CS 161B: Programming and Problem Solving II

Midterm Exam

Academic Integrity

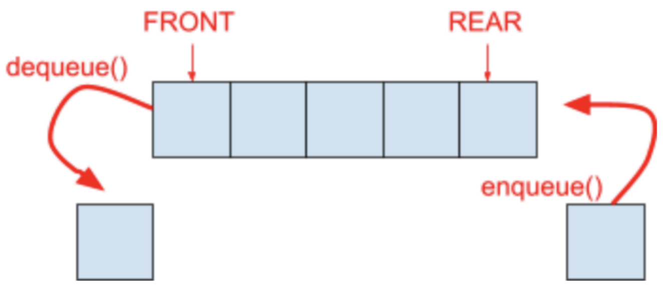
You may NOT, under any circumstances, begin a programming assignment by looking for completed code on StackOverflow or Chegg or any such website, which you can claim as your own. Please check out the Student Code of Conduct at PCC.

The only way to learn to code is to do it yourself. The assignments will be built from examples during the lectures, so ask for clarification during class if something seems confusing. If you start with code from another source and just change the variable names or other content to make it look original, you will receive a zero on the assignment.

I may ask you to explain your assignment verbally. If you cannot satisfactorily explain what your code does, and answer questions about why you wrote it in a particular way, then you should also expect a zero.

A **queue** is a data type with a bounded (predefined) capacity. It is a simple data structure that uses an array and adds elements in one end and removes them from the other end, just like a “queue” or “line” at the market checkout. You enter the line at the rear, and after you’ve paid for your items, you leave at the front. We call this a FIFO (First In First Out) structure. Every time an element is added, it goes in the REAR of the queue, and when an element is removed, all elements before it are also removed (look at the sample run).

For the midterm exam, you will be implementing a **queue** using an array data structure.



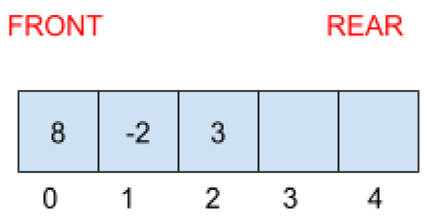
Purpose

The purpose of this midterm exam is to test your knowledge of the concepts learned in this course so far:

- Functions
- Arrays
- Adding, removing, and shifting elements in an array

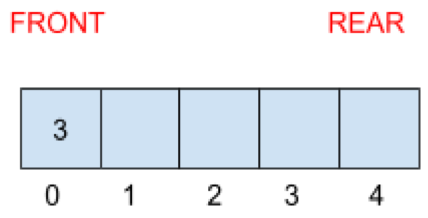
Task

- ❑ Open the Algorithmic Design Document, make a copy, and follow the steps to create your algorithm.
- ❑ You must express your algorithm as **pseudocode** or a **flowchart.**
- ❑ Please take a look at the **sample run** below before you continue!
- ❑ Start with the code in this starter file. You will want to copy it into a directory of your own:
- ❑ You are not allowed to use <string>, <vector> or any other header files in STL.
- ❑ Complete the functions `enqueue()` and `dequeue()`. The function `printQueue()` has been created for you. Please use `printQueue()` to print the queue.
- ❑ Complete the function: `int enqueue(int queue[], int &size, int val)`
    - ❑ Inserts a value into the REAR of a queue of integers.
    - ❑ Takes three arguments:
        - ❑ `queue` The array containing the queue
        - ❑ `size` A reference to the number of elements in the queue
        - ❑ `val` The value to insert
    - ❑ Returns 0 on success, 1 on overflow (queue is full, cannot add).
    - ❑ For example, queue with MAX of 5 and size 3, after integers 8, -2, 3 were added in that order (calls to `enqueue(queue, size, 8), enqueue(queue, size, -2), enqueue(queue, size, 3)`):



- ❑ Complete the function: `int dequeue(int queue[], int &size, int val)`
    - ❑ Removes a value out of a queue of integers.
    - ❑ Integers are removed from the FRONT of the queue.
    - ❑ If the integer is in the queue, ALL integers before are also removed and the remaining elements are shifted to the FRONT of the queue.

    - The first occurrence is removed if there are multiple occurrences.
    - Takes three arguments:
        - `queue` The array containing the queue
        - `size` A reference to the number of elements in the queue
        - `val` The integer to be removed from the queue
    - Returns 0 on success, 1 on underflow (queue is empty, no values to dequeue), 2 if integer is not in the queue.
    - For example, using the queue above, the queue after a call to `dequeue(queue, size, -2)`, notice both elements 8 and -2 were removed from the queue and 3 was moved to the front of the queue, size is now 1:



- Requirements for `enqueue()` and `dequeue()` functions:
    - Do not alter the existing prototypes.
    - Your functions should insert the integers from the REAR of the queue, low indices to high indices; that is, newer elements should be added at higher array indices. FRONT of the queue is at index 0.
    - If an overflow or an underflow occurs, your functions must return an error code; they should not attempt to modify the queue. Handle the return code in the `main()` function, print the error message in `main()`.
    - `enqueue` and `dequeue` should only manipulate the given queue; they should not print anything!
- In your `main()`, an array of size 3 has been declared for you. Using a small array will make it easier to debug all of the conditions. You should then read user commands (input type `char`) and use the queue functions to manipulate the queue accordingly. The program must support the following single-character commands:
    - + Adds an integer into the rear of the queue.
    - - Removes an integer from the front of the queue. All integers before are also removed (see sample run).
    - p Prints the queue.
    - q Quit.
- The queue should be printed using `printQueue()` after every command except q.

❑ If an unknown command is entered, print an error message and DO NOT print the queue.

❑ Use only the concepts we have learned so far.

❑ Please follow ALL the instructions to submit your file in the Criteria for Success section below.

Criteria for Success

❑ Test your program using the following sample runs using the given queue size of 3, making sure you get the same output using the given inputs (in **`blue`**).

`+ 9 + 3 + -2 + 10 - 5 - 3 @ p - -2 - 0 + 23 q`

```
Welcome to the FIFO Queue Program!

Enter option: +
Integer: 9
[9]

Enter option: +
Integer: 3
[9, 3]

Enter option: +
Integer: -2
[9, 3, -2]

Enter option: +
Integer: 10
Error: Queue Overflow!
[9, 3, -2]

Enter option: -
Integer: 5
5 is not in the queue.
[9, 3, -2]

Enter option: -
Integer: 3
[-2]

Enter option: @
Invalid option.

Enter option: p
[-2]

Enter option: -
Integer: -2
[]
```

```
Enter option: -
Integer: 0
Queue Empty.
[]

Enter option: +
Integer: 23
[23]

Enter option: q

Goodbye!
```

- ❏ Complete all sections of your Algorithmic Design Document.
- ❏ Include **pseudocode** or a **flowchart** in part d of the design document.
- ❏ Please open and compare your work with the grading rubric before submitting.
- ❏ Remember to follow all style guidelines.
- ❏ Download your Algorithmic Design Document as a PDF (File -> Download -> PDF), rename it to `midterm.pdf`.
- ❏ Upload your `midterm.cpp` C++ source file and `midterm.pdf` to the D2L assignment folder.
- ❏ Do your own work. Consult the syllabus for more information about academic integrity.