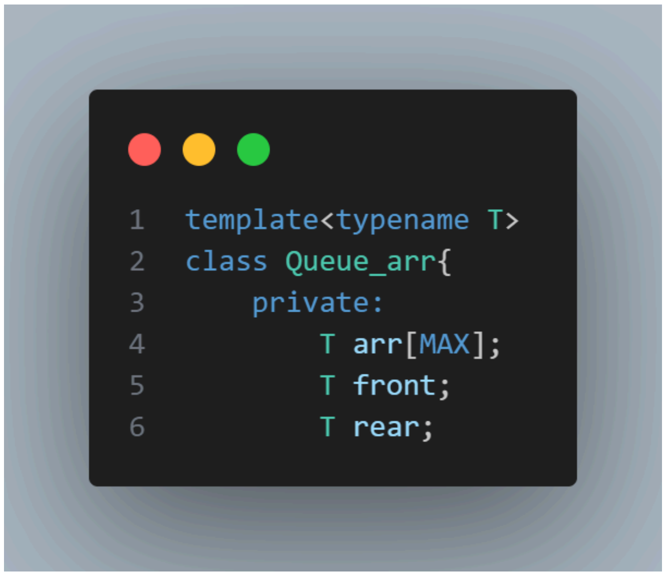
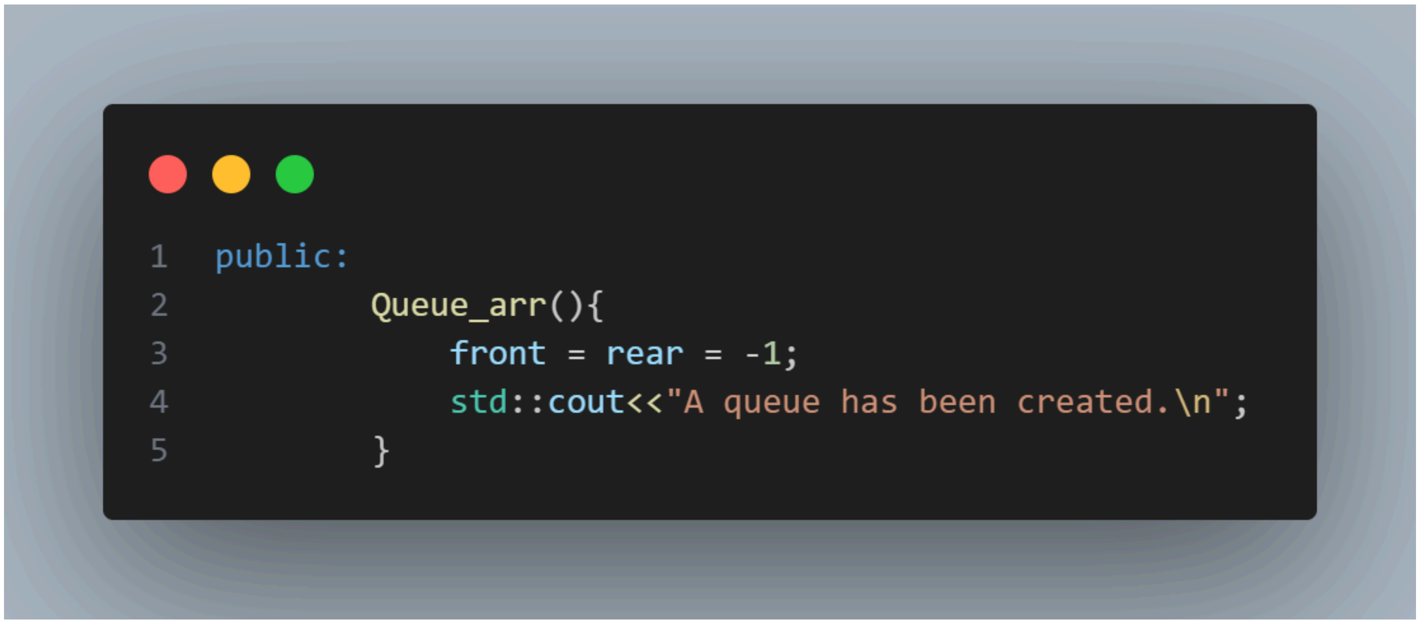
Activity 5.1	
Hands-on-Activity	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 11/09/25**
Section: CPE21S4	**Date Submitted: 11/09/25**
Name(s): Francis Nikko I. Andoy	**Instructor: Jimlord Quejado**

