CS2040C Data Structures and Algorithms

Queue ADT

Outline

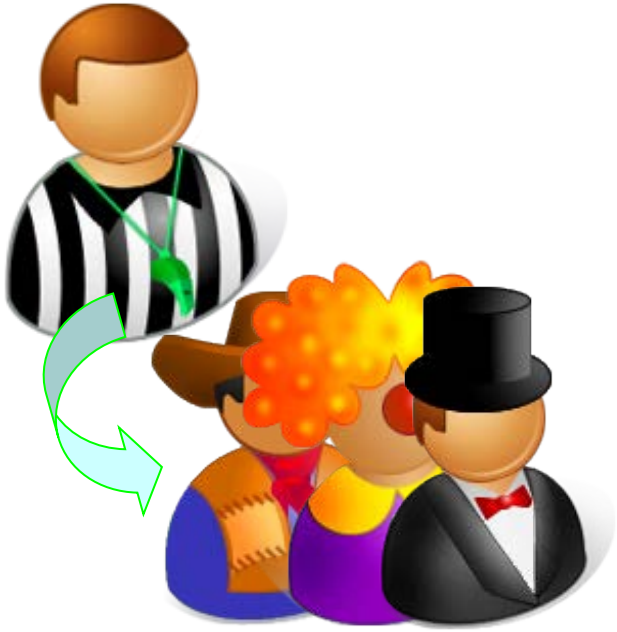
- Queue
  - Introduction
  - Specification
  - Implementations
    - Linked List Based
    - Array Based
  - STL Queue, Deque
  - Application
    - Palindrome checking

What is a queue?

- Real life example:
  - A queue for movie tickets, airline reservation queue, etc.
- First item added will be the first item to be removed
  - Has the **F**irst **I**n **F**irst **O**ut (FIFO) property
- Major Operations:
  - **Enqueue:** Items are added to the **back of the queue**
  - **Dequeue:** Items are removed from the **front of the queue**
  - **Get Front:** Take a look at the first item

Queue: Illustration

A **queue** of 3 persons



Enqueue a new person to the **back of the queue**

Dequeue a person from the **front of the queue**

Queue ADT (Linked List): C++ Specification

```
template<typename T>
class Queue {
   public:
     Queue();

     bool isEmpty() const;
     int size() const;

     void enqueue(const T& newItem) throw (SimpleException);

     void dequeue() throw (SimpleException);
     void dequeue(T& queueFront) throw (SimpleException);


     void getFront(T& queueTop) const
               throw (SimpleException);
   private:
     //Implementation dependent
     //See subsequent implementation slides
};
```

Queue ADT: Design Considerations

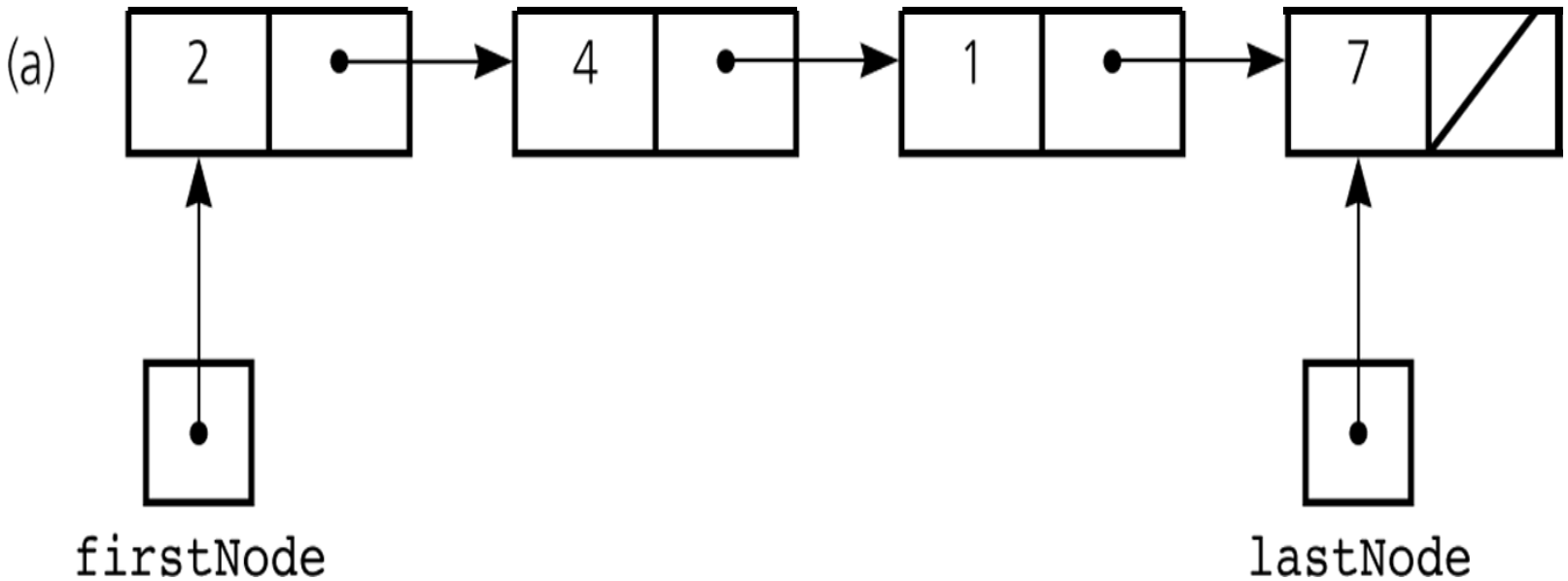
- How about the common choices?
  - Efficiency of **singly linked list implementation**:
    - Removing item at the head is the best case
    - Adding item at the back is the worst case
  - Efficiency of **array based implementation**:
    - Removing item at the head is the worst case
    - Adding item at the back is the best case
- Is it possible to have both efficient ***enqueue*()** and ***dequeue*()** operations?

