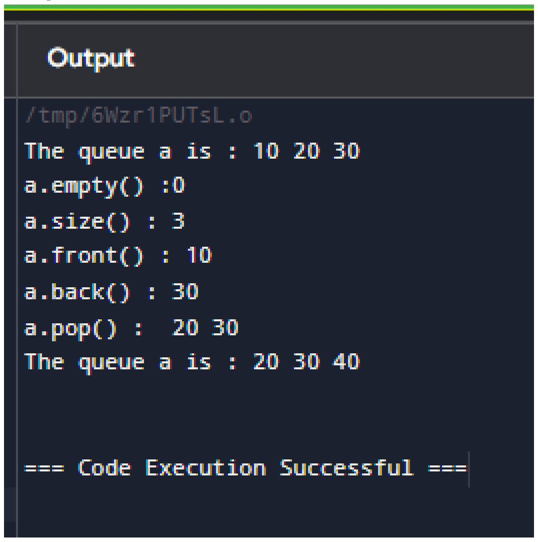
Activity No. 5	
Hands-on Activity 5.1 Queues	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10/07/2024
Section: CPE21S4	**Date Submitted:** 10/08/2024
Name(s): Edson Ray E. San Juan	**Instructor:** Prof. Maria Rizette Sayo
6. Output	

Output:



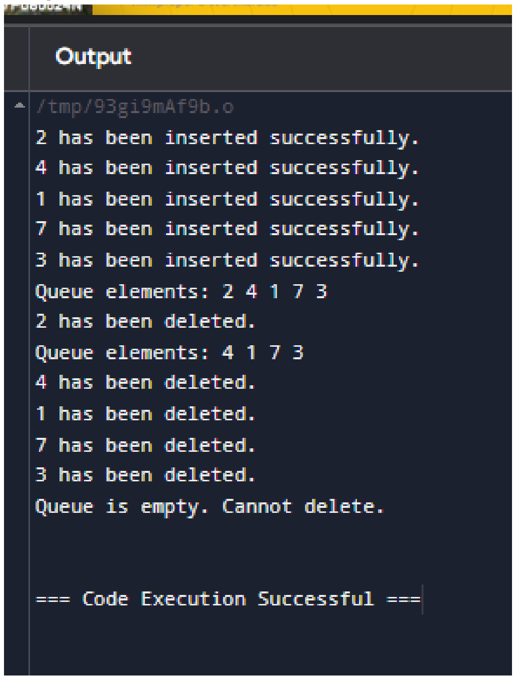
```
/tmp/6Wzr1PUTsL.o
The queue a is : 10 20 30
a.empty() :0
a.size() : 3
a.front() : 10
a.back() : 30
a.pop() :  20 30
The queue a is : 20 30 40


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

Observation:

The provided C++ code demonstrates the use of the Standard Template Library (STL) queue container by initializing a queue, adding integers (10, 20, and 30), and displaying its contents while checking properties like size and emptiness. It showcases basic queue operations, including enqueueing, dequeueing, and maintaining the First-In-First-Out (FIFO) principle, as reflected in the output after each operation.

Table 5-1. Queues using C++ STL

Output:



```
/tmp/93gi9mAf9b.o
2 has been inserted successfully.
4 has been inserted successfully.
1 has been inserted successfully.
7 has been inserted successfully.
3 has been inserted successfully.
Queue elements: 2 4 1 7 3
2 has been deleted.
Queue elements: 4 1 7 3
4 has been deleted.
1 has been deleted.
7 has been deleted.
3 has been deleted.
Queue is empty. Cannot delete.


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

Observation:

The provided C++ code implements a queue using a linked list, allowing for inserting, deleting, and displaying elements while handling both empty and non-empty conditions. It demonstrates the queue operations by initially adding four integers (2, 4, 1, 7) and then inserting an additional integer (3), followed by a series of deletions until the queue is empty.

Table 5-2. Queues using Linked List Implementation

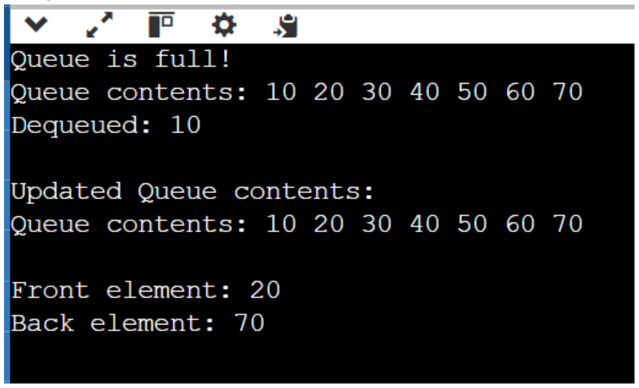
Output:	**Observation:**
 `Queue is full!` `Queue contents: 10 20 30 40 50 60 70` `Dequeued: 10` `Updated Queue contents:` `Queue contents: 10 20 30 40 50 60 70` `Front element: 20` `Back element: 70`	The queue has a maximum capacity of 7 elements, but the program attempted to add 8 elements, resulting in the message "Queue is full!" being displayed. The displayContents function shows all the elements currently in the queue, but after dequeuing (removing) an item, it still lists the original items instead of just the remaining ones. The front element is correctly displayed as 20, while the back element is shown as 70. To improve the code, adjustments are needed to ensure proper handling of the queue's maximum size and accurate display of its contents after modifications.

