**Name:** Akash kafle

**Roll No:** 27

**GitHub ->**[Akash-kafle](https://github.com/Akash-kafle)

**Stack:**

* A stack is a linear data structure where all insertions and deletions are restricted to one end, called the top. It is also known as a Last-In-First-Out (LIFO) list because the last element inserted into a stack is the first element removed. Principal operations on a stack are push (adding an element into a stack) and pop( removing an element from a stack).

Few implementation of Stack are:

* **Expression Evaluation**: Convert and evaluate infix to post-fix expressions.
* **Reversing a List or String:** Reverse the order of elements in a list or characters in a string using a stack.
* **Balancing Symbols:** Check for balanced parentheses and other symbols.
* **Reverse Words in a Sentence**: Use a stack to reverse the order of words or characters in a sentence.

**Queue:**

* A Queue is a First-In-First-Out (FIFO) data structure where the first element inserted is the first element removed. Unlike stacks, insertions and deletions are made at different ends. New elements are added at one end, called the rear, and elements are removed from the other end, called the front. Principal operations on a queue are enqueue (adding an element into a queue) and dequeue (removing an element from a queue).

Few implementation of Queue are:

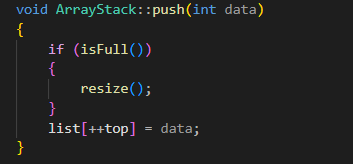
* **Web Server Requests**: Manage incoming HTTP requests in web servers.
* **I/O Buffering:** Queue input/output operations in operating systems.
* **CPU Task Management:** Queue tasks for the CPU in multitasking environments.
* **Email Queueing:** Queue outgoing emails for orderly sending.

1. **Implement Stack data structure using an array as well as a linked list.**

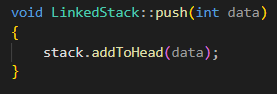
**(a) push(element):**

* Adds an element into the stack

**Array:**

****

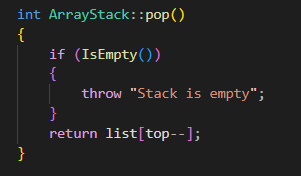
**Linked List:**

****

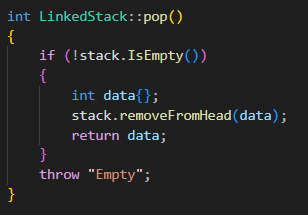
**(b) pop():**

* Removes an element from the stack

**Array:**

****

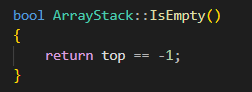
**Linked List:**



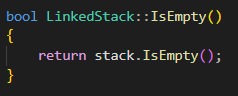
**(c) isEmpty():**

* Checks if the stack is empty

**Array:**

****

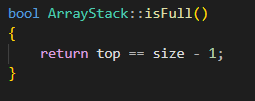
**Linked List:**



**(d) isFull():**

* Checks if the stack is full

**Array:**

****

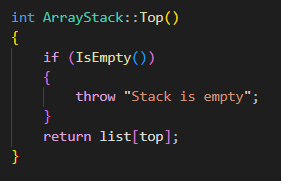
**Linked List:**

* Stack implementation using linked list is never full and is fully dynamic.

