Diagram

Description automatically generated**Stacks and Queues**

**5.0 Stacks Introduction**

**Stacks**

* Def: Stacks are a type of data structures. Data is structured in a single line order. The line can be from top to bottom, left to right or right to left or even bottom to top (but that would make the bottom top and the top bottom)
* When a stack is removed this process is called pop. Similar to the pop process in python, when the last element is removed from a list if .pop() is used
* When a stack adds an element, this process is called push. Similar to the append process in python, when a new last element is added to the list if .append(<The To Be Added Item>) is used
* Stacks follow the standard relationship of LIFO -> Last in First Out
* Which in simple words means most recent, first out
* Stacks are typically not used for scheduling purposes. Because, if you continuously get orders on the stack, you will never get to the first inserted items
* The official names for **Adding** and **Removing** from a stack are **Push** and **Pop**, respectively

**Implement Stacks**

* Stacks can be implemented with arrays or linked lists and stacks in both arrays and linked lists act similarly
* With arrays:
  + We do not need to use circular arrays, because due to the stacks LIFO relationship the front and the back of an array that uses a stack will not change
  + Stacks will add and remove elements from the back of the array or the front of the array. However, the location will always be the same for the addition/subtraction
  + The main issue with arrays is that you might get an error if you try to pop or add elements once your array is empty or full, respectively.
  + If you try to pop from an empty stack you will get an error
* With linked lists:
  + when stacks are implemented, they impact the first node. With each addition of a node the node must be added to the front. Then, when we remove a node, we also remove it starting from the front.
  + The main benefits of linked lists is that it does not need a pre-defined size. Unlike arrays that require predefined sizes. It is more common to have stacks implemented in linked lists
  + If you try to pop from an empty stack, you will get an error

**5.1 Queues Introduction**

**Queues**

* Def: Queues are also a type of data structures and are similar to stacks. But the main difference between a stack and a queue is that it follows the FIFO relationship (First in First out).
* Therefore, when you add an item to a queue list you add it to the end of the list. And when you pop an item from a list you remove it from the start of the list.
* The pop and add features happen at two sides in queues. Stacks have the pop and add features occurring on only one side.
* Queues are great for scheduling tasks and CPUs use them to execute tasks in an order. Therefore, using queues does not allow any item to freeze in the queue because we always working on the oldest respectable item in the list
* The official names for **Adding** and **Removing** from a queue are **Enqueue** and **Dequeue**, respectively

**Implement Queue**

* Like Stacks you can use arrays or linked lists in queues
* With arrays
* The array must be circular, because due to the queue’s FIFO relationship the front of the back of the array will continuously be changing. We can just write the code in python and you can make it satisfy the conditions of a circular array
* We can use a fixed array but fixed array using FIFO will use significant amount of resources, because the while array must move each time, we make a change.
* Shape, circle

  Description automatically generatedBelow is a queue that uses circular array. Please note on how the elements move and change position due to enqueuing or dequeuing
* With linked lists
* We must use a doubly linked list with the tail pointer being activated. Also, regardless of the type of the linked list if it is singly or doubly linked list we will know the start point. The start point is where we will be adding new nodes to the linked list. However, for the end of the linked list, for us to remain aware of its location we need to use a doubly linked list and a tail pointer.
* The tail will let us know where the end of the doubly linked list is, which is what we will need to pop items
* The reason why it is a doubly linked list is because after we pop a node from the linked list. The tail might lose the exact location of the end of the list. But if we have all the nodes connected. Then we won’t lose any data and we can connect the tail to the new “last” node of the linked list

**5.2 Queues and Stacks Run Times**

* Queues and Stacks are not data structures within themselves. They are built off other data structures. Therefore, their run times depends on the data structures they are built on.
* The data structures they are built on are arrays or linked lists

**Queues**

* Both enqueue and dequeue must provide the array and linked list with an O(1) run time. Please note that the array must be circular, other wise you might have an error. The linked list must be a doubly linked list with a tail pointer. Also, the start pointer should also be available in the linked list

**Stack**

* Both pop and push must provide the array and linked list with an O(1) run time

**5.2 Queues and Stacks Real World Examples**

**Queues**

* Ex1: The undo and redo button in various application. The Ctrl z
* Reaching the end of the Ctrl z button meaning you click it and nothing happens, means you have filled up the queue
* The more work you do. You stop being able to go back to the same point each time you click Ctrl z. This is because the redos are being pushed, then popped off the queue. Due to queue’s size, which does not have enough memory to store infinity number of history tabs.
* Ex2: any list of jobs that are scheduled. For example, printing scheduling jobs

**Stack**

* Ex1: A bile of books. The last book we put in the bile is the first book we pick out