Queue ADT using Modified Linked List

Improving the Singly Linked List

- Singly linked list performs badly for ***enqueue*( )**
  - Need to traverse all the way to the last node
  - Takes longer time as the queue grows
- How to avoid the traversal to the last node?
  - Easy: Just need to "know" where is the last node all the time……
- Solutions:
  a) Keep an additional pointer to the last node, OR
  b) Circular linked list with a tail pointer

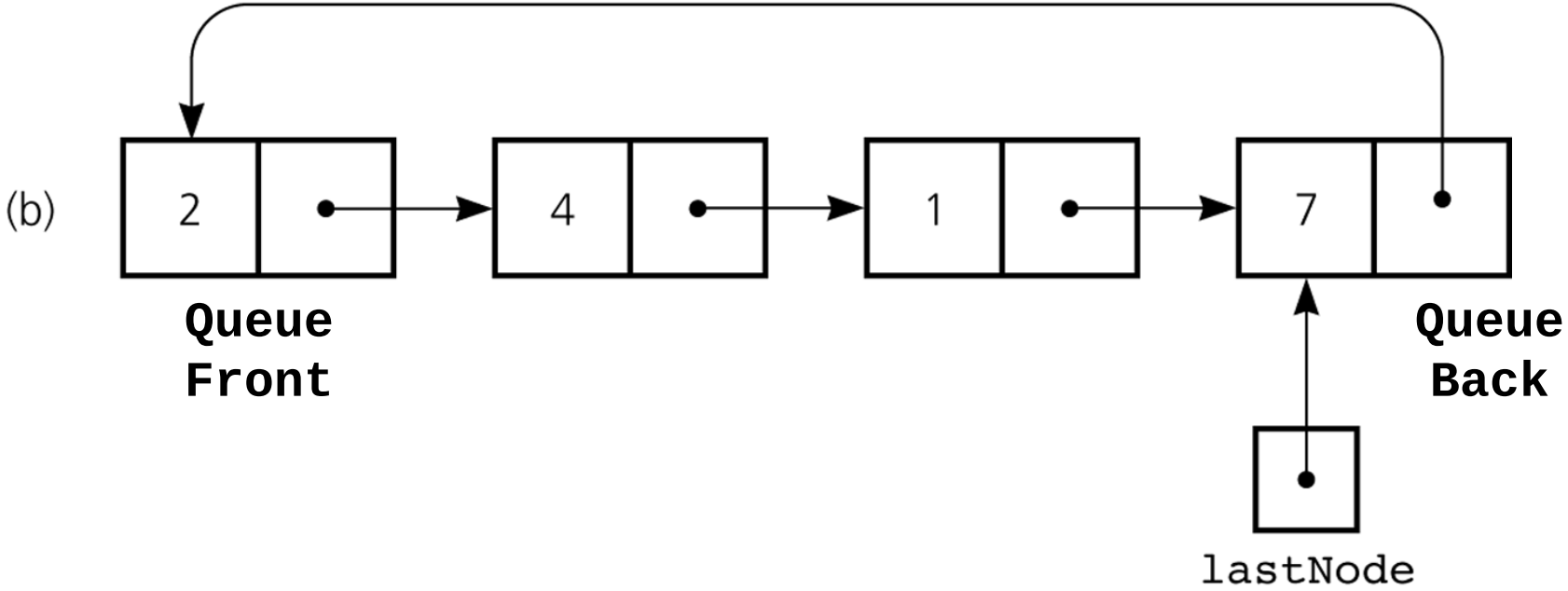
Linked List: with “head” and “tail”