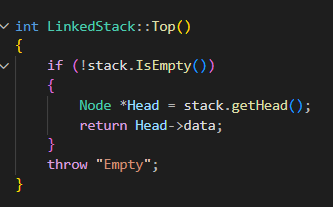
**(e) top():**

* Gives the element at the top

**Array:**

****

**Linked List:**

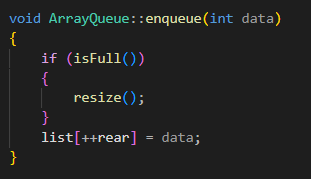


**2. Implement Queue data structure using an array as well as a linked list.**

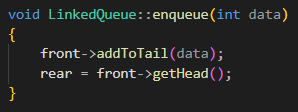
**(a) enqueue(element):**

* Adds an element into the queue

**Array:**

****

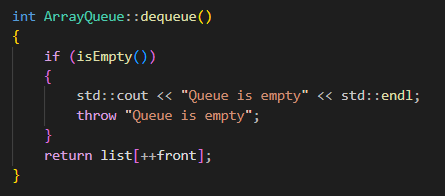
**Linked List:**

****

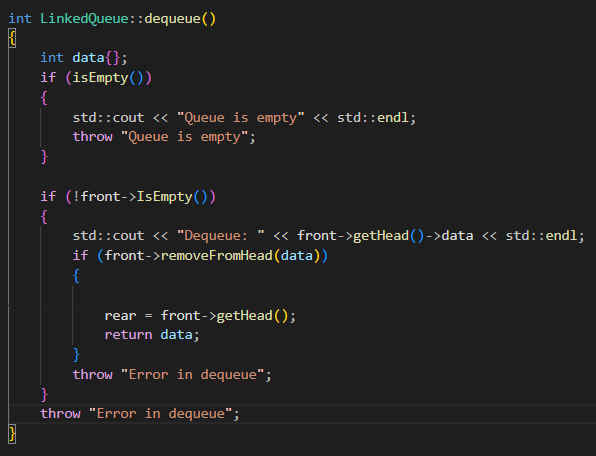
**(b) dequeue():**

* Removes an element from the queue

**Array:**



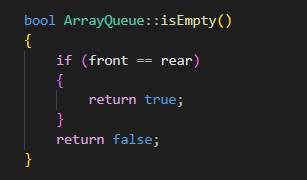
**Linked List:**



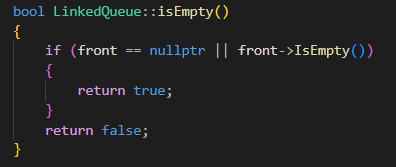
**(c) isEmpty():**

* Checks if the queue is empty

**Array:**

****

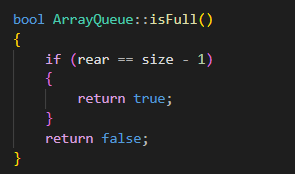
**Linked List:**



**(d) isFull():**

* Checks if the queue is full

**Array:**

****

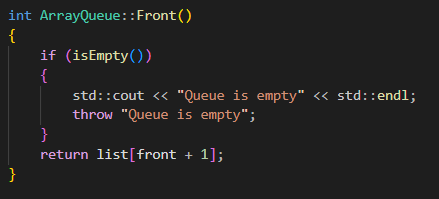
**Linked List:**

* Queue implemented from linked list is never full and is dynamic.

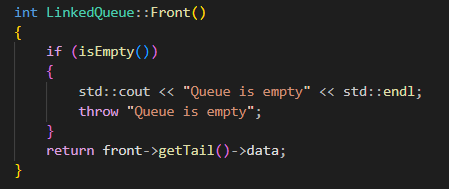
**(e) front()**:

