**DATA STRUCTURES**

**CASE STUDY**

**TOPIC : “Print Job Scheduler”**

**INDEX**

|  |  |
| --- | --- |
| **NAME OF THE TOPIC** | **PAGE NUMBER** |
| **ABSTRACT** | **1** |
| **INTRODUCTION** | **2** |
| **QUEUE** | **2** |
| **QUEUE ALGORITHM** | **3** |
| **COMPLETE CODE** | **4** |
| **EXPLANATION OF CODE** | **8** |
| **OUTPUT & OBSERVATION** | **12** |
| **CONCLUSION** | **13** |

**ABSTRACT**

Printers often handle multiple print jobs simultaneously. The Print Job Scheduler is designed to manage these jobs efficiently, ensuring that they are processed in a fair and timely manner. This case study explores the implementation of a Print Job Scheduler using a queue data structure, focusing on First-In-First-Out (FIFO) scheduling to simulate the order of job processing.

**INTRODUCTION**

The objective of this case study is to simulate a print job scheduler utilizing a queue data structure that operates on a First-In-First-Out (FIFO) basis. In this model, each print job is represented by specific attributes, including a job ID, document name, and the number of pages to be printed. The queue type employed is a straightforward implementation that processes jobs in the order they are received, ensuring fairness and efficiency in handling multiple print requests. This approach not only reflects the practical workings of real-world printers but also provides a foundational understanding of how queue management can be applied in various scheduling scenarios.

# QUEUE

A **queue** is a linear data structure that follows the First-In-First-Out (FIFO) principle, meaning that the first element added to the queue will be the first one to be removed. Think of it like a line of people waiting to buy tickets: the person who arrives first gets served first.

In this case study of a print job scheduler, a queue is used to manage print jobs submitted to a printer. Here’s how it works:

1. **Job Submission**: Each print job, which contains attributes like job ID, document name, and number of pages, is added to the back of the queue when it is submitted.
2. **Job Processing**: When the printer is ready to process a job, it removes the job at the front of the queue and starts printing. This ensures that jobs are handled in the order they were received, reflecting the FCFS scheduling method.
3. **Efficiency and Fairness**: Using a queue allows the print scheduler to handle multiple jobs without any complex priority management. Each job is processed fairly based on its arrival time, which simplifies the management of print requests.

# QUEUE ALGORITHM

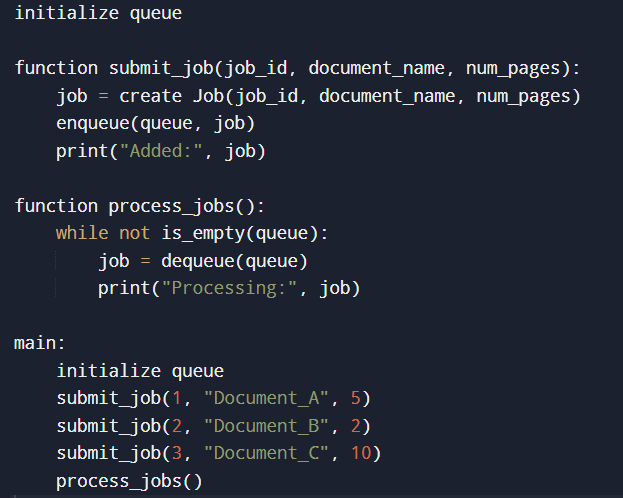
**FIFO Print Job Scheduler Algorithm**

1. **Initialize the Queue**:
   * Create an empty queue to hold print jobs.
2. **Submit Job**:
   * For each incoming print job:
     + Create a new job object with the following attributes:
       - Job ID
       - Document Name
       - Number of Pages
     + Add the job to the end of the queue.
3. **Process Jobs**:
   * While the queue is not empty:
     + Remove the job at the front of the queue.
     + Process the job (e.g., simulate printing, display job details).
4. **Completion**:
   * Once all jobs have been processed and the queue is empty, conclude the scheduling operation.

