Seatwork 5.1	
Linked List Application	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data structures and Algorithm	**Date Performed: 9/9/25**
Section: CPE21S4	**Date Submitted: 9/9/25**
Name(s): Francis Nikko I. Andoy	**Instructor: Engr. Jimlord Quejado**
Output	

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

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

Analysis:
In this we created a class for the Node with template T since we are gonna create a 2nd class we set the modifier to public so the other class can access it. Then we defined 2 data which are data, and next, this is important since we need this to create the literal Node later. Creating a constructor that has a new_data parameter which means that we are assigning the data to the new_data and the next is set to null. we are initializing the Node that we're creating.

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

Analysis:
Then we are creating a class for the literal Queue, then defining the front and rear. We will use Node<T> because we don't have an specific type like string or int.

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

Analysis:
In here we are creating a constructor to print the start of the queue.

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

Analysis:
In this function this will just return a 0 or 1 indicating the True or false.

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

    if(isEmpty()){
        front = rear = new_node;
        std::cout<<"The queue is Empty.\n";
        return;
    }
    rear->next = new_node;
    rear = new_node;
    std::cout<<"May queue.\n";
}
```

Analysis:
In this function we have a parameter called new_data the one we declared in the Node class. Since we don't have a specific data type we are gonna use the <T> then set the new_node to new Node<T> to allocate our queue. Using if statement with the isEmpty function, since if we have an empty Node the front and rear are just set to equal. indicating that the Node is empty.

```
//dequeue
void dequeue(){
    if(isEmpty()){
        return;
    }
    //storing the front to a temporary pointer
    Node<T> *temp = front;

    //check if the after dequeue. the queue is empty
    if(front == nullptr){
        rear = nullptr;
    }

    else{
        front = front->next;
    }
    delete temp;
    std::cout<<"Dequeue.\n";
}
```

Analysis:
In this function we first check if the queue is empty. We create a temporary variable to store the data that we are planning to delete. The one we are deleting is the front, so if the front is equal to nullptr we are gonna set the nullptr since that is how queue works.

```
//getFront
void getFront(){
    if(isEmpty()){
        std::cout<<"The queue is empty.\n";
        return;
    }
    std::cout<<"Front: "<< front->data<<std::endl;
}
```

Analysis:
in this function we are getting the front, firstly we check if the queue is empty, if it is empty then we print the print then return to our function if not then we are gonna print the front in our queue.

