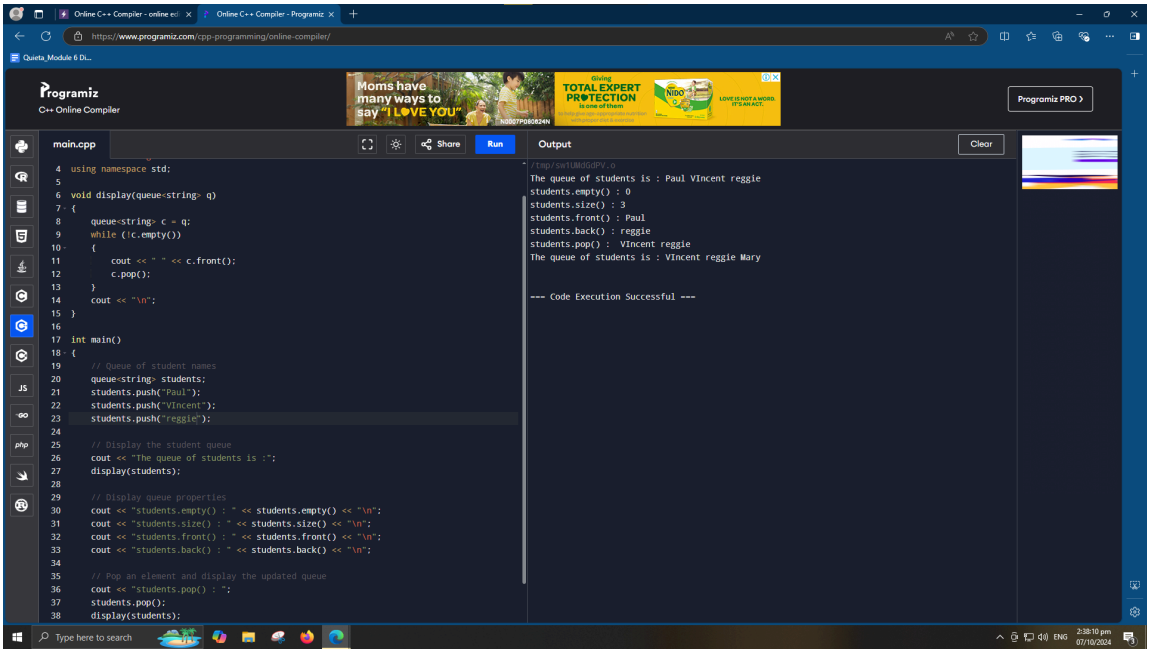
Activity No. <n>

Hands-on Activity 5.1

Queues

Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 10/7/24**
Section: CPE21S4	**Date Submitted:10/7/24**
Name(s): Solis, Paul Vincent M.	**Instructor: Prof.Sayo**

6. Output



C/C++

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

void display(queue<string> q)
{
    queue<string> c = q;
    while (!c.empty())
```

```
    {
        cout << " " << c.front();
        c.pop();
    }
    cout << "\n";
}

int main()
{
    // Queue of student names
    queue<string> students;
    students.push("Paul");
    students.push("Vincent");
    students.push("Reggie");

       cout << "The queue of students is :";
    display(students);

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

      cout << "students.pop() : ";
    students.pop();
    display(students);


    students.push("Mary");
    cout << "The queue of students is :";
    display(students);

    return 0;
}
```

Table 5-1. Queues using C++ STL

```
/tmp/CMwiStP5Vx.o
10 inserted into the queue.
20 inserted into the queue.
30 inserted into the queue.
Queue: 10 20 30
Deleted 10 from the queue.
Queue: 20 30
Deleted 20 from the queue.
Queue: 30
Deleted 30 from the queue.
Queue is empty.


