Problem Solving with ADT
--Queues
Queue ADT

- A queue is a FIFO: first in, first out

<table>
<thead>
<tr>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>front</td>
</tr>
<tr>
<td>back</td>
</tr>
<tr>
<td>items</td>
</tr>
<tr>
<td>createQueue()</td>
</tr>
<tr>
<td>destroyQueue()</td>
</tr>
<tr>
<td>isEmpty()</td>
</tr>
<tr>
<td>enqueue()</td>
</tr>
<tr>
<td>dequeue()</td>
</tr>
<tr>
<td>getFront()</td>
</tr>
</tbody>
</table>
Application: Recognizing Palindromes

- Palindrome is a string of characters that reads the same from left to right as it does from right to left
  - Madam, I’m Adam
  - A man, a plan, a canal, Panama
Example

String: abcbd

Queue:
- a
- b
- c
- b
- d
  - front
  - back

Stack:
- d
- b
- c
- b
- a
  - top
Pseudocode

```cpp
aQueue.createQueue();
aStack.createStack();
while (not end){
    cin >> ch;
    aQueue.enqueue(ch);
    aStack.push(ch);
}
isPal = true;
While(!aQueue.isEmpty() && isPal){
    c1 = aQueue.getFront();
    c2 = aStack.getTop();
    if (c1==c2){
        aQueue.dequeue();
        aStack.pop();
    } else {
        isPal = false;
    }
}
class Queue {
    Public:
        ...
    Private:
        struct QueueNode{
            QueueItem items;
            QueueNode *next;
        };
        QueueNode *backPtr;
        QueueNode *frontPtr;
};
void Queue::enqueue(newItem) {
    QueueNode *newPtr = new QueueNode;
    newPtr->item = newItem;
    newPtr->next = NULL;
    if (isEmpty())
        frontPtr = newPtr;
    else
        backPtr->next = newPtr;
    backPtr = newPtr;
}
void Queue::dequeue(){
    if (isEmpty())
        exit(0);
    QueueNode *temp = frontPtr;
    if (frontPtr==backPtr) {
        frontPtr = NULL;
        backPtr = NULL;
    } else
        frontPtr = frontPtr->next;
    delete temp;
}
Array-Based Implementation

class Queue {
public:
  ...
private:
  QueueItem items[MAX_QUEUE];
  int front, back;
};
Problem with Array-based Implementation

• Rightward shift
  – Queue is full even if it contains few items!
• Solution: shift array items to the left
  – It would dominate the cost of implementation.

• Any good idea?
Solution

- View array as circular
How to Insert and Remove?

- enqueue()
  - Move `back` index clockwise

- dequeue()
  - Move `front` index clockwise
Special Cases

- Queue has one item
  - front == back
- Queue is empty
  - front is one item ahead
Special Cases

- Queue with single empty slot
- Queue is full
  - front is one item ahead, again??
Distinguish Between Empty and Full

- Keep a count of the number of items
- Before enqueue, check if count == MAX_QUEUE
- Before dequeue, check if count == 0

```cpp
class Queue {
public:
    ...
private:
    QueueItem items[MAX];
    int front, back;
    int count;
};
```
Array-Based Implementation

```cpp
void Queue::Queue()
    : front(0), back(MAX-1),
    count(0) {
}

bool Queue::isEmpty(){
    return (count==0);
}

item Queue::getFront(){
    if (isEmpty())
        exit(0);
    else
        return items[front];
}
```
Array-Based Implementation

```c
void Queue::enqueue(Item newItem) {
    if (count==MAX)
        exit(0);
    else {
        back = (back+1)%MAX;
        items[back] = newItem;
        ++count;
    }
}

void Queue::dequeue(){
    if (isEmpty())
        exit(0);
    else{
        front = (front+1)%MAX;
        --count;
    }
}
```
ADT List-Based Implementation

```cpp
class Queue {
public:
    ...
private:
    List aList;
};
```
ADT List-Based Implementation

```cpp
Queue::Queue (const Queue& Q) :
:aList(Q.aList) {}

bool Queue::isEmpty() {
    return aList.isEmpty();
}

void Queue::getFront() {
    if (aList.isEmpty())
        exit(0);
    return (aList.retrieve(1));
}
```
void Queue::enqueue() {
    aList.insert(aList.getLength()+1, newItem);
}

void Queue::dequeue() {
    if (aList.isEmpty())
        exit(0);
    aList.remove(1);
}