6. Output



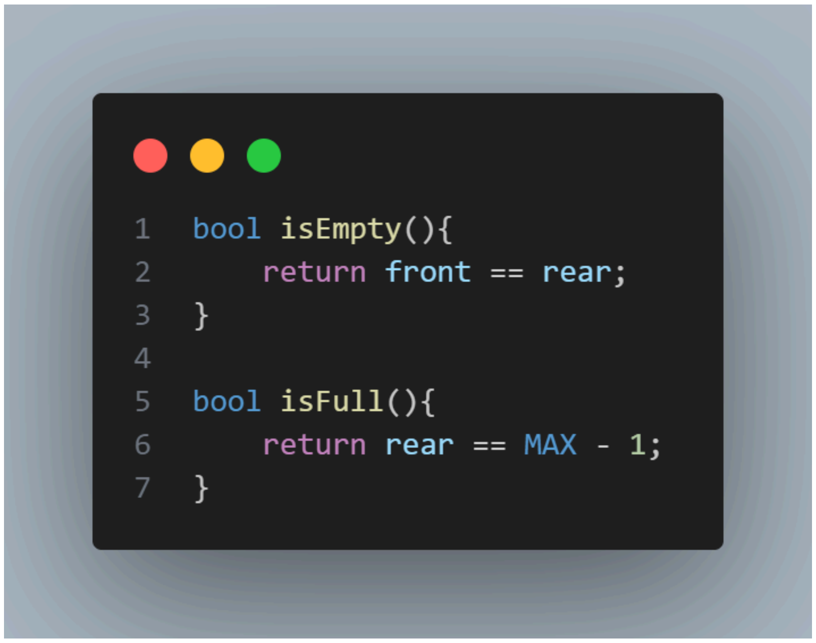
```
template<typename T>
class Queue_arr{
    private:
        T arr[MAX];
        T front;
        T rear;
```

Analysis:
In here I've created a class and set it to private, since I don't have another class that will be used later. I also defined 3 variables.



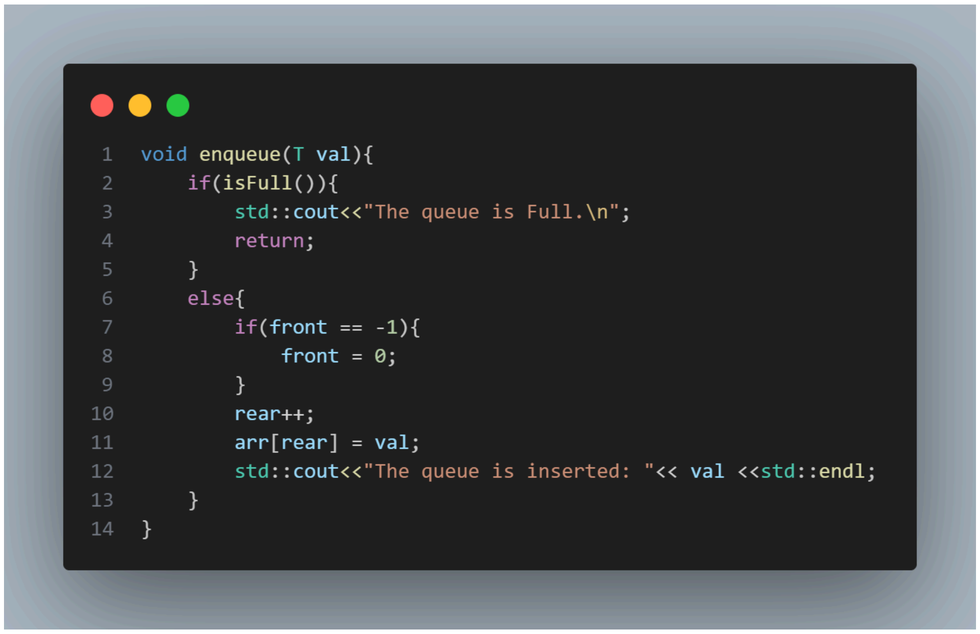
```
public:
        Queue_arr(){
            front = rear = -1;
            std::cout<<"A queue has been created.\n";
        }
```

Analysis:
In here, I've created a constructor that will let us know if we've created a queue.



