# HW 1: The List Interface

CS 102 Data Structures and Algorithms
Habib University
Spring 2022

All questions carry equal weight.

Figure 1: Intimidator Coaster | Yohsin Theme Park

## 1 Coasterheads

The Intimidator Coaster is a popular attraction at Yohsin Theme Park, but there have been too many complaints about long lines. The newly hired ride manager is responsible for reducing the wait time. She decides to dust off her knowledge of the Queue data structure and come up with interesting new ways to handle the lines.

Currently, the Intimidator has two lines - a Regular line and a Priority line - with different tickets. The Priority ticket costs twice as much as the Regular ticket, and gives priority access to the coaster. Everyone in the Priority line gets seated before anyone in the Regular line. But this causes dissatisfaction in the Regular liners, especially since the Priority line has grown longer recently.

To inject a sense of fairplay, the manager plans on a new kind of ticket, called the Lucky ticket. It will be priced in-between the Regular and Priority tickets, and get you assigned uniformly at random to either the Priority or Regular lines. To execute her plan, she first writes pseudocode using only the basic constructs of programming, and the following Queue interface:

```
Queue(rank) #Creates the queue using the rank function
enqueue(x)  #inserts x into the queue
dequeue()   #removes and returns first element of queue
size()      #returns the length of the queue
```

Listing 1: Queue Interface

The `rank` function is not part of the interface, but has access to the interface functions at the back end.

```
rank(x)     #returns the index of x in the queue
```

**Important**: In the questions that follow, you are required to present pseudo-code that only uses functions from Listing 1, and any additional listings made available in the question. The solution to this question should be typed in LaTeX using the `solution_skeleton.tex` file.

### 1.1

A `RandomQueue` is a type of queue where elements are ordered randomly. Implement the `rank1(x)` function that orders elements uniformly at random, to accompany an imaginary implementation of `RandomQueue`. Apart from previous functions, you have access to the following function:

```
random(x,y) #returns a random integer between x and y, inclusive
```

```
FUNCTION rank1(x)
        SET index to random(0,size(RandomQueue))
        REPEAT
        SET index to random(0,size(RandomQueue))
        UNTIL RandomQueue[index] does not already exist.
        Return index
```

## 1.2

A `PriorityQueue` is a type of queue where elements are ordered according to priority. Suppose there are two priorities that can be assigned to an element mutually exclusively. Implement the `rank2(x)` function that orders elements according to their priority, to accompany an imaginary implementation of `PriorityQueue`. Apart from previous functions, you have access to the following function:

```
priority(x) #returns the priority of x
```

```
FUNCTION rank2(x)
        SET index to 0
        SET p to priority(x)
        SET temp_queue to PriorityQueue
        SET iteration to size(PriorityQueue)
        WHILE iteration is greater than 0
                SET temp_val to temp_queue.dequeue()
                IF priority(temp_val)>=p THEN
                        SET index to index+1
                ELSE
                        ENDWHILE
                SET iteration to iteration - 1
        return index
```

## 1.3

A `RegularQueue` is a type of queue where elements are ordered according to the time of insertion. Implement the `rank3(x)` function to accompany an imaginary implementation of `RegularQueue`.

```
FUNCTION rank3(x)
        SET index to size(RegularQueue)
        return index
```

## 1.4

The manager invents a new type of queue and calls it the `IntimidatorQueue`. Here the elements are ordered according to the new ticketing scheme that includes Regular, Priority and Lucky tickets. Implement the `rank4(x)` function to accompany an imaginary implementation of `IntimidatorQueue`. Apart from previous functions, you have access to the following function:

```
isLucky(x) #returns True if x has a Lucky ticket , and False otherwise
```

```
FUNCTION rank4(x)
        CASE ticketType OF
                Regular: SET p to 1
                Priority: SET p to 0
                Lucky: SET check to isLucky(x)
                       IF check = True THEN SET p to 0 ELSE SET p to 1
        SET index = 0
        SET temp_queue to IntimidatorQueue
        SET iteration to size(IntimidatorQueue)
        WHILE iteration is greater than 0
                SET temp_val to temp_queue.dequeue()
                IF priority(temp_val)>=p THEN
                        SET index to index+1
                ELSE
                        ENDWHILE
                SET iteration to iteration - 1
        return index
```

