**1.Implement Stack Using Queues:**

**2.Implement Queues Using Stacks:**

**3.Implementing Queue Using Linked List:**

I can do it.

**// A C program to demonstrate linked list based implementation of queue**

**#include <stdlib.h>**

**#include <stdio.h>**

**// A linked list (LL) node to store a queue entry**

**struct QNode**

**{**

**int key;**

**struct QNode \*next;**

**};**

**// The queue, front stores the front node of LL and rear stores ths**

**// last node of LL**

**struct Queue**

**{**

**struct QNode \*front, \*rear;**

**};**

**// A utility function to create a new linked list node.**

**struct QNode\* newNode(int k)**

**{**

**struct QNode \*temp = (struct QNode\*)malloc(sizeof(struct QNode));**

**temp->key = k;**

**temp->next = NULL;**

**return temp;**

**}**

**// A utility function to create an empty queue**

**struct Queue \*createQueue()**

**{**

**struct Queue \*q = (struct Queue\*)malloc(sizeof(struct Queue));**

**q->front = q->rear = NULL;**

**return q;**

**}**

**// The function to add a key k to q**

**void enQueue(struct Queue \*q, int k)**

**{**

**// Create a new LL node**

**struct QNode \*temp = newNode(k);**

**// If queue is empty, then new node is front and rear both**

**if (q->rear == NULL)**

**{**

**q->front = q->rear = temp;**

**return;**

**}**

**// Add the new node at the end of queue and change rear**

**q->rear->next = temp;**

**q->rear = temp;**

**}**

**// Function to remove a key from given queue q**

**struct QNode \*deQueue(struct Queue \*q)**

**{**

**// If queue is empty, return NULL.**

**if (q->front == NULL)**

**return NULL;**

**// Store previous front and move front one node ahead**

**struct QNode \*temp = q->front;**

**q->front = q->front->next;**

**// If front becomes NULL, then change rear also as NULL**

**if (q->front == NULL)**

**q->rear = NULL;**

**return temp;**

**}**

**// Driver Program to test anove functions**

**int main()**

**{**

**struct Queue \*q = createQueue();**

**enQueue(q, 10);**

**enQueue(q, 20);**

**deQueue(q);**

**deQueue(q);**

**enQueue(q, 30);**

**enQueue(q, 40);**

**enQueue(q, 50);**

**struct QNode \*n = deQueue(q);**

**if (n != NULL)**

**printf("Dequeued item is %d", n->key);**

**return 0;**

**}**

**4.Implementation of Deque using circular array:**

**// C++ implementation of De-queue using circular**

**// array**

**#include<iostream>**

**using namespace std;**

**// Maximum size of array or Dequeue**

**#define MAX 100**

**// A structure to represent a Deque**

**class Deque**

**{**

**int arr[MAX];**

**int front;**

**int rear;**

**int size;**

**public :**

**Deque(int size)**

**{**

**front = -1;**

**rear = 0;**

**this->size = size;**

**}**

**// Operations on Deque:**

**void insertfront(int key);**

**void insertrear(int key);**

**void deletefront();**

**void deleterear();**

**bool isFull();**

**bool isEmpty();**

**int getFront();**

**int getRear();**

**};**

**// Checks whether Deque is full or not.**

**bool Deque::isFull()**

**{**

**return ((front == 0 && rear == size-1)||**

**front == rear+1);**

**}**

**// Checks whether Deque is empty or not.**

**bool Deque::isEmpty ()**

**{**

**return (front == -1);**

**}**

**// Inserts an element at front**

**void Deque::insertfront(int key)**

**{**

**// check whether Deque if full or not**

**if (isFull())**

**{**

**cout << "Overflow\n" << endl;**

**return;**

**}**

**// If queue is initially empty**

**if (front == -1)**

**{**

**front = 0;**

**rear = 0;**

**}**

**// front is at first position of queue**

**else if (front == 0)**

**front = size - 1 ;**

**else // decrement front end by '1'**

**front = front-1;**

**// insert current element into Deque**

**arr[front] = key ;**

**}**

**// function to inset element at rear end**

**// of Deque.