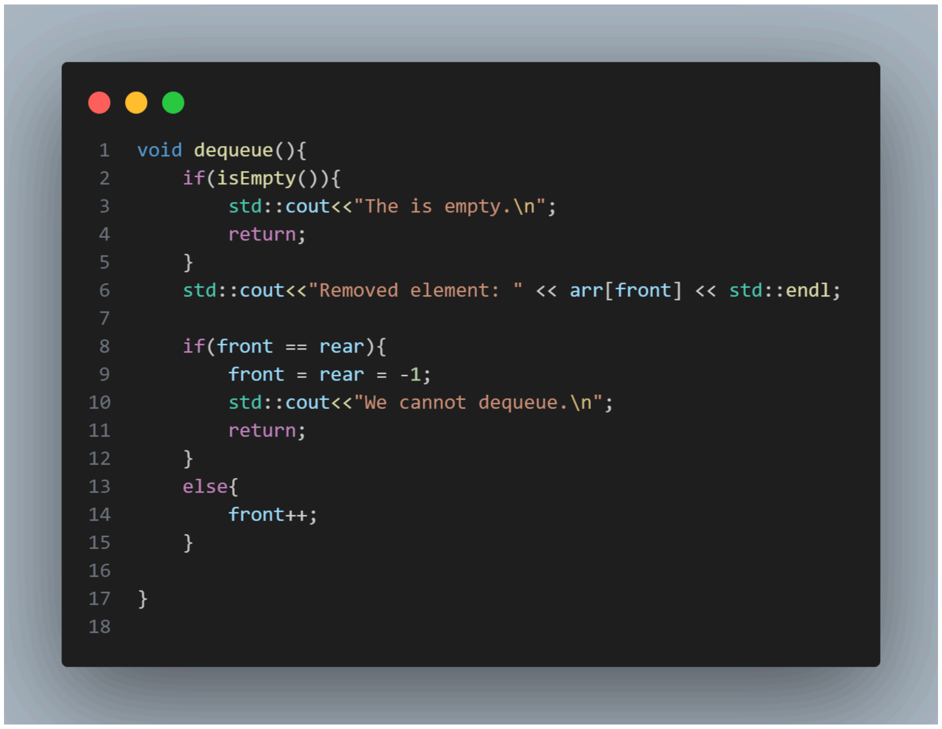
```cpp
bool isEmpty(){
    return front == rear;
}

bool isFull(){
    return rear == MAX - 1;
}
```

Analysis:
Here, I've created 2 functions. This will let us know if the queue is empty or full, this will return 1 or 0 indicating true or false.



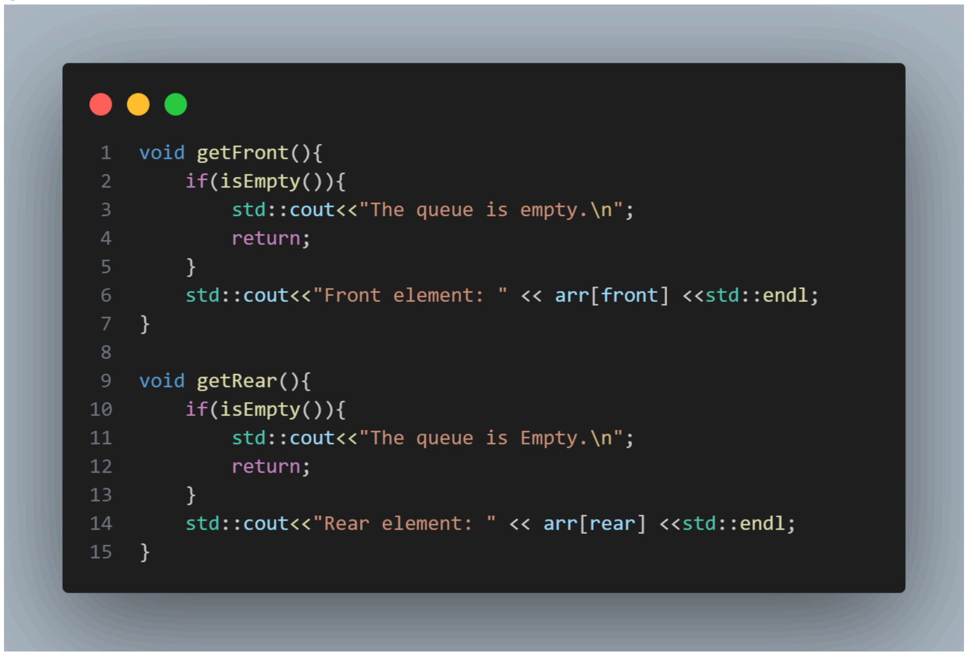
```cpp
void enqueue(T val){
    if(isFull()){
        std::cout<<"The queue is Full.\n";
        return;
    }
    else{
        if(front == -1){
            front = 0;
        }
        rear++;
        arr[rear] = val;
        std::cout<<"The queue is inserted: "<< val <<std::endl;
    }
}
```

Analysis:
Here, since this is the function that will add data to our element. Firstly, we checked if the queue is full, since if the queue is really full, there will be an error. This is a static array so we can't just add data whenever we want. We have to do a dynamic operation to add another data to our static array. Then if the array is not full the if operation will proceed, the condition for our if statement is, if the front is outside of the array then the front will be set to 0 the 0 is the first element in our array. Outside of that is the operation for the rear since we moved that front, we now need to move the rear to move it. We will increment it, then place the value of the val to where our rear is placed. So we now applied the FIFO, where we will add on the rear side.



