CPSC 131
Data Structures Concepts

Dr. Anand Panangadan
apanangadan@fullerton.edu

Goals for today

- The Queue data structure

Queues

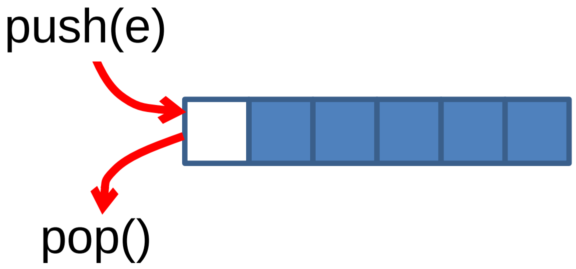
- Two ways of working with data structures
  1. Using a data structure
  2. Implementing a data structure

Queues

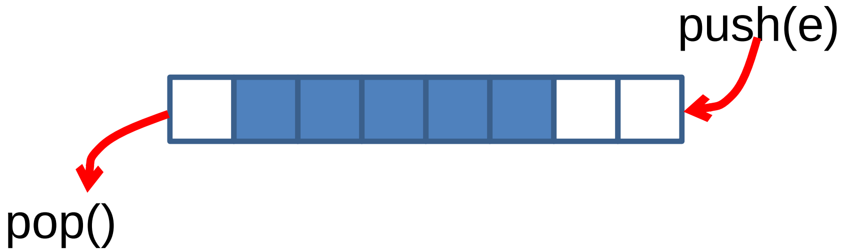
- A Queue stores arbitrary objects
- Insertions and deletions follow the first-in first-out scheme (FIFO): compare with stacks
- Think of a queue

Stack vs Queue

- Stack (Last-In First-Out)



- Queue (First-In First-Out)



Queue operations

- Push(x):
  - inserts x at end of the queue
- Pop ():
  - removes item at front of queue
- Peek():
  - returns the item at front but does not remove it
- IsEmpty():
  - returns true if the queue has no items
- GetLength():
  - Returns the number of items in the queue

Errors

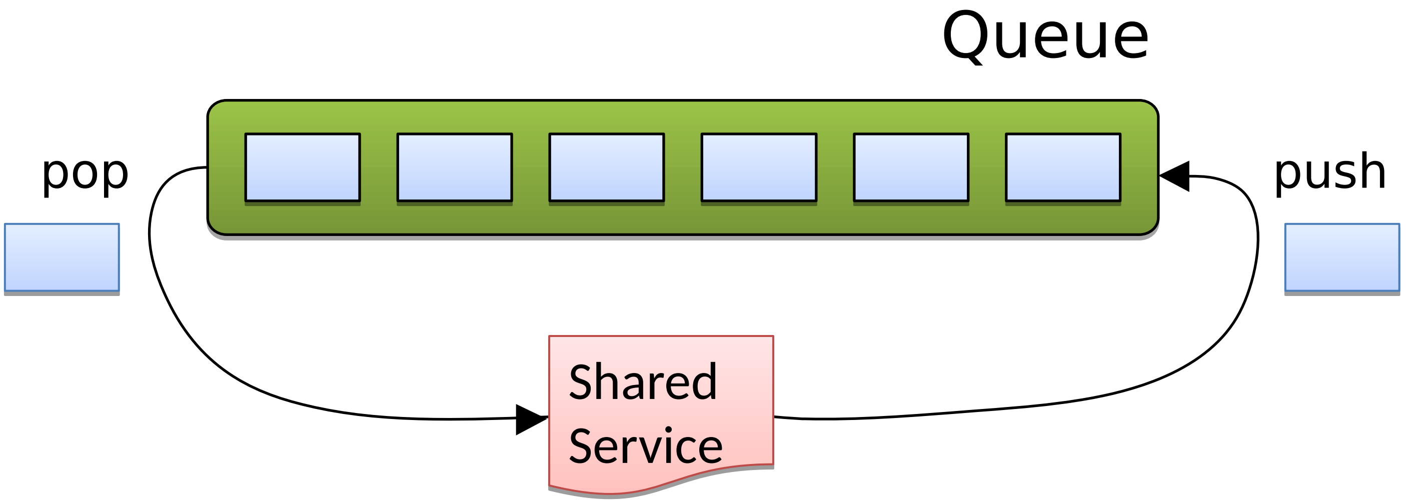
- In a queue, operations Pop() and Peek() cannot be performed if the queue is empty
- Attempting Pop() and Peek() on an empty queue should be caught
  - In C++, use exceptions

Applications of Queues

- Direct applications
  - Waiting lists, bureaucracy
  - Access to shared resources (e.g., printer)
  - Multiprogramming
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

Application: Round Robin Schedulers

- We can implement a round robin scheduler using a queue Q by repeatedly performing the following steps:
  1. e = Q.peek(); Q.pop()
  2. Service element e
  3. Q.push(e)



Queue in C++

```
template <typename E>
class Queue {
public:
   void push (const E& e);
   void pop ();
   E& peek();
      int getlength();
      bool isempty();
}
```

Implementing queues

1. Linked lists ("Wrapper class" implementation)
   - "Wrap a queue around a (singly) linked list"

2. Arrays

Code on GitHub

https://github.com/CSUF-CPSC-131-Spring2019/Data-Structures-Code/

Has both implementations (.h files) and simple main programs to test them (.cpp files)

C++ STL implementation

- C++ Standard Library
- Contains highly optimized implementations of commonly used data structures
  - Including queues
- http://www.cplusplus.com/reference/queue/queue/

