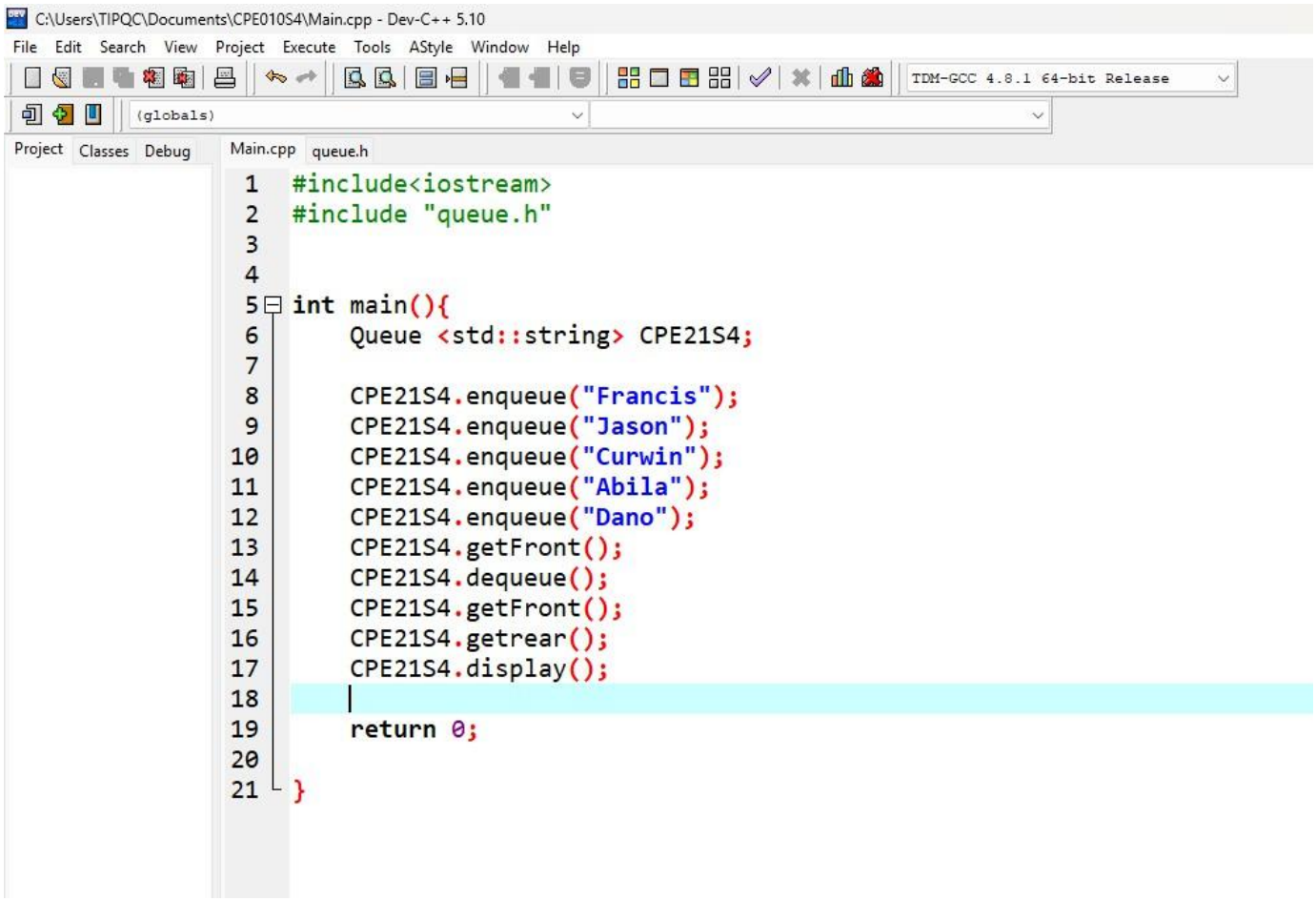
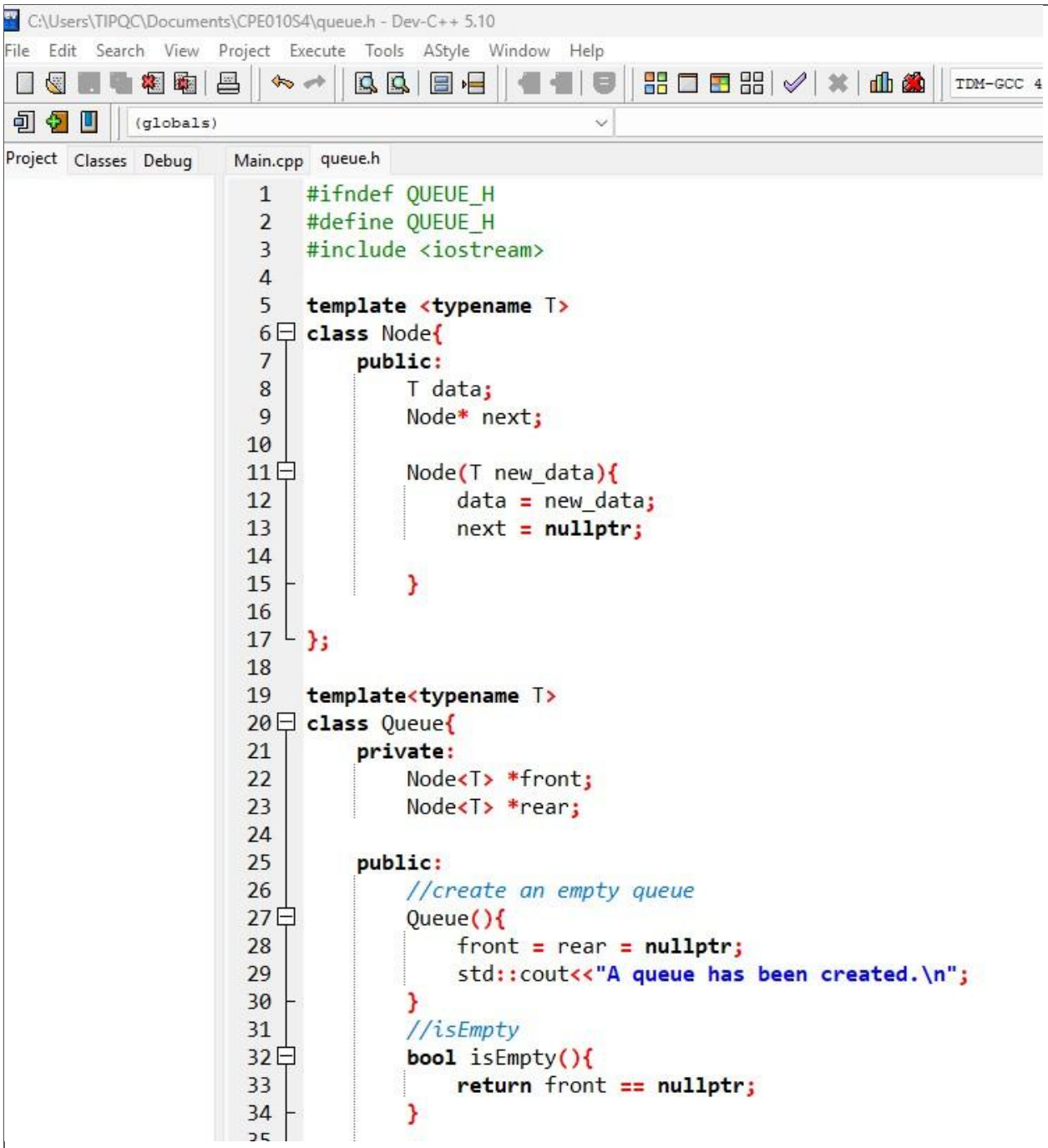
Activity No. <n>	
<Replace with Title>	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 9/9/25**
Section: CPE21S4	**Date Submitted: 9/9/25**
Name(s): QUIOYO, ANGELO M.	**Instructor: Engr. Jimlord Quejado**

