UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 9

Queues

Submitted by:
Dispo, Lei Andrew T.

Instructor:
Engr. Maria Rizette H. Sayo

10, 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 The main difference between a stack and a queue in terms of element removal is that a stack removes elements using the **Last-In, First-Out (LIFO)** method, where the last element added is the first to be removed, while a queue removes elements using the **First-In, First-Out (FIFO)** method, where the first element added is the first to be removed.

2 this is usually handled by checking if the queue is empty before performing a dequeue operation using a condition like:

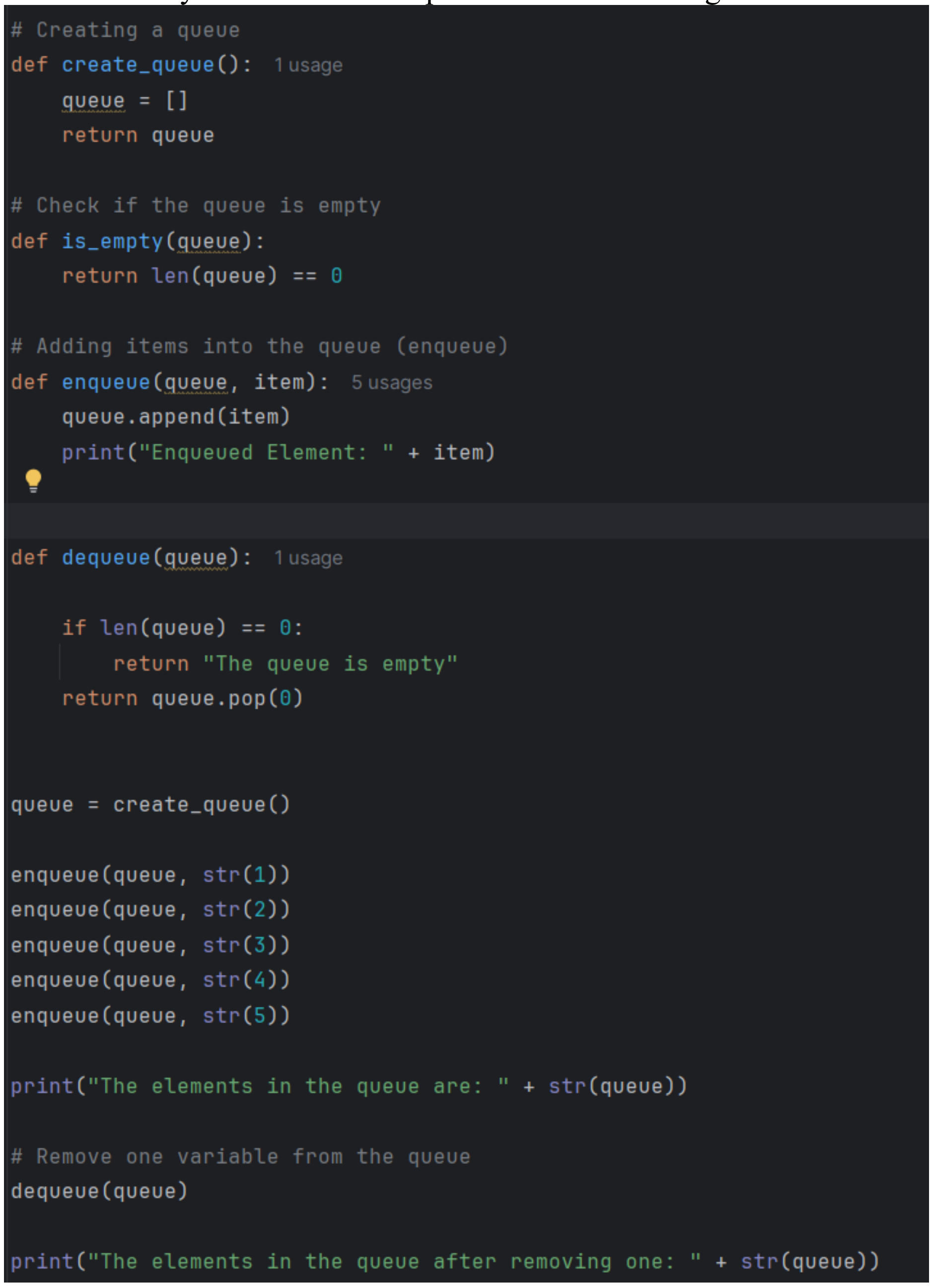
```
if len(queue) == 0:
    print("The queue is empty")
```

3 The queue would begin to behave like a stack if the enqueue operation was changed to insert entries at the beginning instead of the end.
This is because, rather of using the FIFO rule, the most recently added piece would now

be eliminated first, according to the LIFO (Last-In, First-Out) rule. The basic behavior of a queue would be eliminated if the idea of order were flipped.

