linked list
- Private variables
  - node *head_ptr; // Pointer to head of linked list
  - node *tail_ptr; // Pointer to tail of linked list
  - node *cursor; // Pointer to current node
  - node *precursor; // Pointer to node before current
  - size_type many_nodes; // Number of nodes in linked list
- Description of assignment operator is on the second page of the lab doc
  - Please read through it when you go to do the = operator.

Function Clarification

- Node
  - *list_copy_segment()* includes the node at the end position
- Sequence
  - *advance()* can iterate past tail_ptr
    - cursor = NULL
    - precursor = tail_ptr
  - *init()*
    - Helper function that is meant to set the sequence to its default state
      - Pointers to NULL and many_nodes to 0
    - Not necessary to write but may help condense code slightly if you do decide to write it

Hints/Tips/Notes

- Don’t forget to use toolkit functions in sequence class
  - That’s what they are there for
- Implement node functions in the order in which they are tested
  - Easier to test that way
- Precondition for “head_ptr is the head ptr of a linked list” and etc.
  - No need to write assert() or check for this in any way
  - Can simply assume that this is the case when the function is invoked
- Precursor should be NULL if cursor is at head
- sequence.cpp needs to include “node.h” and “sequence.h”
- list_detect_loop()
  - Use **Floyd’s Cycle-Finding Algorithm**

Provided Files

- Node
  - node.h
    - Implemented for you
  - node.cpp
    - Some functions written already
  - node_test.cpp
  - node_out.txt
    - Will be used for demo
- Sequence
  - sequence.h
    - Implemented for you
  - seq_test.cpp
  - seq_out.txt
    - Will be used for demo

Compile/Demo

- Node
  - g++ node.cpp node_test.cpp
  - ./a.out > output.txt
  - diff output.txt node_out.txt
- Sequence
  - g++ sequence.cpp seq_test.cpp node.cpp
  - ./a.out > output.txt
  - diff output.txt sequence_out.txt

Don’t forget

- Demo code to me
  - Either today or next week
  - **Must compile and run on linux servers**
- Submit code to camino by the end of next lab
- Comment code
  - Loops and conditionals
- File with description of lab is on Camino
- Check google sheet to make sure that I didn’t forget to check you off for a demo