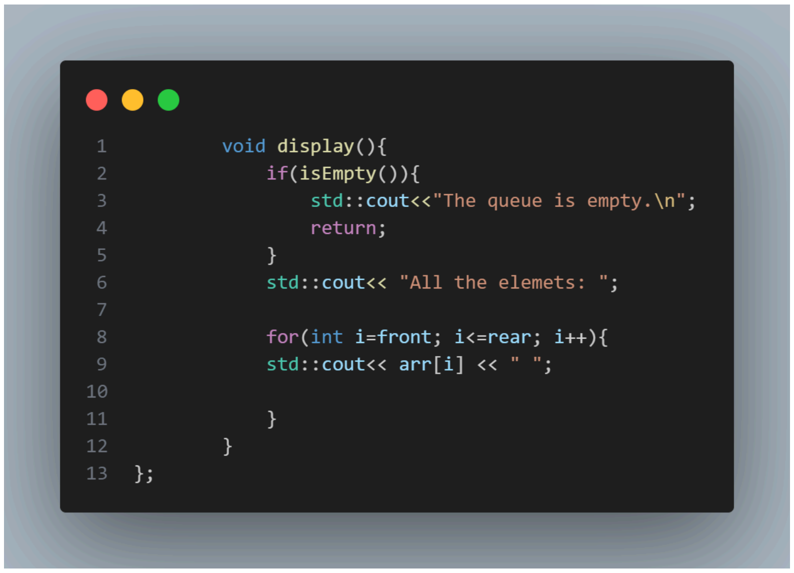
```cpp
void dequeue(){
    if(isEmpty()){
        std::cout<<"The is empty.\n";
        return;
    }
    std::cout<<"Removed element: " << arr[front] << std::endl;

    if(front == rear){
        front = rear = -1;
        std::cout<<"We cannot dequeue.\n";
        return;
    }
    else{
        front++;
    }

}

```

Analysis:
Here, we will check if the queue is empty since we can't do an dequeue operation if the queue is empty. Then print that value inside of the front, since that value will be deleted. Then we will also place a catch when the queue is full.



```cpp
void getFront(){
    if(isEmpty()){
        std::cout<<"The queue is empty.\n";
        return;
    }
    std::cout<<"Front element: " << arr[front] <<std::endl;
}

void getRear(){
    if(isEmpty()){
        std::cout<<"The queue is Empty.\n";
        return;
    }
    std::cout<<"Rear element: " << arr[rear] <<std::endl;
}
```

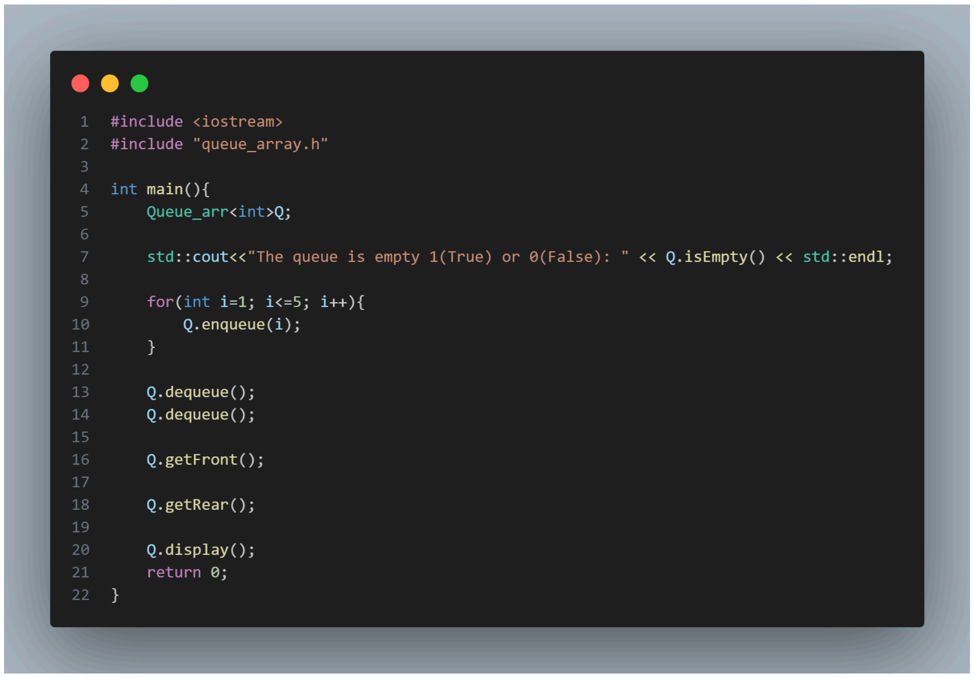
Analysis:
Here we will display the front and the rear by using the is empty function first. If the queue is not empty then we will get or print the front and rear element on our array.



```cpp
        void display(){
            if(isEmpty()){
                std::cout<<"The queue is empty.\n";
                return;
            }
            std::cout<< "All the elemets: ";

            for(int i=front; i<=rear; i++){
            std::cout<< arr[i] << " ";

            }
        }
};
```

Analysis:

Here, we will now display all the elements inside of our array. by using the for loop we will set the i to the front to our rear then increment the i when the condition i is still less than rear.