=== Code Execution Successful ===
```

Table 5-2. Queues using Linked List Implementation

C/C++

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

// Define the structure for a Node in the linked list
struct Node {
    int data;
    Node* next;
};

// Queue class
class Queue {
private:
    Node* front;  // Points to the front of the queue
    Node* rear;   // Points to the rear of the queue

public:
    // Constructor to initialize the queue
    Queue() {
        front = rear = nullptr;
    }

    // Insert an item into the queue
    void enqueue(int value) {
        Node* temp = new Node(); // Create a new node
        temp->data = value;
        temp->next = nullptr;

        // Inserting into an empty queue
        if (rear == nullptr) {
            front = rear = temp;
        }
```

```
        // Inserting into a non-empty queue
        else {
            rear->next = temp;
            rear = temp;
        }
        cout << value << " inserted into the queue.\n";
    }

    // Delete an item from the queue
    void dequeue() {
        if (front == nullptr) {  // Queue is empty
            cout << "Queue is empty. Nothing to delete.\n";
            return;
        }

        Node* temp = front;  // Store the front node
        front = front->next; // Move front to the next node

        // Queue has only one item
        if (front == nullptr) {
            rear = nullptr;  // If the queue is empty after deletion, set rear to null
        }

        cout << "Deleted " << temp->data << " from the queue.\n";
        delete temp;  // Free memory of the old front
    }

    // Display the queue
    void display() {
        if (front == nullptr) {
            cout << "Queue is empty.\n";
            return;
        }

        Node* temp = front;
        cout << "Queue: ";
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << "\n";
    }
};

int main() {
    Queue q;

    // Insert items into the queue
    q.enqueue(10);  // Inserting into an empty queue
    q.enqueue(20);  // Inserting into a non-empty queue
    q.enqueue(30);

    q.display();    // Display the queue

    // Deleting items from the queue
```

```
    q.dequeue();    // Deleting from a queue with more than one item
    q.display();

    q.dequeue();    // Deleting from a queue with more than one item
    q.display();

    q.dequeue();    // Deleting from a queue with only one item
    q.display();

    return 0;
}
```

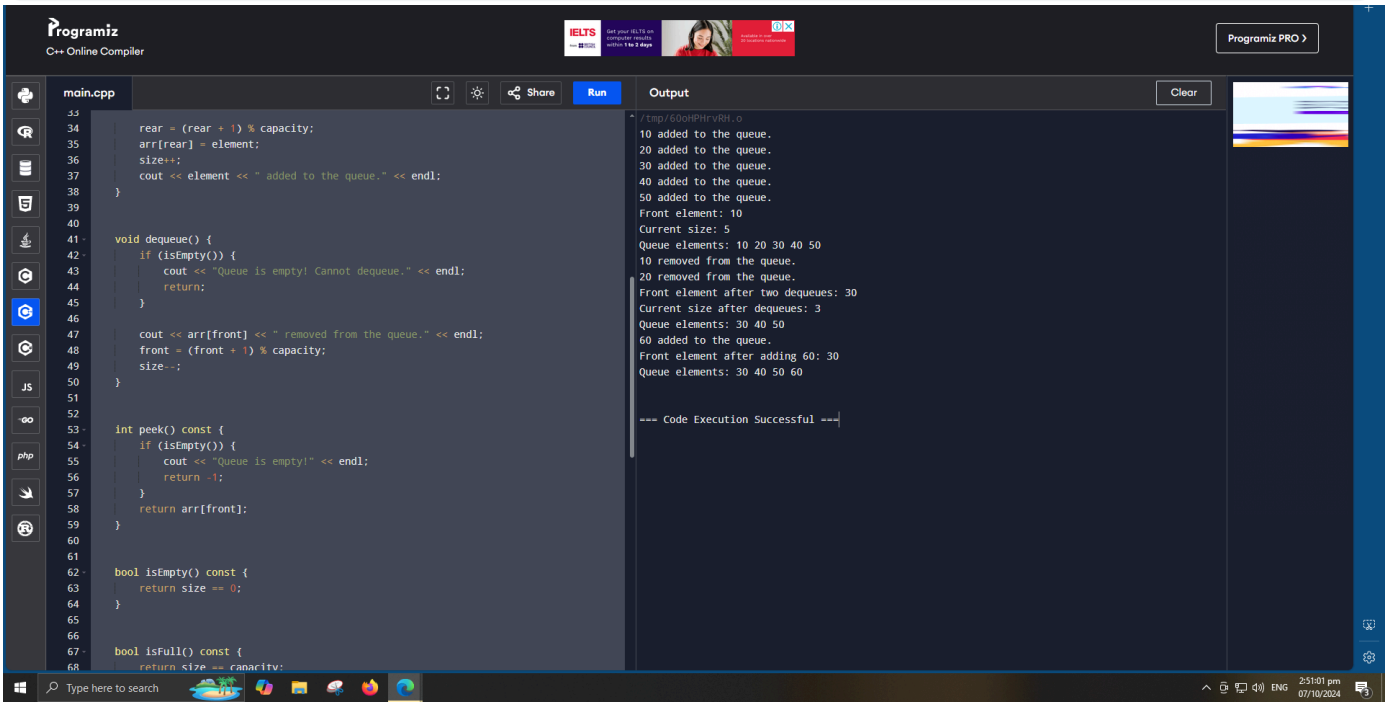
Table 5-3. Queues using Array Implementation

C/C++

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

class Queue {
private:
    int *arr;
    int front;
    int rear;
    int capacity;
    int size;

public:

