Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Tan, Charles Dominic S. | 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?

A stack follows the **LIFO** rule, which means it removes the last item added from the end. A queue follows the **FIFO** rule, which means it removes the first item added 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?

A queue usually works with the FIFO (First In, First Out), adding the new items at the end, and removing from the front. But if you can change it to add new items at the front instead, it becomes LIFO (Last In, First Out). That means the newest item gets removed first and just like a stack. So, changing where you add items can turn the queue into a stack.

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

If using the linked lists for queues is can use more memory and be harder to code, but if they let you add, and remove the items quickly and grow the queue easily Also arrays are faster to access and simpler to make, but removing the items can be slow because arrays have a fixed size or they need resizing, and also elements have to be moved around.

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 in different ways. Stacks take out the newest item first, while queues take out the oldest item first. How you add items can change how a queue works. Queues are helpful in real life for things that need to be done in the order they come, like waiting in line or printing documents.

**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/