Stack and Queue

Lecture 08

Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

Adapted partially from Data Structures and Algorithms in Java, M.T. Goodrich, R.Tamassia and M. H. Goldwasser, Sixth Edition,Wiley; Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning

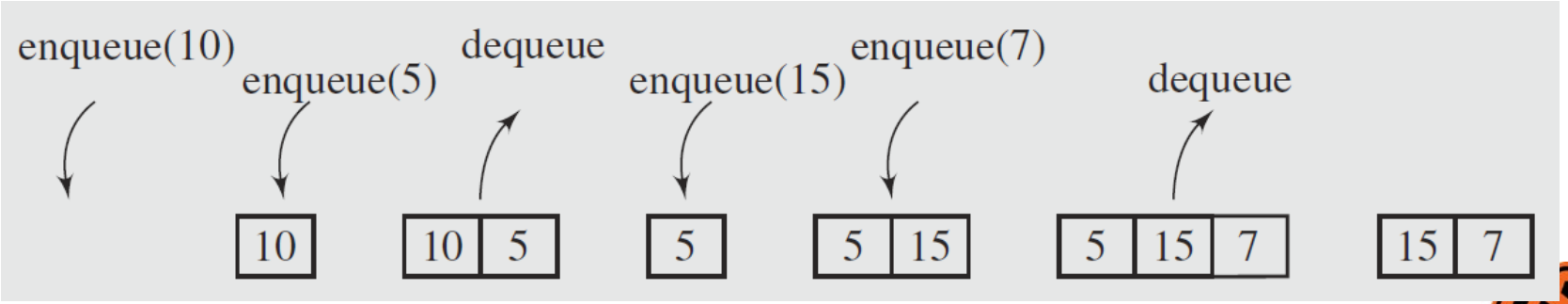
Queues

- A ***queue***??
  - a waiting line
    - grows by adding elements to its end
    - shrinks by taking elements from its front

    } both ends are used
  - use both ends with additions restricted to one end (the ***rear***) and deletions to the other (the ***front***)
    - the last element can be removed when all preceding elements are removed
  - ***first-in first-out (FIFO)*** structure

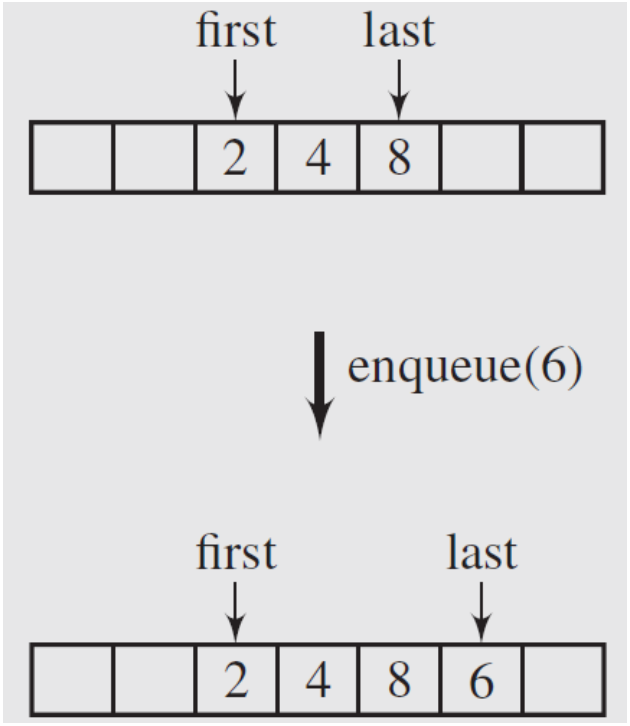
Queues (cont.)

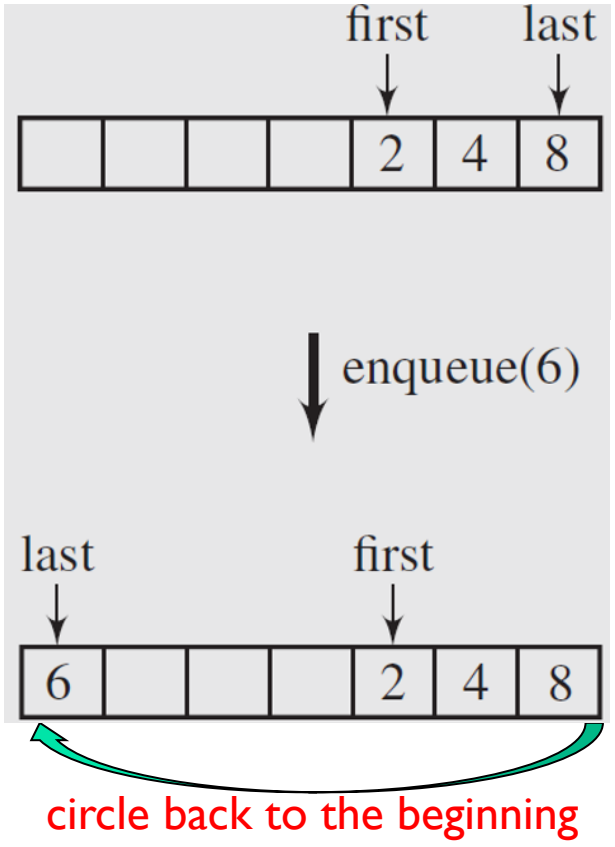
- Queue operations:
  - ***clear( )***: clears the queue
  - ***isEmpty( )***: determines if the queue is empty
  - ***enqueue(el)***: adds the data item ***el*** to the end of the queue
  - ***dequeue( )***: removes the element from the front of the queue
  - ***firstEl( )***: returns the value of the first element of the queue **without removing it**
- E.g., a series of ***enqueues*** and ***dequeues***

