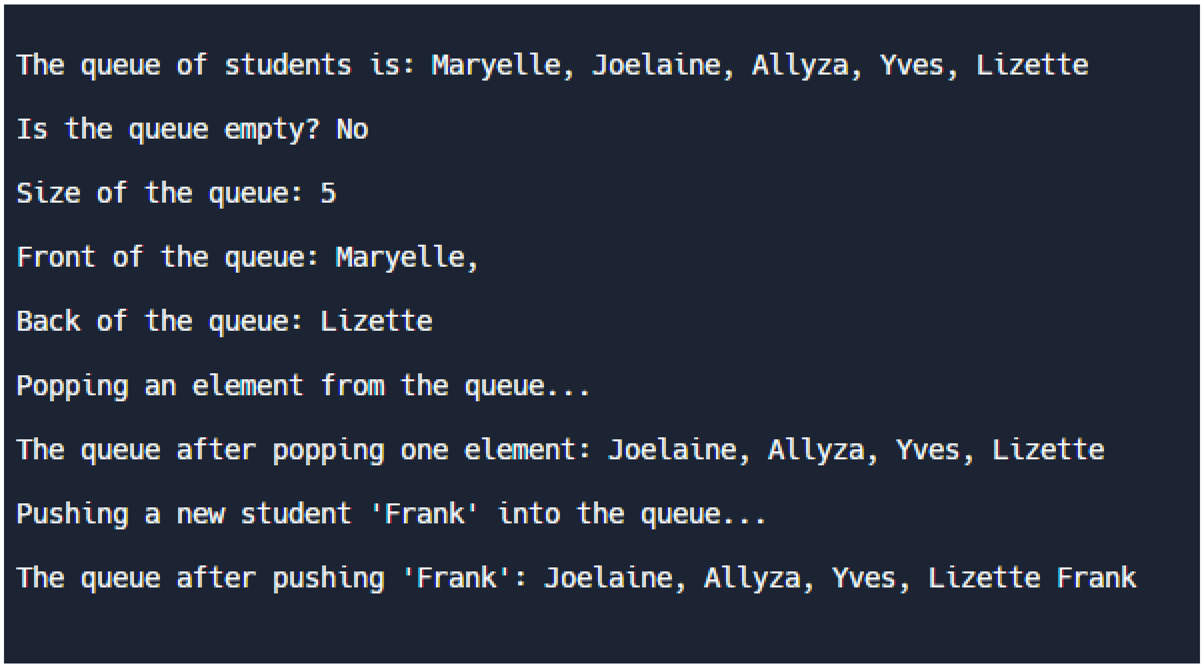
Activity No. 5	
QUEUES	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** Oct 7, 2024
Section: CPE21S4	**Date Submitted:** Oct 8, 2024
Name(s): Danica T. Guariño	**Instructor: Maria Rizette Sayo**

6. Output

C/C++

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

using namespace std;


void display(queue<string> q) {
    while (!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
    cout << endl;
}

int main() {

    queue<string> studentQueue;


    studentQueue.push("Maryelle,");
    studentQueue.push("Joelaine,");
    studentQueue.push("Allyza,");
    studentQueue.push("Yves,");
    studentQueue.push("Lizette");


    cout << "\nThe queue of students is: ";
    display(studentQueue);


    cout << "\nIs the queue empty? " << (studentQueue.empty() ? "Yes" : "No") << endl;
    cout << "\nSize of the queue: " << studentQueue.size() << endl;
    cout << "\nFront of the queue: " << studentQueue.front() << endl;
    cout << "\nBack of the queue: " << studentQueue.back() << endl;


    cout << "\nPopping an element from the queue..." << endl;
    studentQueue.pop();
    cout << "\nThe queue after popping one element: ";
    display(studentQueue);


    cout << "\nPushing a new student 'Frank' into the queue..." << endl;
    studentQueue.push("Frank");
    cout << "\nThe queue after pushing 'Frank': ";
    display(studentQueue);

    return 0;
}
```

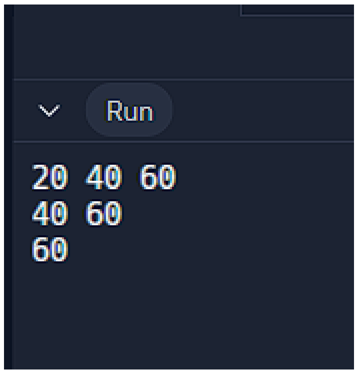


```
The queue of students is: Maryelle, Joelaine, Allyza, Yves, Lizette

Is the queue empty? No

Size of the queue: 5

Front of the queue: Maryelle,

Back of the queue: Lizette

Popping an element from the queue...