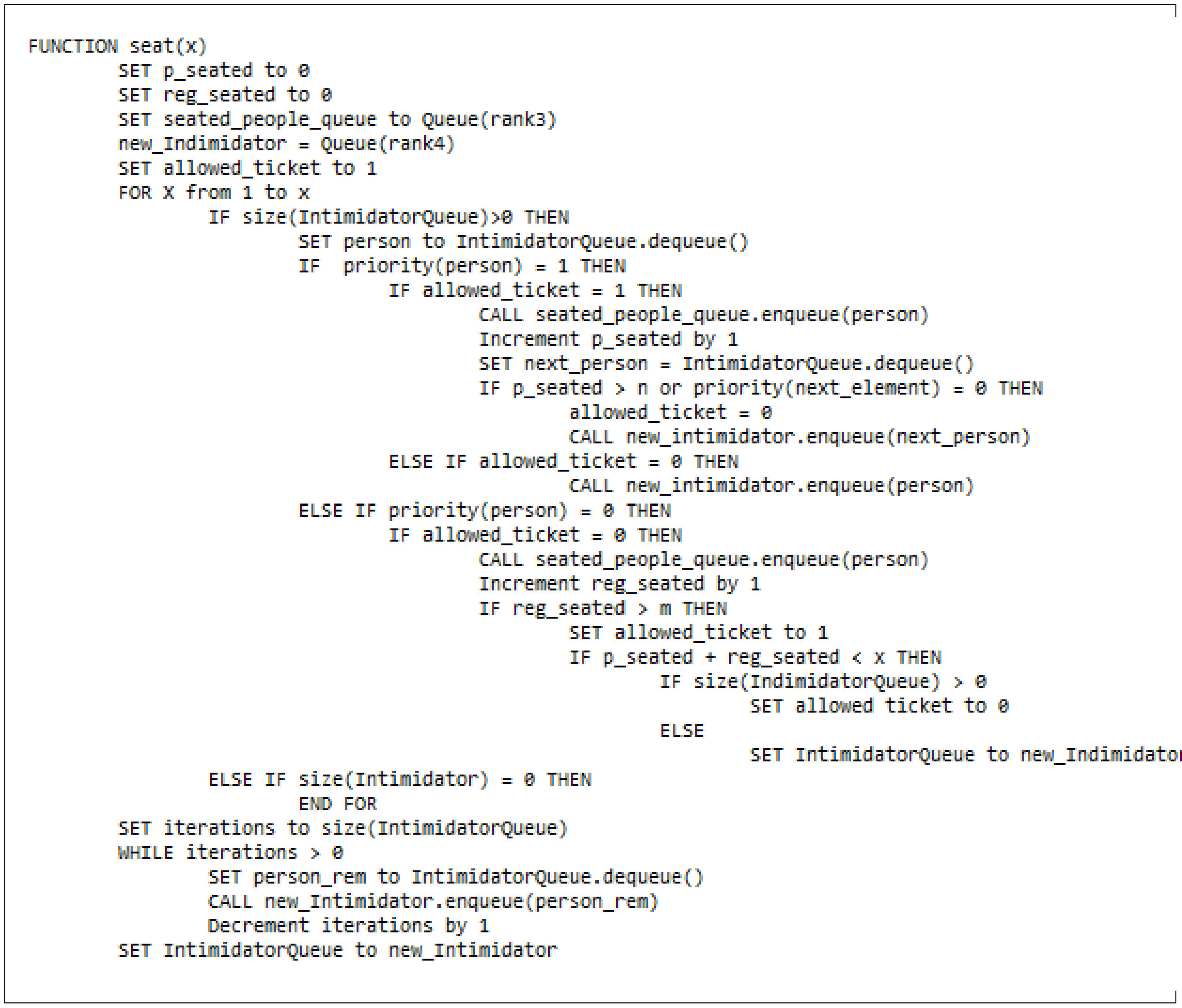
## 1.5

The Priority line is still too long. Before changing the ticket prices, the manager decides a low-key solution: $m$ Regular liners will be seated after every $n$ Priority liners are seated. Initialize an `IntimidatorQueue` by calling the `Queue(rank)` function with the appropriate rank function from before.

```
SET IndimidatorQueue to Queue(rank4)
```