    Queue(int cap) {
        capacity = cap;
        arr = new int[capacity];
        front = 0;
        rear = -1;
        size = 0;
    }


    ~Queue() {
        delete[] arr;
    }


    void enqueue(int element) {
        if (isFull()) {
            cout << "Queue is full! Cannot enqueue " << element << endl;
            return;
        }

        rear = (rear + 1) % capacity;
        arr[rear] = element;
```

```
        size++;
        cout << element << " added to the queue." << endl;
    }


    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty! Cannot dequeue." << endl;
            return;
        }

        cout << arr[front] << " removed from the queue." << endl;
        front = (front + 1) % capacity;
        size--;
    }


    int peek() const {
        if (isEmpty()) {
            cout << "Queue is empty!" << endl;
            return -1;
        }
        return arr[front];
    }


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


    bool isFull() const {
        return size == capacity;
    }


    int getSize() const {
        return size;
    }


    void display() const {
        if (isEmpty()) {
            cout << "Queue is empty!" << endl;
            return;
        }

        cout << "Queue elements: ";
        for (int i = 0; i < size; i++) {
            cout << arr[(front + i) % capacity] << " ";
        }
        cout << endl;
    }
};
```

```cpp
int main() {
    Queue q(5);

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

    cout << "Front element: " << q.peek() << endl;
    cout << "Current size: " << q.getSize() << endl;

    q.display();

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

    cout << "Front element after two dequeues: " << q.peek() << endl;
    cout << "Current size after dequeues: " << q.getSize() << endl;

    q.display();

    q.enqueue(60);
    cout << "Front element after adding 60: " << q.peek() << endl;

    q.display();

    return 0;
}
```



7. Supplementary Activity

1.

```
C/C++
#include <iostream>
using namespace std;

class Job {
private:
    int id;
    string userName;
    int pages;

public:
    Job(int id, string userName, int pages) {
        this->id = id;
        this->userName = userName;
        this->pages = pages;
    }

    int getId() {
        return id;
    }

    string getUserName() {
        return userName;
    }

    int getPages() {
        return pages;
    }
};
```

2.

```
C/C++
class Printer {
private:
    Job* queue;
    int capacity;
    int size;
    int front;
    int rear;

public:
    Printer(int capacity) {
        this->capacity = capacity;
        this->size = 0;
        this->front = 0;
        this->rear = 0;
        queue = new Job[capacity];
    }

    ~Printer() {
```

```
        delete[] queue;
    }

    void addJob(Job job) {
        if (size == capacity) {
            cout << "Printer queue is full.\n";
            return;
        }
        queue[rear] = job;
        rear = (rear + 1) % capacity;
        size++;
    }

    void processJobs() {
        if (size == 0) {
            cout << "No jobs to process.\n";
            return;
        }
        for (int i = 0; i < size; i++) {
            Job job = queue[(front + i) % capacity];
            cout << "Processing job " << job.getId() << " for " << job.getPages() << "
pages by " << job.getUserName() << ".\n";
        }
        size = 0;
        front = 0;
        rear = 0;
    }
};
```

3.

C/C++

```
int main() {
    Printer printer(5);

    Job job1(1, "John", 5);
    Job job2(2, "Alice", 3);
    Job job3(3, "Bob", 2);
    Job job4(4, "Eve", 4);
    Job job5(5, "Charlie", 1);

    printer.addJob(job1);
    printer.addJob(job2);
    printer.addJob(job3);
    printer.addJob(job4);
    printer.addJob(job5);

    printer.processJobs();

    return 0;
}
```

4.

```
Processing job 1 for 5 pages by John.
Processing job 2 for 3 pages by Alice.
Processing job 3 for 2 pages by Bob.
Processing job 4 for 4 pages by Eve.
Processing job 5 for 1 pages by Charlie.
```

5.
Since an array offers constant-time access to any element and a straightforward and effective method of storing and retrieving jobs, I decided to use it to create the printer queue. Moreover, arrays use less memory than linked lists, which makes them a better option for this simulation. The array implementation is a good option for a first-come, first-served method because it is also simple to comprehend and use. All things considered, the array is a suitable fit for this particular use case due to its simplicity and efficiency.

8. Conclusion

To sum up, the Shared Printer Simulation with Queues illustrates how to manage print jobs in a shared printer environment by using a queue data structure effectively. For this simulation, the array-based design makes sense since it offers a straightforward and effective method of storing and retrieving jobs. The first-come, first-served approach, in which jobs are completed in the order they are added to the queue, is demonstrated by the simulation. All things considered, the simulation emphasizes how crucial queue data structures are to the equitable and effective management of tasks and jobs.*CPE010_HOA3_MaamSayo*

9. Assessment Rubric