UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Villanueva, Bryan O.

Instructor:
Engr. Maria Rizette H. Sayo

October 11, 2025

I. Objectives

Introduction

Another fundamental data structure is the queue. It is a close “the same” of the stack, as a queue is a collection of objects that are inserted and removed according to the first-in, first-out (FIFO) principle. That is, elements can be inserted at any time, but only the element that has been in the queue the longest can be next removed.

The Queue Abstract Data Type

Formally, the queue abstract data type defines a collection that keeps objects in a sequence, where element access and deletion are restricted to the first element in the queue, and element insertion is restricted to the back of the sequence. This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in, first-out (FIFO) principle. The queue abstract data type (ADT) supports the following two fundamental methods for a queue Q:

Q.enqueue(e): Add element e to the back of queue Q.

Q.dequeue( ): Remove and return the first element from queue Q;
an error occurs if the queue is empty.

The queue ADT also includes the following supporting methods (with first being analogous to the stack’s top method):

Q.first(): Return a reference to the element at the front of queue Q, without removing it;
an error occurs if the queue is empty.

Q.is empty( ): Return True if queue Q does not contain any elements.

len(Q): Return the number of elements in queue Q; in Python, we implement this with the special method len .

This laboratory activity aims to implement the principles and techniques in:

- Writing Python program using Queues

Writing a Python program that will implement Queues operations

II. Methods

Instruction: Type the python codes below in your Colab. Reconstruct them by implementing Queues (FIFO) algorithm. Hint: You may use Array or Linked List

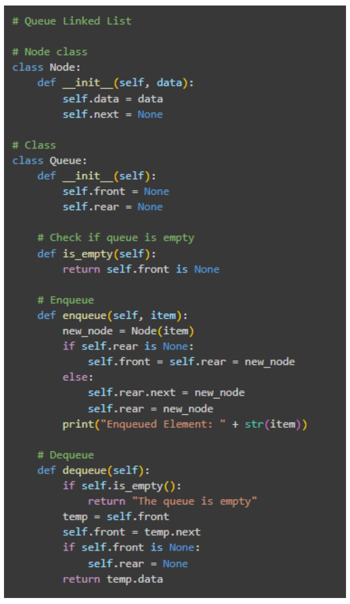
```
# Stack implementation in python

# Creating a stack
def create_stack():
    stack = []
    return stack
```

```
# Creating an empty stack
def is_empty(stack):
    return len(stack) == 0

# Adding items into the stack
def push(stack, item):
    stack.append(item)
    print("Pushed Element: " + item)

# Removing an element from the stack
def pop(stack):
    if (is_empty(stack)):
        return "The stack is empty"
    return stack.pop()

stack = create_stack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
push(stack, str(5))

print("The elements in the stack are:"+ str(stack))
```

Answer the following questions:

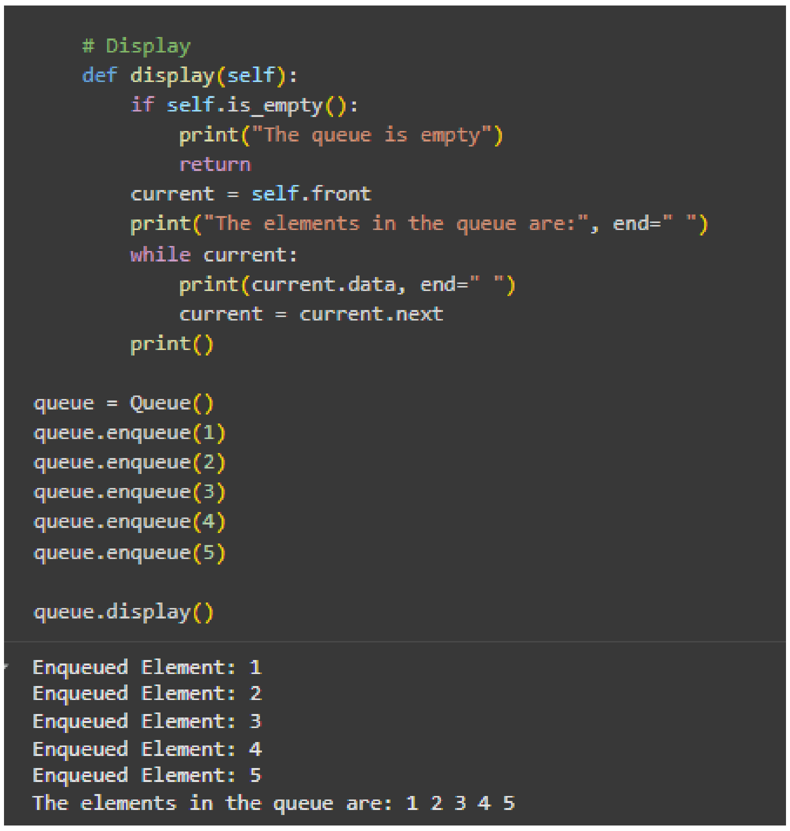
1 What is the main difference between the stack and queue implementations in terms of element removal?
2 What would happen if we try to dequeue from an empty queue, and how is this handled in the code?
3 If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?
4 What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?
5 In real-world applications, what are some practical use cases where queues are preferred over stacks?

