

**Activity No. <5.1>**

**<Queues>**

<b>Course Code:</b> CPE010	<b>Program:</b> Computer Engineering
<b>Course Title:</b> Data Structures and Algorithms	<b>Date Performed:</b> 9/11/2025
<b>Section:</b> CPE21S4	<b>Date Submitted:</b> 9/11/2025
<b>Name(s):</b> Lester Arvid P. Anastacio	<b>Instructor:</b> Engr. Jimlord Quejado

**6. Output**

## TABLE 5-1

CODE:

```
#include <iostream>
#include <queue>
#include <string>

void display(std::queue<std::string> q)
{
    std::queue<std::string> c = q;
    while (!c.empty())
    {
        std::cout << " " << c.front();
        c.pop();
    }
    std::cout << "\n";
}

int main()
{
    std::queue<std::string> a;
    std::string students[] = {"Lester", "Arvid", "Crow", "Diana", "Villa"};
    int n = sizeof(students) / sizeof(students[0]);

    for (int i = 0; i < n; i++)
    {
        a.push(students[i]);
    }

    std::cout << "The queue a is :";
    display(a);

    std::cout << "a.empty() : " << a.empty() << "\n";
    std::cout << "a.size() : " << a.size() << "\n";
    std::cout << "a.front() : " << a.front() << "\n";
    std::cout << "a.back() : " << a.back() << "\n";

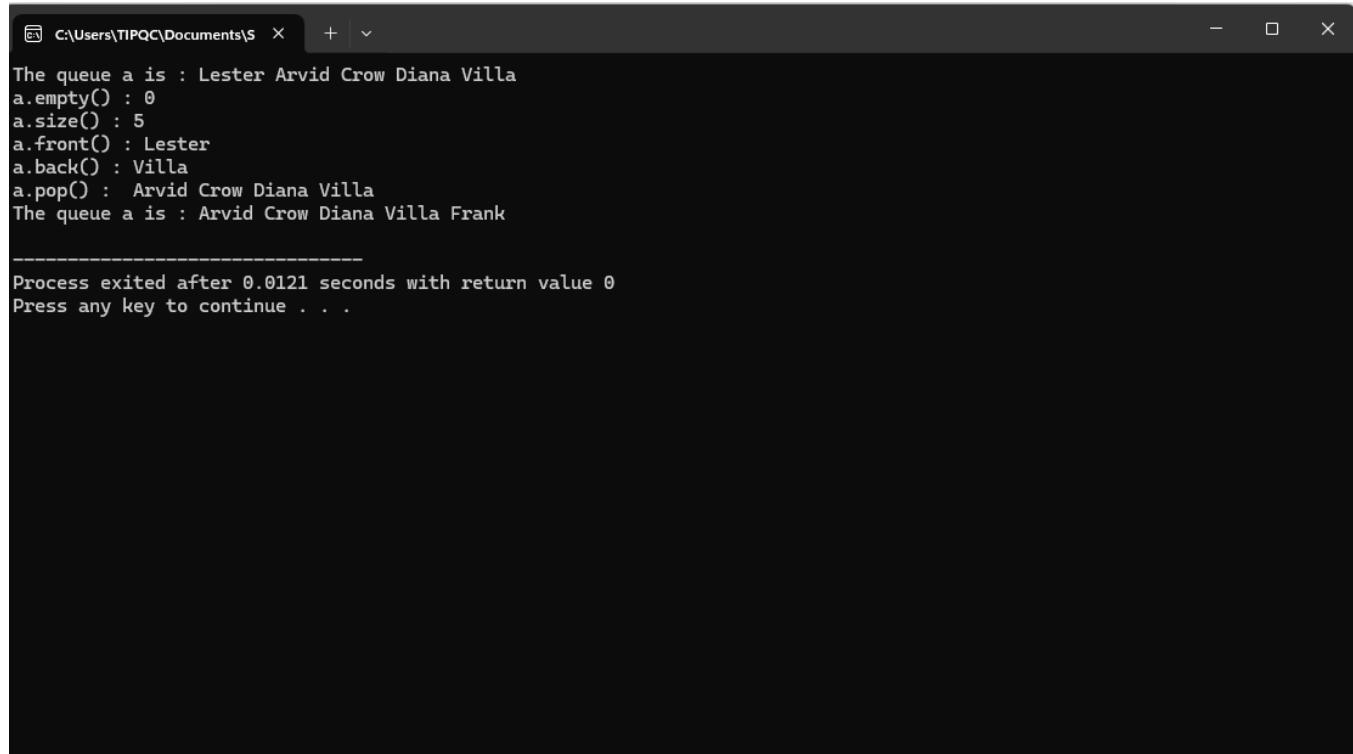
    std::cout << "a.pop() : ";
    a.pop();
    display(a);

    a.push("Frank");
}
```

```
    std::cout << "The queue a is :";
    display(a);

    return 0;
}
```

