UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Maringal, Czer Justine D.

Instructor:
Engr. Maria Rizette H. Sayo

October 11, 2025

I. 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

II. 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?

III. Results

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

a stack is like a pile of plates where you always take the top one off last (LIFO), while a queue is like a line at a cafeteria where the first person in line gets served first (FIFO). In code terms, when we remove from a stack, we take from the end of the list, but when we remove from a queue, we take from the beginning of the list.

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, the program would crash with an error. In code, I prevent this by first checking if the queue is empty using our is_empty function. If it is empty, we return "The queue is empty" instead of letting the program crash.

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

The last person to arrive would always be the first one served, which means our queue would actually behave like a stack (LIFO). The "first-in, first-out" rule would be broken, and we'd lose the main purpose of using a queue.

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

Using a linked list for a queue is like having people stand wherever there's space, with each person remembering who is behind them. The big advantage is that adding and removing people is very efficient. The disadvantage is that it uses more memory because we need to store those "who's behind me" pointers.

Using an array for a queue is like having people stand in a fixed line. The advantage is that it's simple and uses less memory. The disadvantage is that when the first person leaves, everyone else has to shuffle forward, which can be slow for long lines.

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

Printer queues where documents print in the order they were sent.

Customer service call centers where calls are answered in the order received.

Food delivery apps that assign drivers to orders based on who ordered first.

Video streaming where data packets need to arrive in the correct sequence.

IV. Conclusion

This laboratory demonstrated the queue data structure FIFO (First-In-First-Out). Through practical implementation, we confirmed that queues process elements in the exact order they arrive, unlike stacks which process the most recent additions first. Queues prove processing order matters, such as task scheduling, customer service systems, and data streaming.

References

[1] Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). Data Structures and Algorithms in Python. Wiley.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.

[3] Python Software Foundation. (2023). Python Documentation: Data Structures. https://docs.python.org/3/tutorial/datastructures.html

[4]Miller, B. N., & Ranum, D. L. (2011). Problem Solving with Algorithms and Data Structures Using Python. Franklin, Beedle & Associates.