# Chapter 4 Supplement: A Discussion on the Importance of Stacks and Queues

By now you well understand that larger complex data structures are made of smaller building block data structures. For example, multi-dimensional and circular arrays or vectors are made using linear arrays; lists are used to create sets and maps (which is called a dictionary in some languages and associated arrays in some other). However, as building block data structures, stacks and queues are unique in some sense.

For example, a set or a map that you create from a linked list provides additional features over their building block component that is the list. However, stacks and queues are strictly restricted forms of linked list (in rare cases, where you know the maximum number of elements you will put in them, you can use arrays for stacks and queues also). In other words, they do not provide any new features that a linked list does not already have. So it is a natural question why stacks and queues are considered fundamental data structures.

One reason for their importance is that scenarios where you need a linked list behave like a stack or queue is very frequent. For example, you need stacks for language parsing, expression evaluation, function calls, efficient graph traversals, etc. Queues are similarly important for resource scheduling, time sharing of CPU among candidate processes, message routing in network switches and routers, and efficient graph traversal.

Another central reason is that you often do not want to give access to the underlying list data structure that forms a stack or queue to the code that requested an insert or a removal from the stack/queue. If access to the list is provided then the code becomes more insecure. To consider this case, imagine direct access to the function call stack is possible. Then a program can manipulates the stack and jump from the current function to a much earlier function, instead of the immediate predecessor function that called it. Why this might be a problem? Well, this would not be a problem if all the functions in the stack belong to your own program. However, in real-world, library functions, operating system's service functions interleave with user program's functions in the stack. Unprotected jumps to those functions can crash the system.

A third reason to have stack and queues as restricted form of linked lists (or arrays) is that their restricted functions provide smaller code that can fit in very short memory or can even be implemented inside hardware devices directly. This feature is particularly important when you need fast response, for example, in routers and switches. In fact, array based stack and queue implementations are most suited in cases like this.