Data Structure and Algorithm

Laboratory Activity No. 9

Queues

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Nerio, Hannah Grace A. | 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

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 1 Screenshot of program

The screenshot above shows the implementation and output of the queue program. It demonstrates how elements are added using the enqueue operation and removed using the dequeue operation following the First In, First Out (FIFO) rule. The program also checks if the queue is empty before removing any element to avoid errors. In the output, you can see that the first elements added are the first ones removed, showing the proper behavior of a queue. This confirms that the program works correctly and follows the FIFO principle.

**1. What is the main difference between the stack and queue implementations in terms of element removal?**

\* The main difference is how elements are removed. In a stack, the last element that was added is the first one to be removed this is called Last In, First Out (LIFO). In a queue, it’s the opposite the first element that was added is the first one to be removed, which is First In, First Out (FIFO). So basically, stacks remove from the top, while queues 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?**

\* If we try to dequeue from an empty queue, there would be nothing to remove, which could cause an error. In the code, this kind of situation is handled using a check like is empty (). If the queue is empty, it returns a message such as “The queue is empty” instead of continuing. This prevents the program from crashing and makes it safer.

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

\* If elements are added at the beginning instead of the end, the queue would start working like a stack. That’s because the newest element would be removed first instead of the oldest one. In short, it would no longer follow the FIFO rule but would behave like a LIFO structure.

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

\* A linked list queue can grow or shrink as needed since it doesn’t have a fixed size, so there’s no overflow problem. But it uses more memory because each element needs extra space for pointers. An array queue, on the other hand, is simpler and faster for accessing elements but has a fixed size, which can cause issues if it gets full.

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

\* Queues are useful when things need to be done in the order they arrive. Examples include printer tasks, CPU scheduling, customer service lines, and network data handling. They make sure the first request is handled first, which is fair and organized. Stacks are good for tasks like undo actions, but queues are better when order really matters.

# Conclusion

In this laboratory activity, I focused on understanding the concept and implementation of the queue data structure. A queue follows the First In, First Out (FIFO) principle, meaning the first element added is also the first one removed. Through the activity, I learned how enqueue and dequeue operations work and how important it is to check for empty queues to prevent errors. I also gained a better understanding of how the queue differs from the stack, which uses the Last In, First Out (LIFO) principle. By comparing the two, I saw how each structure is useful in different cases the queue is best for managing tasks in order, while the stack is ideal for reversing processes or tracking function calls. This activity also helped me see how queues are used in real-life situations such as printer task management, CPU scheduling, and customer service systems. Overall, the activity deepened my understanding of how queues manage data efficiently, maintain proper order in processing tasks, and play an important role in many computer operations.

**References**

[1] Co Arthur O.. “University of Caloocan City Computer Engineering Department Honor Code,” UCC-CpE Departmental Policies, 2020.

[2] Google Colab. “Google Colab,” google.com. [Online]. Available: <https://colab.research.google.com/drive/1bXVPsuqhMwKEYVoxbdJIuveDVl7gHLj->. [Accessed: Oct. 11, 2025].