Queue

- A Queue is a linear structure which follows a particular order in which the operations are performed.
- The order is First In First Out (FIFO).
- A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first.
- The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.
- A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

Queue

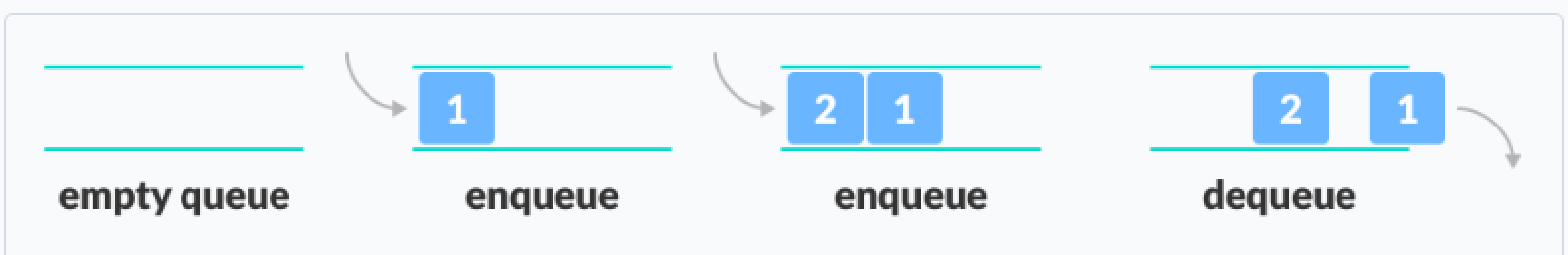
Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue).

LAST IN
LAST OUT
ONE WAY
FIRST IN
FIRST OUT

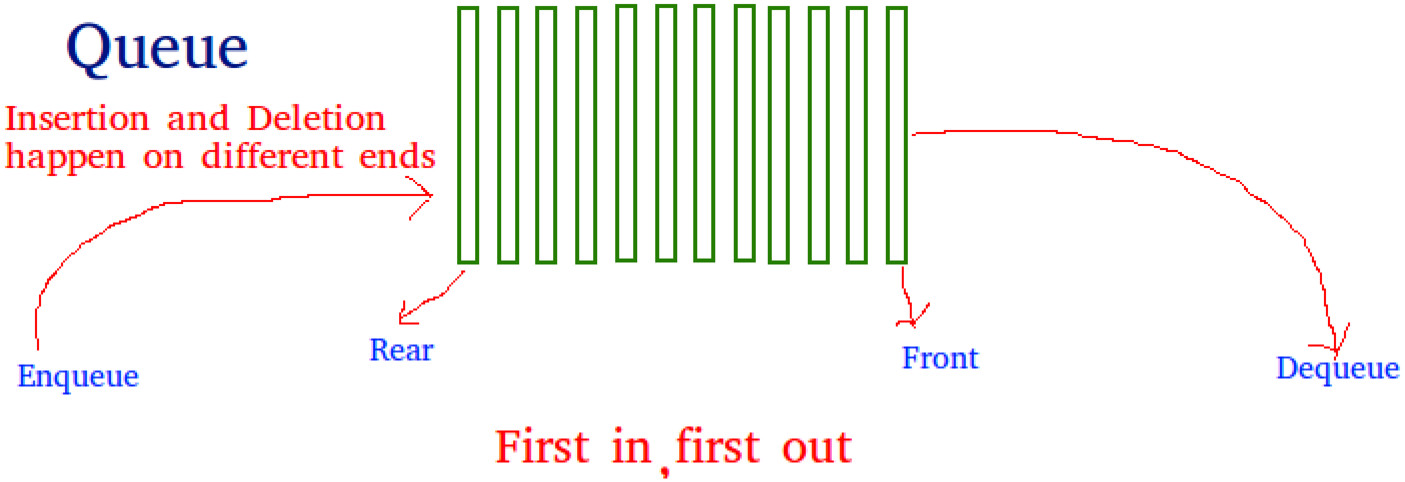
Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

Queue



FIFO Representation of Queue



Basic Operations

enqueue() – add (store) an item to the queue.
dequeue() – remove (access) an item from the queue.

Enqueue Operation

Queues maintain two data pointers, front/first and rear/last. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue −

Step 1 − Check if the queue is full.