The queue after popping one element: Joelaine, Allyza, Yves, Lizette

Pushing a new student 'Frank' into the queue...

The queue after pushing 'Frank': Joelaine, Allyza, Yves, Lizette Frank
```

Table 5-1. Queues using C++ STL

```
C/C++
#include <iostream>

struct Node {
    int data;
    Node* next;

    Node(int data) : data(data), next(nullptr) {}
};
```

```cpp
class Queue {
private:
    Node* front;
    Node* rear;

public:
    Queue() : front(nullptr), rear(nullptr) {}


    void enqueue(int data) {
        Node* newNode = new Node(data);
        if (rear) {
            rear->next = newNode;
        }
        rear = newNode;
        if (!front) {
            front = rear;
        }
    }


    void dequeue() {
        if (!front) {
            std::cout << "Queue is empty\n";
            return;
        }
        Node* temp = front;
        front = front->next;
        if (!front) {
            rear = nullptr;
        }
        delete temp;
    }


    void display() {
        Node* temp = front;
        while (temp) {
            std::cout << temp->data << " ";
            temp = temp->next;
        }
        std::cout << "\n";
    }
};

int main() {
    Queue q;


    q.enqueue(20);
    q.enqueue(40);
    q.enqueue(60);
    q.display();
    q.dequeue();
    q.display();

    q.dequeue();
    q.display();

    q.dequeue();
    q.display();
    return 0;
}
```



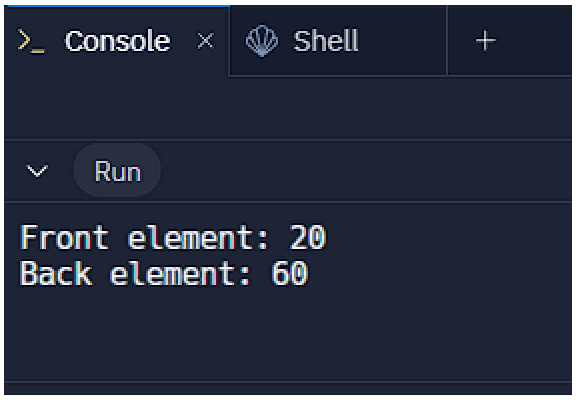
```
20 40 60
40 60
60
```

Table 5-2. Queues using Linked List Implementation

C/C++

```
#include <iostream>
using namespace std;

class CircularQueue {
private:
    int *q_array;
    int q_capacity, q_size, q_front, q_back;

public:
    CircularQueue(int capacity) : q_capacity(capacity), q_size(0), q_front(0),
q_back(capacity - 1) {
        q_array = new int[q_capacity];
    }
```

```cpp
    ~CircularQueue() {
        delete[] q_array;
    }

    bool isEmpty() const {
        return q_size == 0;
    }

    void enqueue(int data) {
        if (q_size == q_capacity) {
            throw runtime_error("Queue is full");
        }
        q_back = (q_back + 1) % q_capacity;
        q_array[q_back] = data;
        q_size++;
    }

    void dequeue() {
        if (isEmpty()) {
            throw runtime_error("Queue is empty");
        }
        q_front = (q_front + 1) % q_capacity;
        q_size--;
    }

    int front() const {
        if (isEmpty()) {
            throw runtime_error("Queue is empty");
        }
        return q_array[q_front];
    }

    int back() const {
        if (isEmpty()) {
            throw runtime_error("Queue is empty");
        }
        return q_array[q_back];
    }
};

int main() {
    CircularQueue q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);

    q.dequeue();
    q.enqueue(60);

    cout << "Front element: " << q.front() << endl; // Output: 20
    cout << "Back element: " << q.back() << endl; // Output: 60
    return 0;
}
```