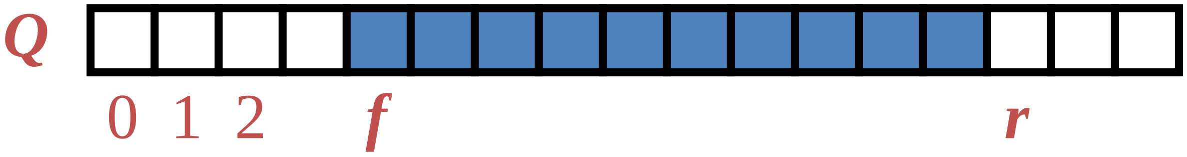
std::queue

ZyBook	std::queue
Push	push()
Pop	pop()
Peek	front()
GetLength	size()
IsEmpty	empty()

```
#include <queue>

int main() {
  std::queue<int> ds;
  ds.push(10);
  ds.push(20);
  ds.pop();
  cout << ds.front();
}
```

Array-based Queue

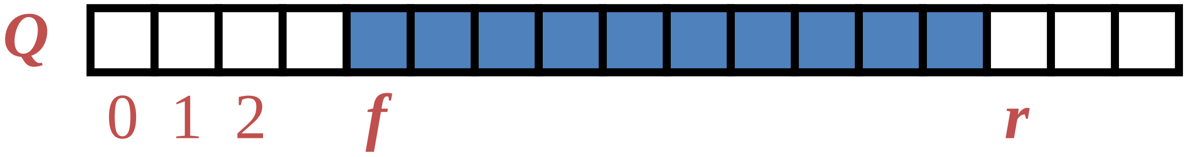
- Use *two* integers to keep track of front and rear of the queue
  - f: index of the front element
  - r: index of the empty location where the next element will enter (the rear of the queue)



Array-based Queue

- Use third integer, *n*, to determine size and emptiness

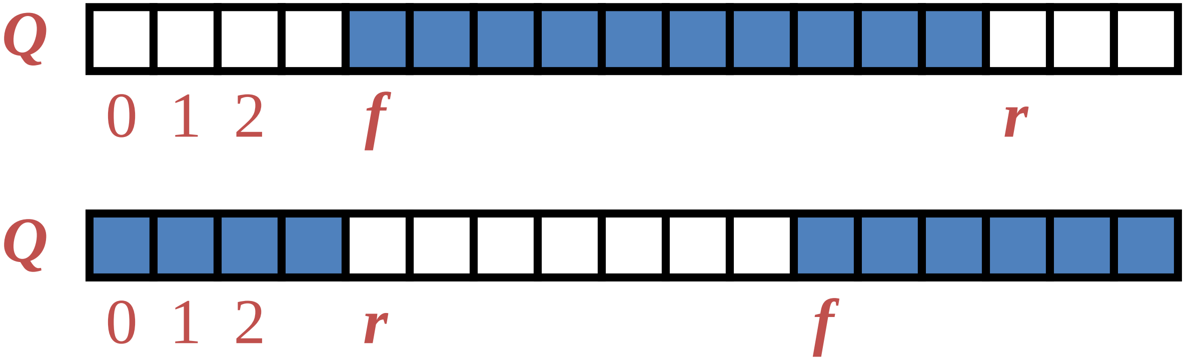
```
**Algorithm** *size*()
    **return** *n*

**Algorithm** *empty*()
    **return** (*n* == 0)
```



Array-based Queue

- Operation Push() throws an exception if the array is full

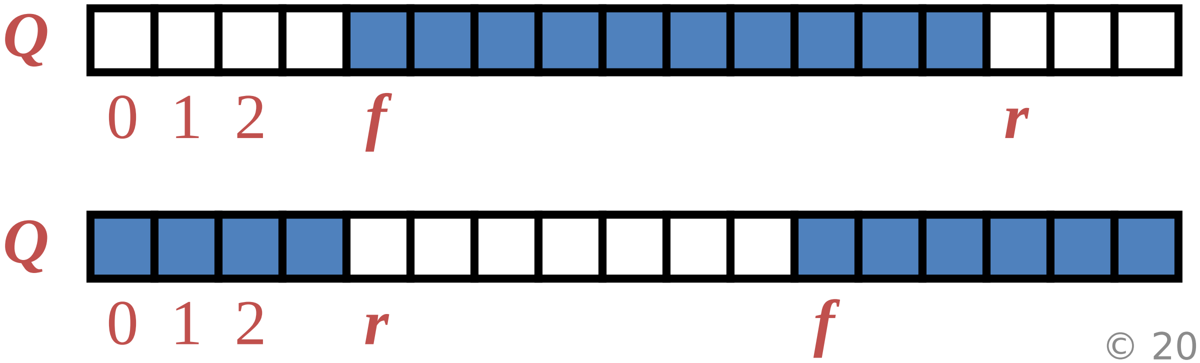
```
Algorithm Push(value)
    if GetLength() == N - 1 then
        throw QueueFull
    else
        Q[r] ← value
        r ← (r + 1) mod N
        n ← n + 1
```



Array-based Queue

- Operation Pop() throws an exception if the queue is empty
- This exception is specified in the queue ADT

```
Algorithm Pop()
    if IsEmpty() then
        throw QueueEmpty
    else
        $f \leftarrow (f + 1) \bmod N$
        $n \leftarrow n - 1$
```


Q
0 1 2
f
r
Q
0 1 2
r
f