# CSC 232: Data Structures and Algorithms

**Asn5-2: Test Suite for DEque ADT**

**Due: See Blackboard**

*Submit your program in through BB by midnight on the due date. Email your programs to me as a last resort if you experience problems with BB.*

**Scenario**

One of the techniques many companies use to get new engineers familiar with code is to have them write test routines. A test routine for a function or method is called a unit test, tests for a class are module tests, and all tests for a program comprise the test suite.

You have just been hired as an entry-level software engineer and been given the code for a double-ended queue (DEque). You are tasked with writing a comprehensive test suite for this code. To help orient you to the code, you will participate in a code walkthrough (Asn5-1).

**Objectives**

* Participate in a code review
* Implement a comprehensive test routine

**Background**

* The Asn4 SLinkedList class has been modified by removing unused methods and renaming existing method names to conform to common usage as implemented in the STL. For example, insertFront() has been renamed to push\_front().
* A DEqueue is implemented by extending (inheritance, see DSA 2.2) SLinkedList. The child class is LinkedDeq (DSA p.218) with the following methods:

size(): Return the number of elements in the deque based on the private field LDsize.

front():Calls SLinkedList front().

push\_front(*e*): Calls SLinkedList push\_front().

pop\_front():Calls SLinkedList pop\_front().

back(): Return the deque’s last element.

push\_back(*e*): Insert *e* at the end of the deque.

pop\_back(): Return the last element of the deque and delete the last node.

* Hierarchical exception handling as described in DSA 2.4 is included. Exceptions print an appropriate error message. When possible, they implement the strong exception safety guarantee by leaving the data structure unchanged in the event of an error.
* The virtual keyword for dynamic binding when methods have the same name is implemented (see DSA 2.2 or my lecture(s) on polymorphism).
* You can use the printDetails() method to expose the structure of the DEque.

**Requirements**

* Create and execute a test routine for LinkedDeque that tests all boundary conditions and all exceptions.
  + Test the methods in the order listed above.
  + Use the Lab4 (sample code below) as a model. Feel free to enhance it by writing individual test functions for some/all test conditions. Any test functions must be called from runAllTests().
* Coding style per the Coding Style Cheat Sheet, including a completed comment template at the top.

**Legal**

You may use only language features discussed in class or presented in the book up to the date the assignment is due. Your submission must be your own work. You may not utilize any code outside of that provided in class or in the book and you may not post any provided code on publicly accessible websites. Submit only what is requested.

/\*

Subset of Lab4 Test suite.

\*/

// A generic test function; prints passed if b is true

// and otherwise prints failed.

void test(const string message, const bool passed) {

cout.width(30);

cout << message;

cout.width(10);

if (passed)

cout << " PASSED\n";

else

cout << " FAILED\n";

}

void runAllTests() {

test("Testing length()...", stringLength("Now") == 3);

test("Testing at()", charAt("Elephant", 3) == 'p');

test("Testing append()", stringAppend("There's a ", "natural mystic.") == "There's a natural mystic.");

test("Testing find()", stringFind("Have to face reality now.", 'o') == 6);

}

int main() {

cout << "Testing your functions...\n\n";

runAllTests();

getchar();

return 0;

}

NOTES :

TEST exceptions on SLL and DEque