Lab 9

Goals-
Implement linear data structures using STL Containers

While standing in line you may have asked yourself, "Is this the best way to organize things"? Specifically, it is better to have one line for all tellers/registers/servers? Or is it better to have one line for each teller/register/server?

You will have 2 simulations. **In one you will have people arriving and being put into a single queue.** Each time "click" you pull out one for each server. If there are fewer people then servers, some of the servers are unused. **In the other simulation you have one queue per server.** Every time click you will put people into the queues. Every time click you will take one person out of each queue. If you have *n* servers what do you do if $n + x$ people arrive? What do you do with the *x* extra people?

KEEP IT SIMPLE. We only care about the queue(s). **A "server" is just taking one person out of a queue.** You need no other code for the servers. **Make one time "click" one repetition of the loop** that drives the simulation. Implement the queues using the STL container. Test your code. If you have *n* servers have *n* people arrive each repetition. Now add the simulation. **The user will enter the number of servers.**

If fewer people are arriving than you have servers there is nothing to simulate. So start with *n* arrivals per repetition. Then increase the number of arrivals. Include a random element; *RND* is a random number between 0 and *n*. Start with *n*, then $2* n + RND$, then $3 * n + RND$, then decrease the number. Basically simulate a workday, or a period of service. You may need to adjust these numbers or the number of cycles (iterations of your loop). You want to start with empty queues and end the simulation with empty queues. **Use whatever scheme you'd like**, **but you need a gradual increase in the number of arrivals and then a decrease**. Your servers are dedicated and take no potty breaks, cigarette breaks, or even eat lunch! ☺ Your customers are also well-behaved and don't jump lines thinking another is better.

Now for the last piece. How are you going to measure the difference, if any? **We are interested in waiting time**. We can approximate that by calculating the number of people waiting. **Think about it and develop a scheme**. It may be as simple as counting the number of people in line for each repetition of your loop (i.e. a time "click"). Maybe slightly more accurate would be to count the number of "clicks" each person is in a queue. This does NOT require a counter for each person that is incremented on each repetition.

Compare the results: *n* servers and 1 queue, *n* servers and *n* queues. Display your times to the user. Does one have less wait time? Could you modify the arrival rate so there was a difference?

What to submit-

You will submit the following files to TEACH-

Code to implement your simulation using the STL Queue container

Grading

Programming style- 1 point

Code to implement your queue(s)- 3 points

Code to vary the number of people arriving each cycle- 2 points

Code to measure the wait time - 2 points

The arrival rate is a function of *n*, the number of servers- 1 point

The results are displayed to the user- 1 point

NOTE- Mathematically, you want a single queue for all servers. At a navy commissary (i.e. a humongous grocery store) they had a single line for more than a dozen check stands. The line went to the back of the store and doubled around. It might have been more efficient, but it was depressing. ☹