* Gives the element at the front

**Array:**

****

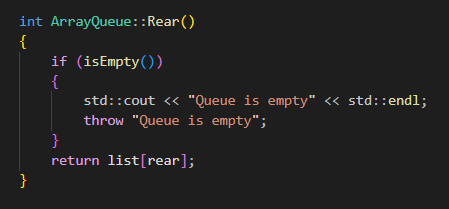
**Linked List:**

****

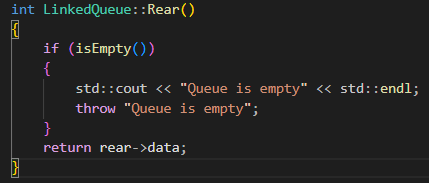
**(f) back():**

* Gives the element at the rear

**Array:**

****

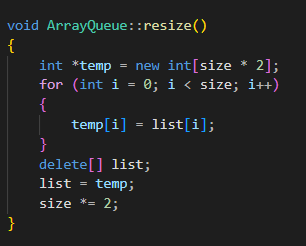
**Linked List:**

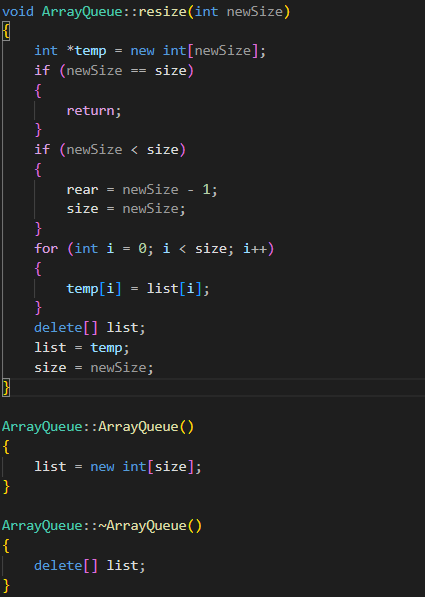
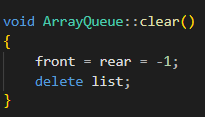
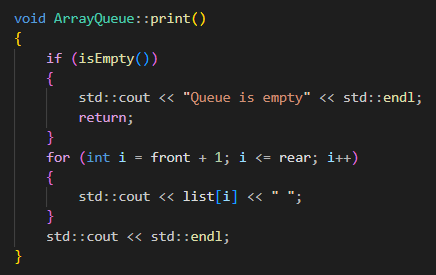


**(g) Other functions:**

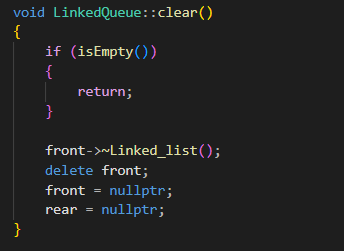
* These are additional functions used in the code respectively for “array” and “linked list” implementations.

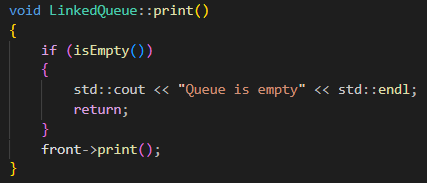
**Array:**



****

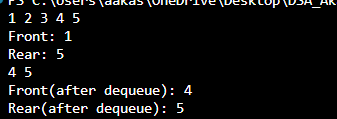
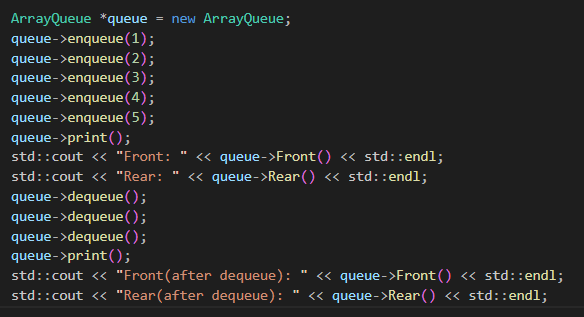
**Linked List:**

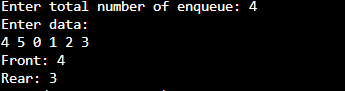
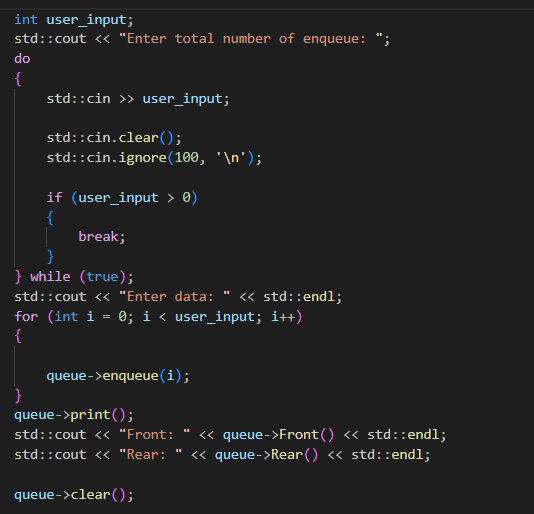
****

