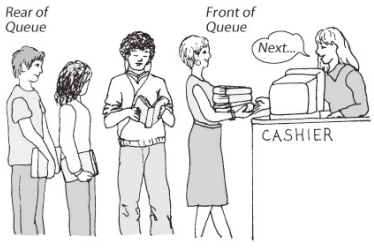
CMPSC 24: Lecture 12
Abstract Data Type: Queues

Divyakant Agrawal
Department of Computer Science
UC Santa Barbara

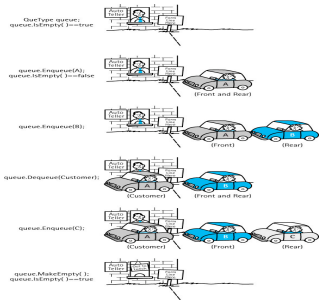
Lecture Plan

- Another common Abstract Data Type:
  - Queues
- Queues Specification
- Queue Implementation:
  - Array Implementation
  - Linked List Implementation

Queues



Queues



4

Queues

•*What do these composite objects all have in common?*

5

Queues

•An abstract data type in which elements are added to the rear and removed from the front; a "first in, first out" (FIFO) structure.

•Applications

- •Checking for a palindrome
- •Assigning priority

6

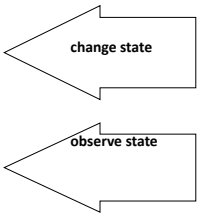
Queues

•*What operations would be appropriate for a queue?*

7

Queues

- **Transformers**
  - MakeEmpty
  - Enqueue
  - Dequeue
- **Observers**
  - ***IsEmpty***
  - ***IsFull***



8

Queue ADT Operations

- MakeEmpty **-- Sets queue to an empty state.**
- IsEmpty **-- Determines whether the queue is currently empty.**
- IsFull **-- Determines whether the queue is currently full.**
- Enqueue (ItemType newItem) **-- Adds newItem to the rear of the queue.**
- Dequeue (ItemType& item) **-- Removes the item at the front of the queue and returns it in item.**

9

Queues

```
class QueType
{
public:
  QueType(int max);
  QueType();
  ~QueType();
  bool IsEmpty() const;
  bool IsFull() const;
  void Enqueue(ItemType
 item);
  void Dequeue(ItemType&
 item);
```

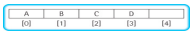
Logical Level

10

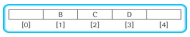
Array-Based Implementation

One data structure: An array with the front of the queue fixed in the first position

Enqueue A, B, C, D



Dequeue



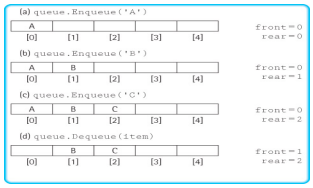
Move elements down

What's wrong with this design?

11

Array-Based Implementation

Another data structure: An array where the front floats

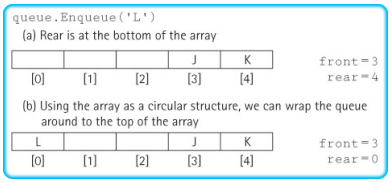


What happens if we add X, Y, and Z ?

12

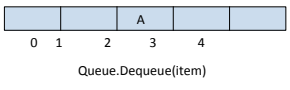
Array-Based Implementation

We can let the queue wrap around in the array; i.e. treat the array as a circular structure



13

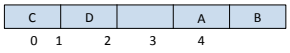
Array-Based Implementation



front = 2
rear = 2

Queue.Dequeue(item)

Empty Queue



front = 3
rear = 1

Queue.Enqueue('E')

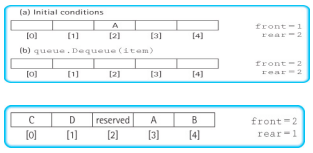
Full Queue

How can we tell the difference?

14

Array-Based Implementation

A third data structure: front indicates the slot preceding the first item; it is reserved and not used



Empty Queue

Full Queue

15

Array-Based Implementation

```
private:
  int front;
  int rear;
  int maxQue;

  ItemType* items;
}
```

Complete implementation level

*To what do we initialize **front** and **rear** ?*

16

Array-Based Implementation

```
QueType::QueType(int max)
{
  maxQue = max + 1;
  front = maxQue - 1;
  rear = maxQue - 1;
  items = new ItemType[maxQue];
}

bool QueType::IsEmpty( )
{ return ( rear == front );}

bool QueType::IsFull( )
{ return ( (rear + 1) % maxQue ==
front )}
```

*Why is the array declared **max + 1** ?*

17

Array-Based Implementation

```
void QueType::Enqueue(ItemType Item)
{
  if (IsFull()) throw FullQueue();
  else
  {
    rear = (rear + 1) % maxQue;
    items[rear] = newItem;
  }
}
```

18

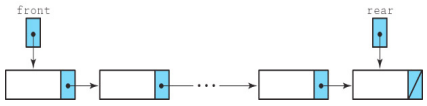
Array-Based Implementation

```
void QueType::Dequeue(ItemType& item)
{
  if (IsEmpty()) throw EmptyQueue();
  else
  {
    front = (front + 1) % maxQue;
    item = items[front];
  }
}
```

19

Linked Implementation

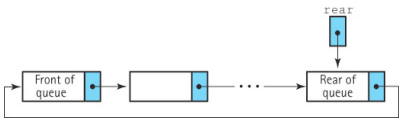
Data structure for linked queue



20

Linked Implementation

A circular linked queue uses only one external pointer: rear



How do you access front?

21