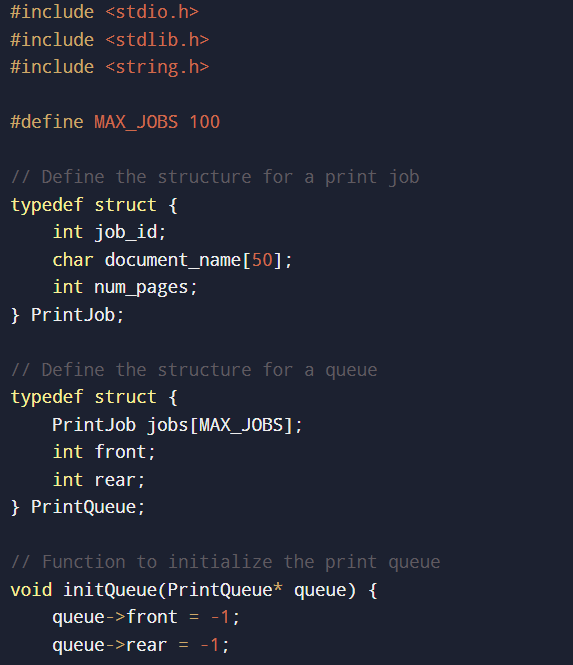
**Key Points**

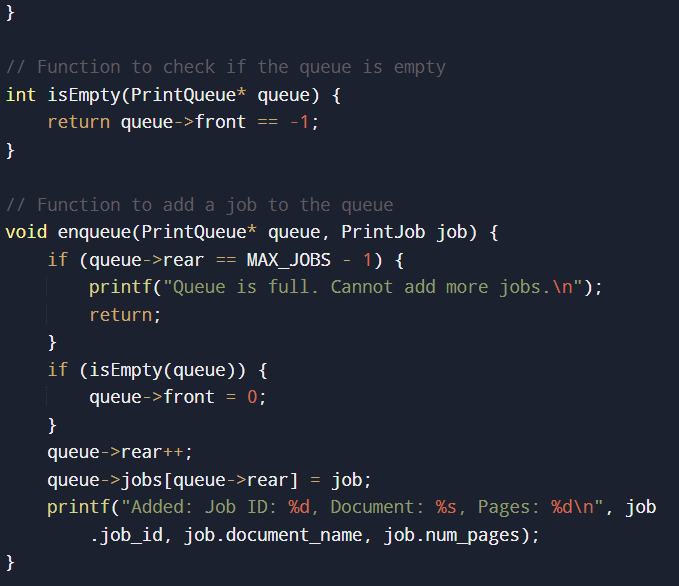
* **Order of Processing**: Jobs are processed in the exact order they were received.
* **Fairness**: Every job gets a turn based on its arrival time, ensuring no job is starved or skipped.
* **Simplicity**: The algorithm is straightforward and easy to implement, requiring minimal overhead.

**PSEUDOCODE REPRESENTATION**:

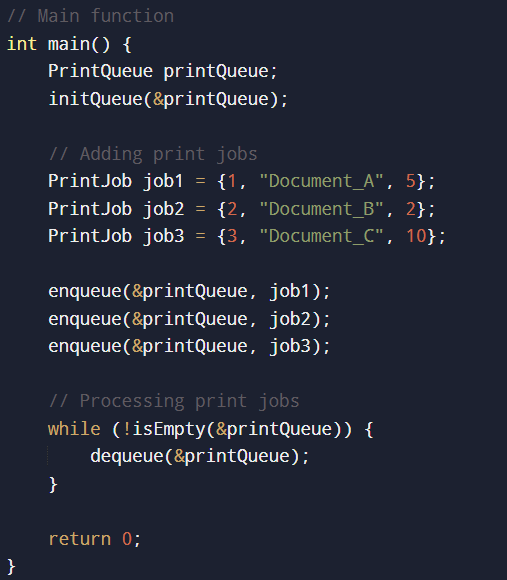


**COMPLETE CODE**

****

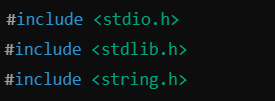






**EXPLANATION OF CODE**

Header Files:

****

stdio.h: Includes standard input/output functions.

stdlib.h: Includes functions for memory allocation and process control.

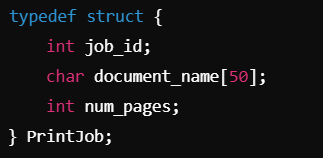
string.h: Includes string handling functions.

Constants:



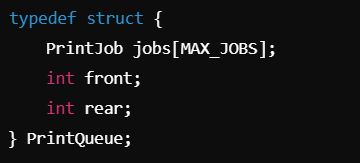
Defines a constant MAX\_JOBS to limit the number of print jobs that can be stored in the queue.

Structures:



Defines a PrintJob structure that contains:

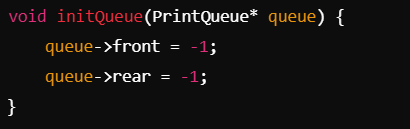
* job\_id: An integer representing the job's unique identifier.
* document\_name: A string for the name of the document.
* num\_pages: An integer indicating how many pages the job contains.



* Defines a PrintQueue structure that contains:
  + jobs: An array to hold the print jobs.
  + front: An index to track the front of the queue.
  + rear: An index to track the rear of the queue.

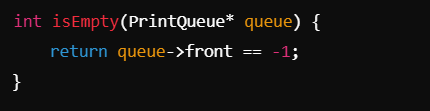
Function Definitions:

Initialize the Queue:



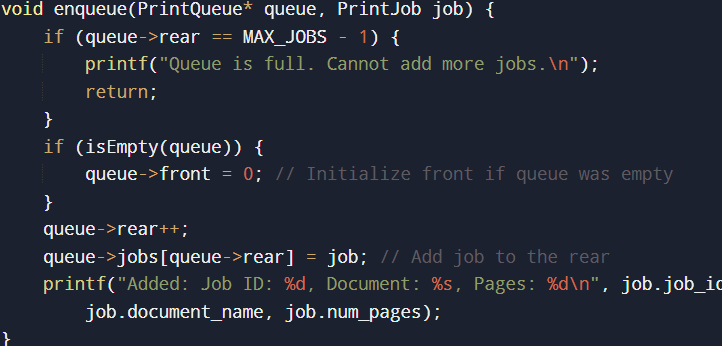
* Sets both front and rear to -1, indicating that the queue is empty.