4 Implementing a queue using a linked list has the advantage of allowing dynamic memory allocation and efficient insertion and deletion operations, but it requires more memory because of additional pointers. In contrast, using an array makes accessing elements faster and uses less memory, but it has a fixed size and may require shifting elements when removing items, which can make operations less efficient.

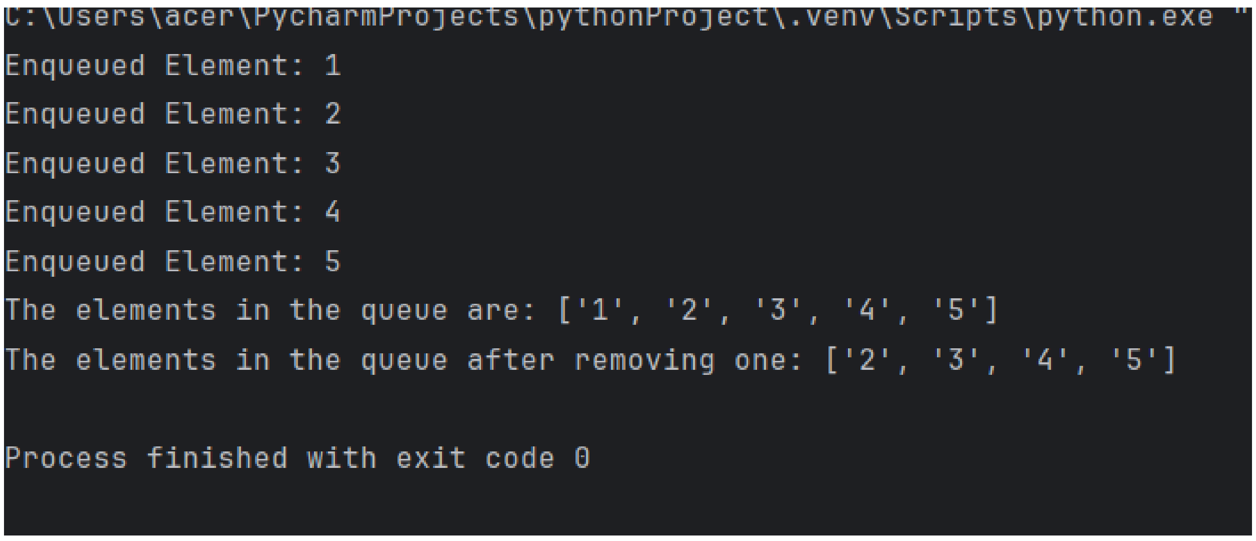
5 -Printer Spooling: Documents are printed in the order they are sent
-CPU Task Scheduling: Processes are executed based on their arrival order.
-Customer Service Systems: The first customer to call is the first to be served.
-Networking (Data Packets): Packets are transmitted in the same order they are received to maintain data integrity.
-Simulation Systems: Events are processed in chronological order.



```python
# Creating a queue
def create_queue():  1 usage
    queue = []
    return queue

# Check if the queue is empty
def is_empty(queue):
    return len(queue) == 0

# Adding items into the queue (enqueue)
def enqueue(queue, item):  5 usages
    queue.append(item)
    print("Enqueued Element: " + item)


def dequeue(queue):  1 usage

    if len(queue) == 0:
        return "The queue is empty"
    return queue.pop(0)


queue = create_queue()

enqueue(queue, str(1))
enqueue(queue, str(2))
enqueue(queue, str(3))
enqueue(queue, str(4))
enqueue(queue, str(5))

print("The elements in the queue are: " + str(queue))

# Remove one variable from the queue
dequeue(queue)

print("The elements in the queue after removing one: " + str(queue))
```

Figure 1

Full picture of the code



```
C:\Users\acer\PycharmProjects\pythonProject\.venv\Scripts\python.exe "
Enqueued Element: 1
Enqueued Element: 2
Enqueued Element: 3
Enqueued Element: 4
Enqueued Element: 5
The elements in the queue are: ['1', '2', '3', '4', '5']
The elements in the queue after removing one: ['2', '3', '4', '5']

Process finished with exit code 0
```

Figure 2

Output of the code

IV. Conclusion

In this laboratory activity, we learned that queues work on the First-In, First-Out (FIFO) principle. We found that stacks use LIFO while queues use FIFO, empty queues must be checked before removing elements, changing the enqueue position can make a queue act like a stack, arrays and linked lists both have pros and cons in implementing queues, and queues are useful in real-world cases like task scheduling and printer spooling.

References

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