Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Monoy, Justin Rhey A. | Engr. Maria Rizette H. Sayo |

October 11, 2025

# 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

# 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?

# Results

1. What is the main difference between the stack and queue implementations in terms of element removal?

In the queue, the first inserted element is removed from the front, whereas in the stack, the last entered element is removed from the end according to a rule called LIFO.

2. What would happen if we try to dequeue from an empty queue, and how is this handled in the code?

A black background with text

AI-generated content may be incorrect.

Figure 1 Screenshot of program

Instead of calling on an empty list, attempting to dequeue from an empty queue will prevent an error and safely provide a message such as "The stack is empty.".

3. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?

First In, First Out (FIFO) becomes Last In, First Out (LIFO) if the enqueue operation is changed to add elements at the beginning instead of the end. The reason for this is that as new elements are added in front and removed from the front, the most recent element is dequeued first, so turning the queue into a stack.

4. What are the advantages and disadvantages of implementing a queue using linked lists versus arrays?

Using linked lists for queues allows you to add and remove items easily and increase the queue as needed, but it also uses more memory and is a little more difficult to implement. Arrays are easier to create and retrieve, but because the size is fixed or requires resizing and moving elements, deleting elements may be slow.

5. In real-world applications, what are some practical use cases where queues are preferred over stacks?

Printer queues, customer service systems, task scheduling, traffic management, and messaging systems

# Conclusion

The most recent item is removed first in stacks, while the earliest item is removed first in queues. A queue's behavior could change if items are added in a different way. Real-world applications for queues include printing and lines, where items must be processed in the order they come. The jobs are arranged and managed with the use of these data structures. Solving issues where timing and order are crucial is made simpler when one is aware of how they operate.

**References**

[1] “W3Schools.com.” <https://www.w3schools.com/python/python_arrays.asp>

[2] GeeksforGeeks, “Queue in Python,” *GeeksforGeeks*, Sep. 20, 2025. <https://www.geeksforgeeks.org/python/queue-in-python/>

[3] D. Bader, “Common Python Data Structures (Guide),” Oct. 21, 2023. <https://realpython.com/python-data-structures/>