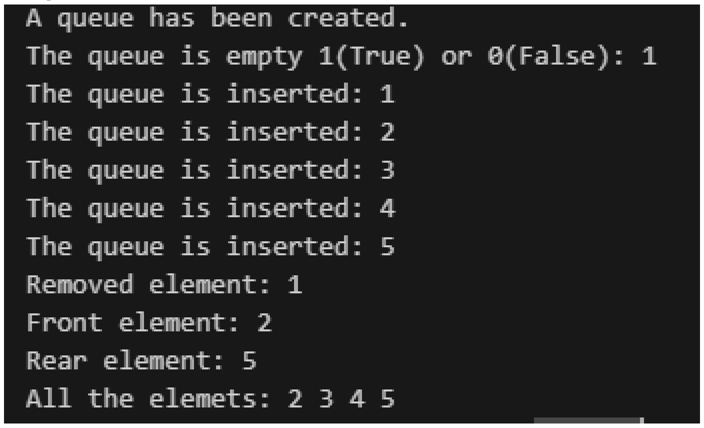


```cpp
#include <iostream>
#include "queue_array.h"

int main(){
    Queue_arr<int>Q;

    std::cout<<"The queue is empty 1(True) or 0(False): " << Q.isEmpty() << std::endl;

    for(int i=1; i<=5; i++){
        Q.enqueue(i);
    }

    Q.dequeue();
    Q.dequeue();

    Q.getFront();

    Q.getRear();

    Q.display();
    return 0;
}
```

Analysis:

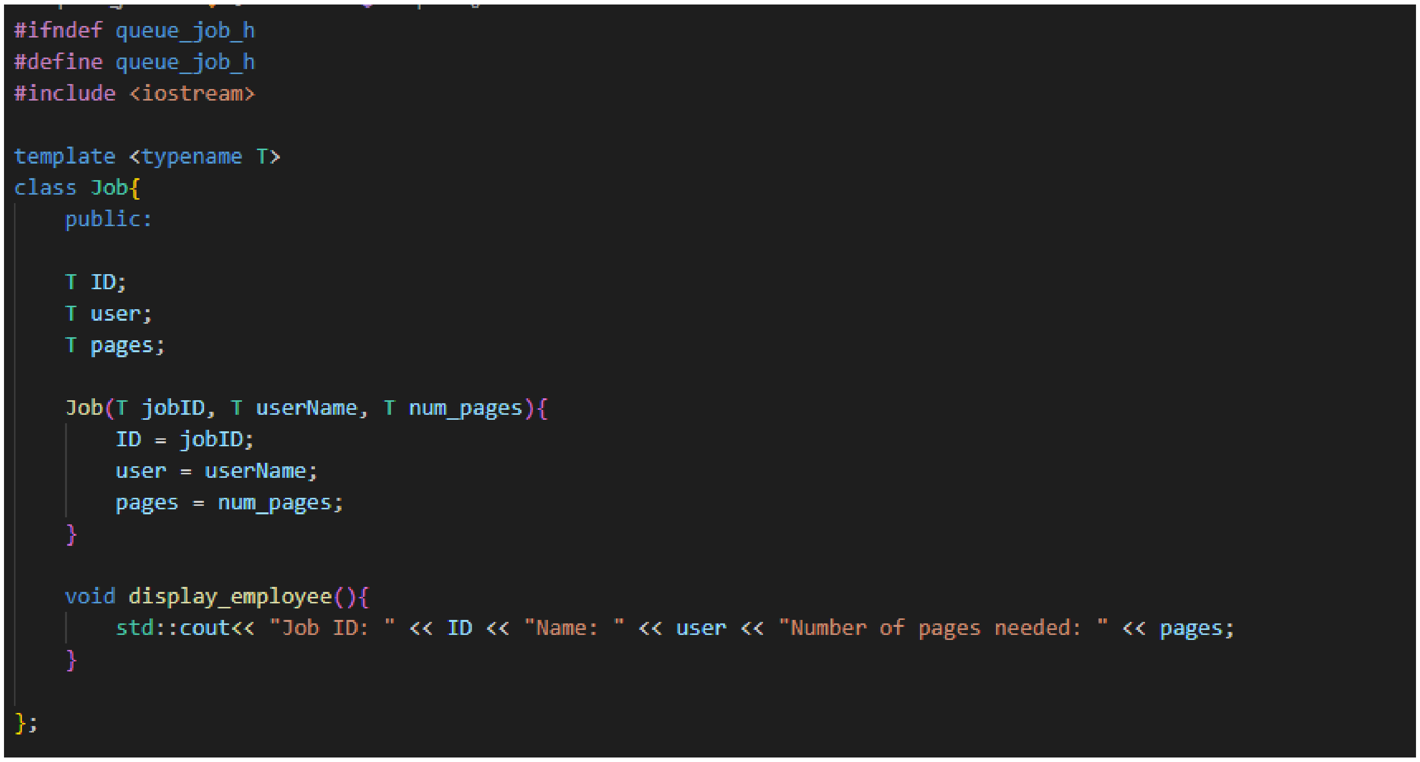
Now we will call the queue class, by calling this we now set it to int since we used a template in our header, we now need to specify what kind of data type we want in the main. Then just call the functions we created.