Queue Front

Queue Back

Circular Linked List



- Only keep tracks of **`lastNode`** pointer
  - **`firstNode`** pointer can be set when needed:
    - **`firstNode = lastNode->next;`**
- We will use circular linked list for subsequent discussion

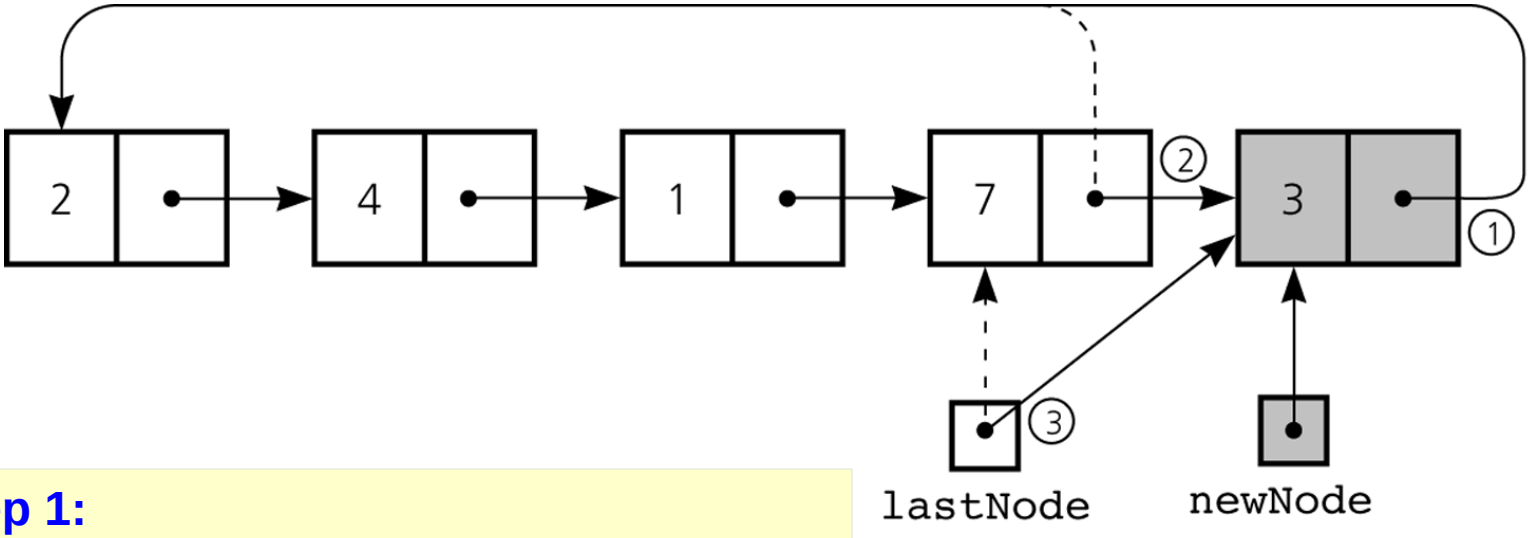
Queue ADT: C++ Specification

```
template<typename T>
class Queue {
   public:
    Queue();
    ~Queue();

     bool isEmpty() const;
     int size() const;

     void enqueue(const T& newItem) throw (SimpleException);

     void dequeue() throw (SimpleException);
     void dequeue(T& queueFront) throw (SimpleException);

     void getFront(T& queueTop) const throw (SimpleException);
private:
   struct QueueNode {
       T item;
       QueueNode *next;
   };

   int _size;
   QueueNode *_lastNode;
};
```

Just like a **ListNode** structure

QueueP.h

Insertion: Non-Empty Queue



Step 1:

```
newNode = new QueueNode;
newNode->next = lastNode->next;
newNode->item = 3;
```

