



## Data Structure and Algorithm Laboratory Activity No. 9

---

# Queues

---

*Submitted by:*  
Villacin, Justine R.

*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

Present the visualized procedures done. Also present the results with corresponding data visualizations such as graphs, charts, tables, or image . Please provide insights, commentaries, or explanations regarding the data. If an explanation requires the support of literature such as academic journals, books, magazines, reports, or web articles please cite and reference them using the IEEE format.

Please take note of the styles on the style ribbon as these would serve as the style format of this laboratory report. The body style is Times New Roman size 12, line spacing: 1.5. Body text should be in Justified alignment, while captions should be center-aligned. Images should be readable and include captions. Please refer to the sample below:

```

▶ # Queue implementation in python using Array

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

# Check if the queue is empty
def is_empty(queue):
    return len(queue) == 0

# Adding items into the queue (enqueue)
def enqueue(queue, item):
    queue.append(item)
    print("Enqueued Element: " + str(item))

# Removing an element from the queue (dequeue)
def dequeue(queue):
    if (is_empty(queue)):
        return "The queue is empty"
    return queue.pop(0)

print("== ARRAY-BASED QUEUE IMPLEMENTATION ==\n")
queue = create_queue()
enqueue(queue, 1)
enqueue(queue, 2)
enqueue(queue, 3)
enqueue(queue, 4)
enqueue(queue, 5)

```

Figure 1 Source Code

```

→ == ARRAY-BASED QUEUE IMPLEMENTATION ==

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 Result of Program

Answers:

- 1 What is the main difference between the stack and queue implementations in terms of element removal?
  - *In a **stack**, elements are removed from the **top** (Last In, First Out – LIFO).*
  - *In a **queue**, elements are removed from the **front** (First In, First Out – FIFO).*
- 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 are no elements to remove.*
- 3 If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?
  - *Adding elements at the **beginning** instead of the **end** would reverse the normal order of processing.*
  - *This would make the queue behave like a **stack** (LIFO), since the most recently added element would be the first to be removed.*
- 4 What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?

#### LINKED LIST

- You can keep adding elements without worrying about the size.
- slightly more complex to implement

#### ARRAY

- Simple to use and understand
- If the array gets full, you can't easily add more elements.

- 5 In real-world applications, what are some practical use cases where queues are preferred over stacks?
  - *Online food delivery system – Orders are prepared and delivered in the order they were received.*

## IV. Conclusion

*In this lab, we explored the difference between **stacks (LIFO)** and **queues (FIFO)**. By reconstructing the stack program into a queue, we learned how the order of element addition and removal changes the data structure's behavior. Queues are widely used when maintaining order of processing is important, while stacks are used for tasks like undo operations or backtracking. Understanding both structures is essential for efficient problem-solving and real-world programming.*

## **References**

- [1] Co Arthur O.. “University of Caloocan City Computer Engineering Department Honor Code,” UCC-CpE Departmental Policies, 2020.