Test 01: Coding Practicum

CSci 520: Data Structures and Algorithms

Spring 2022

Objectives

- Demonstrate ability to use basic stack and queue data structures
- Show use of basic recursive techniques and writing recursive functions
- Demonstrate knowledge of C++ advanced OO features, and using templates in C++
- Show can use basic git and GitHub resources to devlop and commit code.

Description

This tests consists of two tasks, very similar in nature to the tasks you have been performing for our lab assignments so far in this class. You will be reusing the stack and queue abstractions that you worked on and developed in previous assignments for this practicum. In this test, you will demonstrate some of the basic knowledge you should have picked up so far in the class, including using basic data structures like stacks and queue, defining recursive functions, using git and build systems, using C++ OO features and mechanisms, etc.

This test is an individual test, you will not work with your assignment teams for this test. But as usual, you need to accept the test/assignment, configure it, complete the tasks described below, then commit and push your work to your GitHub repository created for this test tasks.

For both tasks you are asked to reverse a queue of items in place. You will first write an iterative solution in task 01 that usues a temporary queue to hold the values and put them back into the queue in reverse order. Then in task 02 you will use recursion to dequeue the front item from the queue, recursively reverse the queue, then enqueue the front item back to the end of the queue in order to reverse its order.

Overview and Setup

For this test you will be given the following files that you will be using and adding code to for this assignment. We have not listed all of the files that are in this test project. You have been given all of the Queue and Stack files you developed and worked with in previous assignments. You will be using our Stacks and Queues to complete the tasks for this assignment.

But besides the data structure files, you have the following files you will need to work with.

File Name	Description
`include/task01-library.hpp`	Header/declarations of the function for task 01
`src/task01-library.cpp`	Implementation of the task 01 function
`src/test-task01.cpp`	Unit tests you need to pass for task 01
`include/task02-library.hpp`	Header/declarations of the function for task 02
`src/task02-library.cpp`	Implementation of the task 02 function
`src/test-task02.cpp`	Unit tests you need to pass for task 02

You will put the declaration/prototype for task 01 in the `task01-library.hpp` header file, and implement the task/function in the `task01-library.cpp` file. Your implementation will need to pass the unit tests given in the `test-task01.cpp` file. Likewise the declaration, implementation and unit tests for task 02 are in the `task02` files.

As usual, before starting on the test tasks proper, you need to make sure you have completed the following setup steps:

1. Copy the test repository on GitHub using the provided assignment invitation link for 'Test 01' for our current class semester and section.
2. Clone the repository using the SSH URL to your local class DevBox development environment.
3. Configure the project by running the `configure` script from a terminal.
4. Confirm that the project builds and runs, though no tests will be defined or run initially. If the project does not build on the first checkout, please inform the instructor.
5. You should create the issue for Task 01 and/or for all tasks for the assignment now before beginning the first task. On your GitHub account, go to issues, and create it/them from the issue templates for the assignment. Also make sure that you link each issue you create with the `Feedback` pull request on your GitHub classroom.

Test Tasks

Task 01: Using Stacks to Reverse Queues

In Task 01 you are to implement a regular C/C++ function that will take a queue as input and reverse the order of the items on the given queue. For example, if the queue initially has the items

```
front: 1 2 3 4 :back
```

then after reversing the queue, 4 will be at the front, 1 at the back, and the items will then be orderd like this:

```
front: 4 3 2 1 :back
```

You need to implement this function as a template function. The function for task 01 will be named `reverseQueueIterative()`. Your function should take a `Queue<T>` as its one and only input parameter. Make sure you declare your input queue parameter to be passed in by reference, because you will be rearranging the order of the items in place in the queue given as input. Your function will be a `void` function, it does not return an explcit result. Instead, as we said, the queue you give as input will be modified to reverse the order of the items in the queue.

You are required to use an iterative solution for task 1 using the following pseudocode.

1. Create a local temporary stack.
2. Iterate over the queue, removing all items from the queue and pushing all items onto the temporary stack.
3. Then once all items have been pushed to the stack, reverse them by iterating over the stack, and removing the top item and putting it back on the end of the original queue.

Because the stack is FILO (first in last out), while the queue is FIFO (first in first out), the items will be reversed in order as a result of moving them to the stack then back onto the queue.

The following are requirements for task 01

- You are required to create a regular C/C++ template function for this task.
- You are required to create and demonstrate using a local temporary stack.
- You are required to iterated over the queue and stack using the queue and stack API methods (e.g. `isEmpty()`, `getSize()`, `top()`, `front()`, `push()`, `enqueue()`, etc.).

You are also required to make at least 1 commit of only your task 01 work. No work for task 02 should be in any of your task 01 commits. You can of course make more than 1 commit. It is suggested that you create and commit a stub function at least, to ensure you can compile your code and run the tests. You will get partial credit at least for a successful commit of a stub function that compiles and runs tests successfully.

In addition you are required to follow the usual class requirements. You must provide function documentation for the code you write. Your code must be correctly formatted and pass through the code style checker.

NOTE: As usual the unit tests are commented out initially for task 01. Start by uncommenting them. There is also some code commented out at the bottom of the `task01-library.cpp` file that needs to be uncommented in order to get the code to compile.