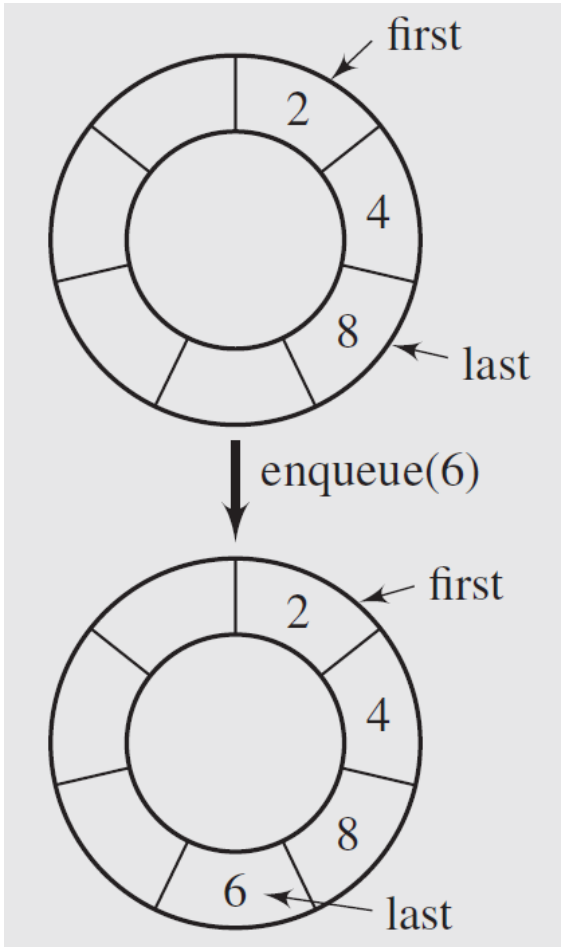


Queues (cont.)

- Implementing using an ***array*** (*not the best option*)
  - treats the array as though it were **circular**

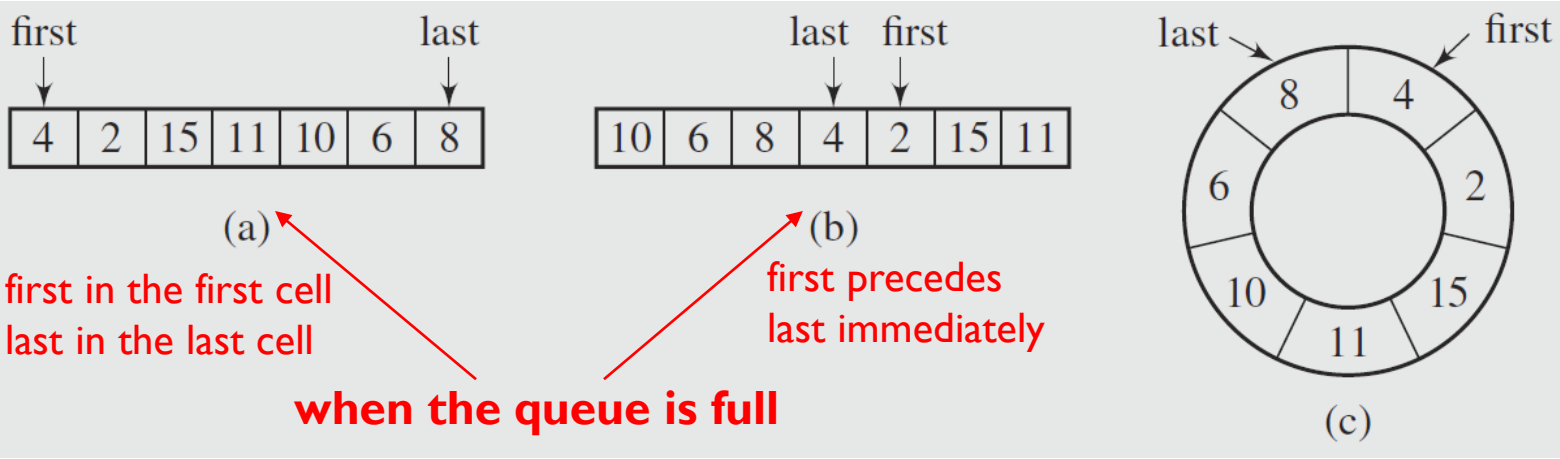






Queues (cont.)