Output:

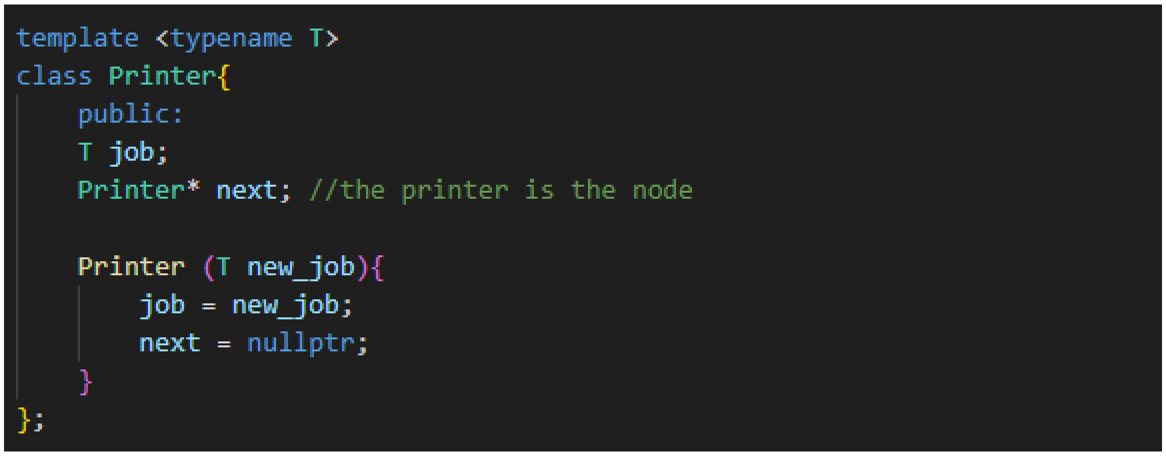


```
A queue has been created.
The queue is empty 1(True) or 0(False): 1
The queue is inserted: 1
The queue is inserted: 2
The queue is inserted: 3
The queue is inserted: 4
The queue is inserted: 5
Removed element: 1
Front element: 2
Rear element: 5
All the elemets: 2 3 4 5
```