This value is just an example only

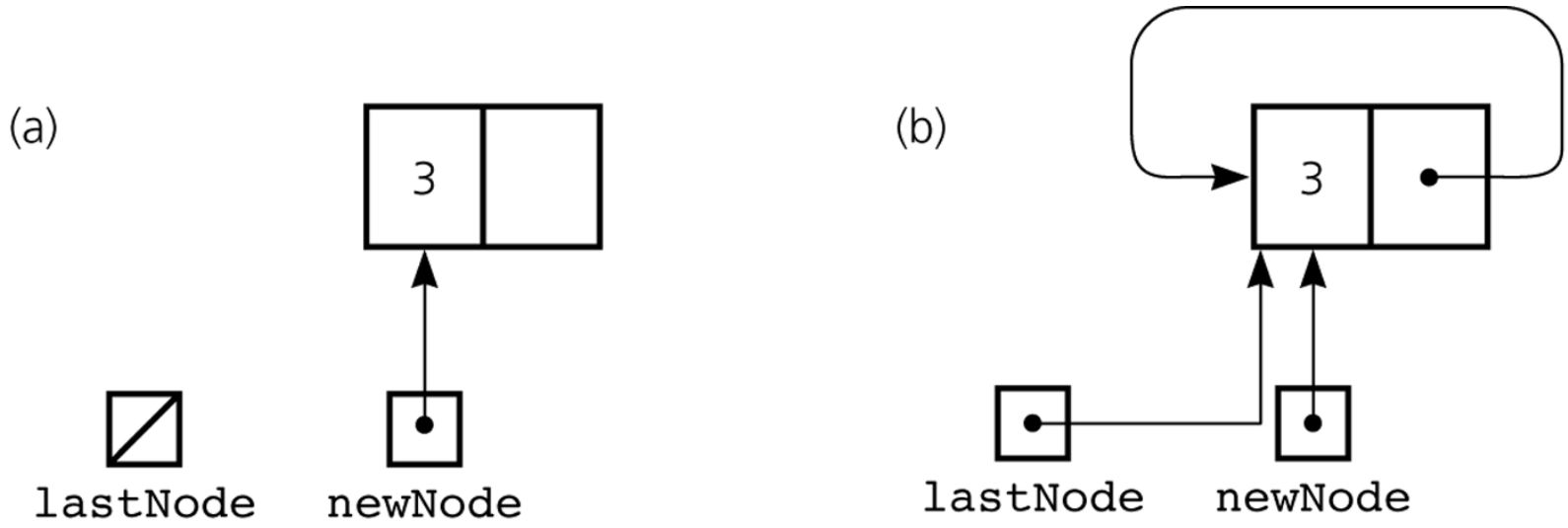
Step 2:

```
lastNode->next = newNode;
```

Step 3:

```
lastNode = newNode;
```

Insertion: Empty Queue



Step (a):

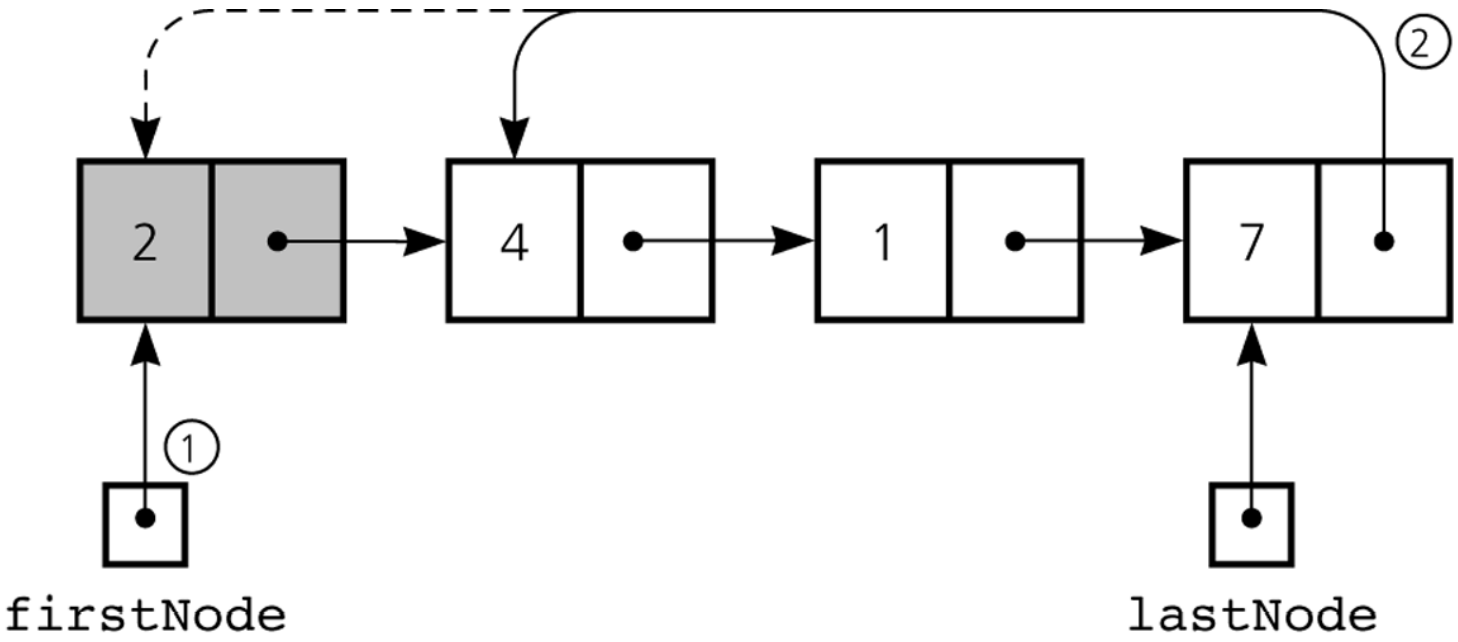
```
newNode = new QueueNode;
newNode->item = 3;
```

