Stacks and queues in different programming languages

Introduction

A stack is a fundamental data structure that limits the way data is stored in an application. Namely, items must adhere to the entry or exit policy of last-in-first-out. Different languages have different implementations, though overall, the general result is the same. A queue is very similar to a stack; however, the order of entry is different, favoring a first-in-first-out policy.

In this reading, you will learn about some of the inherent differences in both stacks and queues across different programming languages.

Stacks

Stacks are an example of an abstract data type. In computer science, this means that there are some very important characteristics that need to be enforced when implementing it, but there are no actual built-in versions that you can just import. For stacks, the important principle is LIFO or last-in-first-out. The analogy for stacks can be to visualize a pile of plates on a washing board. As each plate is cleaned, it is placed on top of the previous one. For drying them, each plate must be taken from the top. So, the last plate added to the pile is dried first, and the first plate on the pile is dried last. A common usage for stacks is to keep your browser history. Each time you hit back, the previously visited page loads and, going forward, then reads it to the pile.

A common implementation of a stack in JavaScript is through using an array and making sure that it acts only in a way that a stack would act. Taking a data structure and using it to build another type of data structure is said to use a container adapter. So here, the array is the container, and the adaptation forces it to behave as a stack should. The important features of a stack are push, pop, peek and count. Push adds an item to the top of the stack. Using an array means that the count must always know where on the array the last item went. Equally, for the pop method, having a count for the array is important, as pop only returns the last item entered on the stack.

Queues

A queue, like a stack, is a linear data structure that retains the order in which things were entered. When one talks about a linear structure, it means that items are stored in the order in which they were added. Just like a stack, the queue has a strict implementation of how items are added and removed. While a stack implements a LIFO approach, queues work with a first-in-first-out (FIFO) approach. So, the first item that is added to the queue will be the first item that is removed. Like a customer queuing to make a purchase, queues look to implement a policy that places importance on the time of arrival. This has some real-world benefits when implementing things like CPU and disk scheduling, where it is important to deal with tasks as they arrive.

A queue is an abstract data type in Swift, which means that to use the functionality of a queue, you would first have to code up a structure that will act in that way. Two important terms associated with queues are enqueued (to add an item to the queue) and dequeue (to remove an item from the back of a list). The simplest approach for creating a queue is to use an array as the container adaptor. Because a queue only pops elements from the front of the queue, the look-up time will always be O(1), so any application built using this approach can expect very quick service. As with stacks, the available methods for a queue are kept minimal, with peak, pop, push and size being the most important ones.

Python implements several variant queue classes that are all synchronized. Synchronization is a computer science concept that means that all access to the data found there is managed so that the data structure can only be accessed by one process at a time. This is important when you think about cloud computing and having many different sources trying to look up and change data at the same time. On top of FIFO and LIFO, Python queue implementation includes a priority queue. A priority queue, as the name suggests, is about assigning importance to the elements found in the queue. So instead of a first-come-first-serve basis, service is based on importance. This is a bit like the VIP line at the concert.

Conclusion

Selecting the appropriate data structure for the appropriate task is a very important programming decision, as it can affect the entire project. You would select a stack when there is a need to keep track of the order in which an item is entered and to return them in a very specific way. A queue is used for maintaining order, but it will abide by different rules than a stack. Finally, if you want a data structure where you can assign importance, then a priority queue is worth looking at.