## 1.6

Implement the `seat(x)` function that dequeues $x$ people from the `IntimidatorQueue`. Once the `seat(x)` function is done, the Regular and Priority lines should both be shorter in accordance with the $m$-Regular $n$-Priority dequeueing scheme.



```
FUNCTION seat(x)
        SET p_seated to 0
        SET reg_seated to 0
        SET seated_people_queue to Queue(rank3)
        new_Indimidator = Queue(rank4)
        SET allowed_ticket to 1
        FOR X from 1 to x
                IF size(IntimidatorQueue)>0 THEN
                        SET person to IntimidatorQueue.dequeue()
                        IF  priority(person) = 1 THEN
                                IF allowed_ticket = 1 THEN
                                        CALL seated_people_queue.enqueue(person)
                                        Increment p_seated by 1
                                        SET next_person = IntimidatorQueue.dequeue()
                                        IF p_seated > n or priority(next_element) = 0 THEN
                                                allowed_ticket = 0
                                                CALL new_intimidator.enqueue(next_person)
                                ELSE IF allowed_ticket = 0 THEN
                                                CALL new_intimidator.enqueue(person)
                        ELSE IF priority(person) = 0 THEN
                                IF allowed_ticket = 0 THEN
                                        CALL seated_people_queue.enqueue(person)
                                        Increment reg_seated by 1
                                        IF reg_seated > m THEN
                                                SET allowed_ticket to 1
                                                IF p_seated + reg_seated < x THEN
                                                        IF size(IndimidatorQueue) > 0
                                                                SET allowed ticket to 0
                                                        ELSE
                                                                SET IntimidatorQueue to new_Indimidator
                ELSE IF size(Intimidator) = 0 THEN
                        END FOR
        SET iterations to size(IntimidatorQueue)
        WHILE iterations > 0
                SET person_rem to IntimidatorQueue.dequeue()
                CALL new_Intimidator.enqueue(person_rem)
                Decrement iterations by 1
        SET IntimidatorQueue to new_Intimidator
```

## 1.7

The lines are already looking better after some experimentation, but the manager wants to tweak the Lucky ticket further. Instead of assigning people to the two lines uniformly at random, she makes the assignment based on the ratio $m : n$. The Lucky person gets assigned the Priority line $\frac{m}{n}$ of the times and the Regular line $1 - \frac{m}{n}$ of the times ($n$ is always greater than $m$). Implement the `rank5(x)` function according to this new scheme. Assume that all prior functions are available.

```
FUNCTION rank5(x)
        SET total_lucky to 0
        SET total_unlucky to 0
        SET max_luck to (m/n)*size(seated_people_queue)
        SET max_unluck to (1-(m/n))*size(seated_people_queue)

        CASE ticketType OF
                Regular: SET p to 1
                Priority: SET p to 0
                Lucky: SET loop to TRUE
                           SET check = random(0,1)
                                IF check = 1 THEN
                                        IF total_lucky < max_luck THEN
                                                SET p to 1
                                                Increment total_lucky by 1
                                        ELSE
                                                IF total_unlucky < max_unluck
                                                        SET p to 0
                                                        Increment total_unlucky by 1
                                                ELSE
                                                        SET p to 1
                                                        SET total_lucky by 1
                                ELSE
                                        IF total_unlucky <  THEN
                                                SET p to 0
                                                Increment total_unlucky by 1
                                        ElSE
                                                IF total_lucky < max_luck
                                                        SET p to 1
                                                        Increment total_lucky by 1
                                                ELSE
                                                        SET p to 0
                                                        SET total_unlucky by 1

        SET index = 0
        SET temp_queue to IntimidatorQueue
        SET iteration to size(IntimidatorQueue)
        WHILE iteration is greater than 0
                SET temp_val to temp_queue.dequeue()
                IF priority(temp_val)>=p THEN
                        SET index to index+1
                ELSE
                        ENDWHILE
                SET iteration to iteration - 1
        return index
```

## 2 Paranthetical

Ponyo's friends made him a birthday present — a bracket sequence! Ponyo was quite disappointed with his gift, because he dreamt of correct bracket sequences, yet his friends were uncultured in the art of correct bracket sequences.

To make everything right, Ponyo is going to move at most one paranthesis from its original place in the sequence to any other position. Reversing the paranthesis (e.g. turning "(" into ")" or vice versa) isn't allowed.

A bracket sequence, $s$, is called correct if:

1. $s$ is empty.

2. $s$ is equal to $(t)$, where $t$ is a correct bracket sequence.

3. $s$ is equal to $t_1t_2$, where $t_1$, $t_2$ are correct bracket sequences.

For example, "(()())", "()" are correct, while ")(" and "())" are not. Help Ponyo fix his birthday present and understand whether he can move one paranthesis so that the sequence becomes correct. You are required to submit code for this question using the HackerRank website here

## 3 Balanced Braces

One of the most important applications of Stacks is to check if the parentheses are balanced in a given expression. You are designing a compiler for a programming language and need to check that braces in any given file are balanced. Braces in a string are considered to be balanced if the following criteria are met:

1. Braces come in pairs of the form (), {} and []. The left brace opens the pair, and the right one closes it. All brace pairs must be closed.
2. In any set of nested braces, the braces between any pair must be closed. For example, [{}] is a valid grouping of braces but []]{} is not.

You are required to submit code for this question using the HackerRank website here

## Credits

This homework and related files are due in part to Aisha Batool.