Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Aquino, Jester J. | 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?

The stack has a rule like (LIFO) it removes the last inserted element to removes from the end while the queue also has a rule like (FIFO) it removes the first inserted element to remove from the front.

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

If we try to dequeue from an empty queue it will prevents an error and it will safety return a message like “The stack is empty” instead of calling on an empty list.

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

The queue procedure changes from FIFO (First In, First Out) to LIFO (Last In, First Out) if we change the enqueue operation to add elements at the beginning rather than the end. This is because new elements are added in front and removed from the front, which causes the most recently added element to be dequeued first, essentially converting the queue into a stack.

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

While it consumes more memory and is a little more difficult to code, using linked lists for queues allows you to add and delete items fast and expand the queue as needed. Although arrays are quicker to access and easier to create, deleting things might be slow because the size is fixed or requires resizing and elements must be moved.

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

Customer service lines / Waiting lines, Printer, Online order,

# Conclusion

Stacks and queues store data differently stacks remove the newest item first, while queues remove the oldest. Changing how items are added can change a queue’s behavior. Queues are useful in real life when things need to be handled in the order they arrive, like in lines, printing.

**References**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Python* (2nd ed.). Wiley.

GeeksforGeeks. (n.d.). *Queue Data Structure*. Retrieved from https://www.geeksforgeeks.org/queue-data-structure/