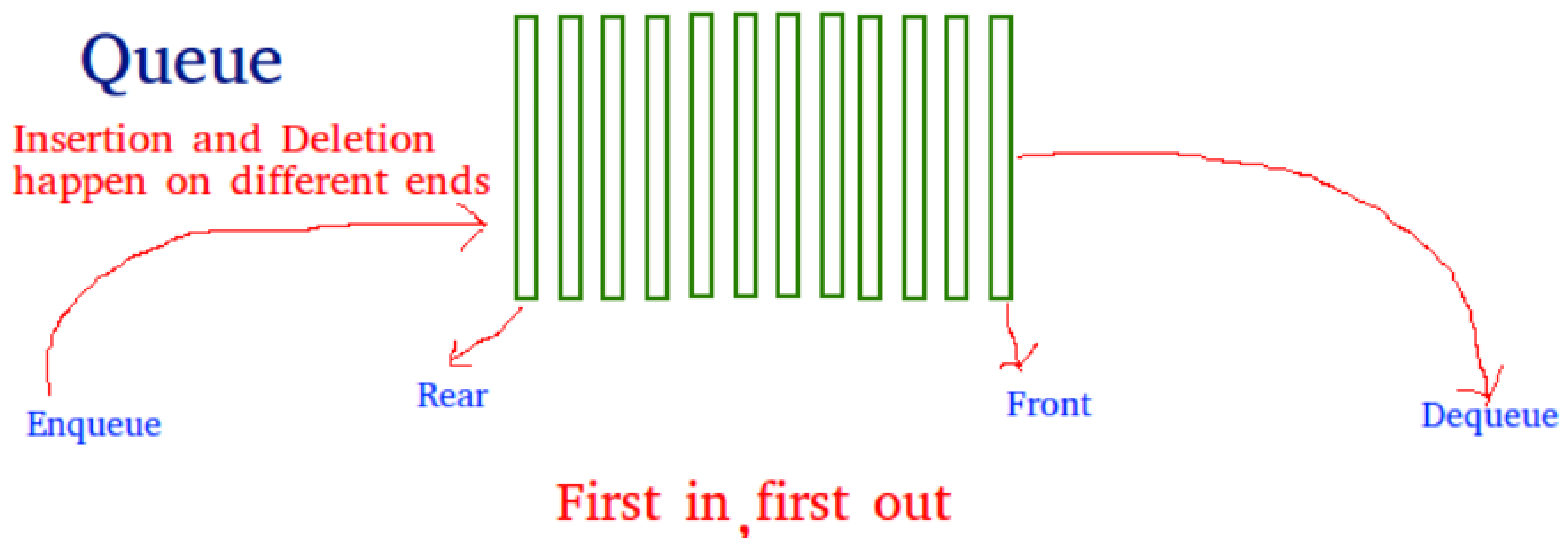
Queue in C++ STL

In general, Queue is a linear data structure where elements are inserted and deleted in the FIFO order. FIFO stands for First In First Out, which means that the first element to be inserted in the queue will be the first one to be removed.

C++ STL has an in-built container adaptor that implements the queue data structure internally. We can directly use this container to implement queues in our programs and perform a number of operations available.

Insertion and Deletion in a queue are done from opposite ends. Insertions in a queue are done at the rear end whereas deletions are done at the front.

To understand how a queue works, consider a queue at a ticket counter or bus stop in real life. Persons who came first are served first and the persons coming after the current person waits after them in the queue.



Example:

```
Let's say we inserted 4 items, {10, 20, 30}
in a queue:

The queue will look like:
++++++++
+  30  +
++++++++
+  20  +
++++++++
+  10  +
++++++++

On deleting element 10, queue will be:
++++++++
+  30  +
++++++++
+  20  +
++++++++

Again, on deleting 20, the queue will be:
++++++++
+  30  +
++++++++

Now, if we insert an element 40, it will
be inserted again at the rear end:
++++++++
+  40  +
++++++++
+  30  +
++++++++
```

Syntax:

```
queue<data_type> queue_name;
```

Note: The queue class is available in the header file . Therefore, one must include this header file in programs while using the queue class of C++ STL.

Some important functions of Queue in C++ STL: Below is a list of some of the most commonly used functions which are available with Queue in C++ STL.

The functions associated with Queue are:

- **push(g)**: The push function adds an element 'g' passed as parameter to it at the rear end of the queue.
- **pop()**: The pop() function deletes the front element of the queue.
- **front()**: This function returns a reference to the front element of the queue.
- **back()**: This function returns a reference to the last element of the queue.
- **size()**: The size() function returns the size of the queue, i.e., the number of elements present in the Queue.
- **empty()**: This function is used to check whether the queue is empty. If the Queue is empty it will return True otherwise it will return False.

Below program illustrate the above-discussed functions of Queue in C++ STL:

```cpp

// C++ program to illustrate Queue in STL

#include<iostream>
#include<queue>

using namespace std;

int main()
{
    // Creating a Queue using STL
    queue<int> q;

    // The push function inserts elements
    // at the end of Queue
    q.push(10);
    q.push(20);
    q.push(30);

    // After above operations, Queue will
    // look like:
    //       ******
    //       * 30 * <- Queue's rear end
    //       * 20 *
    //       * 10 * <- Queue's front
    //       ******

    // This will print the current size
    // of queue which is 3
    cout<<q.size()<<endl;
```

Run

Output:

```
3
10 30
20 30
```

Queue Traversal

We cannot access queue elements randomly as they are not stored at contigous memory locations. Hence, we cannot use loops or iterators to traverse queues.

However, in some cases, if we want to remove a few elements from a queue or empty a queue we can do that by using some built-in functions we discussed above.

The idea is:

- While, queue is not empty, i.e., there are some elements in the queue.
  1. Print the front element.

  2. Pop the front element.

  Repeat the above 2 steps until queue is not empty.

Below program illustrate traversal of the Queue:

```
// C++ program to illustrate Queue traversal

#include<iostream>
#include<queue>

using namespace std;

int main()
{
    // Creating a Queue using STL
    queue<int> q;

    // The push function inserts elements
    // at the rear end of Queue
    q.push(10);
    q.push(20);
    q.push(30);

    // After above operations, queue will
    // look like:
    //      ******
    //      * 30 *  <- Queue's rear end
    //      * 20 *
    //      * 10 *  <- Queue's front
    //      ******

    // While Queue is not empty
    while(!q.empty())
    {
```

Run

Output:

```
10 30
20 30
30 30
```

Time Complexities and Internal Working

All of the above six functions we discussed works in constant time complexity.

```
push()    -----
pop()     -----  \
front()   -----    ---  O(1) Time Complexity.
back()
size()    -----  /
empty()   -----
```

Container Adaptor: In C++ STL, container adaptor is a term used for containers that can be built upon other containers. That is, they can be implemented using other containers already available in C++ STL.

The **queue** is a container adaptor and can be implemented using any other container which supports below operations:

- Inserting elements to the end.
- Deleting elements from the front.
- Returning size of the container.
- Returning element from the end.
- Returning element from the front.
- Checking if the container is empty.

Note: In C++ STL there are two containers **list** and **deque** which supports the above-mentioned functionalities in O(1) time complexity and hence can be used to implement Queue.