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

• What do these composite objects all have in common?

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
Queues

• What operations would be appropriate for a queue?

Queues

• Transformers
  – MakeEmpty
  – Enqueue
  – Dequeue

• Observers
  – IsEmpty
  – IsFull

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.
Queues

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

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?

Array-Based Implementation

Another data structure: An array where the front floats

What happens if we add X, Y, and Z?
Array-Based Implementation

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

![Diagram](image)

Array-Based Implementation

**Empty Queue**

0 1 2 3 4
Queue.Dequeue(item)
front = 2
rear = 2

**Full Queue**

C D A B
Queue.Enqueue('E')
front = 3
rear = 1

How can we tell the difference?

Array-Based Implementation

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

![Diagram](image)

**Empty Queue**

Empty Queue

**Full Queue**

Full Queue
Array-Based Implementation

private:
  int front;
  int rear;
  int maxQue;
  ItemType* items;
}

To what do we initialize front and rear?

Array-Based Implementation

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

Why is the array declared max + 1?

Array-Based Implementation

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

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

Linked Implementation

Data structure for linked queue

[Diagram of linked queue]

Linked Implementation

A circular linked queue uses only one external pointer: rear

[Diagram of circular linked queue]

How do you access front?