6 Results



```
# Queue Linked List

# Node class
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# Class
class Queue:
    def __init__(self):
        self.front = None
        self.rear = None

    # Check if queue is empty
    def is_empty(self):
        return self.front is None

    # Enqueue
    def enqueue(self, item):
        new_node = Node(item)
        if self.rear is None:
            self.front = self.rear = new_node
        else:
            self.rear.next = new_node
            self.rear = new_node
        print("Enqueued Element: " + str(item))

    # Dequeue
    def dequeue(self):
        if self.is_empty():
            return "The queue is empty"
        temp = self.front
        self.front = temp.next
        if self.front is None:
            self.rear = None
        return temp.data
```

Figure 1 Program Input



```
    # Display
    def display(self):
        if self.is_empty():
            print("The queue is empty")
            return
        current = self.front
        print("The elements in the queue are:", end=" ")
        while current:
            print(current.data, end=" ")
            current = current.next
        print()

queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
queue.enqueue(4)
queue.enqueue(5)

queue.display()
```

```
Enqueued Element: 1
Enqueued Element: 2
Enqueued Element: 3
Enqueued Element: 4
Enqueued Element: 5
The elements in the queue are: 1 2 3 4 5
```

Figure 2 Program Output

SOURCE CODE:

```
# Queue Linked List
# Node class
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# Class
class Queue:
    def __init__(self):
        self.front = None
        self.rear = None

    # Check if queue is empty
    def is_empty(self):
        return self.front is None

    # Enqueue
    def enqueue(self, item):
        new_node = Node(item)
        if self.rear is None:
            self.front = self.rear = new_node
        else:
            self.rear.next = new_node
            self.rear = new_node
        print("Enqueued Element: " + str(item))

    # Dequeue
    def dequeue(self):
        if self.is_empty():
            return "The queue is empty"
        temp = self.front
        self.front = temp.next
        if self.front is None:
            self.rear = None
        return temp.data

    # Display
    def display(self):
        if self.is_empty():
            print("The queue is empty")
            return
        current = self.front
        print("The elements in the queue are:", end=" ")
        while current:
            print(current.data, end=" ")
            current = current.next
        print()

queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
queue.enqueue(4)
queue.enqueue(5)
queue.display()
```

ANSWERS:

1. What is the main difference between the stack and queue implementations in terms of element removal?
   - The main difference is that in a stack, the last element added is the first one removed or Last In, First Out. In a queue, the first element added is the first one removed or First In, First Out.

2. What would happen if we try to dequeue from an empty queue, and how is this handled in the code?
   - If we try to dequeue from an empty queue, there will be nothing to remove. Usually, the code checks if the queue is empty first and gives a message like "Queue is empty" instead of trying to remove something.

3. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?
   - If we change the enqueue operation to add elements at the beginning, the new element will always go to the front. This means the queue will act more like a stack, because the most recently added element will be removed first.

4. What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?
   - Using a linked list for a queue makes it easy to add and remove elements without worrying about size limits. But it uses more memory. Using an array is simpler and faster to access items, but it has a fixed size and may need resizing when it gets full.

5. In real-world applications, what are some practical use cases where queues are preferred over stacks?
   - Queues are used in real life cases like printing jobs, customer service lines, task scheduling, or data buffering, where the first task that comes should be handled first.

III. Conclusion

In this activity, I learned how to use and apply the concept of queues in Python. I understood that a queue follows the First In, First Out or FIFO rule, where the first element added is also the first one to be removed. I was able to practice how to add elements using enqueue and remove elements using dequeue. This lab helped me see how queues are different from stacks and how they can be used in real life situations where order and sequence are important.