Step (b):

```
newNode->next = newNode;
lastNode = newNode;
```

Set up the “loop”

Deletion: Queue size larger than one



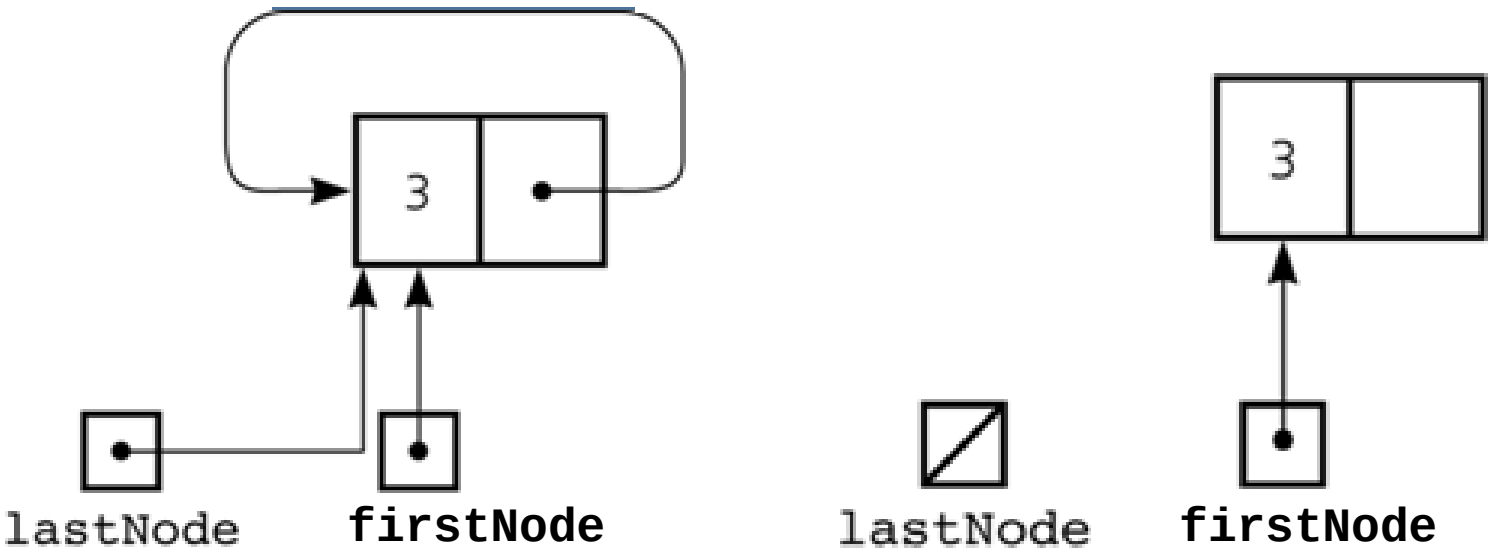
Step 1:

```
QueueNode* firstNode = lastNode->next;
```

Step 2:

```
lastNode->next = firstNode->next;
delete firstNode;
```

Deletion: Queue size equal to one?