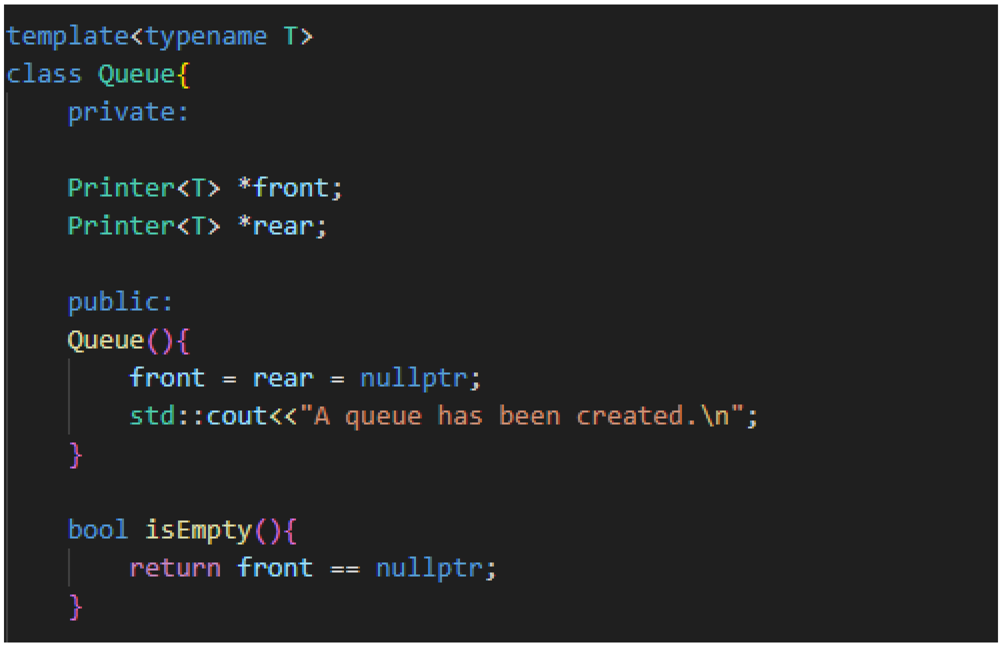
7. Supplementary Activity



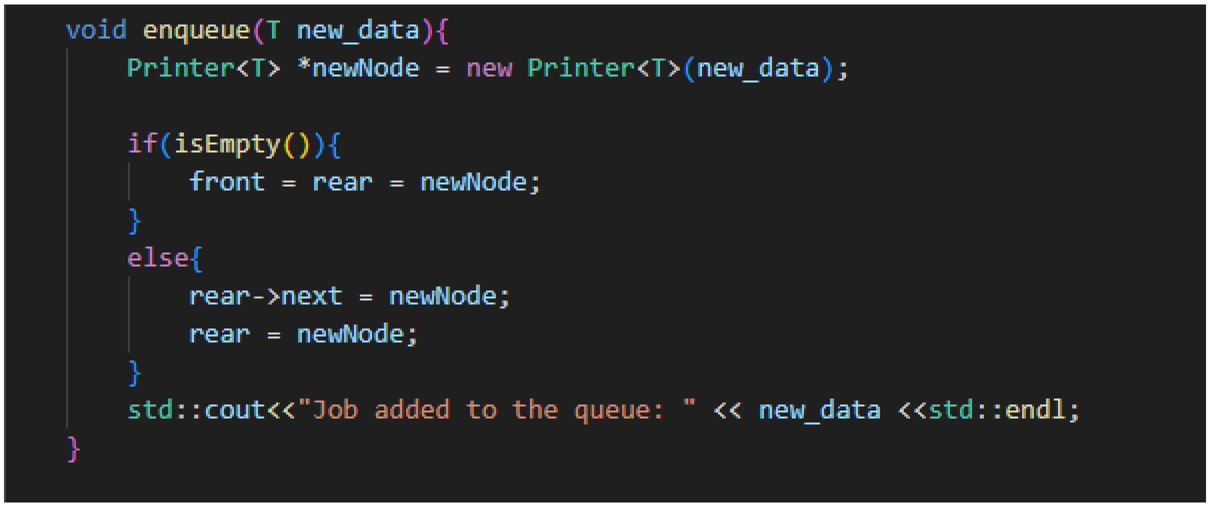
```cpp
#ifndef queue_job_h
#define queue_job_h
#include <iostream>

template <typename T>
class Job{
    public:

    T ID;
    T user;
    T pages;

    Job(T jobID, T userName, T num_pages){
        ID = jobID;
        user = userName;
        pages = num_pages;
    }

    void display_employee(){
        std::cout<< "Job ID: " << ID << "Name: " << user << "Number of pages needed: " << pages;
    }

};
```



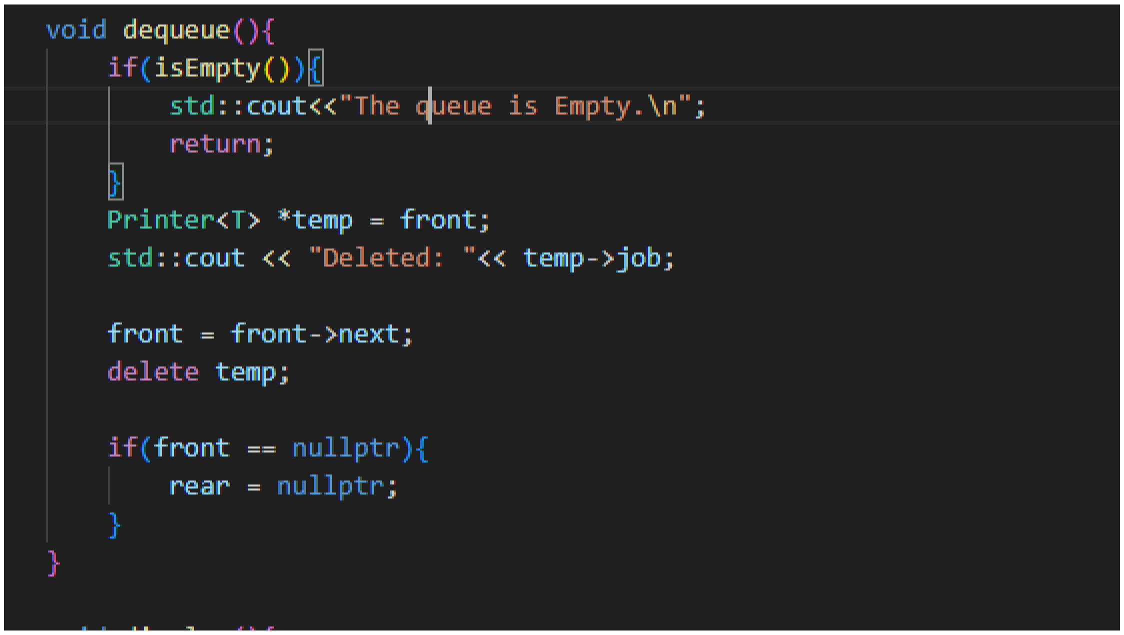
```cpp
template <typename T>
class Printer{
    public:
    T job;
    Printer* next; //the printer is the node

    Printer (T new_job){
        job = new_job;
        next = nullptr;
    }
};
```



