HABIB UNIVERSITY

Data Structures & Algorithms

CS/CE 102/171 Spring 2023

Instructor: Maria Samad

Implementing Queues

Student 1: _____

INSTRUCTIONS:

1. **Project Pitch**
   - Ms. Maria Samad is a very popular teacher of the department (AS IF 😜) She is teaching 2 sections of DSA this semester, which due to her popularity have been completely filled. However, there are 14 more students who still want to join her class. RO makes a request to her to allow at least 7 more students in each of her section in order to accommodate the remaining students. Ms. Maria Samad, being an enthusiast as she is, gives her approval to enroll all the students (I KNOW, RIGHT 🤦‍♀️)
   - The main task is to go through a series of activities and get all the students enrolled in one of the two sections
2. **Requirements**
   - **LISTEN/READ** attentively, and then **ANALYZE** carefully
3. **Solution**
   - Follow the requirements and design an algorithm to implement, and show the outputs at each stage in the design
4. **Restrictions**
   - Student IDs and their section choice (‘A’ or ‘B’) of the waiting students is provided in the form of list of tuples
   - Each section’s list of students must be represented by QUEUES, i.e. QueueA & QueueB
   - Pending students later would be placed in another QUEUE, i.e. QueueWait
   - Allowed to use only ONE extra Queue OR ONE extra Stack
   - The number of waiting students is 14 for this example, but the design should work for any size of data
   - **MUST NOT USE ANY OTHER DATA STRUCTURES EXCEPT FOR THE ONES MENTIONED IN RESTRICTIONS!**

ACTIVITIES:

1. Activity 1
2. Activity 2
3. Activity 3

ACTIVITY 1:

The students have entered their IDs (numeric) and choices (character: either ‘A’ or ‘B’), and the system has generated a list of tuples. Extract the IDs from the list and define two separate sequences representing each section. Even though a list of tuples is provided to you for reference, but that is there only to visualize/understand the problem and its inputs/outputs. However, when designing an algorithm, you must generalize your design, and not make it specific to just this list of students. **DO KEEP THE RESTRICTIONS IN MIND WHEN DESIGNING THE ALGORITHM**

Test input:

- waiting_list = [(21, ‘A’), (95, ‘B’), (17, ‘A’), (16, ‘A’), (76, ‘B’), (30, ‘B’), (33, ‘A’), (8, ‘A’), (87, ‘B’), (92, ‘B’), (57, ‘A’), (25, ‘B’), (66, ‘B’), (49, ‘A’)]

ACTIVITY 2:

- RO enrolls the students according to the Queues defined in the previous task. However, 3 students from Section 'A' have some time clashes, so they request RO to place them in Section 'B' instead. Section 'B' is already full, so RO suggests that they can be placed in a waiting sequence on first-come-first-served basis, and as soon as there are students in Section 'B', who are willing to swap their section, they will be placed in Section 'B' instead.
- Assume the **smallest 3 odd-valued IDs** to be the ones requesting for change in section
- The order of student IDs even after removing the smallest 3 elements MUST NOT CHANGE!
- Use QueueWait to hold the waiting sequence of student IDs.

DO KEEP THE RESTRICTIONS IN MIND WHEN DESIGNING THE ALGORITHM

ACTIVITY 3:

- Few days later, 3 students from Section B decided to swap their sections with the waiting students, so RO changes their sections
- Assume the **first 3 even-valued IDs** from QueueB to be the ones willing to swap for change in section
- The order of student IDs even after removing the first 3 elements

DO KEEP THE RESTRICTIONS IN MIND WHEN DESIGNING THE ALGORITHM