Check if the Queue is Empty:



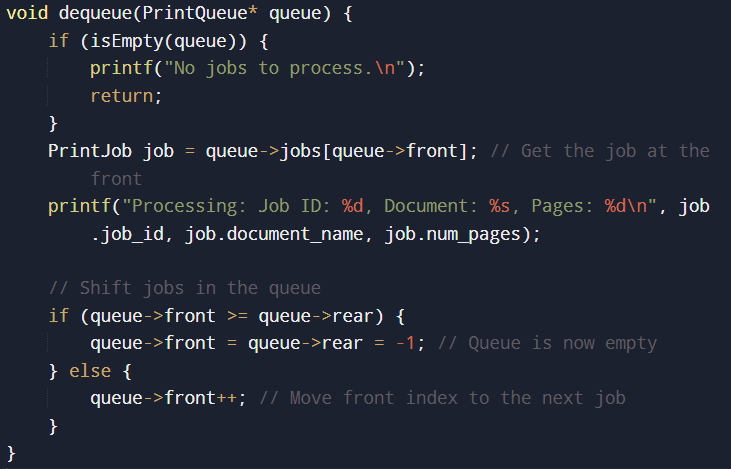
Returns true if the front index is -1, meaning there are no jobs in the queue.

Enqueue (Add a Job):



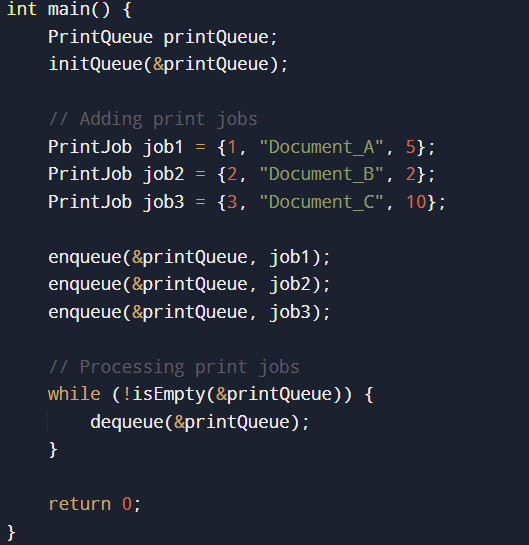
* Checks if the queue is full. If not, it updates the front index if the queue was previously empty and then adds the job at the rear of the queue.

Dequeue (Process a Job):



Checks if the queue is empty. If not, it retrieves the job at the front, prints its details, and then updates the front index. If processing the last job, it resets both front and rear to -1.

Main Function:



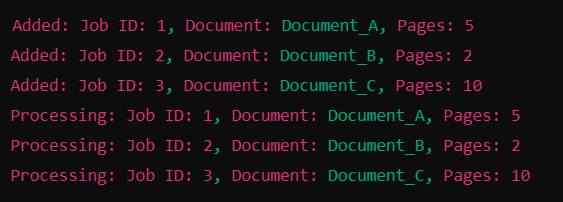
Initializes the PrintQueue.

Creates several PrintJob instances with sample data.

Enqueues the jobs into the queue using the enqueue function.

Processes the jobs in the queue one by one using a loop that continues until the queue is empty.

**OUTPUT & OBSERVATION**



**FIFO Behavior**:

* The output clearly demonstrates that the print jobs are processed in the order they were received, which is the core principle of FCFS scheduling.

**Simplicity and Efficiency**:

* The program is straightforward and easy to follow, making it suitable for educational purposes. It effectively illustrates how a queue can manage print jobs.

**Scalability**:

* The program can handle up to MAX\_JOBS (100 in this case). If more jobs need to be managed, the MAX\_JOBS constant can be adjusted.

**Potential Improvements**:

* The program currently does not handle error scenarios, such as memory allocation issues or invalid job data. Adding more robust error handling could improve reliability.
* It could also be extended to include features like job priorities or estimated printing times, enhancing its functionality for more complex scheduling needs.

**CONCLUSION**

The Print Job Scheduler implemented in C successfully demonstrates the use of a queue to manage print jobs using the First-In, First-Out (FIFO) scheduling algorithm. It effectively handles jobs in the order they arrive, ensuring fairness and simplicity. The program is straightforward and educational, showcasing basic data structures and algorithms. While it manages a fixed number of jobs efficiently, there is potential for enhancements, such as error handling and job prioritization. Overall, this project highlights the practical application of queues in real-world printing systems and provides a solid foundation for further exploration of scheduling algorithms.