Table 5-3. Queues using Array Implementation

7. Supplementary Activity

ILO C: Solve problems using queue implementation

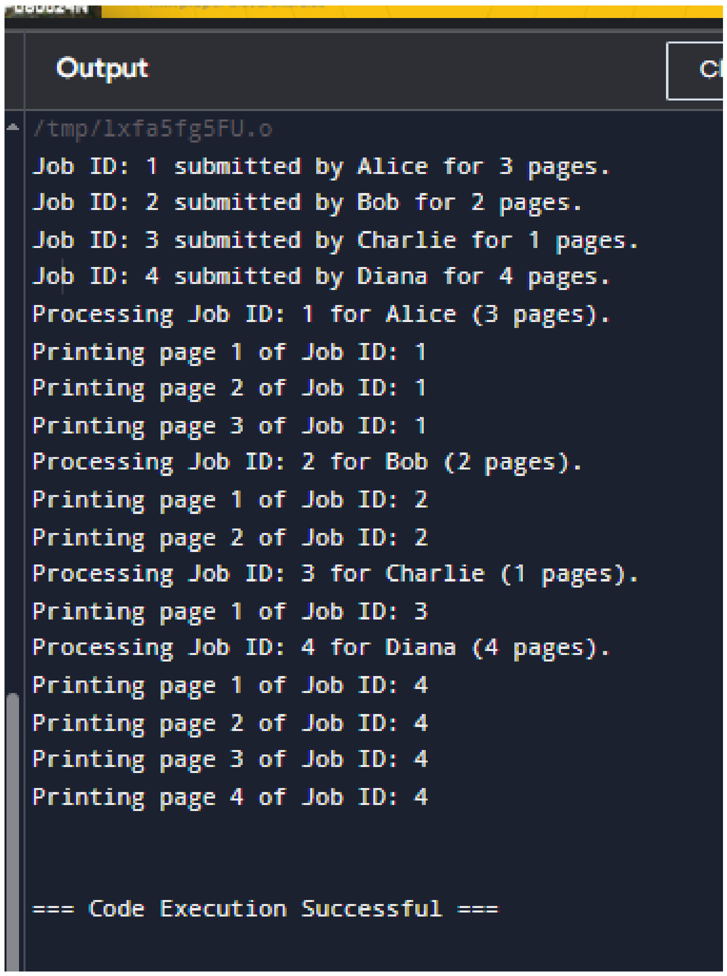
Problem Title: Shared Printer Simulation using Queues
Problem Definition: In this activity, we'll simulate a queue for a shared printer in an office. In any corporate office, usually, the printer is shared across the whole floor in the printer room. All the computers in this room are connected to the 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.

Output:


Output

```
/tmp/1xfa5fg5FU.o
Job ID: 1 submitted by Alice for 3 pages.
Job ID: 2 submitted by Bob for 2 pages.
Job ID: 3 submitted by Charlie for 1 pages.
Job ID: 4 submitted by Diana for 4 pages.
Processing Job ID: 1 for Alice (3 pages).
Printing page 1 of Job ID: 1
Printing page 2 of Job ID: 1
Printing page 3 of Job ID: 1
Processing Job ID: 2 for Bob (2 pages).
Printing page 1 of Job ID: 2
Printing page 2 of Job ID: 2
Processing Job ID: 3 for Charlie (1 pages).
Printing page 1 of Job ID: 3
Processing Job ID: 4 for Diana (4 pages).
Printing page 1 of Job ID: 4
Printing page 2 of Job ID: 4
Printing page 3 of Job ID: 4
Printing page 4 of Job ID: 4


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

Analysis:

The code effectively simulates a printer queue using a linked list to manage print jobs, ensuring that jobs are processed in a first-come, first-served manner. Each job is encapsulated in a Job class, and the Printer class manages the linked list of jobs, displaying clear messages as each job is added and processed. Overall, the implementation efficiently handles memory allocation and deallocation, providing a realistic representation of how print jobs are managed in an office setting.

8. Conclusion

Provide the following:

- Summary of lessons learned
  - I learned in the laboratory how to implement and use different types of queues (STL, linked list, and arrays), and reinforced the importance of FIFO in data handling.
- Analysis of the procedure
  - Implementing queues through C++ STL, linked lists, and arrays provided insights into memory management and handling edge cases, such as exceeding queue capacity.
- Analysis of the supplementary activity
  - The printer simulation using a linked list effectively demonstrated dynamic memory management and processing jobs in a first-come, first-served manner, proving more flexible than arrays.
- Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?
  - I think I did well in this laboratory activity, but improvements can be made in handling boundary cases and refining output accuracy. I will work on improving debugging and optimizing memory management in future implementations.

9. Assessment Rubric