

Queues

Jyh-Shing Roger Jang (張智星)
CSIE Dept, National Taiwan University

Queue

- What is a queue?
 - An ordered list where insertions occur at the rear while deletions occur at the front.
 - Also known as FIFO (first-in-first-out) list.
- Real-world examples



ADT OF QUEUES

- Data

- Arbitrary objects

- Operations

- **enqueue(object)**: inserts an element at the end
 - **dequeue()**: removes the element at the front
 - object **front()**: returns the element at the front without removing it
 - integer **size()**: returns the number of elements stored
 - boolean **empty()**: indicates whether no elements are stored

- Exceptions

- **dequeue()** and **front()** cannot be performed if the queue is empty.
 - **enqueue(object)** cannot be performed if the queue is full.



Array-based Queues

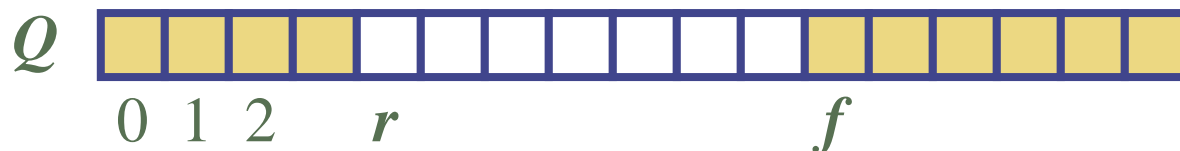
To avoid
element movement

- Use an array of size N in a circular fashion
- Three variables to keep track of the front and rear
 - f: index of the front element
 - r: index immediately past the rear element
 - n: number of items in the queue

normal configuration



wrapped-around configuration



Queue Interface in C++

- C++ interface corresponding to our Queue ADT
- Requires the definition of exception QueueEmpty
- Different from the built-in C++ STL class queue

```
template <typename E>
class Queue {
public:
    int size() const;
    bool empty() const;
    const E& front() const
        throw(QueueEmpty);
    void enqueue (const E& e);
    void dequeue()
        throw(QueueEmpty);
};
```

Queue Operations

Algorithm *size()*

return n

Algorithm *empty()*

return $(n == 0)$

Algorithm *dequeue()*

if *empty()* then

throw *QueueEmpty*

else

$f \leftarrow (f + 1) \bmod N$

$n \leftarrow n - 1$

Algorithm *enqueue(o)*

if *size()* = $N - 1$ then

throw *QueueFull*

else

$Q[r] \leftarrow o$

$r \leftarrow (r + 1) \bmod N$

$n \leftarrow n + 1$

Why
not N ?

Modulo operator
(" \bmod " in C/C++)

STL Queue

- STL provides an implementation of a queue

- Based on the STL vector class
- Declaration

```
#include <queue>
using std::queue;           // make queue accessible
queue<float> myQueue;       // a queue of floats
```

- Operators

`size()`: Return the number of elements in the queue.
`empty()`: Return true if the queue is empty and false otherwise.
`push(e)`: Enqueue *e* at the rear of the queue.
`pop()`: Dequeue the element at the front of the queue.
`front()`: Return a reference to the element at the queue's front.
`back()`: Return a reference to the element at the queue's rear.

No exception is thrown
if something
goes wrong!

Extra for queues!

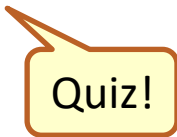
<i>Operation</i>		<i>Output</i> <i>Q</i>
enqueue(5)	—	(5)
enqueue(3)	—	(5, 3)
dequeue()	5	(3)
enqueue(7)	—	(3, 7)
dequeue()	3	(7)
front()	7	(7)
dequeue()	7	()
dequeue()	"error"	()
isEmpty()		true
enqueue(9)	—	(9)
enqueue(7)	—	(9, 7)
size()	2	(9, 7)
enqueue(3)	—	(9, 7, 3)
enqueue(5)	—	(9, 7, 3, 5)
dequeue()	9	(7, 3, 5)

Other Implementation of Queues

- We can also use linked lists to implement queues
 - Do not require contiguous memory
 - More codes for pointer manipulation
 - See textbook for more info.

Applications of Queues

- Access to shared resources
 - Printer
 - Requests received at a web server
- Breadth-first search (BFS) for graph traversal
- Many, many more...



Quiz!