```
//getRear
void getRear(){
    if(isEmpty()){
        std::cout<<"The queue is empty.\n";
        return;
    }
    std::cout<<"Rear: " << rear->data<<std::endl;
}
```

Analysis:
In this function we are gonna create the rear or the tail of the queue. this is just like the front.

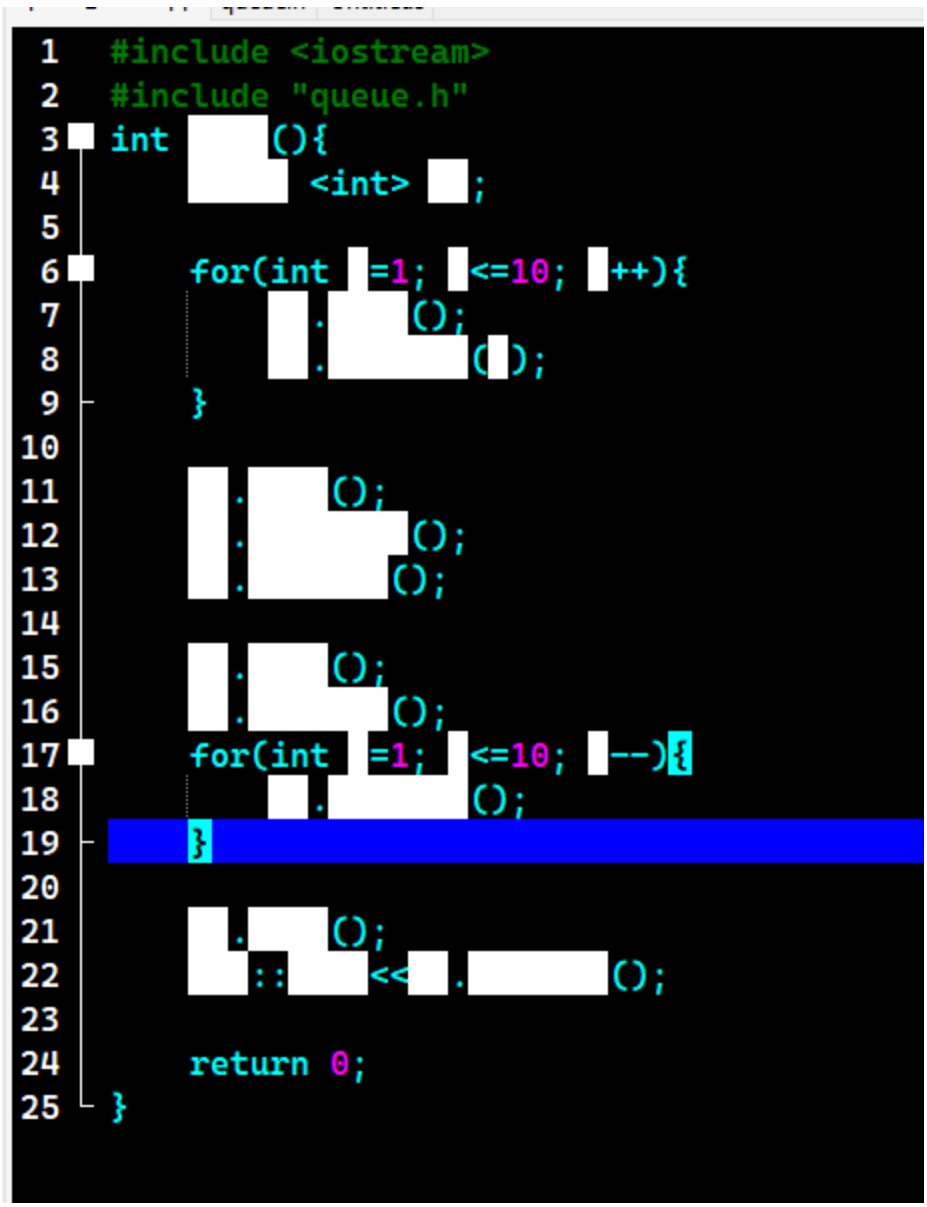
```
//display
void display(){
    if(isEmpty()){
        std::cout<<"The queue is empty.\n";
        return;
    }
    Node<T> *temp = front;
    while(temp != nullptr){
        std::cout<< temp->data <<std::endl;
        //update the loop
        temp = temp->next;
    }
}
```

Analysis:
In here we are displaying all the data inside our queue. We defined the temp to store the front of our data and if it is not equal to pointer it will keep on printing the data.