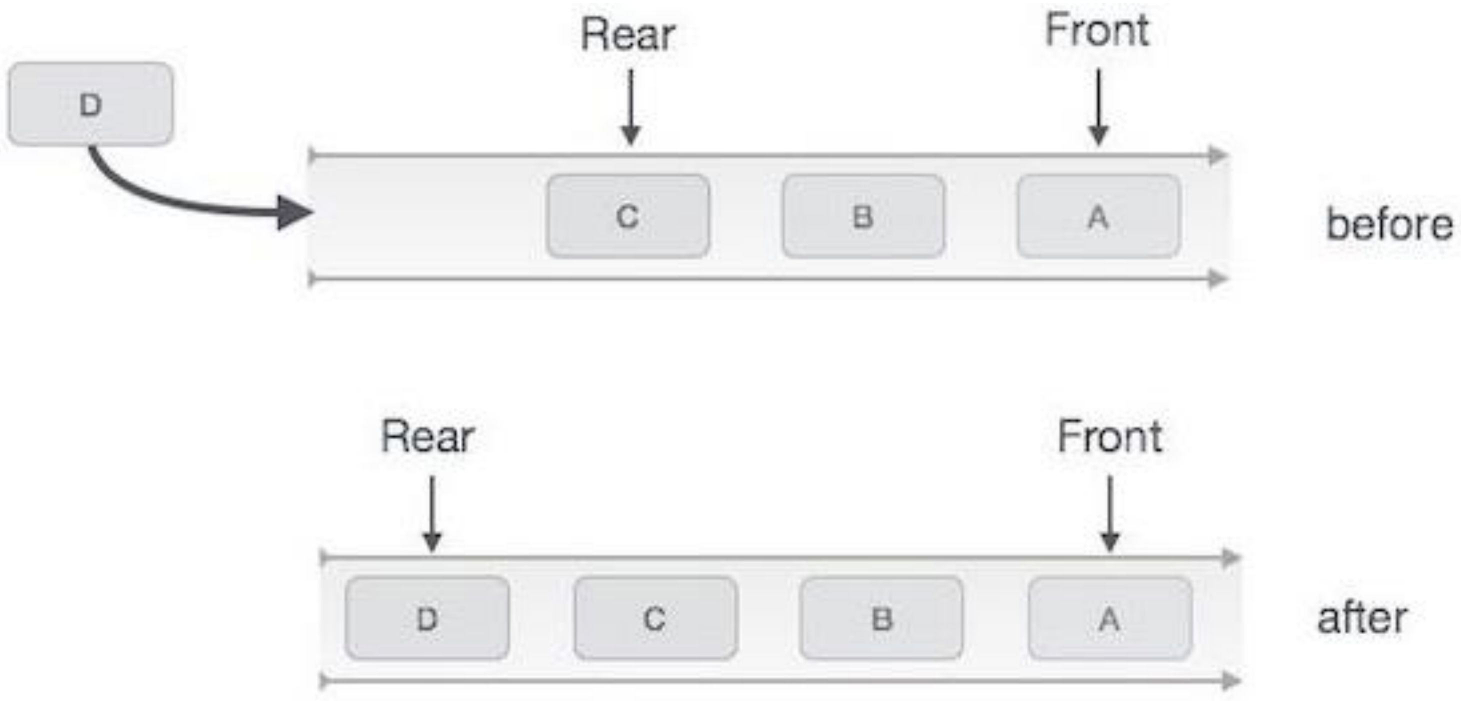
Step 2 − If the queue is full, produce overflow error and exit.

Step 3 − If the queue is not full, increment rear pointer to point the next empty space.

Step 4 − Add data element to the queue location, where the rear is pointing.

Step 5 − return success

Enqueue Operation



Queue Enqueue

Dequeue Operation

Accessing data from the queue is a process of two tasks − access the data where front is pointing and remove the data after access. The following steps are taken to perform dequeue operation −

Step 1 − Check if the queue is empty.

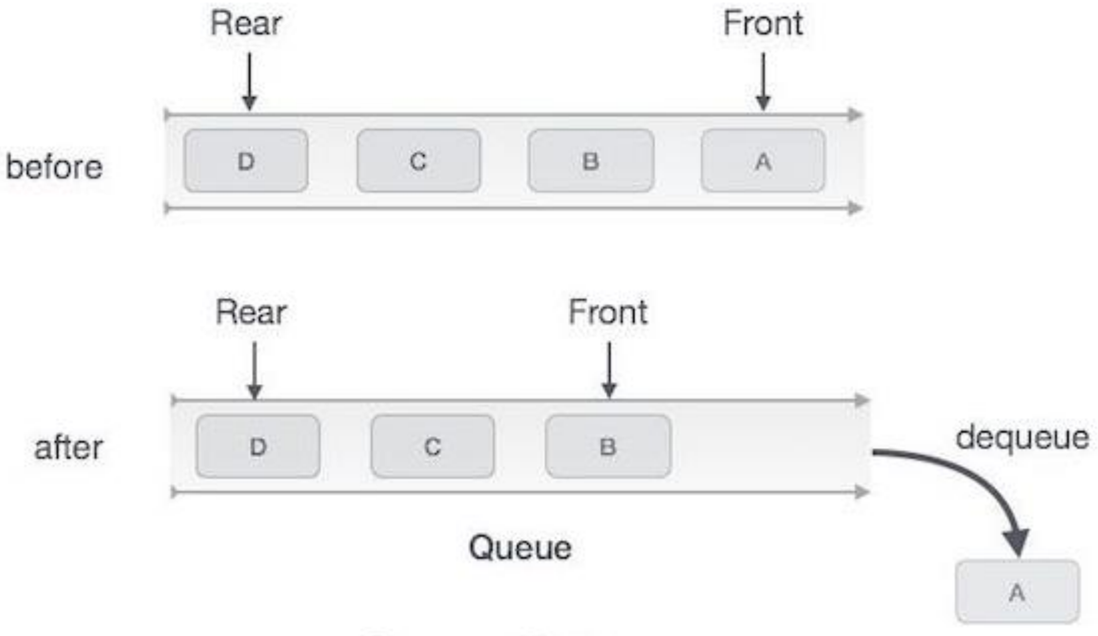
Step 2 − If the queue is empty, produce underflow error and exit.