Step 1:

```
QueueNode* firstNode = lastNode->next;
```

Step 2:

```
lastNode = null;
delete firstNode;
```

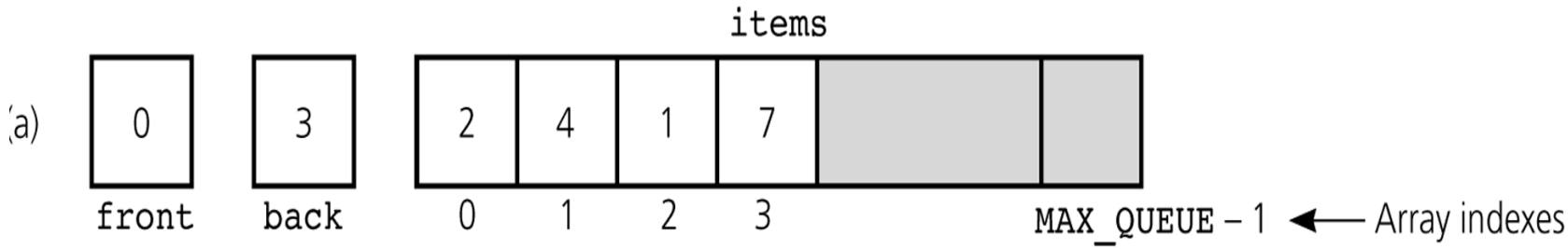
Queue ADT using Array

Array Implementation Issues

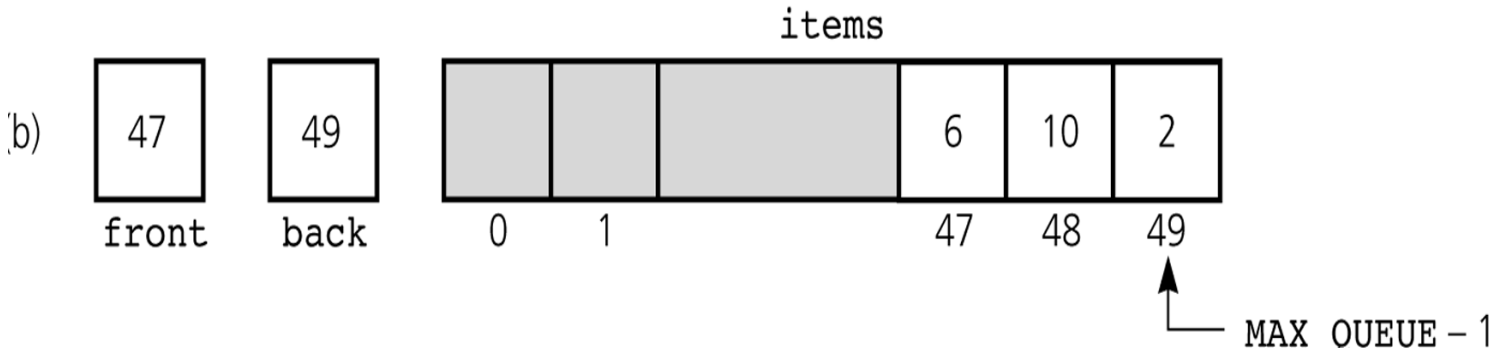
- Removing item from the front is inefficient
  - **Shifting items is too expensive**
- Basic Idea:
  - The reason for shifting is:
    - Front is assumed to be at index 0
  - Instead of shifting items:
    - **Shift the front index**
- So we have two indices:
  - **Front:** index of the queue front
  - **Back:** index of the queue back