**

**void Deque ::insertrear(int key)**

**{**

**if (isFull())**

**{**

**cout << " Overflow\n " << endl;**

**return;**

**}**

**// If queue is initially empty**

**if (front == -1)**

**{**

**front = 0;**

**rear = 0;**

**}**

**// rear is at last position of queue**

**else if (rear == size-1)**

**rear = 0;**

**// increment rear end by '1'**

**else**

**rear = rear+1;**

**// insert current element into Deque**

**arr[rear] = key ;**

**}**

**// Deletes element at front end of Deque**

**void Deque ::deletefront()**

**{**

**// check whether Deque is empty or not**

**if (isEmpty())**

**{**

**cout << "Queue Underflow\n" << endl;**

**return ;**

**}**

**// Deque has only one element**

**if (front == rear)**

**{**

**front = -1;**

**rear = -1;**

**}**

**else**

**// back to initial position**

**if (front == size -1)**

**front = 0;**

**else // increment front by '1' to remove current**

**// front value from Deque**

**front = front+1;**

**}**

**// Delete element at rear end of Deque**

**void Deque::deleterear()**

**{**

**if (isEmpty())**

**{**

**cout << " Underflow\n" << endl ;**

**return ;**

**}**

**// Deque has only one element**

**if (front == rear)**

**{**

**front = -1;**

**rear = -1;**

**}**

**else if (rear == 0)**

**rear = size-1;**

**else**

**rear = rear-1;**

**}**

**// Returns front element of Deque**

**int Deque::getFront()**

**{**

**// check whether Deque is empty or not**

**if (isEmpty())**

**{**

**cout << " Underflow\n" << endl;**

**return -1 ;**

**}**

**return arr[front];**

**}**

**// function return rear element of Deque**

**int Deque::getRear()**

**{**

**// check whether Deque is empty or not**

**if(isEmpty() || rear < 0)**

**{**

**cout << " Underflow\n" << endl;**

**return -1 ;**

**}**

**return arr[rear];**

**}**

**// Driver program to test above function**

**int main()**

**{**

**Deque dq(5);**

**cout << "Insert element at rear end : 5 \n";**

**dq.insertrear(5);**

**cout << "insert element at rear end : 10 \n";**

**dq.insertrear(10);**

**cout << "get rear element " << " "**

**<< dq.getRear() << endl;**

**dq.deleterear();**

**cout << "After delete rear element new rear"**

**<< " become " << dq.getRear() << endl;**

**cout << "inserting element at front end \n";**

**dq.insertfront(15);**

**cout << "get front element " << " "**

**<< dq.getFront() << endl;**

**dq.deletefront();**

**cout << "After delete front element new "**

**<< "front become " << dq.getFront() << endl;**

**return 0;**

**}**

Now, check the condition for isFull

**bool Deque::isFull()**

**{**

**return ((front == 0 && rear == size-1)||**

**front == rear+1);**

**}**

Now, front==0 && rear==size-1 is normal condition that no elements get deleted and read end in which elements are added. Now, front==rear+1 this is a special condition since we are implementing dequeue using circular array.

**5.Circular Queue:**I can do that.

**6.Implement K Queues In An Array Of N:**

Following are the three extra arrays are used:

1) front[]: This is of size k and stores indexes of front elements in all queues.

2) rear[]: This is of size k and stores indexes of rear elements in all queues.

2) next[]: This is of size n and stores indexes of next item for all items in array arr[].

Here arr[] is actual array that stores k stacks.

Together with k queues, a stack of free slots in arr[] is also maintained. The top of this stack is stored in a variable ‘free’.

All entries in front[] are initialized as -1 to indicate that all queues are empty. All entries next[i] are initialized as i+1 because all slots are free initially and pointing to next slot. Top of free stack, ‘free’ is initialized as 0.

I can do the rest with clear thinking.

**7.Interleave the first half of the queue with second half**

Given a queue of integers of even length, rearrange the elements by interleaving the first half of the queue with the second half of the queue.

Following are the steps to solve the problem:

1.Push the first half elements of queue to stack.

2.Enqueue back the stack elements.

3.Dequeue the first half elements of the queue and Enqueue them back.

4.Again push the first half elements into the stack.

5.Interleave the elements of queue and stack.

(Remember, that it requires an auxiliary stack to perform the job)

**8.Sliding Window Maximum (Maximum of all subarrays of size k)**

We create a Dequeue, Qi of capacity k, that stores only useful elements of current window of k elements. An element is useful if it is in current window and is greater than all other elements on left side of it in current window. We process all array elements one by one and maintain Qi to contain useful elements of current window and these useful elements are maintained in sorted order. The element at front of the Qi is the largest and element at rear of Qi is the smallest of current window.