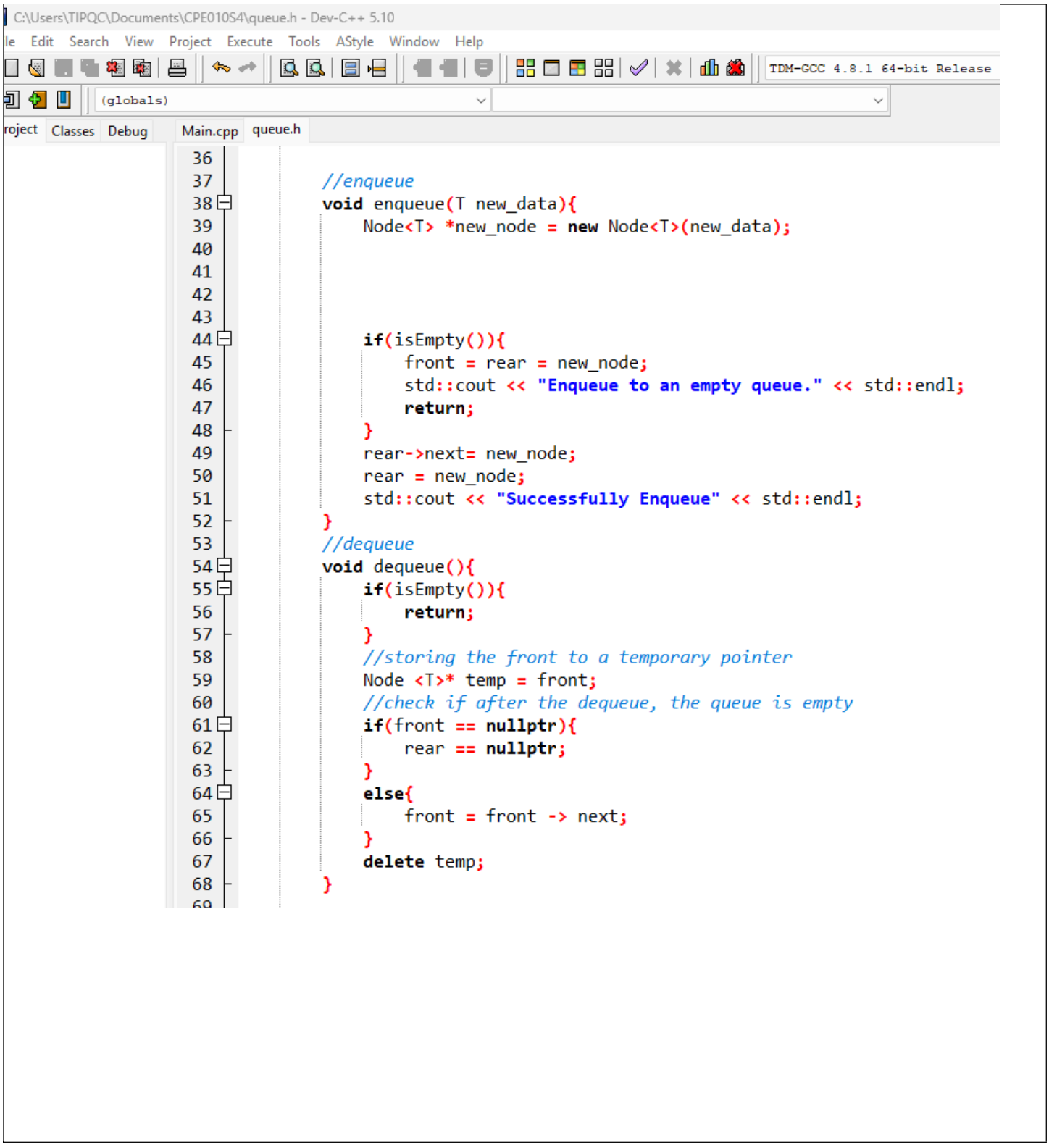
6. Output

Main Function:



Queue.h:

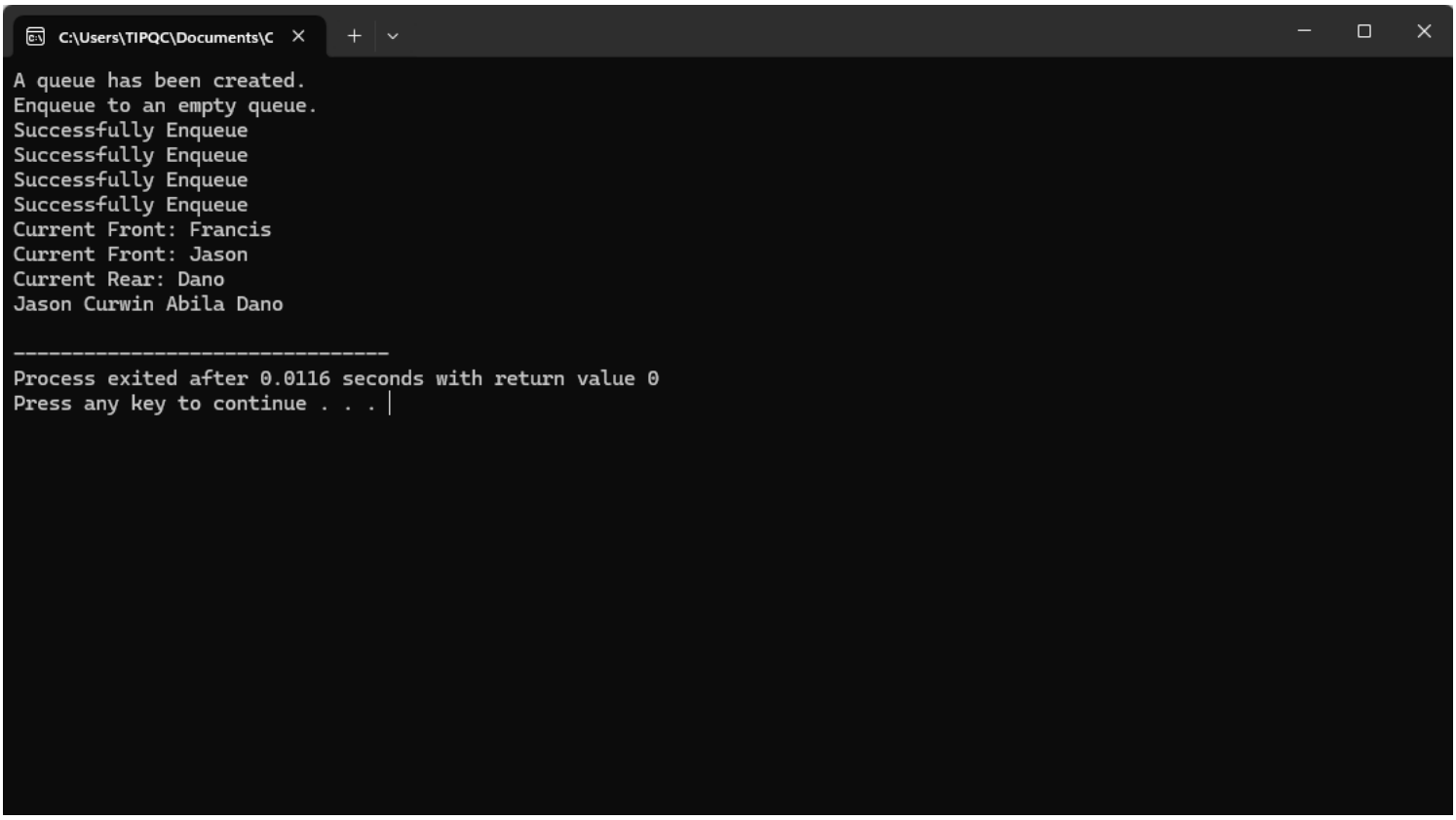

C:\Users\TIPQC\Documents\CPE010S4\queue.h - Dev-C++ 5.10
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4
(globals)
Project Classes Debug
Main.cpp queue.h
#ifndef QUEUE_H
#define QUEUE_H
#include <iostream>
template <typename T>
class Node{
public:
T data;
Node* next;
Node(T new_data){
data = new_data;
next = nullptr;
}
};
template<typename T>
class Queue{
private:
Node<T> *front;
Node<T> *rear;
public:
//create an empty queue
Queue(){
front = rear = nullptr;
std::cout<<"A queue has been created.\n";
}
//isEmpty
bool isEmpty(){
return front == nullptr;
}


C:\Users\TIPQC\Documents\CPE010S4\queue.h - Dev-C++ 5.10
le Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.8.1 64-bit Release
(globals)
roject Classes Debug
Main.cpp queue.h
36
37 //enqueue
38 void enqueue(T new_data){
39 Node<T> *new_node = new Node<T>(new_data);
40
41
42
43
44 if(isEmpty()){
45 front = rear = new_node;
46 std::cout << "Enqueue to an empty queue." << std::endl;
47 return;
48 }
49 rear->next= new_node;
50 rear = new_node;
51 std::cout << "Successfully Enqueue" << std::endl;
52 }
53 //dequeue
54 void dequeue(){
55 if(isEmpty()){
56 return;
57 }
58 //storing the front to a temporary pointer
59 Node <T>* temp = front;
60 //check if after the dequeue, the queue is empty
61 if(front == nullptr){
62 rear == nullptr;
63 }
64 else{
65 front = front -> next;
66 }
67 delete temp;
68 }

```
70
71          //getfront
72          void getFront(){
73              if (isEmpty()){
74                  std::cout<<"The queue is empty." << std::endl;
75                  return;
76              }
77              std::cout<<"Current Front: " << front -> data << std::endl;
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          //display
88          void display(){
89              if (isEmpty()){
90                  std::cout << "The queue is empty.\n";
91                  return;
92                  }
93
94          Node<T> *temp=front;
95          while (temp !=nullptr){
96              std::cout << temp -> data << " ";
97              temp = temp -> next;
98          }
99
100         std::cout<<std::endl;
101 }
```

```
1   }
2
3   ~Queue(){
4       while (!isEmpty()){
5           dequeue();
6       }
7   }
8   };
9
0   #endif
```

OUTPUT:


C:\Users\TIPQC\Documents\C
A queue has been created.
Enqueue to an empty queue.
Successfully Enqueue
Successfully Enqueue
Successfully Enqueue
Successfully Enqueue
Current Front: Francis
Current Front: Jason
Current Rear: Dano
Jason Curwin Abila Dano

Process exited after 0.0116 seconds with return value 0
Press any key to continue . . .

7. Supplementary Activity

8. Conclusion

In the activity, we learned how to replace the front with the new data and adding rear with new data. Just like stacks and arrays we use public and private for the typename and the Node which is the new data. Queue is the one who waits for the front to be done and the next front is the next or new node. Filling the blanks or the box if it is empty or not, we can add more and remove one for the next rear to be called upon. We identify the rear and front using void if the front is empty and the rear will be identify.

9. Assessment Rubric