Queue

This data structure works on the FIFO technique, where FIFO stands for First In First Out. The element which was first inserted will be extracted at the first and so on. There is an element called as 'front' which is the element at the front most position or say the first position, also there is an element called as 'rear' which is the element at the last position. In normal queues insertion of elements take at the rear end and the deletion is done from the front.

Syntax

template<class T, class Container = deque<T> > class queue;

Queues in the application areas are implied as the container adaptors.

The containers should have a support for the following list of operations:

empty

size

push_back

pop_front

front

back

Template Parameters

T: The argument specifies the type of the element which the container adaptor will be holding.

Container: The argument specifies an internal object of container where the elements of the queues are held.

Member Types

Given below is a list of the queue member types with a short description of the same.

Member Types	Description
value_type	Element type is specified.
container_type	Underlying container type is specified.
size_type	It specifies the size range of the elements.
reference	It is a reference type of a container.
const_reference	It is a reference type of a constant container.

Functions

With the help of functions, an object or variable can be played with in the field of programming. Queues provide a large number of functions that can be used or embedded in the programs. A list of the same is given below:

Function	Description
(constructor)	The function is used for the construction of a queue container.
empty	The function is used to test for the emptiness of a queue. If the queue is empty the function returns true else false.
size	The function returns the size of the queue container, which is a measure of the number of elements stored in the queue.
front	The function is used to access the front element of the queue. The element plays a very important role as all the deletion operations are performed at the front element.
back	The function is used to access the rear element of the queue. The element plays a very important role as all the insertion operations are performed at the rear element.
push	The function is used for the insertion of a new element at the rear end of the queue.
pop	The function is used for the deletion of element; the element in the queue is deleted from the front end.
emplace	The function is used for insertion of new elements in the queue above the current rear element.
swap	The function is used for interchanging the contents of two containers in reference.
relational operators	The non member function specifies the relational operators that are needed for the queues.
uses allocator<queue>	As the name suggests the non member function uses the allocator for the queues.

Example: A simple program to show the use of basic queue functions.

```
#include <iostream>
#include <queue>
using namespace std;
void showsg(queue <int> sg)
{
```

```
    queue <int> ss = sg;
    while (!ss.empty())
    {
        cout << '\t' << ss.front();
        ss.pop();
    }
    cout << '\n';
}

int main()
{
    queue <int> fquiz;
    fquiz.push(10);
    fquiz.push(20);
    fquiz.push(30);

    cout << "The queue fquiz is : ";
    showsg(fquiz);

    cout << "\nfquiz.size() : " << fquiz.size();
    cout << "\nfquiz.front() : " << fquiz.front();
    cout << "\nfquiz.back() : " << fquiz.back();

    cout << "\nfquiz.pop() : ";
    fquiz.pop();
    showsg(fquiz);

    return 0;
}
```

Output:

```
The queue fquiz is : 10     20      30
fquiz.size() : 3
fquiz.front() : 10
fquiz.back() : 30
fquiz.pop() : 20      30
```