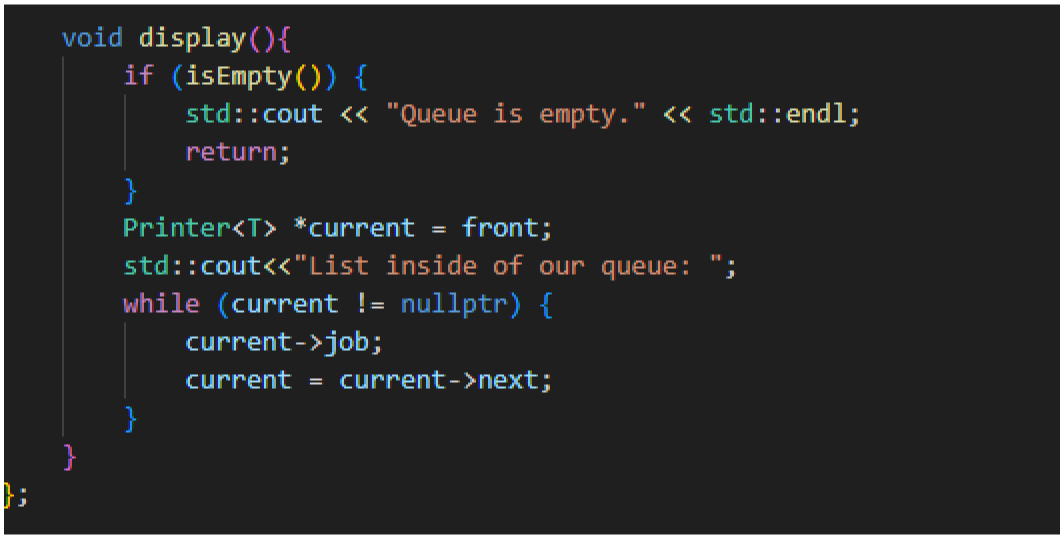
```cpp
template<typename T>
class Queue{
    private:

    Printer<T> *front;
    Printer<T> *rear;

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

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



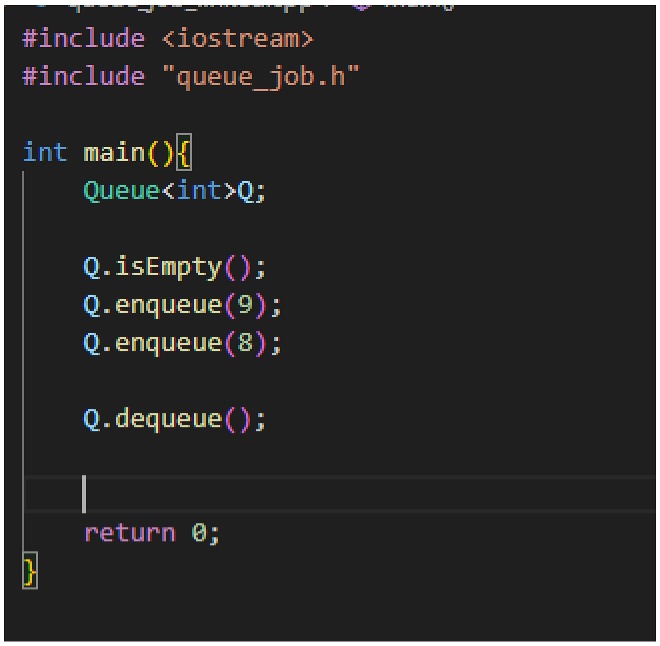
```cpp
    void enqueue(T new_data){
        Printer<T> *newNode = new Printer<T>(new_data);

        if(isEmpty()){
            front = rear = newNode;
        }
        else{
            rear->next = newNode;
            rear = newNode;
        }
        std::cout<<"Job added to the queue: " << new_data <<std::endl;
    }
```



```cpp
  void dequeue(){
      if(isEmpty()){
          std::cout<<"The queue is Empty.\n";
          return;
      }
      Printer<T> *temp = front;
      std::cout << "Deleted: "<< temp->job;

      front = front->next;
      delete temp;

      if(front == nullptr){
          rear = nullptr;
      }
  }
```



```
    void display(){
        if (isEmpty()) {
            std::cout << "Queue is empty." << std::endl;
            return;
        }
        Printer<T> *current = front;
        std::cout<<"List inside of our queue: ";
        while (current != nullptr) {
            current->job;
            current = current->next;
        }
    }
};
```



```
#include <iostream>
#include "queue_job.h"

int main(){
    Queue<int>Q;

    Q.isEmpty();
    Q.enqueue(9);
    Q.enqueue(8);

    Q.dequeue();


    return 0;
}
```

8. Conclusion

in conclusion, ive learned how to use queue we need to apply the FIFO if we want to use this data structure and the functions that we need for this are enqueue, dequeue, isFull, isEmpty, and size. Although the size is just for the array since we don't need the size if we use the linked list.

9. Assessment Rubric