ECE 2574: Project 4.1

Ryan M. Gerdes
{rgerdes}@vt.edu

Virginia Tech — 2020-11-08

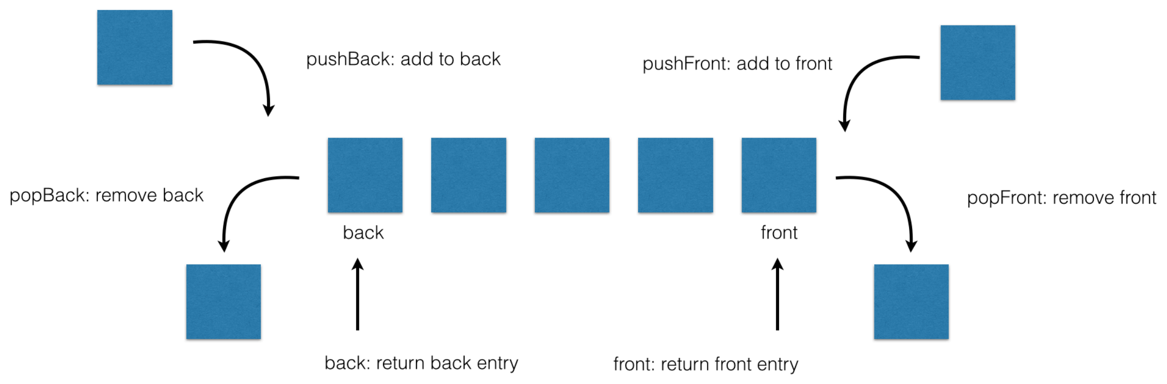
Revision: Wednesday 3rd November, 2021 19:58

Introduction

For our fourth project we'll be implementing the Deque (double-ended queue) ADT. A Deque is a generalization of a queue that allows for enqueueing/dequeueing from both the front and back of the queue (Figure 1).

A couple of points to note:

1. You must implement the ADT deque by using a dynamic array to contain the items in the deque; average complexity must be $O(1)$ with worst case $O(n)$.
2. The array implementation will feature dynamic memory. The default constructor will create a deque capable of storing 64 items. Once the capacity is exceeded you must implement an array growing strategy that doubles the size of the array.
3. You must follow the rule of three.



As always, you need to write unit tests for your class.

1 The `Deque` Class

The file `abstract_deque.hpp` in the stater code defines the templated interface, AbstractDeque. You should define a template Deque that publicly inherits from the interface in the header file `Deque.hpp`. This template should override the following methods from the interface (see `abstract_deque.hpp` for details)

- isEmpty
- pushFront

- popFront
- front
- pushBack
- popBack
- back

You will need to define the methods and implement all methods in the `Deque.hpp/.txx` file. You should add appropriate comment blocks to each method, as well. You will need to write tests in the `student_tests.cpp` using the Catch testing framework, as described in class. The included `CMakeLists.txt` file sets up everything for you. Just generate your build directory for your development environment as described in the course workflow tutorial.

2 Grader

We will be using an automatic grader to help you determine your assignment's completeness and correctness. A portion of each assignment grade will be determined by the number of passing tests as determined by the autograder, with our evaluation filling in the rest. This means you know before you turn in your submission that all is well. You can submit to the autograder as many times as you like, but it is rate limited (five submissions every hour) to keep you from using it as your compiler. See this canvas for a summary of how to use the grader (Note is is not WebCAT, which many of you may be familiar with).

For this assignment you should upload a zip file containing only the files: `Deque.hpp`, `Deque.txx` and `student_tests.cpp`. There is a build target called "submission" configured by default to create this file with the correct contents in your build directory.

3 Submission

Once you are satisfied with your code, upload a zip file containing (only) `Deque.hpp`, `Deque.txx` and `student_tests.cpp` through Canvas at the assignment link. You should not submit the other files from the starter code, nor your build directory. There is a build target called "submission" configured by default to create this file with the correct contents in your build directory.

4 Grading

There are 40 points allocated to this assignment.

- Correctly submitting the required files: 2 points
- Your tests compile: 3 points
- Your tests pass: 10 points (proportional)
- Instructor tests compile with your code: 3 points
- Instructor tests pass: 15 points (proportional)
- No tests leak memory: 2 points
- Design requirements met (e.g., code and test quality, comments): 5 points

As per the syllabus, good faith efforts must be made to write comprehensive tests. Since there is no way to test for this at compile or run time (at this point), should the grader (person) deem your tests to be woefully lacking you will lose not only design points but also the points given by the autograder. Each method should be tested.