Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Polestico,Paul Justine D. | 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?

Answer:

In stacks, the last element to be added would be the first to be removed from the list **(Last In,First out**). In Queues, the first element to be added would be the first to be removed from the list(**First In,First Out**).

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

Answer:

If we try to dequeue from an empty queue, the code first checks whether the queue is empty. If it is empty, the program will not attempt to remove any element. Instead, it will either return a message indicating that the queue is empty or handle the situation gracefully to prevent an error or crash.

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

Answer:

If elements are added at the beginning, the data structure behaves like a stack (LIFO), not a queue (FIFO). This change makes it unusual and inefficient for queue operations.

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

Answer:

While arrays are easier to understand,they have limitations on their size and the data type allowed. Linked lists require more complex coding and understanding of data types,but offer dynamic size and can store various types of data.

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

Answer:

An example would be in fast food chains like Jolibee and McDonalds where if you arrive first,you are the first to be served. Another example would be in printing shops where maintaining the order of jobs is essential to reduce confusion among customers.

# Results

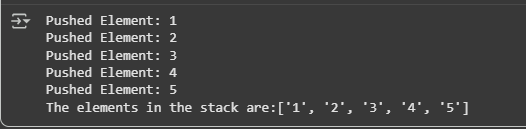


Figure 1 Stacks

A black and white screen with white text

AI-generated content may be incorrect.

Figure 2 Queues

Conclusion

Queues are essential for maintaining order especially in large businesses due to its ability to arrange data in sequential order,providing improved production and quality of service. Without Queues, it can increase chances of confusion and decrease in quality of service among its customers,affecting both sections of the business.

**References**

[1] *W3Schools.com*. (n.d.). <https://www.w3schools.com/python/python_dsa_queues.asp>

[2] GeeksforGeeks. (2025, August 31). *Stack Data Structure*. GeeksforGeeks. <https://www.geeksforgeeks.org/dsa/stack-data-structure/>

[3] GeeksforGeeks. (2025a, July 23). *Queue data structure*. GeeksforGeeks. <https://www.geeksforgeeks.org/dsa/queue-data-structure/>