Task 2: Recursive Implementation of Queue Reversal

In task 02 you will demonstrate writing a recursive function. All of the work for task 02, including declaring the function prototype, the function implementation and the unit tests are in the `task02` set of files. Make sure all and only work for task 02 is done in these 3 files.

The function name for task 02 will be `reverseQueueRecursive()`. However this function has the same signature as the task 01 function. It will be a template function, templatized on `<class T>` It should take a `Queue<T>` as its one and only input parameter, a queue of some type of `T` values. This parameter needs to be passed in by reference. And the function will be a `void` function again as it does not return an explicit result, it simply reverses the queue again as a side effect of calling the function.

For task 02 perform the reversal of the queue items by writing a recursive function. The base case for the recursion is that, if the queue is empty, you should simply return. Otherwise if the queue has 1 or more items, you do the following for the general case:

1. Remove the item from the front of the queue.
2. Reversed the remaining items on the queue by recursive calling this function on the smaller queue.
3. Place the item you removed on the end of the (now reversed) queue of items.

This should successfully reverse the items of the queue. For example, the front item is removed by this algorithm, then the remaining items are reversed (recursively), and then the front item gets put onto the back, thus it is now reversed from its original position.

The following are requirements for task 02:

- You must perform the reversal using a recursive solution.
- You must implement the recursion as described, where the base case happens for an empty queue and recursion happens for queues of size 1 or bigger.
- You are required to use the queue API functions to manipulate the queue successfully, to remove and put back items on the queue.

As with task 01, you are also required to create function documentation for this function, you code is required to be properly formatted using the class code style checker, and you must have only work for task 02 in 1 or more commits for this part of the test.

Test Submission

For this class, the submission process is to correctly create pull request(s) with changes committed and pushed to your copied repository for grading and evaluation. For the assignments, you may not be able to complete all tasks and have all of the tests successfully finishing. This is ok. However, you should endeavor to have as many of the tasks completed before the deadline for the assignment as possible. Also, try and make sure that you only push commits that are building and able to run the tests. You may loose points for pushing a broken build, especially if the last build you submit is not properly compiling and running the tests.

In this test, up to 20 points will be given for having at least 1 commit that compiles and runs the tests (and at least some attempt was made to work on the first task). Thereafter 40 points are awarded for completing each of the 2 tasks. However you should note that the autograder awards either all point for passing all tests, or no points if any test is failing for one of the tasks. Also note that even if you pass all tests, when the instructor evaluates your assignment, they may remove points if you don't follow the requirements for implementing the code (e.g. must reuse functions here as described, need to correctly declare parameters or member functions as `const` where needed, must have function documentation correct). You may also loose points for style issues. The instructor may give back comments in pull requests and/or create new issues for you if you have issues such as these, so it is good to have work committed early before the due date, so that the instructor may give feedback requesting you to fix issues with your current submission.

Program Style

At some point you will be required to follow class style and formatting guidelines. The VSCode environment has been set up to try and format your code for some of these guidelines automatically to conform to class style requirements. But not all style issues can be enforced by the IDE/Editor. The instructor may give you feedback in your pull comments and/or create issues for you for the assignment that you need to address and fix. You should address those if asked, and push a new commit that fixes the issue (or ask for clarification if you don't understand the request). In general the following style/formatting issues will be required for programs for this class:

1. All programs must be properly indented. All indentation must be consistent and lined up correctly. Class style requires 2 spaces with no embedded tabs for all code indentation levels. The editor style checker should properly indent your code when you save it, but if not you may need to check or correct this if code is misaligned or not properly indented.
2. Variable and function names must use `camelCaseNameingNotation`. All variable and function names must begin with a lowercase letter. Do not use underscores between words in the variable or function name. Often function names will be given to you, but you will need to create variables, and maybe some functions, that conform to the naming conventions.
   - Global constants should be used instead of magic numbers. Global constants are identified using `ALL_CAPS_UNDERLINE_NAMING`.
   - User defined types, such as classes, structures and enumerated types should use camel case notation, but should begin with an initial upper case letter, thus `MyUserDefinedClass`.
3. You are required to use meaningful variable and function names. Choosing good names for code items is an important skill. The code examples and starting code tries to give examples of good and meaningful names. In general, do not use abbreviations. Single variable names should be avoided, except maybe for generic loop index variables `i`, `j`, etc. Make your code readable, think of it as writing a document to communicate with other developers (and with your instructor who will be evaluating your code).
4. There are certain white space requirements. In general there should usually never be more than 1 blank line in a row in your code. Likewise there should usually not be more than 1 blank space on a line. There should be 1 blank space before and after all binary operators like `+`, `*`, `=`, `or`.
5. Function documentation is required for all regular functions and all class member functions. You need to follow the correctly formatted Doxygen function documentation format. We will use function documentation generation, and you should be sure your documentation can be built without emitting warnings or errors. Likewise all files should have a file header documentation at the top. You should edit the file header of files where you add in new code (not simply uncommenting existing code). Make sure the information has your correct name, dates, and other information.
6. Practice using proper Git commit messages. You should refer to issues and tasks correctly in commit messages.