Namal University, Mianwali
Department of Electrical Engineering
Course Title: EE-253 Data Structures and Algorithms

LAB MANUAL

Lab 7 Queues in Python

In previous lab, we have studied the concept of stacks and their implementation in python. Stack works on the process of Last In First Out (LIFO).

1. Lab Objectives

The objective of this lab is to introduce students to the concept of Queue.

2. Lab Outcomes

- CLO:1 Recognize the usage of fundamental Data structure using Python Programming Language.
- CLO:3 Demonstrate solution to real life problems using appropriate data structures.

3. Equipment

- Software
    - IDLE (Python 3.11)

4. Instructions

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as 'Example') are for practice purposes only while others (marked as 'Task') have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
4. The report should be submitted on the given template, including:
    a. Code (copy and pasted, NOT a screenshot)
    b. Output figure (as instructed in 3)
    c. Explanation where required
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. Plagiarism or any hint thereof will be dealt with strictly. Any incident where plagiarism is caught, both (or all) students involved will be given zero marks, regardless of who copied whom. Multiple such incidents will result in disciplinary action being taken.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X_DSA_LabReportX.pdf).

5. Introduction

5.1 Queue:

As stack works on the concept of Last In First Out (LIFO), Queue is a linear data structure that works on the principle of First In First Out (FIFO). Elements can be inserted at any time from one end and the elements that has been longest in the queue will be removed first.
Real time application of Queue can be passengers standing on the counter of bus station for purchasing tickets.

Graphical Representation of Queue is given below in Figure 1.

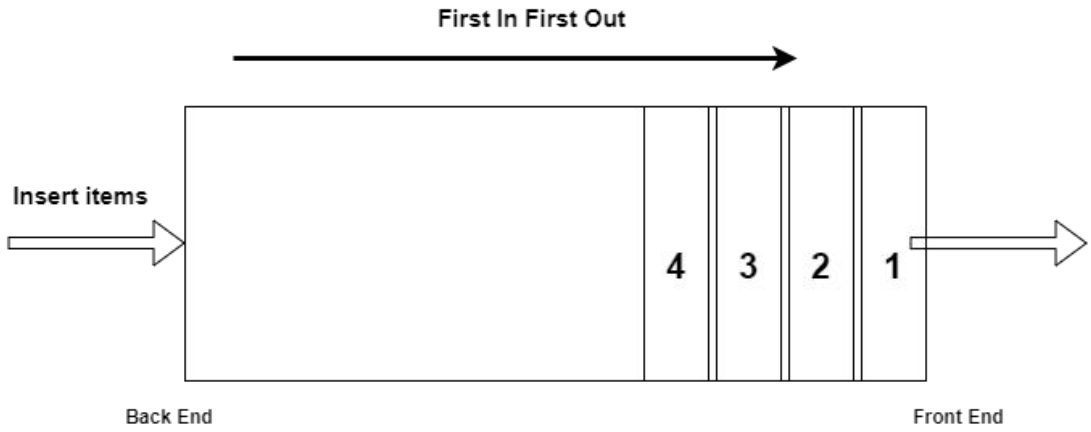

Figure 1 Graphical Representation of Queue

There are two processes being conducted inside Queue,

a) Enqueue: Add elements to the back of the queue.
b) Dequeue: Remove elements from the front of the Queue.

5.2 Implementation of Queue in Python:

The components required for Implementing queue through circular array requires following components.

Components:

1. Class
   1.a) __init__ method
   1.b) Method for length
   1.c) Method that checks if queue is empty
   1.d) Method that returns the element of queue at front
   1.e) dequeue method
   1.f) enqueue method
   1.g) resize method

1.h) print the whole queue

2. Main

Python Code:

```
class ArrayQueue:
   #FIFO queue implementation using a Python list as underlying storage."""

   DEFAULT_CAPACITY = 5 # moderate capacity for all new queues

   def __init__(self):
      #Create an empty queue."""
       self.data = [None] * ArrayQueue.DEFAULT_CAPACITY
       self.size = 0
       self.front = 0

   def __len__(self):
      #Return the number of elements in the queue."""
      return self.size

   def is_empty(self):
      #Return True if the queue is empty."""
      return self.size == 0

   def first(self):
      #Return (but do not remove) the element at the front of the queue.

     # Raise Empty exception if the queue is empty.

      if self.is_empty():
         raise Empty("Queue is empty")
      return self.data[self.front]

   def dequeue(self):
      #Remove and return the first element of the queue (i.e., FIFO).

      #Raise Empty exception if the queue is empty.
     # Students are required to write dequeue method by their own

   def enqueue(self, e):
      #Add an element to the back of queue."""
      if self.size == len(self.data):
         print("control comes inside enqueue")
         self.resize(2 * len(self.data)) # double the array size
      print("self.front="+str(self.front))
```

```
        print("self.size="+str(self.size))
        print("length of self.data"+str(len(self.data)))
        avail = (self.front + self.size) % len(self.data)
        print("avail="+str(avail))
        self.data[avail] = e
        self.size += 1

    def resize(self, cap): # we assume cap >= len(self)
        #Resize to a new list of capacity >= len(self)."""
        print("control comes inside resize method")
        old = self.data # keep track of existing list
        print(old)
        self.data = [None] * cap # allocate list with new capacity
        print(self.data)
        walk = self.front
        print("walk = "+str(walk))
        print("self.size"+str(self.size))
        for k in range(self.size): # only consider existing elements
            self.data[k] = old[walk] # intentionally shift indices
            walk = (1 + walk) % len(old) # use old size as modulus
        self.front = 0 # front has been realigned
    def print(self): # print the queue
        # Students are required to design this method by themselves.

# Make code for the main
```

* print commands in the above example is added to make your understanding better about the code.

6 Lab Tasks

Task 1: Make DEQUEUE method for the class provided in Example.

Task 2: Design PRINT method for printing the whole queue in Example.

Task 3: Write the main portion for making a queue. Call all the methods to see the behavior of the code.

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-2**	**Marginal 3-5**	**Satisfactory 6-8**	**Exemplary 9-10**	**Allocated Marks**
Psychomotor	**CLO:1 R2**	Implementation with Results **(P)**	Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem.	Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result.	Refines solutions suggested by others. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced.	Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend for the instructor. Results are correctly produced. Student incorporated comments in the code.	
Affective	**CLO:3 R3**	Lab Report **(A)**	Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated.	Code of the problem is given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around. More than two incidents of talking non-lab related stuff in laband/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab activity.	Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff in lab Voice level exceeding theappropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceedingthe appropriate level, use of cell phones and involvementin any non-lab related activity.	Stayed in seat and got up for a specific lab related reason. Tookcare of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity.	