Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Gabuyo, Ivan love D. | Engr. Maria Rizette H. Sayo |

OCT, 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?  
     
   Consider a stack to be analogous to the "undo" function in a text message or document. Your most recent action is the first to be undone. We call it "last in, first out" because it always works with the most recent addition.

Conversely, a queue functions similarly to a coffee shop line. The first person to get coffee and head out is the first one to arrive. The oldest task is always processed first in this equitable "first come, first served" system.

1. What would happen if we try to dequeue from an empty queue, and how is this handled in the code?  
     
   It's like reaching into an empty bag when you try to take something out of an empty queue; you get nothing and are left perplexed.

The code is clever to prevent this. It checks inside the queue first. It doesn't attempt to grab anything that isn't there if it discovers that the queue is empty. Rather, it simply prints the unambiguous message, "The queue is empty."

1. If we modify the enqueue operation to add elements at the beginning instead of the end, how would that change the queue behavior?  
   By removing the very feature that makes a queue unique, this one change effectively flips the entire processing order.

This single modification essentially reverses the entire processing order by eliminating the very characteristic that distinguishes a queue.

By removing the very feature that makes a queue unique, this one change effectively flips the entire processing order.

The personality of the line will completely change if we begin incorporating new elements at the front rather than the back. The last item added would be the first item removed if it operated more like a stack and disregarded the "first in, first out" principle.

This single modification essentially reverses the entire processing order by eliminating the very characteristic that distinguishes a queue.

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

Because everything else in the line must move when the items at the front are removed, the line may become sluggish. However, creating a queue with an array (or Python list) is easy to understand and program. However, adding and removing items from a linked list is much faster because nothing needs to be moved. Linked lists have the disadvantage of using more memory and being a little more challenging to implement because each element in a linked list also stores a reference to the one that comes after it.

1. In real-world applications, what are some practical use cases where queues are preferred over stacks?  
     
   In many real-world scenarios where items must be handled in the order they arrive, queues are utilized. For instance, the first file sent to the printer is printed first when printing documents. They are also utilized in computer networks for customer support systems, data packet control, and task scheduling. Programming algorithms like Breadth-First Search (BFS), where processing items in a particular order is crucial, also frequently use queues. That is, when fairness and order are important, lines are preferred.

# Results

Present the visualized procedures done. Also present the results with corresponding data visualizations such as graphs, charts, tables, or image . Please provide insights, commentaries, or explanations regarding the data. If an explanation requires the support of literature such as academic journals, books, magazines, reports, or web articles please cite and reference them using the IEEE format.

Please take note of the styles on the style ribbon as these would serve as the style format of this laboratory report. The body style is Times New Roman size 12, line spacing: 1.5. Body text should be in Justified alignment, while captions should be center-aligned. Images should be readable and include captions. Please refer to the sample below:

A black screen with white text

AI-generated content may be incorrect.\

Figure 1 Screenshot of program

The output shows the result of the Python program that implements basic queue operations. The program starts by creating an empty queue and then adds the elements **1, 2, 3, 4,** and **5** one after another. Each time an element is added, the program displays a message confirming the action. After all the elements are enqueued, the final print statement shows the complete list of elements in the queue: **['1', '2', '3', '4', '5']**. This confirms that the program follows the First In, First Out (FIFO) rule, where new elements are added to the back of the queue in the same order they were inserted.

# Conclusion

A Python script makes it possible to witness the essential functioning of a queue and the way it accepts inputs. The application processes the inputs sequentially one at a time in a void queue and for each input returned, a confirmation message is sent. Finally, the application displays the complete queue which holds the strings "1, 2," "3, 4," and "5.". This way, the queue follows the FIFO principle as it can be returned to the input state all at once. The queue, so to speak, discloses and affirms that the last input was the last one to come out and the first input was the first one to go out.

**References**

Lafore, R. (2017). *Data structures and algorithms in Python*. Sams Publishing.

Python Software Foundation. (2024). *Python documentation: Data structures*. Retrieved from <https://docs.python.org/3/tutorial/datastructures.html>

If your school prefers that you only cite **online references**, here’s a version using just the Python official site:

**Python Software Foundation. (2024).** *Python documentation: Data structures.* Retrieved from <https://docs.python.org/3/tutorial/datastructures.html>