- Implementing using an ***array*** (*not the best option*)
  - when is a queue ***full***??



Queues: Array Implementation

```
3  #ifndef ARRAY_QUEUE
4  #define ARRAY_QUEUE
5
6  template<class T, int size = 100>
7  class ArrayQueue {
8  public:
9      ArrayQueue() {
10         first = last = -1;
11     }
12     void enqueue(T);
13     T dequeue();
14     bool isFull()  {
15         return first == 0 && last == size-1 || first == last + 1;
16     }
17     bool isEmpty() {
18         return first == -1;
19     }
20 private:
21     int first, last;
22     T storage[size];
23 };
```

class template: a class defines something that is independent of the data type

ref: https://www.geeksforgeeks.org/templates-cpp/

size of queue

two possible scenarios for full queue

two "pointers" for queue

Queues: Array Implementation

```
template<class T, int size>
void ArrayQueue<T,size>::enqueue(T el) {
    if (!isFull())
        if (last == size-1 || last == -1) {
            storage[0] = el;
            last = 0;
            if (first == -1)
                first = 0;
        }
        else storage[++last] = el;
    else cout << "Full queue.\n";
}
```

check to see whether circle back

circle back to the beginning of array

queue is empty before enqueue?

not circle back

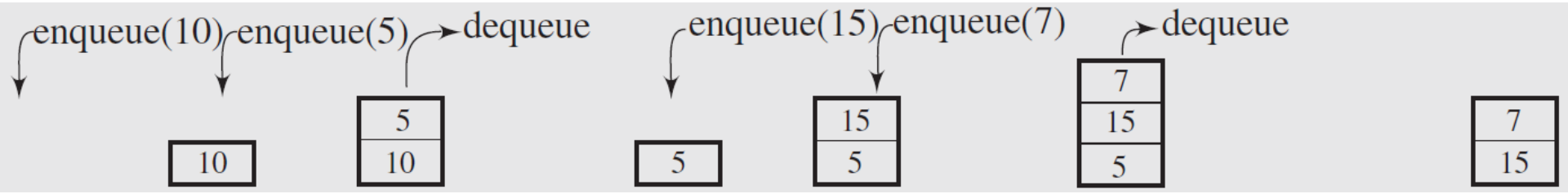
```
template<class T, int size>
T ArrayQueue<T,size>::dequeue() {
    T tmp;
    tmp = storage[first];
    if (first == last)
        last = first = -1;
    else if (first == size-1)
        first = 0;
    else first++;
    return tmp;
}
```

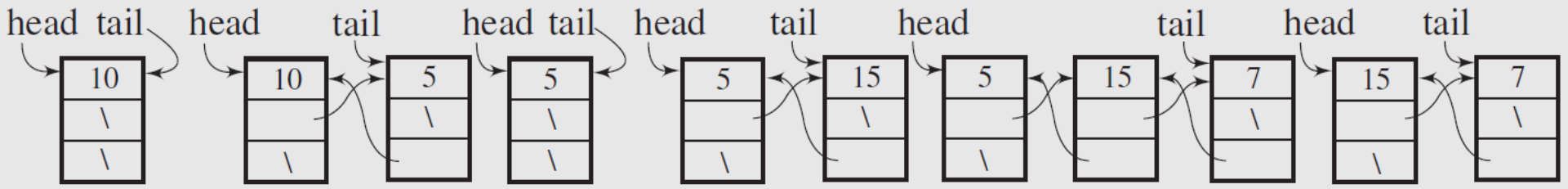
one element?

the last element of array is the first element of queue?

Queues (cont.)

- Implementing queue using
  - array (e.g., vector) – enqueue and dequeue "**pointers**"
  - ***double linked list*** with "***head***" and "***tail***" ***pointers***
    - inserting at the end of list
    - deleting at the beginning of list





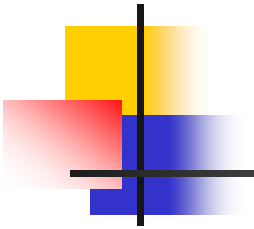
Queues: Linked List Implementation

list::member functions

ref: https://cplusplus.com/reference/list/list/

```
#include <list>

template<class T>
class Queue {
public:
    Queue() {
    }
    void clear() {
        lst.clear();
    }
    bool isEmpty() const {
        return lst.empty();
    }
    T& front() {
        return lst.front();
    }
    T dequeue() {
        T el = lst.front();
        lst.pop_front();
        return el;
    }
    void enqueue(const T& el) {
        lst.push_back(el);
    }
private:
    list<T> lst;
};
```



Queues (cont.)

- Used in a wide variety of applications
  - especially in studies of service simulations
  - e.g., a very advanced body of mathematical theory, called ***queuing theory***
    - various scenarios are analyzed and models are built that use queues

Priority Queues

- In some circumstances,
  - ***priorities*** associated the elements of the queue → affect the order of processing
- A ***priority queue*** ??
  - elements are removed based on priority and position
  - difficulty in implementing such a structure
    - trying to accommodate the priorities while still maintaining efficient enqueuing and dequeuing