Incorrect Implementation

- At the beginning, with 4 items queued



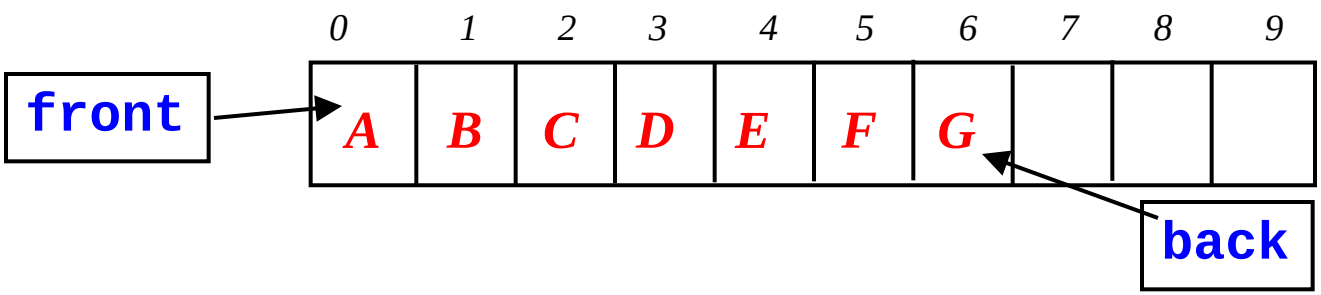
- After many queue operations



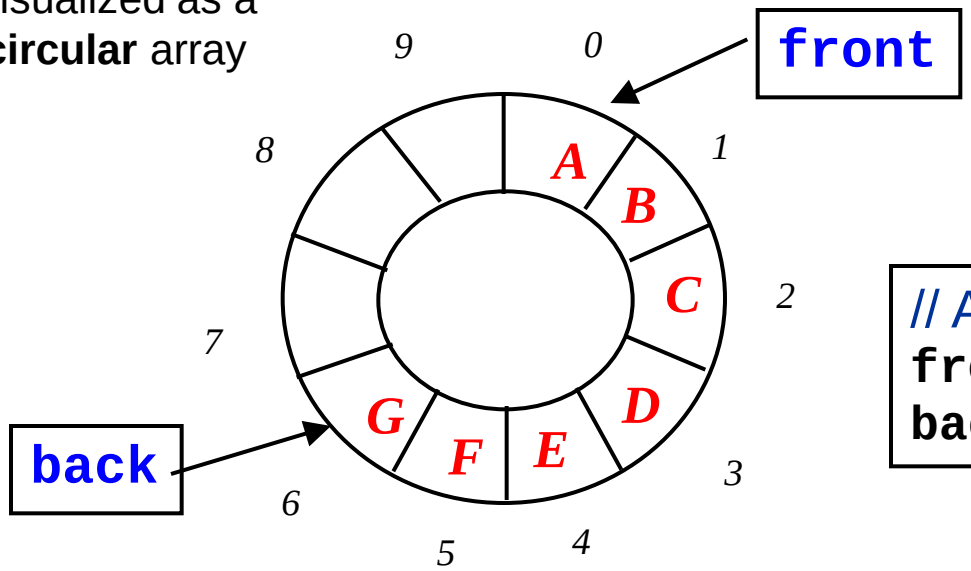
- The front index will drift to the right,
  - Most array locations empty and unusable

Circular Array

- Allow both indices to “wrap” back to index 0 when they reach the end of array
  - Effectively making the array “circular”



Visualized as a **circular** array




```
// Advancing front and back index
front = (front+1) % maxsize;
back = (back+1) % maxsize;
```

Queue ADT (Array): C++ Specification

```
const int MAX_QUEUE = 50;

template<typename T>
class Queue {
   public:
     Queue();
    bool isEmpty() const;
     int size() const;

     void enqueue(const T& newItem) throw (SimpleException);

     void dequeue() throw (SimpleException);
     void dequeue(T& queueFront) throw (SimpleException);

     void getFront(T& queueTop) const throw (SimpleException);
private:
   T _items[MAX_QUEUE];

   int _front, _back, _count;

};
```

QueueA.h

Implement Queue ADT (Array): 1/2

```
template<typename T>
Queue<T>::Queue( )
{
   _front = 0;
   _back = MAX_QUEUE - 1;
   _count = 0;
}

template<typename T>
void Queue<T>::enqueue( const T& newItem )
    throw (SimpleException)
{
   if ( _count == MAX_QUEUE )
       throw SimpleException("queue full");
   else   {
      _back = ( _back + 1 ) % MAX_QUEUE;
      _items[_back] = newItem;
      ++_count;
   }
}
```

Only selected methods implementation are shown. You should be able to code the rest easily by now ☺.

QueueA.cpp

Implement Queue ADT (Array): 2/2

```
template <typename T>
void Queue<T>::dequeue( T& queueFront )
    throw (SimpleException)
{
   if ( isEmpty() )
       throw SimpleException("Empty queue");
   else   {
      queueFront = _items[ _front ];
      _front = ( _front+1 ) % MAX_QUEUE;
      --_count;
   }
}
```

QueueA.cpp

STL Queue

Queue has a standard implementation

STL class queue

```
template <class T>
class queue {
public:
      bool empty() const;
      size_type size() const;

      T& front();
      T& back();

      void push (const T& t);
      void pop();
};
```

enqueue() is known as **push()** in STL Queue

This is the **dequeue()** equivalence

STL Deque

Both ends!

Deque

- Double-ended queue – is a list that allows for direct access to both ends of the list, to insert and delete elements
- Can be implemented as a doubly linked list with pointer data members head and tail
- STL deque – allows random access to any position of the deque (like in arrays and vectors)

STL class deque - Example

```
#include <iostream>
#include <algorithm>
#include <deque>
using namespace std;

int main (){
   deque<int> dq1;
   dq1.push_front(1);
   dq1.push_front(2);
   dq1.push_back(3);
   dq1.push_back(4);
   cout >> "First item is " >> front() >> endl;
   cout >> "last item is " >> back() >> endl;

    // continued next slide
```

STL class deque - Example cont'd

```
   deque<int> dq2(dq1.begin()+1, dq1.end()-1);
   dq1[1] = 5;
   dq1.erase(dq1.begin());
   dq1.insert(dq1.end() -1, 2, 6);

   sort(dq1.begin(), dq1.end());
   deque<int> dq3;
   dq3.resize(dq1.size()+dq2.size());
   merge(dq1.begin(), dq1.end(), dq2.begin(), dq2.end();
   dq3.begin());

   return 0;

};
```

Queue Application

Checking for palindrome

Palindrome : Problem Description

- **Palindrome** is a string which reads the same either *left to right*, or *right to left*
  - **Palindromes:** “r a d a r” and “d e e d”
  - **Counter Example:** “d a t a”
- Many solutions:
  - We use the two newly learned ADTs
  - Highlight the difference between LIFO and FIFO property
- Main Idea:
  - Use *stack* to reverse the input
  - Use *queue* to preserve the input
  - The two sequences should be the same for palindrome

