CSE30: Lab #10 – Queues

Overview

In this lab, we will explore a new useful data structure: **Queues**. Queues are usually referred to as a **FIFO (First In First Out)** structure, where the first element inserted into the data structure is the first one to be removed from it. Queues can be implemented with any type of container (e.g. arrays, linked lists, etc.). In this lab, you will implement the Queues **with your implementation of the Linked Lists in Lab#8**. In fact, you can think of Queues as Linked Lists, with the FIFO constraints: you will implement them using inheritance, by inheriting from a Linked List class.

Getting started

Create a new directory in your main development directory (probably on Desktop/CSE30) called Lab_10. Try to use the terminal on your own, without getting help from the TA to setup the new directories (try to learn/remember the terminal commands).

The g++ syntax to compile classes is slightly different than for a single program comprised of a main (and potential functions):

g++ *class1.h class1.cpp class2.h class2.cpp mainSource.cpp* -o *executable*

where:

- g++ is the compiler (installed on your Linux system) of C++ source files,
- ***mainSource.cpp*** is the source file for your main program (main function),
- ***class1.h*** is the class declaration for your 1st class,
- ***class1.cpp*** is your 1st class definition,
- ***class2.h*** is the class declaration for your 2nd class,
- ***class2.cpp*** is your 2nd class definition (and so on...),
- -o tells the compiler that you want to give the executable its own name, and
- ***executable*** is the name you want to give your program.

As an example, if we have a main source file called ***Exercise1.cpp***, the class declaration called ***LinkedList.h***, the class definition called ***LinkedList.cpp***, and want to create an executable called ***aTestProgram***, you would type:

g++ *LinkedList.h LinkedList.cpp Exercise1.cpp* -o *aTestProgram*

Assuming that your program compiled successfully (i.e. no errors were found), you can run your program as you normally would by typing "***./aTestProgram***" in the terminal/console.

Good coding practices (worth 2 points!)

Writing code that is understandable by humans is as important as being correct for compilers. Writing good code will help you complete the code, debug it and … get good grades. It is very important to learn as soon as possible, because bad habits are hard to get rid of and good habits become effortless. Someone (guess who) reads your code will be in a better mood if it is easy to understand … leading to better grades! This lab will include 2 points (10% for code quality):

- Explanations with comments
- Meaningful names
- Indenting of blocks { } and nesting …
- Proper use of spaces, parentheses, etc. to
- Visible, clear logic
- One / simple statements per line
- Anything that keeps your style consistent

(Exercise 1)

In this part of the lab, you will be implementing your own **Queue** class, comprised of a class declaration (**Queue.h**) and class definition (**Queue.cpp**). You will also create a simple main program to test your queue class (**Exercise1.cpp**). This part of the lab will use ***int*** as the type for elements in the Queue (Linked List nodes). Similar to your stack implementation in Lab#11, **your Queue class will inherit the LinkedList class** so that any functions available inside the LinkedList class can be used by the Queue class. Take a look at the ***Inheritance_example_files*** from lecture to see how to use inheritance. As you will see when you implement the Queue class, inheriting the LinkedList class will make this part of the lab very simple. Your Queue class will be comprised of the following functions, which you need to implement (note that the Queue class does not need any variables, since those from the inherited LinkedList are all you need):

- Default Constructor (**Queue()**): does nothing
- Destructor (**~Queue()**): does nothing
- **void enqueue(int value)**: inserts a new element (value) at the **back** of the Queue, **by calling the appropriate LinkedList function**.
- **int dequeue()**: removes the first element of the Queue, **by calling the appropriate LinkedList function**. It also returns the value of the element that has been dequeued.
- **int& front()**: returns a reference to the front element of the Queue.

You also need to create your own main program (**Exercise1.cpp**) to test the various Queue functions:

- Create a queue of at least 10 elements in your main program and call various member functions to manipulate the queue (as you have seen in your HW2 problem).
- Call ***enqueue*** to insert integer to the queue
- Call ***front*** to return to the front of the queue
- Call ***dequeue*** to remove elements from the queue
- Check the size of the queue by calling a function inherited from Linked List.
- Check if the queue is empty by calling a function inherited from Linked List.
- Print the content of the queue by calling a function inherited from Linked List and verify if the member functions work correctly.

Note:

- Your Queue implementation should be very simple and short. It will be similar to your implementation of Stack in Lab#11. Remember that since you are inheriting the LinkedList class, you can call any functions/variables that were defined inside the LinkedList class.
- Unlike your Stack implementation in Lab#11, **you must generate exceptions where appropriate**: you should throw exceptions in both dequeue and front functions whenever these functions would fail. You will need to catch the exceptions and print the following error messages if an exception is generated: "Call to front() generated an exception, because the queue is empty" OR "Call to dequeue() generated an exception, because the queue is empty"
    - You should throw an exception (an integer value) inside the appropriate Queue's functions. (i.e. in Queue.cpp) See the ***online tutorial in exceptions*** for examples on how to implement exceptions. All you need to do is to ***throw*** an integer value when an error arises.
    - http://www.cplusplus.com/doc/tutorial/exceptions/
    - For each call to the ***dequeue*** and ***front*** functions, you should try to catch the exception (the integer value thrown from the functions) in the main file (Exercise.cpp) and print the appropriate error to the screen.

(Exercise 2)

In this part of the lab, you will make the following changes to the code developed in Exercise 1:

- Change your Queue and Linked List classes so that they use ***char*** type instead of ***int***. Name them Queue_char.h/Queue_char.cpp and LinkedList_char.h/ LinkedList_char.cpp to avoid the confusion between the classes created in Exercise 1.
- Create a main program (Exercise2.cpp) to produce the following output similar to what was illustrated in the lecture.
- You do not need to produce the table lines, but need to use tabs in between columns and newlines for each row. Columns do not need to be aligned between rows.

Function call	Output	Front Queue Rear
dequeue()	exception	
enqueue(D)		D
enqueue(A)		D, A
dequeue()	D	A
size()	1	A
enqueue(D)		A, D
isEmpty()	false	A, D
front()	A	A, D
enqueue(T)		A, D, T
front()	A	A, D, T

What to hand in

When you are done with this lab assignment, you are ready to submit your work. Make sure you have included the following ***before*** you press Submit:

- Your **LinkedList.h**, **LinkedList.cpp**, **LinkedList_char.h**, **LinkedList_char.cpp , Queue.h, Queue.cpp, Queue_char.h, Queue_char.cpp, Exercise1.cpp**, **Exercise2.cpp** , and a list of Collaborators.
- Documentation (in a text file) of code you used from the internet. You may want to cut-and-paste the original code as well.