**Queue:**

**#include <bits/stdc++.h>**

**using namespace std;**

**struct Queue {**

**int front, rear, capacity;**

**int\* queue;**

**Queue(int c)**

**{**

**front = rear = 0;**

**capacity = c;**

**queue = new int;**

**}**

**~Queue() { delete[] queue; }**

**// function to insert an element**

**// at the rear of the queue**

**void queueEnqueue(int data)**

**{**

**// check queue is full or not**

**if (capacity == rear) {**

**printf("\nQueue is full\n");**

**return;**

**}**

**// insert element at the rear**

**else {**

**queue[rear] = data;**

**rear++;**

**}**

**return;**

**}**

**// function to delete an element**

**// from the front of the queue**

**void queueDequeue()**

**{**

**// if queue is empty**

**if (front == rear) {**

**printf("\nQueue is empty\n");**

**return;**

**}**

**// shift all the elements from index 2 till rear**

**// to the left by one**

**else {**

**for (int i = 0; i < rear - 1; i++) {**

**queue[i] = queue[i + 1];**

**}**

**// decrement rear**

**rear--;**

**}**

**return;**

**}**

**// print queue elements**

**void queueDisplay()**

**{**

**int i;**

**if (front == rear) {**

**printf("\nQueue is Empty\n");**

**return;**

**}**

**// traverse front to rear and print elements**

**for (i = front; i < rear; i++) {**

**printf(" %d <-- ", queue[i]);**

**}**

**return;**

**}**

**// print front of queue**

**void queueFront()**

**{**

**if (front == rear) {**

**printf("\nQueue is Empty\n");**

**return;**

**}**

**printf("\nFront Element is: %d", queue[front]);**

**return;**

**}**

**};**

**// Driver code**

**int main(void)**

**{**

**// Create a queue of capacity 4**

**Queue q(4);**

**// print Queue elements**

**q.queueDisplay();**

**// inserting elements in the queue**

**q.queueEnqueue(20);**

**q.queueEnqueue(30);**

**q.queueEnqueue(40);**

**q.queueEnqueue(50);**

**// print Queue elements**

**q.queueDisplay();**

**// insert element in the queue**

**q.queueEnqueue(60);**

**// print Queue elements**

**q.queueDisplay();**

**q.queueDequeue();**

**q.queueDequeue();**

**printf("\n\nafter two node deletion\n\n");**

**// print Queue elements**

**q.queueDisplay();**

**// print front of the queue**

**q.queueFront();**

**return 0;**

**}**

Queue Linked List:

#include <bits/stdc++.h>

using namespace std;

struct QNode {

int data;

QNode\* next;

QNode(int d)

{

data = d;

next = NULL;

}

};

struct Queue {

QNode front, \*rear;

Queue()

{

front = rear = NULL;

}

void enQueue(int x)

{

QNode temp = new QNode(x);

if (rear == NULL) {

front = rear = temp;

return;

}

rear->next = temp;

rear = temp;

}

void deQueue()

{

if (front == NULL)

return;

QNode\* temp = front;

front = front->next;

if (front == NULL)

rear = NULL;

delete (temp);

}

};

int main()

{

Queue q;

q.enQueue(10);

q.enQueue(20);

q.deQueue();

q.deQueue();

q.enQueue(30);

q.enQueue(40);

q.enQueue(50);

q.deQueue();

cout << "Queue Front : " << (q.front)->data << endl;

cout << "Queue Rear : " << (q.rear)->data;

}

Queue Program using class:

#include <bits/stdc++.h>

using namespace std;

class Queue {

public:

int front, rear, size;

unsigned capacity;

int\* array;

};

Queue\* createQueue(unsigned capacity)

{

Queue\* queue = new Queue();

queue->capacity = capacity;

queue->front = queue->size = 0;

queue->rear = capacity - 1;

queue->array = new int[(

queue->capacity \* sizeof(int))];

return queue;

}

int isFull(Queue\* queue)

{

return (queue->size == queue->capacity);

}

int isEmpty(Queue\* queue)

{

return (queue->size == 0);

}

void enqueue(Queue\* queue, int item)

{

if (isFull(queue))

return;

queue->rear = (queue->rear + 1)

% queue->capacity;

queue->array[queue->rear] = item;

queue->size = queue->size + 1;

cout << item << " enqueued to queue\n";

}

int dequeue(Queue\* queue)

{

if (isEmpty(queue))

return INT\_MIN;

int item = queue->array[queue->front];

queue->front = (queue->front + 1)

% queue->capacity;

queue->size = queue->size - 1;

return item;

}

int front(Queue\* queue)

{

if (isEmpty(queue))

return INT\_MIN;

return queue->array[queue->front];

}

int rear(Queue\* queue)

{

if (isEmpty(queue))

return INT\_MIN;

return queue->array[queue->rear];

}

int main()

{

Queue\* queue = createQueue(1000);

enqueue(queue, 10);

enqueue(queue, 20);

enqueue(queue, 30);

enqueue(queue, 40);

cout << dequeue(queue)

<< " dequeued from queue\n";

cout << "Front item is "

<< front(queue) << endl;

cout << "Rear item is "

<< rear(queue) << endl;

return 0;

}

Double-ended queue:

#include <iostream>

#include <cmath>

#include <bits/stdc++.h>

#include <climits>

#include <deque>

using namespace std;

int main()

{

deque<int> dq = {10, 20, 30};

dq.push\_front(5);

dq.push\_back(50);

for(auto x: dq)

cout << x << " ";

cout << dq.front() << " " << dq.back();

return 0;

}

Double Ended Queue Operations:

#include <iostream>

#include <cmath>

#include <bits/stdc++.h>

#include <climits>

#include <deque>

using namespace std;

int main()

{

deque<int> dq = {10, 15, 30, 5, 12};

auto it = dq.begin();

it++;

dq.insert(it, 20);

dq.pop\_front();

dq.pop\_back();

cout<<dq.size();

return 0;

}

Double-ended Queue operations 2:

#include <iostream>

#include <cmath>

#include <bits/stdc++.h>

#include <climits>

#include <deque>

using namespace std;

int main()

{

deque<int> dq {10, 20, 5, 30};

auto it = dq.begin();

it = dq.insert(it, 7);

it = dq.insert(it, 2, 3);

it = dq.erase(it + 1);

cout << (\*it) << endl;

for(int i = 0; i < dq.size(); i++)

cout<<dq[i]<<" ";

return 0;

}