```
Front element: 20
Back element: 60
```

Table 5-3. Queues using Array Implementation

7. Supplementary Activity

ILO C: Solve problems using queue implementation
Problem Title: Shared Printer Simulation using Queues
Problem Definition: In this activity, we'll simulate a queue for a shared printer in an office. In any corporate office, usually, the printer is shared across the whole floor in the printer room. All the computers in this room are connected to the same printer. But a printer can do only one printing job at any point in time, and it also takes some time to complete any job. In the meantime, some other user can send another print request. In such a case, a printer needs to store all the pending jobs somewhere so that it can take them up once its current task is done.

Perform the following steps to solve the activity. **Make sure that you include a screenshot of the source code for each.** From the get-go: You must choose whether you are making a linked list or array implementation.
You may NOT use the STL.

1. Create a class called Job (comprising an ID for the job, the name of the user who submitted it, and the number of pages).
2. Create a class called Printer. This will provide an interface to add new jobs and process all the jobs added so far.
3. To implement the printer class, it will need to store all the pending jobs. We'll implement a very basic strategy – first come, first served. Whoever submits the job first will be the first to get the job done.
4. Finally, simulate a scenario where multiple people are adding jobs to the printer, and the printer is processing them one by one.
5. Defend your choice of internal representation: Why did you use arrays or linked list?

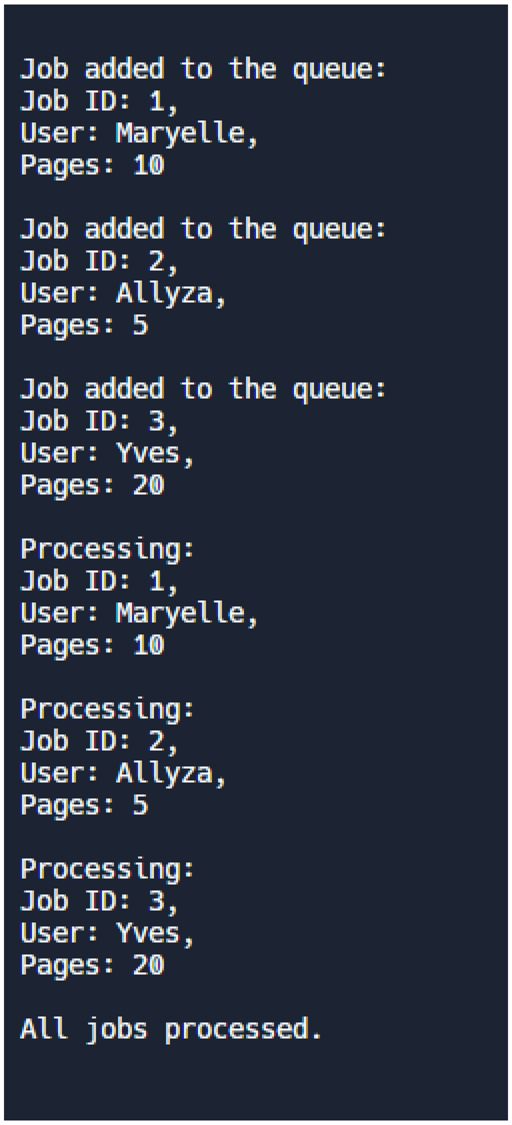
Output Analysis:

• Provide the output after performing each task above.

• Include your analysis: focus on why you think the output is the way it is.

C/C++

```cpp
#include <iostream>
#include <string>


class Job {
public:
    int jobID;
    std::string userName;
    int numPages;

    Job(int id, const std::string& name, int pages) : jobID(id), userName(name),
numPages(pages) {}

    void printJobInfo() const {
        std::cout << "\nJob ID: " << jobID << ", \nUser: " << userName << ", \nPages: " <<
numPages << std::endl;
    }
};


struct Node {
    Job* job;
    Node* next;

    Node(Job* j) : job(j), next(nullptr) {}
};


class Printer {
private:
    Node* front;
    Node* rear;

public:
    Printer() : front(nullptr), rear(nullptr) {}


    void addJob(Job* newJob) {
        Node* newNode = new Node(newJob);
        if (rear == nullptr) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        std::cout << "\nJob added to the queue: ";
        newJob->printJobInfo();
    }
    void processJobs() {
        while (front != nullptr) {
            std::cout << "\nProcessing: ";
            front->job->printJobInfo();
            Node* temp = front;
            front = front->next;
            delete temp;
        }
        std::cout << "\nAll jobs processed." << std::endl;
    }
};

int main() {
    Printer printer;


    printer.addJob(new Job(1, "Maryelle", 10));
    printer.addJob(new Job(2, "Allyza", 5));
    printer.addJob(new Job(3, "Yves", 20));


    printer.processJobs();

    return 0;
}
```



```
Job added to the queue:
Job ID: 1,
User: Maryelle,
Pages: 10

Job added to the queue:
Job ID: 2,
User: Allyza,
Pages: 5

Job added to the queue:
Job ID: 3,
User: Yves,
Pages: 20

Processing:
Job ID: 1,
User: Maryelle,
Pages: 10

Processing:
Job ID: 2,
User: Allyza,
Pages: 5

Processing:
Job ID: 3,
User: Yves,
Pages: 20

All jobs processed.
```

8. Conclusion

In conclusion, this activity is simple if you understand the concept of a queue. At first, I was confused, but as I worked on it, I realized it was simple and interesting. The more I practiced, the clearer everything got. It's satisfying to see how elements flow in and out of the queue in an organized manner.

9. Assessment Rubric