UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Directo, Hannah Thea B.

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?

III. Results

```python
# Creating a queue
def create_queue():
    queue = []
    return queue

def is_empty(queue):
    return len(queue) == 0

def enqueue(queue, item):
    queue.append(item)
    print("Enqueued Element: " + item)

def dequeue(queue):
    if is_empty(queue):
        return "The queue is empty"
    return queue.pop(0)


queue = create_queue()
enqueue(queue, str(1))
enqueue(queue, str(2))
enqueue(queue, str(3))
enqueue(queue, str(4))
enqueue(queue, str(5))

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

while not is_empty(queue):
    print("Dequeued Element: " + dequeue(queue))

print("Queue after dequeuing all elements: " + str(queue))
```

Figure 1 Screenshot of Program

```
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']
Dequeued Element: 1
Dequeued Element: 2
Dequeued Element: 3
Dequeued Element: 4
Dequeued Element: 5
Queue after dequeuing all elements: []
```

Figure 2 Screenshot of Output

1. The main difference between a stack and a queue is in how items are added and removed. In a stack, items are added to the top and also removed from the top—this means the last item added is the first one taken out (Last In, First Out or LIFO). In a queue, items are added to the back and removed from the front—so the first item added is the first one removed (First In, First Out or FIFO).

2. If we try to remove (or dequeue) an item from an empty queue, there's nothing to remove. The program checks first if the queue is empty using a condition. If it is, it returns a message like "The queue is empty" instead of trying to remove something, which helps avoid errors in the program.

3. If we modify enqueue so that new items are added at the front instead of the back, the queue will stop working like a proper queue. This change would cause the most recent item to be removed first, which is how a stack works. It would break the first-in, first-out rule and make it behave more like a last-in, first-out structure.

4. Queues can be made using arrays or linked lists. When using an array, the items are stored next to each other, so it's faster to find or update them. But arrays have a fixed size, so if the queue gets too full, it's harder to add more items unless we make a new, bigger array. On the other hand, linked lists can easily grow or shrink because each item is connected by links. This

makes them more flexible, but they use more memory and can be a little slower because the computer must follow the links between items.

5. Queues are helpful when things need to be done in order they arrive. For example, when people line up at a store, the first person in line is served first. In computers, queues are used for things like printing documents, where the first file sent to the printer gets printed first, or for running tasks in order, so everything is done fairly and correctly.

IV. Conclusion

In this activity, we learned how to create and use a queue, which follows the rule that the first item added is the first one removed (FIFO). We saw how queues are different from stacks, where the last item added is removed first (LIFO). By writing Python code to add and remove items from a queue, we better understood how queues work and why they are useful in many real-life situations like waiting lines and task scheduling.