COMP 2231_SW3 - Data Structures and Algorithms (Fall 2020 Carruthers)

Assignment 2: Stacks & Queues (8%)

This programming project should be completed and submitted by Monday of Week 6 if you are following the suggested course schedule. It is worth 8% of your final grade. Please refer to the "Assessments Overview" tab for details on submission of your work. Your overall course assessment information is found in your Course Guide.

1. The `ArrayStack` implementation in Chapter 12 uses the top variable to point to the next array position available in the stack i.e. above the actual top of the stack. Modify the array implementation such that `stack[top]` is the actual top of the stack. Do not introduce any other counter variables. Make sure your driver demonstrates any method that you modify. (Adapted from PP 12.4.)

   Hint: Start with the `ArrayStack` code that you completed as part of Lab 3. Consider starting top at a value other than 0 and altering where top may be incremented or decremented.

2. There is a data structure called a **drop-out stack** that behaves like a stack in every respect except that the size is fixed. If the stack size is n, the bottom element is lost when the $n+1$ element is pushed onto the top. Implement a drop-out stack using links, by modifying the `LinkedStack` code that you completed as part of Lab 3.

   The driver should create a stack of moderate size (say five) and push on that many String elements consisting of people's names (include your own). Output the contents of the stack and the results of the `size()` and `peek()` operations. Push two more elements onto the stack, outputting the contents and the results of the `size()` and `peek()` operations for each. (Adapted from PP 13.8.)

3. A double-ended queue, or **deque** (pronounced like "deck"), is introduced in Section 14.8. With a deque you can add, remove, or view elements from both ends of the queue. Rather than use the Deque interface supplied by the Java API, design your own `DequeADT` interface (patterned after `QueueADT`). Then, implement a deque using links.

   Hint: Start with the `LinkedQueue` code that you completed as part of Lab 4. Also, each node will need both a next and a previous reference.

   The driver should create a deque of moderate size (say five or six). It should repetitively add elements to the front, and then the rear. For each addition, output the contents of the deque as well as the size, front element, and last element. Then, repetitively remove elements from the deque. For each removal, output the contents of the deque as well as the size, front element, and last element. (Adapted from PP 14.6 and PP 14.7.)

Assignment Marking Criteria	Weighting
Correctness of solution: Algorithm is implemented and produces correct results for the stated problem.	/4
Testing: Submission of test exhibits to indicate the solution works for a range of cases (e.g., minimum and maximum inputs) and handles unexpected exceptions.	/2

Comments and documentation: Source code contains comments that explain in plain English what the code is intended to do. /2

Note

Javadoc style is not required.

Total **/8**

Submission status

Attempt number	This is attempt 1.
Submission status	No attempt
Grading status	Not graded
Last modified	-
Submission comments	▸ Comments (0)

Add submission

You have not made a submission yet.

◀ Assignment 1: Analysis of Algorithms & Searching and Sorting (8%)

Jump to...

Assignment 3: Lists & Iterators (8%) ▶

805 TRU Way
Kamloops, BC V2C 0C8
Canada
Contact Us

TRU Student Links

Student Services
Financial Aid
Library

Bookstore
Student Email
Blackboard Learn
Self-service Password Portal
IT Support

Student Moodle Support

Logging In
Getting Started

Faculty Moodle Support

Logging In
Requesting a Course
(Campus Faculty Only)
Course Search
Campus Faculty Support
OLFM Support

TRU's Kamloops campus is situated on the traditional and unceded lands of the Tk'emlúps te Secwépemc within Secwépemc'ulucw, the traditional territory of the Secwépemc people.