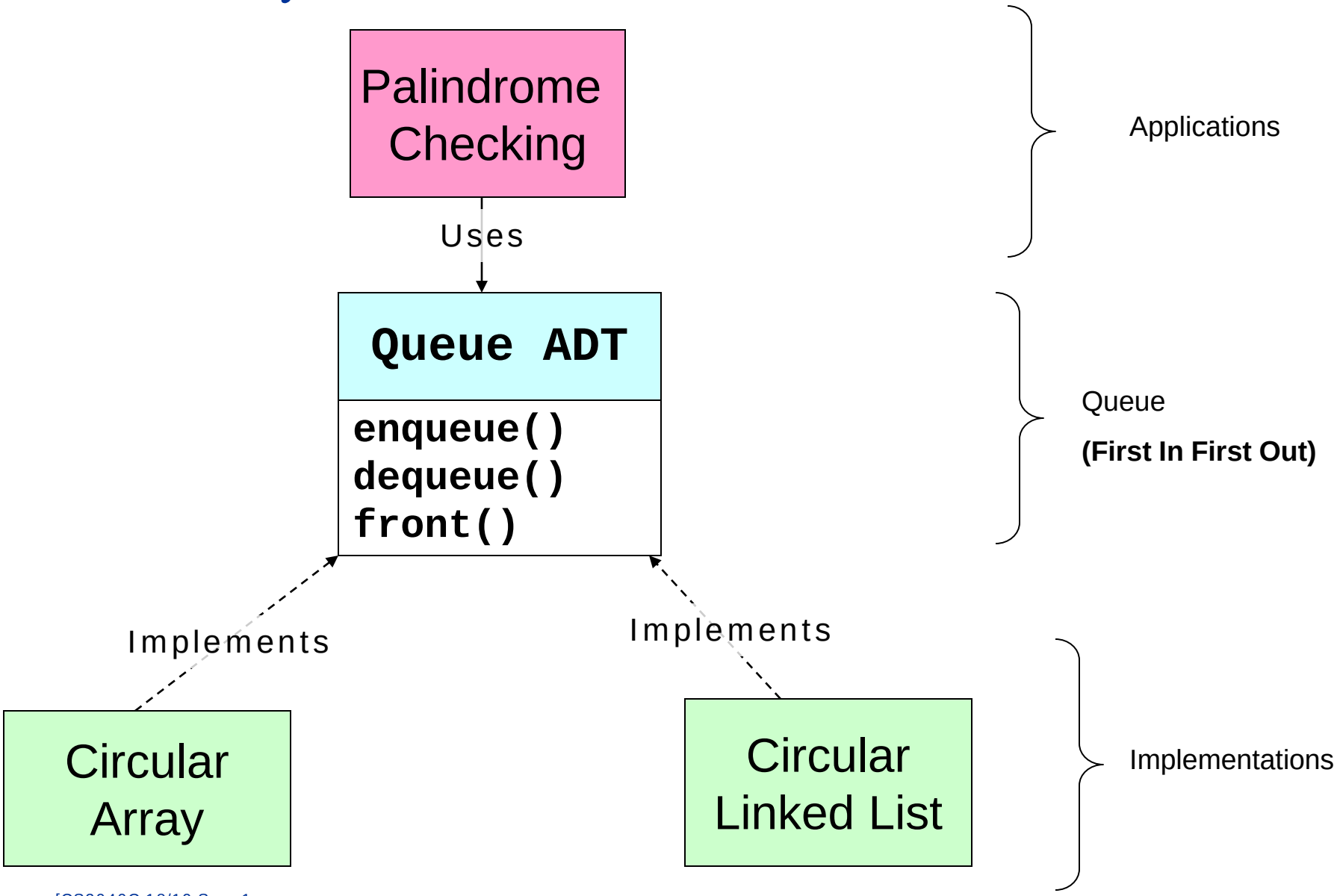
Palindrome: Implementation

```
#include <queue>
#include <stack>
using namespace std;

bool palindrome(string input) {
   stack <char> s ;
   queue <char> q ;

   for (int j=0; j < input.size(); j++) {
        s.push (input[j] );
        q.push (input[j] );
   }
    while (!s.empty()) {
        if ( s.top() != q.front() )
            return false;
        s.pop();
        q.pop();
    }
    return true;
}
```

Push the same character into both queue and stack

Queue has the original sequence, Stack has the reversed sequence. Compare to make sure they are the same

Summary


Palindrome Checking
Uses
Queue ADT
enqueue()
dequeue()
front()
Implements
Implements
Circular Array
Circular Linked List
Applications
Queue
(First In First Out)
Implementations