Seatwork 5.1

Queue - Linked List Application

Course Code: CPE010	**Program:** BSCPE
Course Title: Object Oriented Programming	**Date Performed:** September 9, 2025
Section: CPE21S4	**Date Submitted:** September 9, 2025
Name: Bautista,Mariela S.	**Instructor:** Engr. Quejado

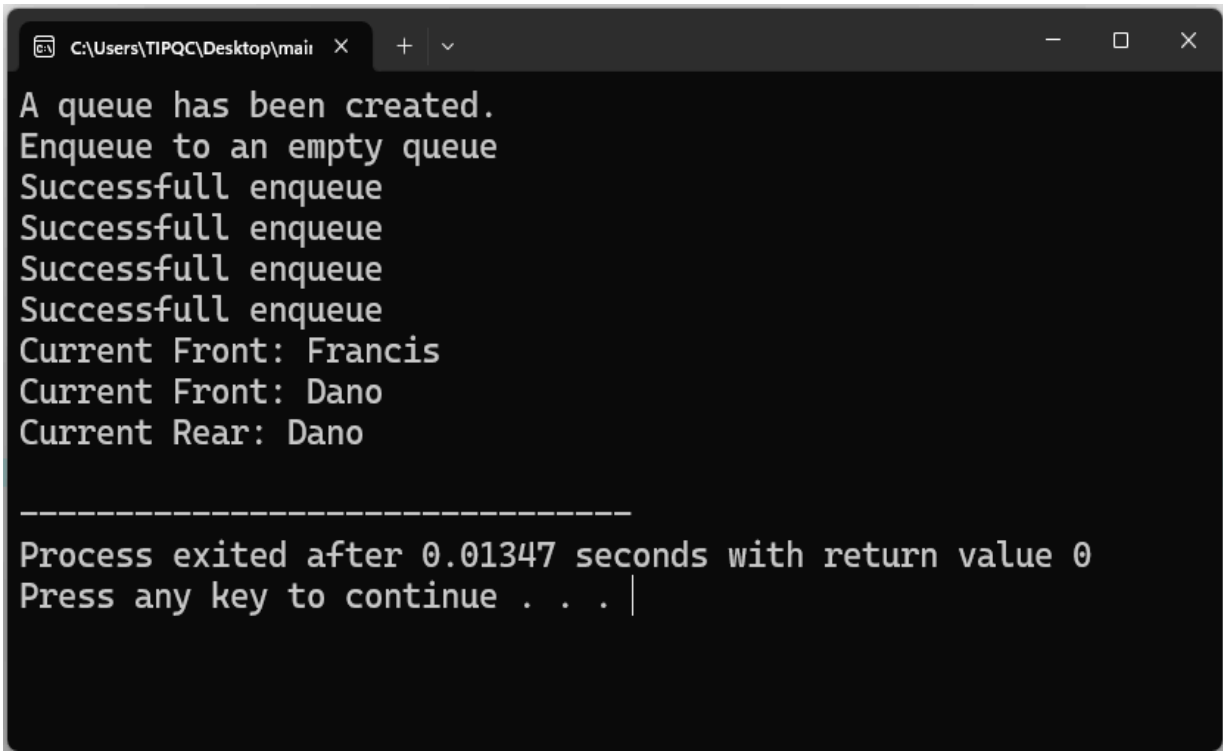
Objective(s):

Documentation requirements:

- Ensure that the code and outputs are documented in the **output** section of the activity template
- Create an in-depth discussion for **each linked list operations** that the class created during the class activity. Put your answers in the supplementary activity
- Submit pdf file, main.cpp and the header file

Use the activity template for the activity.

Output:



Supplementary Activity:

main.q.cpp [*] queue.h

```cpp
#ifndef QUEUE_H
#define QUEUE_H
#include <iostream>

template<typename T>
class Node{
    public:
        T data;
        Node* next;

        Node(T new_data){
            data = new_data;
            next = nullptr;

        }
};

template<typename T>
class Queue{
    private:
        Node<T> *front;
        Node<T> *rear;

    public:
        //creates an empty queue
        Queue(){
            front = rear = nullptr;
            std::cout << "A queue has been created. \n";
        }

        //isEmpty
        bool isEmpty(){
            return front == nullptr;
        }

        //enqueue
        void enqueue(T new_data){
            Node<T> *new_node = new Node<T>(new_data);


            if(isEmpty()){
                front = rear = new_node;
                std::cout << "Enqueue to an empty queue " << std::endl;
                return;
            }
            rear-> next = new_node;
            rear = new_node;
            std::cout <<"Successfull enqueue " <<std::endl;
```

main.q.cpp [*] queue.h

```
49          }
50
51          //dequeue
52          void dequeue(){
53              //checking if queue is empty
54              if (isEmpty()){
55                  return;
56              }
57              //storing the front to a temporary pointer
58              Node<T> *temp = front;
59
60              //check if after the dequeue, the queue is empty
61              if (front == nullptr) rear == nullptr;
62
63              else{
64              //reassign the front to the next node
65                  front = front-> next;
66              }
67              delete temp;
68          }
69
70          //getFront
71          void getFront(){
72              if (isEmpty()){
73                  std::cout << "The queue is Empty " <<std::endl;
74                  return;
75              }
76              std::cout << "Current Front: " << front->data <<std::endl;
77          }
78
79          //getRear
80          void getRear(){
81              if (isEmpty()){
82                  std::cout << "The queue is empty. \n";
83                  return;
84              }
85              std::cout << "Current Rear: " << rear -> data << std::endl;
86          }
87
88          //display
89          void display(){
90              if (isEmpty()){
91                  std::cout << "The queue is empty. \n";
92                  return;
93              }
94              Node<T> *temp = front;
95              while (temp !=nullptr){
96                  std::cout << temp -> data << " ";
```

```
97                  temp = temp -> next;
98              }
99              std::cout <<std::endl;
100         }
101
102 };
103
104 #endif
```

Conclusion:

What i've learned in this program is we create a simple queue. First of; Node: we created a function and then made address it as classNode. After that we declare a public inside then named the as T data, made the pointer or *Node next and made the Node and declare the data as new_data, data is equal to new_data; next = nullptr;.

We create a template and named it as typename T, declared a queue or class Queue. Inside is private and public (this creates an empty queue) functions.

isEmpty function just checks if its empty or not.

Enqueue function is to push a data to an empty queue. Which making the empty queue to create a value inside making the rear to become the next new node, which then the output is then successfullyenqueued.

Dequeue function, we create and then check if the queue is empty if it is then return. We created a node setting the pointer T as the front which is just storing the front to a temporary pointer. We then check if after the dequeue, the queue is empty which if front is equal to null pointer and rear is equal to null pointer. Or else we reassign the front to the next node.

GetFront is just checking the the queue is empty then printing what is the current front.

GetRear is checking also if its empty and then printing what is the current front.

Then lastly we display if the queue is empty and then we return and or while, we check the nullpointer as the temp.