OUTPUT:



```
The queue a is : Lester Arvid Crow Diana Villa
a.empty() : 0
a.size() : 5
a.front() : Lester
a.back() : Villa
a.pop() : Arvid Crow Diana Villa
The queue a is : Arvid Crow Diana Villa Frank

-----
Process exited after 0.0121 seconds with return value 0
Press any key to continue . . .
```

## TABLE 5-2

CODE:

```
#ifndef QHEADER_H
#define QHEADER_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.." << std::endl;
        }

        //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 << "Successfully Enqueued. " << std::endl;
        }

        //dequeue
        void dequeue(){
            if (isEmpty()){
                std::cout << "The Queue is Empty" << std::endl;
                return;
            }

            //storing the front to a temporary pointer
            Node<T>* temp = front;

            //check if after the dequeue, the queue is empty
            if (front == nullptr){
                rear == nullptr;
            }
            else{
                //reassign the front to the next node
                front = front->next;
            }
        }
}

```

```

        delete temp;
    }

//getfront
void getFront(){
    if (isEmpty()){
        std::cout<<"The Queue is Empty"<<std::endl;
        return;
    }

    std::cout<<"Current Front: "<< front -> data << std::endl;
}

//getrear
void getRear(){
    if (isEmpty()){
        std::cout<<"The Queue is Empty"<<std::endl;
        return;
    }

    std::cout<<"Current Rear: "<< rear -> data << std::endl;
}

//display
void Display(){
    if (isEmpty()){
        std::cout<<"The Queue is Empty"<<std::endl;
        return;
    }

    Node<T> *temp = front;
    while (temp !=nullptr){
        std::cout<< temp -> data << " ";
        temp = temp -> next;
    }
    std::cout<<std::endl;
}

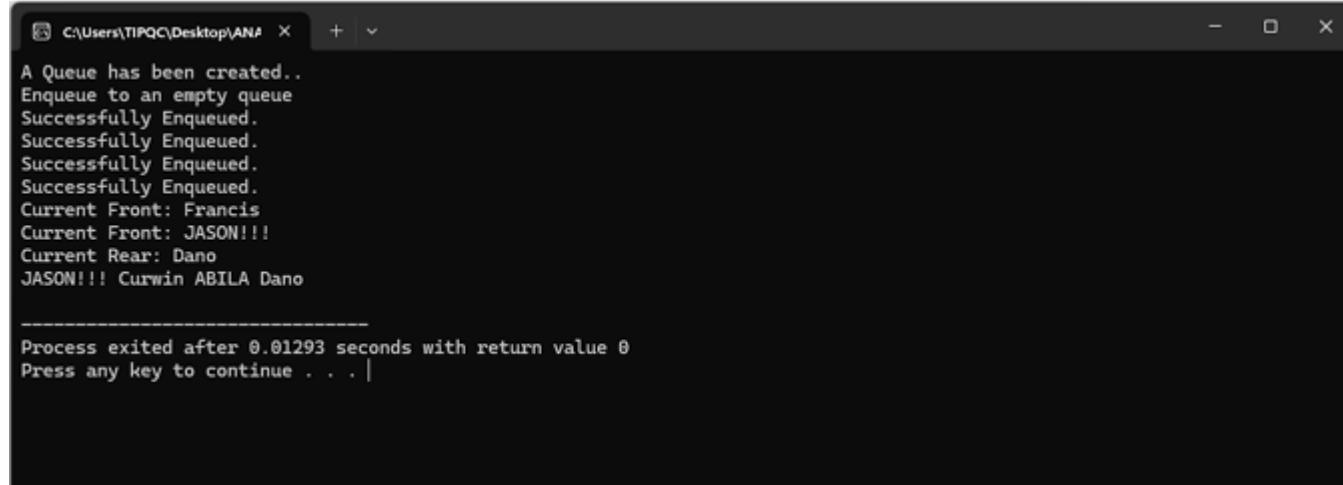
//to deallocate memory
~Queue(){
    while(!isEmpty()){
        dequeue();
    }
}

};


