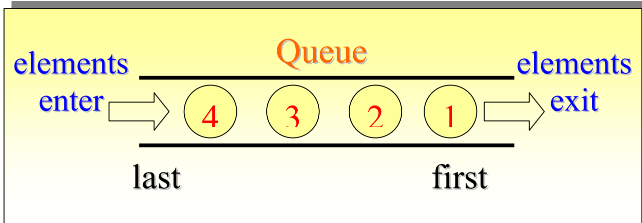
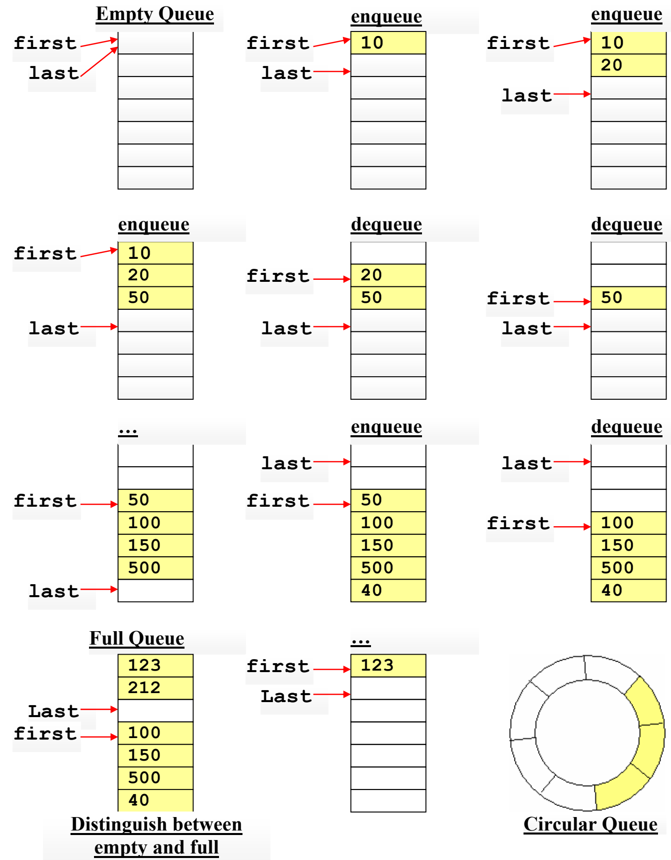
Queue…
It is an ordered group of homogeneous items of elements… Queues have two ends:

- ➢Elements are added at one end…
- ➢Elements are removed from the other end…

The element added first is also removed first
(*FIFO*: *F*irst *I*n, *F*irst *O*ut).



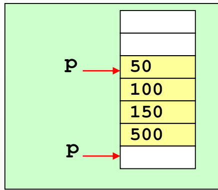


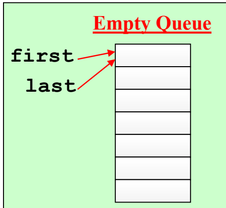
Non-Template implementation.

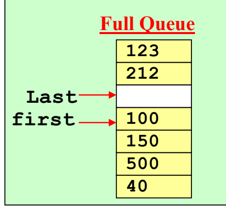
```
#include <iostream.h>

const int maxQ = 5;

class Queue{
 public:
   Queue();
   bool enqueue(const int &);
   bool dequeue(int &);
   bool front(int &);
   bool isEmpty();
   bool isFull();
   void printQ();
 private:
   int nextPos(int);
   int first;     // marks the front of the queue
   int last;      // marks the rear of the queue
   int Qarr[maxQ];
};
```

…

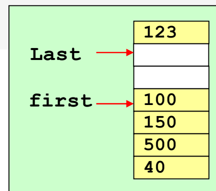

```
// Constructor. Start both front and rear at 0
Queue::Queue() {
   first = 0;
   last = 0;
}

// Return next empty position or 0 if Top….
// Used for enqueue and dequeue…
int Queue::nextPos(int p) {
   if (p == maxQ - 1)
      return 0; // at end of circle
   else
      return p+1;
}

// Return true if the Queue is empty, that is, if front is the same as rear
bool Queue::isEmpty() {
   return bool(first == last);
}

// Return true if the Queue is full, that is Full…
bool Queue::isFull() {
   return ((nextPos(last)) == first);
}
```