Step 3 − If the queue is not empty, access the data where front is pointing.

Step 4 − Increment front pointer to point to the next available data element.

Step 5 − Return success.

Dequeue Operation



Queue Dequeue

Queue using Arrays

```
#include <iostream>
#include <conio.h>
const int size=10;
using namespace std;

class queue
{
        private:
                int array[size];
                int first,last,count;
                public:
                        queue();
                        void insert(int);
                        int remove();

};
```

Queue using Arrays

```
queue::queue():first(0),last(-1),count(0)
{
}
```

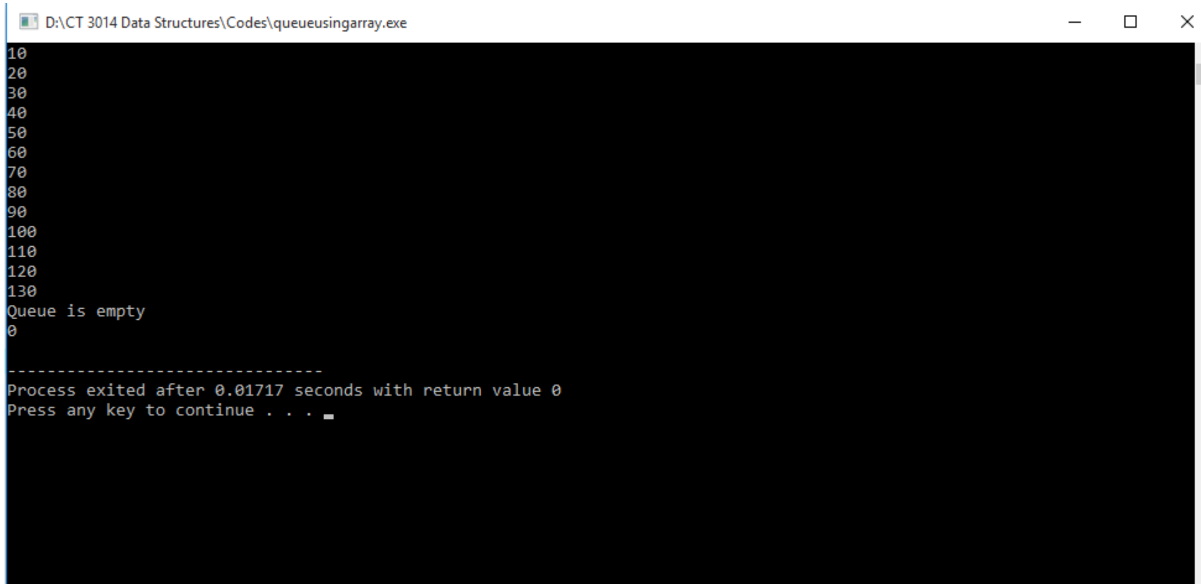
```
void queue::insert(int value)

{

        if(count>=size)

        {

        cout<<"Queue is full\n";

        return;

        }

        if(last>=size-1)

        {

                last=-1;

        }

        array[++last]=value;

        count++;

}
```

Queue using Arrays

```
int queue::remove()
{
        if(count<=0)
        {
                cout<<"Queue is empty\n";
                return NULL;
        }
        if(first>=size)
        {
                first=0;
        }
        count--;
        return array[first++];
}
```

Queue using Arrays

```
int main()
{
        queue q;
        q.insert(10);
        q.insert(20);
        q.insert(30);
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        q.insert(40);
        q.insert(50);
        q.insert(60);
        q.insert(70);
        q.insert(80);
        q.insert(90);
        q.insert(100);
        q.insert(110);
        q.insert(120);
        q.insert(130);
cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
        cout<<q.remove()<<endl;
}
```



```
10
20
30
40
50
60
70
80
90
100
110
120
130
Queue is empty
0

--------------------------------
Process exited after 0.01717 seconds with return value 0
Press any key to continue . . .
```