```

```
#endif
```

OUTPUT:



```
C:\Users\TIPQC\Desktop\ANA X + ▾
A Queue has been created..
Enqueue to an empty queue
Successfully Enqueued.
Successfully Enqueued.
Successfully Enqueued.
Successfully Enqueued.
Current Front: Francis
Current Front: JASON!!!
Current Rear: Dano
JASON!!! Curwin ABILA Dano

Process exited after 0.01293 seconds with return value 0
Press any key to continue . . . |
```

### TABLE 5-3

CODE:

*Main.cpp*

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

```
int main() {
    // Create a queue of integers
    Queue<int> q;

    // Enqueue elements
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    // Display queue contents
    std::cout << "Queue contents: ";
    q.Display();

    // Show front and rear
    q.getFront();
    q.getRear();

    // Show size
    std::cout << "Queue size: " << q.Size() << std::endl;

    // Dequeue an element
    q.dequeue();
    std::cout << "After one dequeue:" << std::endl;
    q.Display();

    // Clear the queue
}
```

```

q.Clear();
std::cout << "After clearing the queue." << std::endl;
q.Display();

// Test copy constructor
q.enqueue(100);
q.enqueue(200);
Queue<int> q2(q);
std::cout << "Copied queue contents (q2): ";
q2.Display();

// Test assignment operator
Queue<int> q3;
q3 = q;
std::cout << "Assigned queue contents (q3): ";
q3.Display();

return 0;
}

```

### ***Q\_Header.h***

```

#ifndef Q_HEADER_H
#define Q_HEADER_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;
    int q_size;

public:
    // Constructor
    Queue() : front(nullptr), rear(nullptr), q_size(0) {
        std::cout << "A Queue has been created.." << std::endl;
    }

    // Copy Constructor

```

```

Queue(const Queue& other) : front(nullptr), rear(nullptr), q_size(0) {
    Node<T>* temp = other.front;
    while (temp != nullptr) {
        enqueue(temp->data);
        temp = temp->next;
    }
}

// Copy Assignment Operator
Queue& operator=(const Queue& other) {
    if (this != &other) {
        Clear();
        Node<T>* temp = other.front;
        while (temp != nullptr) {
            enqueue(temp->data);
            temp = temp->next;
        }
    }
    return *this;
}

// Destructor
~Queue() {
    Clear();
}

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

// Size
int Size() const {
    return q_size;
}

// Clear
void Clear() {
    while (!isEmpty()) {
        dequeue();
    }
}

// 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;
    } else {
        rear->next = new_node;
        rear = new_node;
    }
}