**#include <iostream>**

**#include <deque>**

**using namespace std;**

**// A Dequeue (Double ended queue) based method for printing maixmum element of**

**// all subarrays of size k**

**void printKMax(int arr[], int n, int k)**

**{**

**// Create a Double Ended Queue, Qi that will store indexes of array elements**

**// The queue will store indexes of useful elements in every window and it will**

**// maintain decreasing order of values from front to rear in Qi, i.e.,**

**// arr[Qi.front[]] to arr[Qi.rear()] are sorted in decreasing order**

**std::deque<int> Qi(k);**

**/\* Process first k (or first window) elements of array \*/**

**int i;**

**for (i = 0; i < k; ++i)**

**{**

**// For very element, the previous smaller elements are useless so**

**// remove them from Qi**

**while ( (!Qi.empty()) && arr[i] >= arr[Qi.back()])**

**Qi.pop\_back(); // Remove from rear**

**// Add new element at rear of queue**

**Qi.push\_back(i);**

**}**

**// Process rest of the elements, i.e., from arr[k] to arr[n-1]**

**for ( ; i < n; ++i)**

**{**

**// The element at the front of the queue is the largest element of**

**// previous window, so print it**

**cout << arr[Qi.front()] << " ";**

**// Remove the elements which are out of this window**

**while ( (!Qi.empty()) && Qi.front() <= i - k)**

**Qi.pop\_front(); // Remove from front of queue**

**// Remove all elements smaller than the currently**

**// being added element (remove useless elements)**

**while ( (!Qi.empty()) && arr[i] >= arr[Qi.back()])**

**Qi.pop\_back();**

**// Add current element at the rear of Qi**

**Qi.push\_back(i);**

**}**

**// Print the maximum element of last window**

**cout << arr[Qi.front()];**

**}**

**// Driver program to test above functions**

**int main()**

**{**

**int arr[] = {12, 1, 78, 90, 57, 89, 56};**

**int n = sizeof(arr)/sizeof(arr[0]);**

**int k = 3;**

**printKMax(arr, n, k);**

**return 0;**

**}**

Now,

**while ( (!Qi.empty()) && Qi.front() <= i - k)**

**{**

**Qi.pop\_front(); // Remove from front of queue**

**}**

Now, this step is very important. Because, in dequeue, we are storing indices of maximum elements. Now, if current index is I and I-stored index>k then it will not be a part of current and upcoming windows. Hence, delete it.

**9.Find the largest multiple of 3 | Set 1 (Using Queue)**

Before putting into selected queue based on remainder, we need to sort appropriately.

**10.First negative integer in every window of size k:**

Given an array and a positive integer k, find the first negative integer for each and every window(contiguous subarray) of size k. If a window does not contain a negative integer, then print 0 for that window.

I can do that.

**11.Sum of minimum and maximum elements of all subarrays of size k.**

I can do that.

**12.Find the first circular tour that visits all petrol pumps:**

Suppose there is a circle. There are n petrol pumps on that circle. You are given two sets of data.

1. The amount of petrol that every petrol pump has.

2. Distance from that petrol pump to the next petrol pump.

Calculate the first point from where a truck will be able to complete the circle (The truck will stop at each petrol pump and it has infinite capacity). Expected time complexity is O(n). Assume for 1 litre petrol, the truck can go 1 unit of distance.

For example, let there be 4 petrol pumps with amount of petrol and distance to next petrol pump value pairs as {4, 6}, {6, 5}, {7, 3} and {4, 5}. The first point from where truck can make a circular tour is 2nd petrol pump. Output should be “start = 1” (index of 2nd petrol pump).

We can use a Queue to store the current tour. We first enqueue first petrol pump to the queue, we keep enqueueing petrol pumps till we either complete the tour, or current amount of petrol becomes negative. If the amount becomes negative, then we keep dequeueing petrol pumps till the current amount becomes positive or queue becomes empty.

Instead of creating a separate queue, we use the given array itself as queue. We maintain two index variables start and end that represent rear and front of queue

**13.An Interesting Method to Generate Binary Numbers from 1 to n**

**A simple method is to run a loop from 1 to n, call decimal to binary inside the loop.**

1) Create an empty queue of strings

2) Enqueue the first binary number “1” to queue.

3) Now run a loop for generating and printing n binary numbers.

……a) Dequeue and Print the front of queue.

……b) Append “0” at the end of front item and enqueue it.

……c) Append “1” at the end of front item and enqueue it.

**14.Stack Permutations (Check if an array is stack permutation of other)**

.

The idea to do this is we will try to convert the input queue to output queue using a stack, if we are able to do so then the queue is permutable otherwise not.

Below is the step by step algorithm to do this:

Continuously pop elements from the input queue and check if it is equal to the top of output queue or not, if it is not equal to the top of output queue then we will push the element to stack.

Once we find an element in input queue such the top of input queue is equal to top of output queue,

we will pop a single element from both input and output queues, and compare the top of stack and top of output queue now. If top of both stack and output queue are equal then pop element from both stack and output queue. If not equal, go to step 1.

Repeat above two steps until the input queue becomes empty. **At the end if both of the input queue and stack are empty then the input queue is permutable otherwise not.**