```
/* HEADER FILE - ARRAY IMPLEMENTATION OF A QUEUE */

#include "underflowoverflow.h"

template <class Object>
class Queue
{
    public:
        // Constructor for a queue of a given capacity
        Queue(int capacity);
        // Mutator - enqueue an item - error if full
        void enqueue(Object item) throw(Overflow);
        // Mutator - remove an item - error if empty
        void dequeue() throw(Underflow);
        // Accessor for front item - error if empty
        Object front() const throw(Underflow);
```

```
        // Accessor - true if queue is empty
        bool empty() const;
        // Accessor - true if queue is full
        bool full() const;
        // Make the queue empty - discarding all items on it
        void makeEmpty();
    private:
        int _capacity, _front, _back;
        bool _empty;
        Object * _theArray;
};
```

```
/* IMPLEMENTATION FILE - ARRAY IMPLEMENTATION OF A QUEUE */

#include "queuea.h"

template <class Object>
Queue<Object>::Queue(int capacity)
: _capacity(capacity),
  _front(0), _back(0), _empty(true),
  _theArray(new Object [capacity])
{ }
```

```
template <class Object>
void Queue<Object>::enqueue(Object item) throw(Overflow)
{
    if (_front == _back && ! _empty)
        throw Overflow();
    _theArray[_back] = item;
    _back = (_back + 1) % _capacity;
    _empty = false;
}
```

```
template <class Object>
void Queue<Object>::dequeue() throw(Underflow)
{
    if (_empty)
        throw Underflow();

    _front = (_front + 1) % _capacity;
    _empty = (_front == _back);
}
```

```
template <class Object>
Object Queue<Object>::front() const throw(Underflow)
{
    if (_empty)
        throw Underflow();

    return _theArray[_front];
}
```

```
template <class Object>
bool Queue<Object>::empty() const
{ return _empty; }
```

```
template <class Object>
bool Queue<Object>::full() const
{
    return _front == _back && ! _empty;
}
```

```
template <class Object>
void Queue<Object>::makeEmpty()
{
    _front = 0;
    _back = 0;
    _empty = true;
}
```