Queue

Like *Stack* data structure, ***Queue*** is also a linear data structure that follows a particular order in which the operations are performed. The order is **F**irst **I**n **F**irst **O**ut (FIFO), which means that the element that is inserted first in the queue will be the first one to be removed from the queue. A good example of queue is any queue of consumers for a resource where the consumer who came first is served first.

The difference between stacks and queues is in removing. In a stack, we remove the most recently added item; whereas, in a queue, we remove the least recently added item.

Operations on Queue: Mainly the following four basic operations are performed on queue:

- **Enqueue:** Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
- **Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.
- **Front:** Get the front item from queue.
- **Rear:** Get the last item from queue.

Array implementation Of Queue: For implementing a *queue*, we need to keep track of two indices - front and rear. We enqueue an item at the rear and dequeue an item from the front. If we simply increment front and rear indices, then there may be problems, the front may reach the end of the array. The solution to this problem is to increase front and rear in a circular manner.

Consider that an Array of size **N** is taken to implement a queue. Initially, the size of the queue will be zero(0). The total capacity of the queue will be the size of the array i.e. N. Now initially, the index *front* will be equal to 0, and *rear* will be equal to N-1. Every time an item is inserted, so the index *rear* will increment by one, hence increment it as: **rear = (rear + 1)%N** and everytime an item is removed, so the front index will shift to right by 1 place, hence increment it as: **front = (front + 1)%N**.

Time Complexity: Time complexity of all operations such as enqueue(), dequeue(), isFull(), isEmpty(), front(), and rear() is O(1). There is no loop in any of the operations.

Applications of Queue: Queue is used when things don't have to be processed immediatly, but have to be processed in **F**irst **I**n**F**irst **O**ut order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios:

1. When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

Queue in C++ STL

The Standard template Library in C++ offers a built-in implementation of the Queue data structure for simpler and easy use. The STL implementation of queue data structure implements all basic operations on queue such as enque(), deque(), clear() etc.

Syntax:

```
queue< data_type > queue_name;

where,
data_type is the type of element to be stored
in the queue.
queue_name is the name of the queue data structure.
```

The functions supported by std::queue are :

- **empty()** – Returns whether the queue is empty.
- **size()** – Returns the size of the queue.
- **swap()**: Exchange the contents of two queues but the queues must be of same type, although sizes may differ.
- **emplace()**: Insert a new element into the queue container, the new element is added to the end of the queue.
- **front() and back()**: front() function returns a reference to the first element of the queue. back() function returns a reference to the last element of the queue.
- **push(g) and pop()**: The push() function adds the element 'g' at the end of the queue. The pop() function deletes the first element of the queue.

Implementation of Queue Using Array

Circular Array :

```
//using circular array

#include<iostream>
#include<queue>
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 INT32_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 INT32_MIN;
    return queue->array[queue->front];
}

int rear(Queue *queue)
{
    if(isEmpty(queue))
        return INT32_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;
}
```

OUTPUT :

10 enqueued to queue

20 enqueued to queue

30 enqueued to queue

40 enqueued to queue

10 dequeued from queue

Front item is 20

Rear item is 40

Implementation of Queue Using Linked List:

```
#include<iostream>
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;

    return 0;
}
```

OUTPUT :

Queue Front : 40

Queue Rear : 50

Queue in C++ STL

Push, front, back, pop :

```cpp
#include<iostream>
#include<queue>
using namespace std;

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

    cout<<q.front()<<" "<<q.back()<<endl;

    q.pop();

    cout<<q.front()<<" "<<q.back()<<endl;


    return 0;
}
```

OUTPUT :

10 30

20 30

Queue in C++ STL

Push, front, back, pop, empty :

```cpp
#include<iostream>
#include<queue>
using namespace std;

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

    while (q.empty()==false)
    {
        cout<<q.front()<<" "<<q.back()<<endl;

        q.pop();
    }

    return 0;

}
```

OUTPUT :

10 30

20 30

30 30

Queue in C++ STL

size :

```cpp
#include<iostream>
#include<queue>
using namespace std;

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

    cout<<q.size();

    return 0;
}
```

OUTPUT :

3

Implementation of stack using Queue

```cpp
#include<iostream>
#include<queue>
using namespace std;

struct Stack
{
    queue<int>q1,q2;
    int curr_size;

    public:
    Stack()
    {
        curr_size=0;
    }

    void push(int x)
    {
        curr_size++;

        q2.push(x);

        while (!q1.empty())
        {
            q2.push(q1.front());
            q1.pop();
        }

        queue<int> q=q1;
        q1=q2;
        q2=q;
    }

    void pop()
    {
        if(q1.empty())
            return;
        q1.pop();
        curr_size--;
    }

    int top()
    {
        if(q1.empty())
            return -1;
        return q1.front();
    }
```

```
    int size()
    {
        return curr_size;
    }
};

int main()
{
    Stack s;
    s.push(10);
    s.push(5);
    s.push(15);
    s.push(20);

    cout<<"current size: "<<s.size()<<endl;
    cout<<s.top()<<endl;
    s.pop();
    cout<<s.top()<<endl;
    s.pop();
    cout<<s.top()<<endl;

    cout<<"current size: "<<s.size()<<endl;
    return 0;
}
```

OUTPUT :

current size: 4

20

15

5

current size: 2

Reversing a Queue

Iterative :

```
#include<iostream>
#include<queue>
#include<stack>
using namespace std;

void print(queue<int> &Queue)
{
    while (!Queue.empty())
    {
        cout<<Queue.front()<<" ";
        Queue.pop();
    }

}

void reverseQueue(queue<int> &Queue)
{
    stack<int> Stack;
    while(!Queue.empty())
    {
        Stack.push(Queue.front());
        Queue.pop();
    }

    while (!Stack.empty())
    {
        Queue.push(Stack.top());
        Stack.pop();
    }

}


int main()
{
    queue<int> q;
    q.push(12);
    q.push(5);
    q.push(15);
    q.push(20);

    reverseQueue(q);
    print(q);
}
```

OUTPUT : 20 15 5 12

Recursive for reverse a Queue :

```
#include<iostream>
#include<queue>
using namespace std;

void print(queue<int> &Queue)
{
    while (!Queue.empty())
    {
        cout<<Queue.front()<<" ";
        Queue.pop();
    }

}

void reverse(queue<int> &q)
{
    if(q.empty())
        return;

    int x=q.front();
    q.pop();

    reverse(q);
    q.push(x);
}

int main()
{
    queue<int> q;
    q.push(12);
    q.push(5);
    q.push(15);
    q.push(20);

    reverse(q);
    print(q);
}
```

OUTPUT : 20 15 5 12

Generate a Number with Given Digit :

```cpp
//Given a number n, print first n number(in increasing order)
// such that all these numbers have digits in set {5, 6}

#include<iostream>
#include<queue>
using namespace std;

void printFirstN(int n)
{
    queue<string>q;
    q.push("5");
    q.push("6");

    for(int i=0;i<n;i++)
    {
        string curr=q.front();

        cout<<curr<<" ";

        q.pop();

        q.push(curr + "5");
        q.push(curr + "6");
    }
}

int main()
{
    int n=5;
    printFirstN(n);

    return 0;
}
```

OUTPUT :

5 6 55 56 65