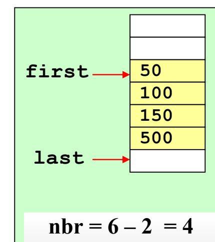


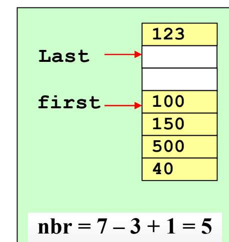


```
// Add e to the rear of the queue, advancing last to next position
bool  Queue:: enqueue(const int &e) {
   if (isFull()) {
      cout << "Queue is FULL\n";
      return false;
   }
   else {
      Qarr[last] = e;
      last = nextPos(last);
      return true;
   }
}

// Advance front of queue, return in e, the value of element at the front
bool Queue::dequeue(int &e ) {
   if (isEmpty()) {
      cout << "Queue is Empty\n";
      return false;
   }
   else {
      e = Qarr[first];
      first = nextPos(first);
      return true;
   }
}
```




```
void Queue::printQ() {
   int nbr,ndx;

   // Get the number of Items in the Queue…
   if (last > first)
      nbr = last - first;
   else
      nbr = maxQ - first + last;

   cout <<"First=" << first <<"-Last=" << last << endl;
   ndx= first;
   while (nbr > 0) {
      cout << Qarr[ndx]<< " ";
      ndx = nextPos(ndx);
      nbr -=1;
   }
   cout << endl;
}
```






```
// Return value of element at the front
bool Queue::front(int &e ) {
   if (isEmpty()) {
      cout << "Queue is Empty\n";
      return false;
   }
   else {
      e = Qarr[first];
      return true;
   }
}
```


Template implementation.

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

const int maxQ = 5;

template < class qT >
class Queue {
 public:
   Queue();
   bool enqueue(const qT &);
   bool dequeue(qT &);
   bool front(qT &);
   bool isEmpty();
   bool isFull();
   void printQ();
 private:
   int nextPos(int);
   int first;        // marks the front of the queue
   int last;         // marks the rear of the queue
   qT Qarr[maxQ];
};
```

```
// Constructor. Start both front and rear at 0
template < class qT >
Queue < qT >::Queue() {
   first = 0;
   last = 0;
}

// add e to the rear of the queue, advancing last to next position
template < class qT >
bool  Queue< qT > :: enqueue(const qT &e) {
   if (isFull()) {
      cout << "Queue is FULL\n";
      return false;
   }
   else {
      Qarr[last] = e;
      last = nextPos(last);
      return true;
   }
}
// Advance front of queue, return in e, the value of element at the front
template < class qT >
bool Queue < qT >::dequeue(qT &e ) {
   if (isEmpty()) {
      cout << "Queue is Empty\n";
      return false;
   }
   else {
      e = Qarr[first];
      first = nextPos(first);
      return true;
   }
}
```



```
// Return value of element at the front
template < class qT >
bool Queue < qT >::front(qT &e ) {
   if (isEmpty()) {
      cout << "Queue is Empty\n";
      return false;
   }
   else {
      e = Qarr[first];
      return true;
   }
}

// Return true if the Queue is empty, that is, if front is the same as rear
template < class qT >
bool Queue < qT >::isEmpty() {
   return bool(first == last);
}

// Return true if the Queue is full, that is Full…
template < class qT >
bool Queue < qT >::isFull() {
   return ((nextPos(last)) == first);
}
```



```
// Return next empty position or 0 if full.
template < class qT >
int Queue < qT >::nextPos(int p) {
   if (p == maxQ - 1)
      return 0;      // at end of circle
   else
      return p+1;
}

template < class qT >
void Queue < qT >::printQ() {
   int nbr,ndx;
   if (last > first)
      nbr = last - first;
   else
      nbr = maxQ - first + last;
   cout <<"First=" << first <<"-Last=" << last << endl;
   ndx= first;
   while (nbr > 0) {
      cout << Qarr[ndx]<< " ";
      ndx = nextPos(ndx);
      nbr -=1;
   }
   cout << endl;
}
```



```
void main() {
   Queue<int> intQ;
   int x;

   intQ.enqueue(1);
   intQ.enqueue(2);
   intQ.enqueue(3);
   intQ.enqueue(4);
   intQ.dequeue(x);
   cout << x << endl;
   intQ.dequeue(x);
   cout << x << endl;
   intQ.dequeue(x);
   cout << x << endl;
   intQ.enqueue(5);
   intQ.enqueue(6);
   intQ.enqueue(7);
   intQ.printQ();
   intQ.dequeue(x);
   cout << x << endl;
   intQ.dequeue(x);
   cout << x << endl;
   intQ.dequeue(x);
   cout << x << endl;
   intQ.enqueue(8);
   intQ.enqueue(9);
   intQ.enqueue(10);
   intQ.enqueue(11);
   intQ.printQ();
   intQ.front(x);
   cout << x << endl;
}
```