CSE 20212 Fundamentals of Computing II
Spring 2016

Lab handout for Week of March 21
Due 4/1 by 7pm for everyone

Objectives

1. Practice templating, data hiding and operator overloading
2. Implement two simple data structures
3. Have fun!

Part 1: Writing our own vectors

1. Report to lab **on time**. Attendance will be taken at the scheduled lab time.

2. Develop a simple vector-like array class called "NDVector" using new/delete, with the following member functions/interface for an arbitrary data type (aka use templating):

    - Non-default constructor with "Fall back" value of 10 elements
    - Assignment operator (operator=; see end of Feb notes and text)
    - Copy constructor (also see end of Feb notes and text)
    - Deconstructor
    - Random access to return i^{th} element (operator[], should be const)
    - Add to back (push_back)
    - Remove from back (pop_back)
    - Size of NDVector (size, should also be const)
    - Access last element (back)
    - Clear NDVector (clear)

3. Check out with a TA when the end of the lab period is reached. If you have additional time, you can proceed to Part II.

Part 2: Simple binary trees

1. Modify code from the course website or the text such that the Node object has data, a string label, and a left and right pointer (child).

2. Write a new function that when given a parent node, it sets the value and label of that parent node to whichever child item has the larger data value. If there is a tie, either always pick the left child or pick one at random.

3. Develop a simple Binary Tree object in C++ that will accept a list of 2^n pairs (data and label) and build a binary tree with $n + 1$ levels. Each parent node should have the label and data value using the function from #2 above

4. To your Binary Tree class, add a function to search for a specific data value

5. Write traversal functions inorder(), preorder(), and postorder()

Part 3: Testing our simple new data structures

1. Rework your CardDeck class from Lab 4 to use an internal templated NDVector instead of an internal STL deque (to again practice data hiding) as outlined below. Please be aware of the following potential issues:

   a. Make sure your push_back() method in NDVector allocates additional space when it is full, just like a STL vector often does, so all *n* - 1 values can be successfully added. As a concrete example, if the 11th element is inserted with 10 "slots" allocated, reallocate 2X (20) slots prior to adding the new element.

   b. Since you are using new/delete you will have to implement your own random_shuffle method in NDVector to support the prior **`shuffle`** method in the STL. We recommend Knuth's shuffle (see below), which swaps each element of the NDVector with a randomly selected prior element. **srand**() must still be called to change the random number seed for a more random shuffle.

      For each element *j* of the NDvector (1 … size())
      - Pick a random prior element *k* (index between 0 and *j*)
      - Swap element *j* and *k* in the NDVector

   c. inOrder() should still work if you implement operator[] properly, but note that it might not compile unless size and operator[] are made const.

   Submit a similar test program as Lab 4 that initializes an array of 52 cards, ensures they are in order, and prints the deck before and after a shuffle.

2. Use your new Binary Tree class to automatically fill out a "bracket" of your choice. You should provide a file with at least 16 lines with a data value and a label. You can fall back to something simple like Men's or Women's NCAA seeds or have a completely different set of items (food, books, movies, etc.). Use your imagination with the following requirements.

   a. You will be required to either write a new function or provide separate functionality to display the "final four", the championship, and who ever wins according to your algorithm.

   b. You should do all three traversals of your complete binary tree (inorder, preorder and postorder)

   c. Demonstrate that your search function written for part 2 works as intended.

Coder challenge

Due to the Easter break in between lab these two weeks there will be no coder challenge, but you are encouraged to make more complicated weightings to simulate your brackets more accurately if you are interested.