```
    ~        (){
        while(!         ()){
            dequeue();
        }
    }
};
```

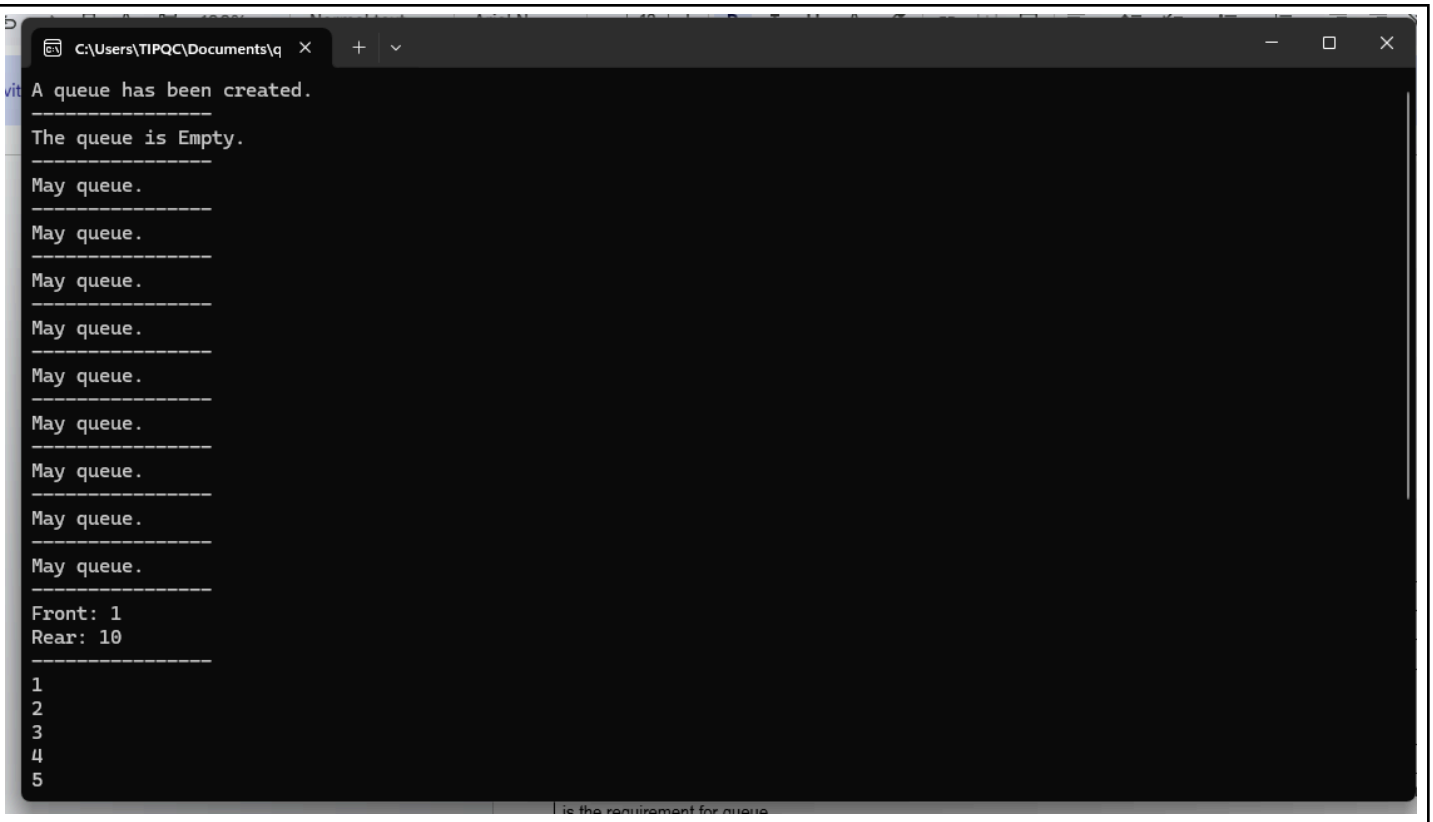
Analysis:
this is to delete our data. deconstructor.

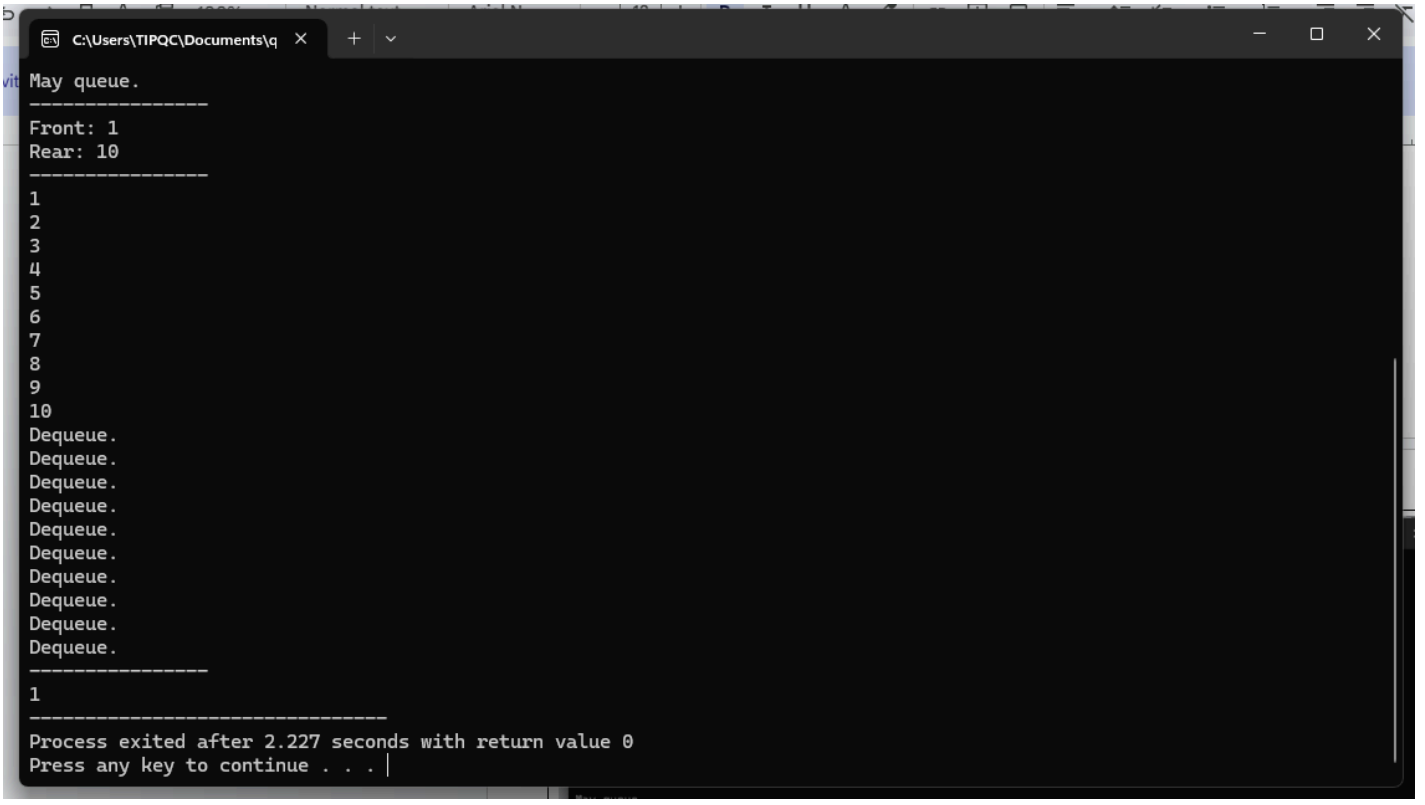
MAIN:



```
#include <iostream>
#include "queue.h"
int [redacted](){
    [redacted]<int> [redacted];

    for(int [redacted]=1; [redacted]<=10; [redacted]++){
        [redacted].[redacted]();
        [redacted].[redacted]([redacted]);
    }

    [redacted].[redacted]();
    [redacted].[redacted]();
    [redacted].[redacted]();

    [redacted].[redacted]();
    [redacted].[redacted]();
    for(int [redacted]=1; [redacted]<=10; [redacted]--){
        [redacted].[redacted]();
    }

    [redacted].[redacted]();
    [redacted]::[redacted]<<[redacted].[redacted]();

    return 0;
}
```

OUTPUT:


C:\Users\TIPQC\Documents\q
A queue has been created.
The queue is Empty.
May queue.
May queue.
May queue.
May queue.
May queue.
May queue.
May queue.
May queue.
May queue.
Front: 1
Rear: 10
1
2
3
4
5


C:\Users\TIPQC\Documents\q
May queue.
Front: 1
Rear: 10
1
2
3
4
5
6
7
8
9
10
Dequeue.
Dequeue.
Dequeue.
Dequeue.
Dequeue.
Dequeue.
Dequeue.
Dequeue.
Dequeue.
Dequeue.
1
Process exited after 2.227 seconds with return value 0
Press any key to continue . . .

Conclusion :
We learned how the queue works, how pointer and templating works. We used a lot of functions for our class, since that is the requirement for the queue. We also used a temporary variable to store a data inside of our queue, then we are gonna focus on that temp variable to do the operations needed.