Queue using Linked List

```
#include <iostream>
#include <conio.h>

using namespace std;

class node
{
		public:
				int data;
				node *link;
				node(){data=0;link=NULL; }
};
```

Queue using Linked List

```
class queue
{
        private:
                node *first,*last;
                public:
                        queue():first(NULL),last(NULL){
        }
                        void insert(int);
                        int remove();
                        ~queue();
};
```

Queue using Linked List

```
void queue::insert(int value)
{
        node *ptr=NULL;
        ptr=new node;
        if(ptr==NULL)
        {
                cout<<"Queue is full\n";
                return;
        }
if(first==NULL)
        {
                first=ptr;
                first->data=value;
                first->link=NULL;
                last=first;
                return;
        }
        last->link=ptr;
        last=ptr;
        last->data=value;
        last->link=NULL;
}
```

Queue using Linked List

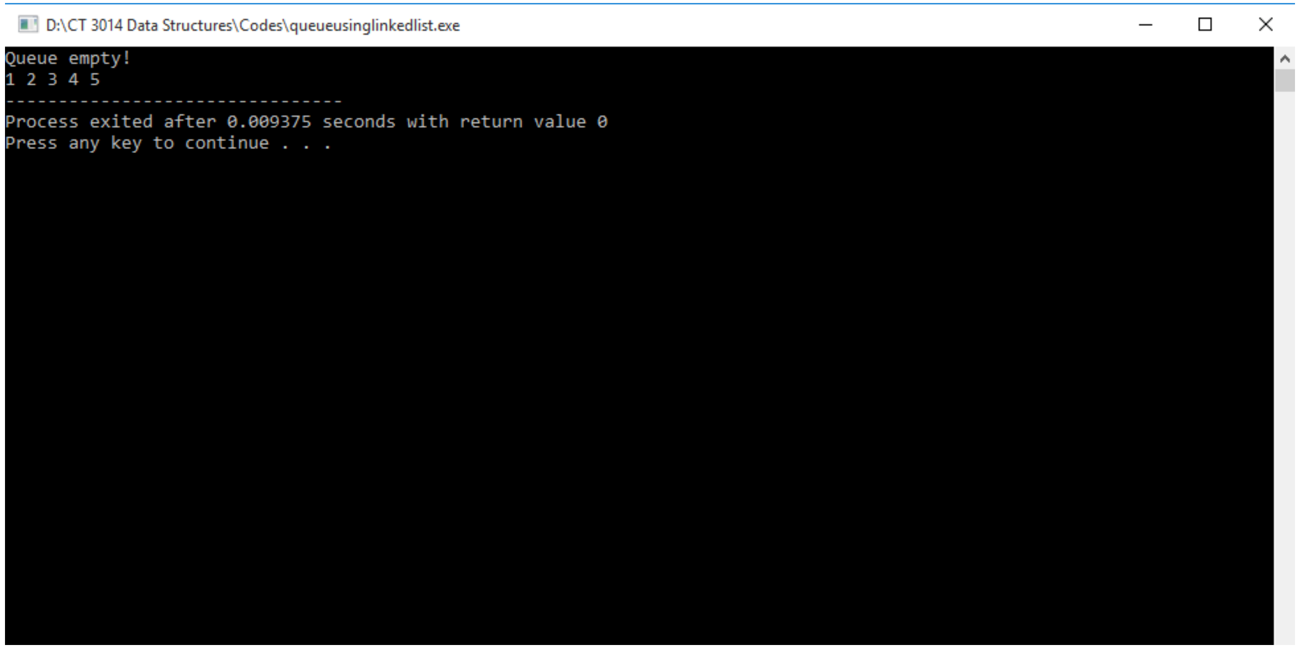
```
int queue::remove()
{
        if(first==NULL)
        {
                cout<<"Queue empty!\n";
                return NULL;
        }
        node *ptr=first;
        first=first->link;
        int value = ptr->data;
        delete ptr;
        return value;
}
```

Queue using Linked List

```
queue::~queue()
{
        node *ptr=first;
        while(first!=NULL)
        {
                first=first->link;
                delete ptr;
                ptr=first;
        }
}
```

Queue using Linked List

```
int main()
{
        queue q;
        int i;
        q.remove();
        for(i=1;i<=5;i++)
        q.insert(i);
        for(i=1;i<=5;i++)
        cout<<q.remove()<<" ";
}
```



```
Queue empty!
1 2 3 4 5
--------------------------------
Process exited after 0.009375 seconds with return value 0
Press any key to continue . . .
```