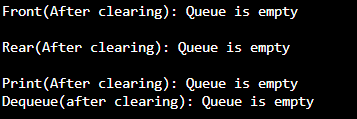
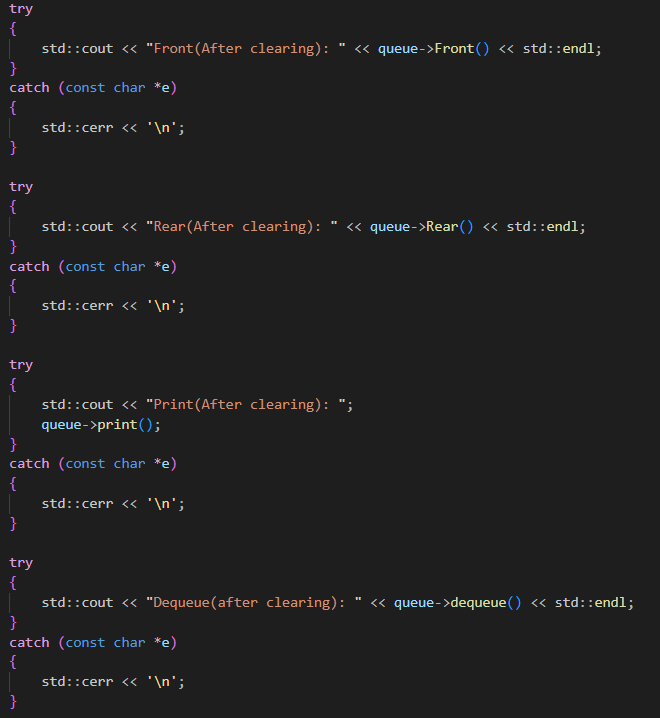
****

**#MAIN Functions used for all:**

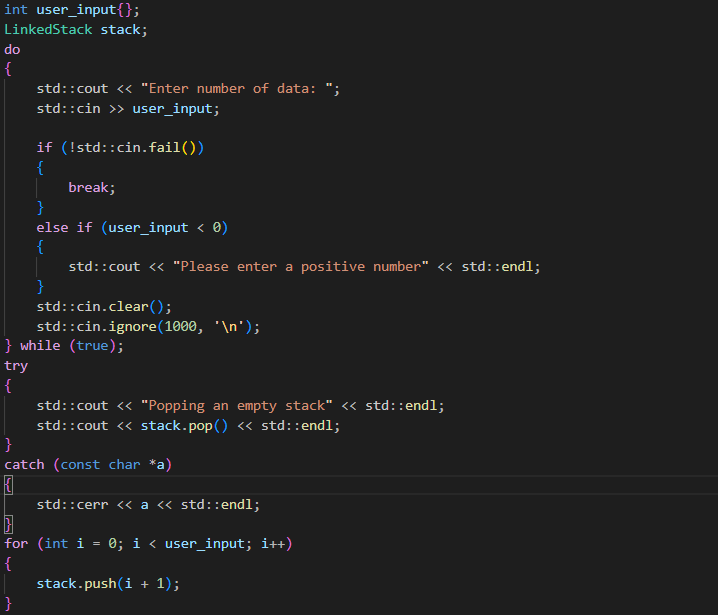
**Queue:**

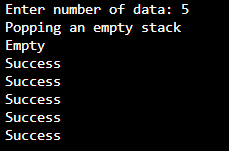
****

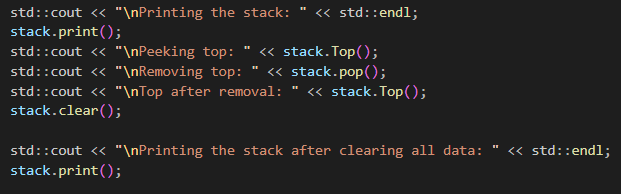
****

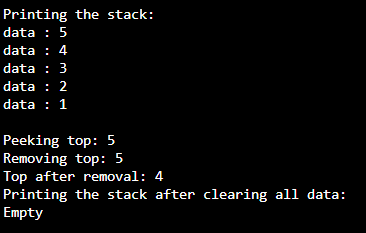
****

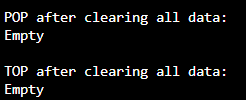
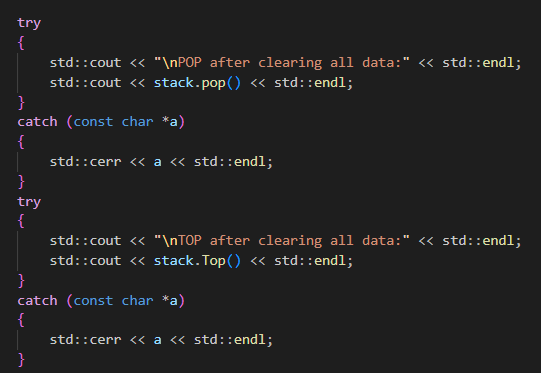
**Stack:**

****

****

****

****

****