Riphah International University
I-14 Main Campus
Faculty of Computing

Class:	Fall-2024	**Subject:**	Data Structures & Algorithms
Course Code:	CS 2124	**Lab Instructor:**	Zeeshan Ali

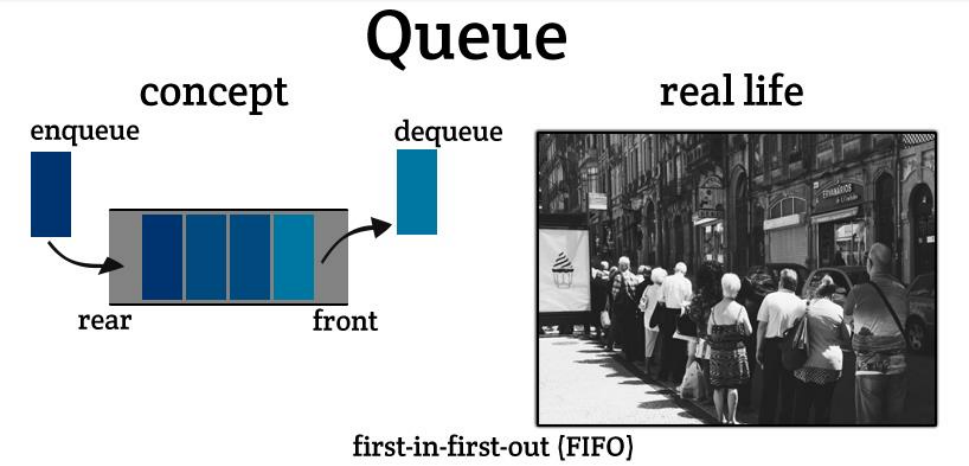
Learning Objective:

- What is Queue?
- Queue in C++.
- Operations of Queue
- Use of Queue
- Implementation of Queue using Link list in C++.
- Tasks

Queue:

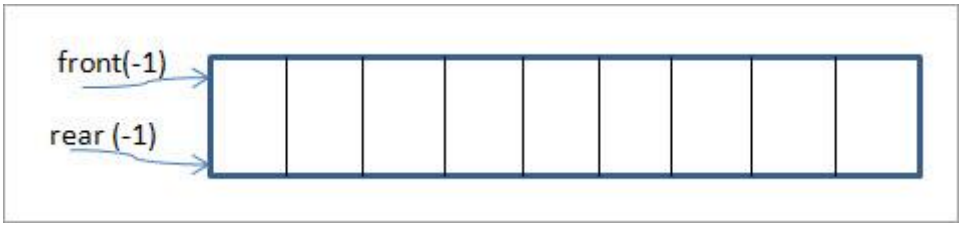
The queue is a basic data structure just like a stack. In contrast to stack that uses the LIFO approach, queue uses the FIFO (first in, first out) approach. With this approach, the first item that is added to the queue is the first item to be removed from the queue. Just like Stack, the queue is also a linear data structure.

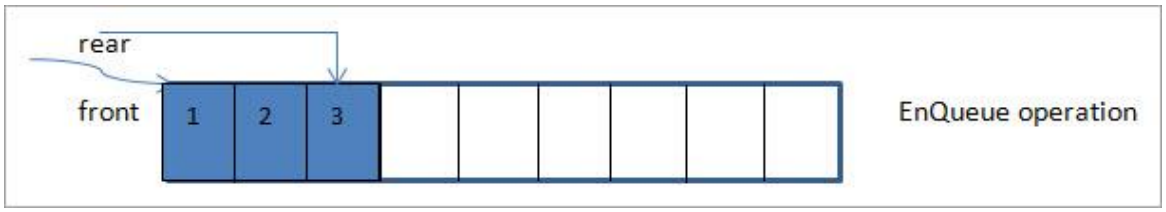
In a real-world analogy, we can imagine a bus queue where the passengers wait for the bus in a queue or a line. The first passenger in the line enters the bus first as that passenger happens to be the one who had come first.

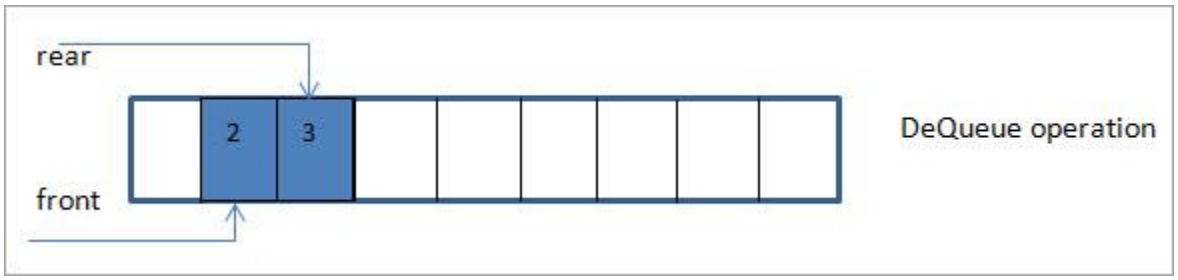


Queue in C++

In software terms, the queue can be viewed as a set or collection of elements as shown below. The elements are arranged linearly.







Operations of Queue

- **EnQueue:** Adds an item to the queue. Addition of an item to the queue is always done at the rear of the queue.
- **DeQueue:** Removes an item from the queue. An item is removed or de-queued always from the front of the queue.

Use of Queue

- CPU scheduling
- Synchronization
- Handling of interrupts in real-time systems

Implementation of Queue using link list in C++.

```cpp
#include<iostream>
using namespace std;
struct Node
{
    int data;
    Node *next ;
};
class Queue
{
    Node *front, *rear;
public:
    Queue()
    {
        front = rear = NULL;  //   Initially
    }
    void Enqueue(int data)    // for insertion from rear
    {
        Node *newnode;
        newnode = new Node;
        newnode->data = data;
        newnode->next = NULL;

        if(front == NULL)
            front = rear = newnode;
        else
        {
            rear->next = newnode;
            rear = newnode;
        }
    }
```

```
31      void Dequeue()            // for deletion from front
32      {
33          Node *temp;
34          if(front == NULL)
35              cout<<"Queue is Empty";
36          else
37          {
38              temp= front;
39              front = front->next;
40              delete temp;
41          }
42      }
43      void display()
44      {
45          Node *temp;
46          temp= front;
47          while(temp!=NULL) // (temp!= rear->next)
48          {
49              cout<<temp->data<<"\t";
50              temp = temp->next;
51          }
52          cout<<endl;
53      }
54  };
55  int main()
56  {
57      Queue Q1;
58      Q1.Enqueue(10);
59      Q1.Enqueue(20);
60      Q1.Enqueue(30);
61      Q1.Enqueue(40);
62      cout<<"Queue after Enqueue :"<<endl;
63      Q1.display();
64      Q1.Dequeue();
65      cout<<"Queue after Dequeue :"<<endl;;
66      Q1.display();
67  }
```

Output

```
Queue after Enqueue :
10  20  30  40
Queue after Dequeue :
20  30  40
```

Tasks

1. Modify the Queue to Handle Overflow
2. Count Elements in the Queue
3. Implement a method to clear the entire queue.