no curves or EC in midterm grades
pre-work on whiteboards mandatory
bonus points to those that share solutions
must move on as group

lets use class templates!

CptS 122 – Data Structures

Lab 7: Developing a (Linked) Queue Class in C++

Assigned: Week of February 26, 2024
Due: At the end of the lab session

challenges for bonus points
-class templates
-recursive stream insertion overload

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Design, implement and test a dynamic queue implemented with a singly linked list in C++
- Compare and contrast dynamic linked lists, dynamic linked stacks, and dynamic linked queues
- Summarize the advantages of applying a queue within certain applications
- Describe the operations applied to a queue including
    1. enqueue ( )
    2. dequeue ( )

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Analyze a basic set of requirements for a problem
- Create test cases for a program
- Design, implement and test classes in C++
- Declare and define *constructors*
- Declare and define *destructors*
- Compare and contrast *public* and *private* access specifiers in C++
- Describe what is an *attribute* or data member of a class
- Describe what is a *method* of a class
- Apply and implement *overloaded* functions
- Distinguish between pass-by-*value* and pass-by-*reference*
- Discuss *classes* versus *objects*
- Apply and implement *pointers* in C++
- Apply and implement *dynamic* memory in C++
- Design and implement singly linked *lists* in C++

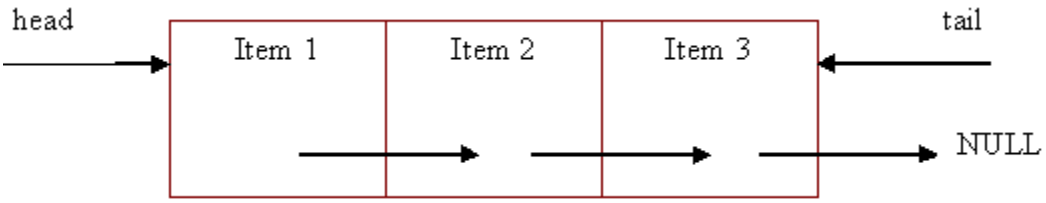
III. Overview & Requirements:

This lab, along with your TA, will help you navigate through designing, implementing, testing a List class in C++. It will also help you with understanding how to apply a Queue object to an application.

Labs are held in a “closed” environment such that you may ask your TA questions. Please use your TAs knowledge to your advantage. You are required to move at the pace set forth by your TA. Have a great time! Labs are a vital part to your education in CptS 122 so work diligently.

Tasks:

This lab, along with your TA, will help you navigate through designing, implementing, and testing a dynamic queue implemented with a linked list. Recall, a queue data structure is a restricted linked list, where only the front or head node in the queue may be processed and then removed, at any given time. However, only nodes may be added to the end, back, or tail of the queue. A queue is referred to as a first-in, first-out (FIFO) structure as a result of this constraint. Furthermore, the operations of a queue must adhere to this restriction. An `enqueue()` operation adds a node to the tail of the queue and a `dequeue()` operation removes a node from the head of the queue. We will visualize a queue in the following way:



Labs are held in a "closed" environment such that you may ask your TA questions. Please use your TAs knowledge to your advantage. You are required to move at the pace set forth by your TA. Please help other students in need when you are finished with a task. Have a great time! Labs are a vital part to your education in CptS 122 so work diligently.

Tasks:

1. Define a class `QueueNode` with data of type `std::string`. Implement the core constructors, destructor, setters, and getters for the class.

2. Implement a class `Queue` with the following operations for your queue data structure:
    1. `isEmpty()` - a predicate function which checks to see if the queue is empty; returns true if the queue is empty; false otherwise
    2. `enqueue()` - inserts a node to the tail of the queue; the node is allocated dynamically; returns true if the memory was allocated for a node, false otherwise
    3. `dequeue()` - deletes a node from the head of the queue; returns the data in the node; precondition: queue is not empty (isEmpty () must be called before dequeue () is called)
    4. `printQueueRecursive()` - recursively prints out the data in the queue

3. Test your application. In the same project, create one more header file `testQueue.hpp` and source file `testQueue.cpp` (for a total of at least seven files). The `testQueue.hpp` file should contain function prototypes for test functions you will use on your queue functions. The `testQueue.cpp` source file should contain the implementations for these test functions. You will have at least one test function per application function. For example, you will have an application function called `enqueue()` (or a function very similar) that is used to insert a node into the tail of the queue. In this task, you will need to create a test function called `testEnqueue()` that passes in various data directly into `enqueue()` to see if it works as intended. You will also want to test these functions on empty and non-empty queues.

4. Work on the current programming assignment.

IV. Submitting Labs:

- You are not required to submit your lab solutions, unless you are unable to attend them synchronously. You should keep them in a folder that you may continue to access throughout the semester.

V. Grading Guidelines:

- This lab is worth 10 points. Your lab grade is assigned based on completeness and effort. To receive full credit for the lab you must show up on time, work in a team, complete 2/3 of the problems, and continue to work on the problems until the TA has dismissed you.