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

#include "underflowoverflow.h"

template <class Object>
class Queue
{
    public:
        // Constructor for a queue - can grow to any size
        Queue();
        // Mutator - enqueue an item
        void enqueue(Object item);
        // 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 - never true for list
        bool full() const;
        // Make the queue empty - discarding all items on it
        void makeEmpty();

        // Methods needed for list - called automatically by
        // compiler. Not normally called by user code
        Queue(const Queue & rhs);
        const Queue & operator = (const Queue & rhs);
        ~Queue();
private:
        class Node;
        Node * _back;
};
```

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

#include "queuel.h"
#ifndef NULL
#define NULL 0
#endif

template <class Object>
class Queue<Object>::Node
{
    public:

        Node(Object item)
        : _item(item), _link(NULL)
        { }

        Object _item;
        Node * _link;
};
```

```
template <class Object>
Queue<Object>::Queue()
: _back(NULL)
{ }
```

```
template <class Object>
void Queue<Object>::enqueue(Object item)
{
    Node * n = new Node(item);
    if (_back == NULL)
        n -> _link = n;
    else
    {
        n -> _link = _back -> _link;
        _back -> _link = n;
    }
    _back = n;
}
```

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

    Node * front = _back -> _link;
    // See if we're removing last node
    if (_back -> _link == _back)
        _back = NULL;
    else
        _back -> _link = front -> _link;
    delete front;
}
```

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

    return _back -> _link -> _item;
}
```

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

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

```
template <class Object>
void Queue<Object>::makeEmpty()
{
    while (_back != NULL)
        dequeue();
}
```

```
template <class Object>
Queue<Object>::Queue(const Queue & rhs)
{
    * this = rhs;
}
```

```
template <class Object>
const Queue<Object> & Queue<Object>::operator=(const Queue & rhs)
{
    if (this != &rhs)
    {
        // Assignment requires discarding all existing nodes, and
        // then making a deep copy

        makeEmpty();
        if (rhs.back == NULL)
            return * this;

        Node * rhsBack = rhs._back;
        Node * p = rhsBack;
        do
        {
            p = p -> _link;
            enqueue(p -> _item);
        } while (p != rhs._back);
    }

    return * this;
}
```

```
template <class Object>
Queue<Object>::~Queue()
{
    makeEmpty();
}
```