Jason Luttrell

CSD 420T301

M1: Assignment

**The Role of Stacks and Queues in Data Flow and Process Scheduling**

Stacks and queues are fundamental data structures that play crucial roles in managing data flow and scheduling processes in computing systems. Their contrasting operational rules, Last-In-First-Out (LIFO) for stacks and First-In-First-Out (FIFO) for queues, make them suitable for different kinds of tasks (GeeksforGeeks, 2025). In particular they are suitable for tasks that require ordered processing. Understanding these structures is essential for efficient software design and system performance.

A stack operates on a LIFO basis, meaning the last element added is the first to be removed. This makes stacks ideal for managing nested or recursive operations. A prime example is the function call stack in programming languages. When a function is called, its context (such as variables and the return address) is pushed onto the stack. As nested functions are called, each context is stacked on top of the previous one. Once a function completes, the system pops its context off the stack and returns control to the previous function. This structure allows computers to efficiently track and return from nested function calls (Farney, 2018). Additionally, stacks are commonly used to implement undo functionality in applications, where each user action is recorded and can be reversed in reverse order.

Queues, by contrast, follow FIFO logic. This means that the first element added is the first to be processed. This structure is commonly used in scenarios where fairness and order are important. In operating systems, queues are used in process scheduling to manage CPU execution. When multiple processes need CPU time, they are placed in a queue. The process at the front gets executed first, followed by the next one, ensuring that all processes get a fair chance at execution (Programiz, n.d.). Queues are also essential in data streaming, such as managing network packets, buffering data, or handling background tasks asynchronously.

Both stacks and queues can be implemented using arrays or linked lists. Arrays offer faster access times but fixed size, while linked lists provide dynamic sizing at the cost of more complex memory handling. Circular queues and double-ended queues (deques) are variations that provide additional flexibility in data management.

The importance of these data structures is evident in many algorithmic processes. For example, stacks are critical in algorithms such as depth-first search, expression evaluation, and backtracking problems. Queues, in contrast, are used in breadth-first search algorithms and scenarios that require level-order traversal or parallel task management.

In conclusion, stacks and queues are more than academic concepts; they are foundational to real-world computing. Their inherent order-handling capabilities support essential functionalities such as process scheduling, function execution, and asynchronous task handling. As systems become increasingly complex, a strong understanding of these structures remains essential for developers and engineers working across all areas of computer science.

# References

Farney, R. M. (2018, May 23). *Breaking Down the Call Stack*. Retrieved from Medium: https://medium.com/@ryanfarney/breaking-down-the-call-stack-e68b5633fbad

GeeksforGeeks. (2025, Mar 1). *https://www.geeksforgeeks.org/dsa/stack-data-structure/*. Retrieved from GeeksforGeeks.org: https://www.geeksforgeeks.org/dsa/stack-data-structure/

Programiz. (n.d.). *Queue Data Structure*. Retrieved from Programiz: https://www.programiz.com/dsa/queue