Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Hermosura, Leigh B. | 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 main difference between the two abstract data types in terms of element removal is how they follow different approaches. Stack uses a Last-in First-out (LIFO) method of removing elements while a queue uses a First-in First-out (FIFO) method

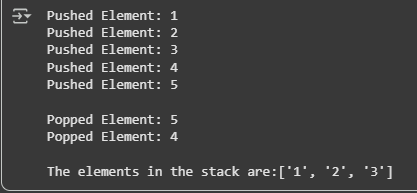
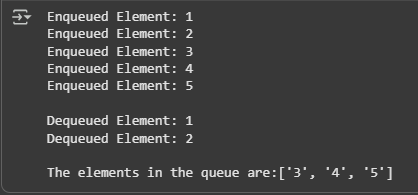


Figure 1 Removing Items in a Stack Figure 2 Removing Items in a Queue

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

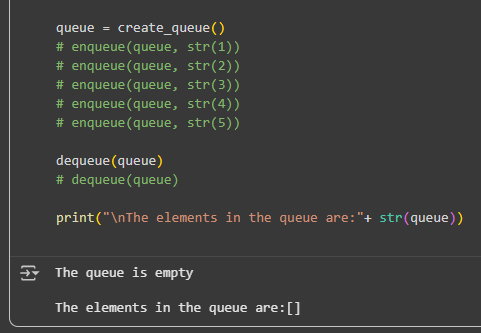
* If we try to dequeue an empty queue, the function first examines if the queue *is\_empty()* by checking if its length is equal to 0. Once confirmed to be 0, the function returns the message “The queue is empty”.

Figure 3 Calling the Dequeue Function on an Empty Queue

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

* If the enqueue operation is modified to add elements at the beginning, since the program’s structure is simple and is coded to dequeue the first element in the queue, it would turn into a LIFO process which is a method used in stacks, not queues. If we wish to follow the FIFO procedure, we would also need to modify the *dequeue()* function.

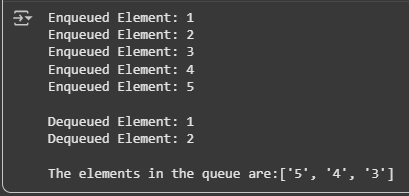
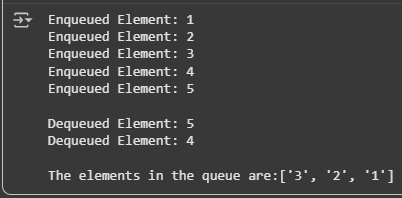


Figure 4 Modified Enqueue Operation Figure 5 Modified Enqueue and Dequeue Operation

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

* The advantages of implementing a queue using arrays is that it has faster access time and less memory is used due to not having extra pointer fields. The disadvantage on the other hand is that it stores data in a container with a fixed size, it has trouble in resizing as it’s complex and it can lower performance.
* The advantages of implementing a queue using linked lists is that it’s dynamic, allowing to shrink or grow without the need for complex resizing unlike in arrays. It also excels at insertion and deletion of elements if node pointers are structured properly. The disadvantage of using linked lists is that it uses more memory for the pointer fields.

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

* Practical use cases where queues are preferred over stacks to benefit from their FIFO structure are web servers where the client requests are prioritized in order by which came first. Another example would be printer queues where print jobs are added to the queue and processed in the order they’re received [1]. These are important in urgent situations where the first to be queued will be the first to be processed.

# Conclusion

The understanding between abstract data types like stacks and queues is essential for selecting the optimal data structure for different problems. Stacks follow a Last-in First-out (LIFO) procedure whereas queues operate using First-in First-out (FIFO). Mastery of these concepts will allow for efficient and effective use of each data types in real-world applications.

**References**

[1] GeeksforGeeks, “Applications of Queue Data Structure,” *GeeksforGeeks*, Mar. 2011. https://www. geeksforgeeks.org/dsa/applications-of-queue-data-structure/