```

```
        std::cout << "Successfully Enqueued." << std::endl;
    }
    ++q_size;
}

// Dequeue
void dequeue() {
    if (isEmpty()) {
        std::cout << "The Queue is Empty" << std::endl;
        return;
    }
    Node<T>* temp = front;
    front = front->next;
    if (front == nullptr) {
        rear = nullptr;
    }
    delete temp;
    --q_size;
}

// getFront
void getFront() const {
    if (isEmpty()) {
        std::cout << "The Queue is Empty" << std::endl;
        return;
    }
    std::cout << "Current Front: " << front->data << std::endl;
}

// getRear
void getRear() const {
    if (isEmpty()) {
        std::cout << "The Queue is Empty" << std::endl;
        return;
    }
    std::cout << "Current Rear: " << rear->data << std::endl;
}

// Display
void Display() const {
    if (isEmpty()) {
        std::cout << "The Queue is Empty" << std::endl;
        return;
    }
    Node<T>* temp = front;
    while (temp != nullptr) {
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    std::cout << std::endl;
}
```

```
#endif
```

OUTPUT:

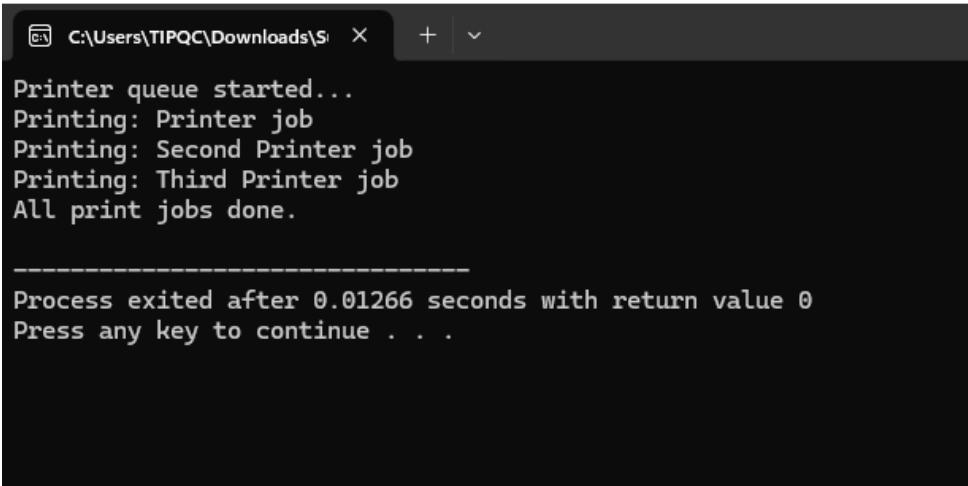
```
C:\Users\TIPQC\Documents\S X + | ▾  
A Queue has been created..  
  
Enqueue to an empty queue  
Successfully Enqueued.  
Successfully Enqueued.  
Queue contents: 10 20 30  
Current Front: 10  
Current Rear: 30  
Queue size: 3  
After one dequeue:  
20 30  
After clearing the queue:  
The Queue is Empty  
Enqueue to an empty queue  
Successfully Enqueued.  
Enqueue to an empty queue  
Successfully Enqueued.  
Copied queue contents (q2): 100 200  
A Queue has been created..  
  
Enqueue to an empty queue  
Successfully Enqueued.  
Assigned queue contents (q3): 100 200  
  
-----  
Process exited after 0.01015 seconds with return value 0  
Press any key to continue . . . |
```

## 7. Supplementary Activity

CODE:

```
1 #include <iostream>
2 #include <queue>
3 #include <string>
4
5 int main() {
6     std::queue<std::string> printerQueue;
7
8     // Adding print jobs
9     printerQueue.push("Printer job");
10    printerQueue.push("Second Printer job");
11    printerQueue.push("Third Printer job");
12
13    std::cout << "Printer queue started...\n";
14
15    // Process the queue
16    while (!printerQueue.empty()) {
17        std::cout << "Printing: " << printerQueue.front() << std::endl;
18        printerQueue.pop(); // Remove the job after printing
19    }
20
21    std::cout << "All print jobs done.\n";
22
23    return 0;
24 }
```

OUTPUT:



```
C:\Users\TIPQC\Downloads\$ + ^
Printer queue started...
Printing: Printer job
Printing: Second Printer job
Printing: Third Printer job
All print jobs done.

Process exited after 0.01266 seconds with return value 0
Press any key to continue . . .
```

## 8. Conclusion

## 9. Assessment Rubric