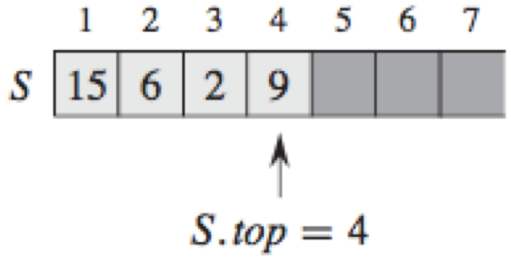
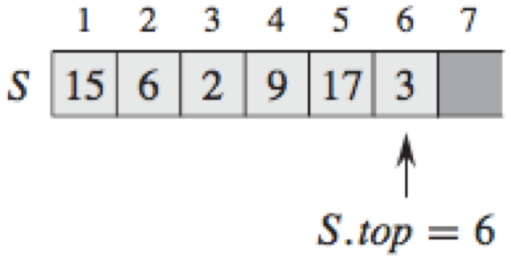
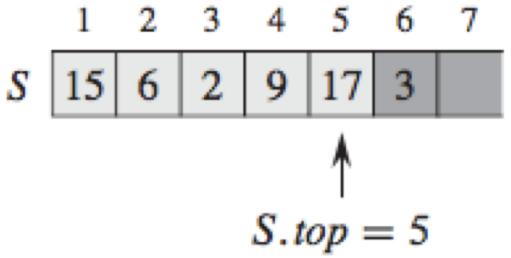
Introduction to the Stack ADT

Static Stack: fixed size, implemented as array





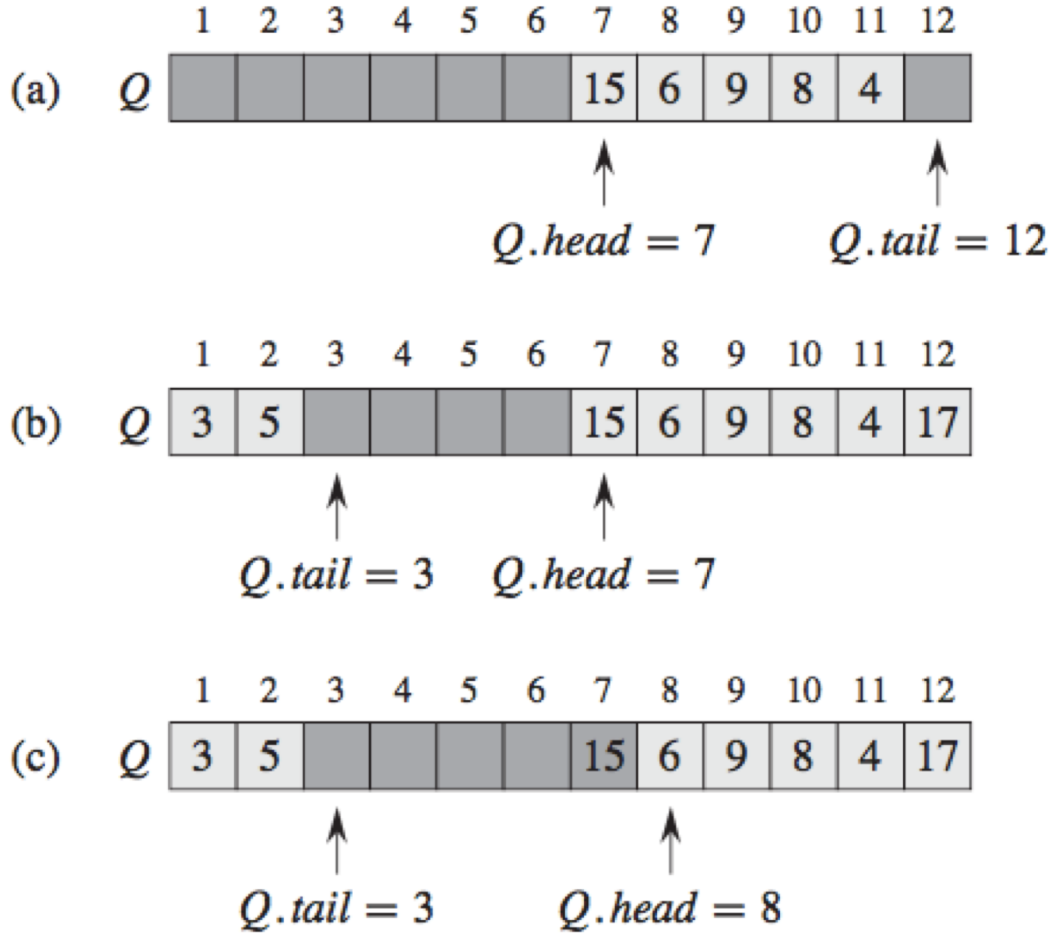


PUSH(S, x)

POP(S)

STACK-EMPTY(S)

Static Queue: fixed size, implemented as array



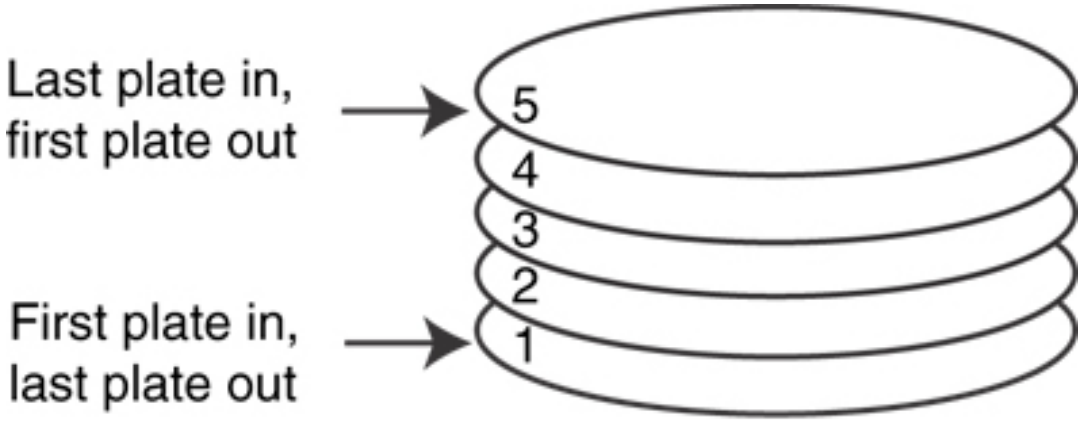
ENQUEUE(Q, x)

DEQUEUE(Q)

Introduction to the Stack ADT

- Stack: a LIFO (last in, first out) data structure
- Examples:
  - plates in a cafeteria
  - return addresses for function calls
- Implementation:
  - static: fixed size, implemented as array
  - dynamic: variable size, implemented as linked list

A LIFO Structure

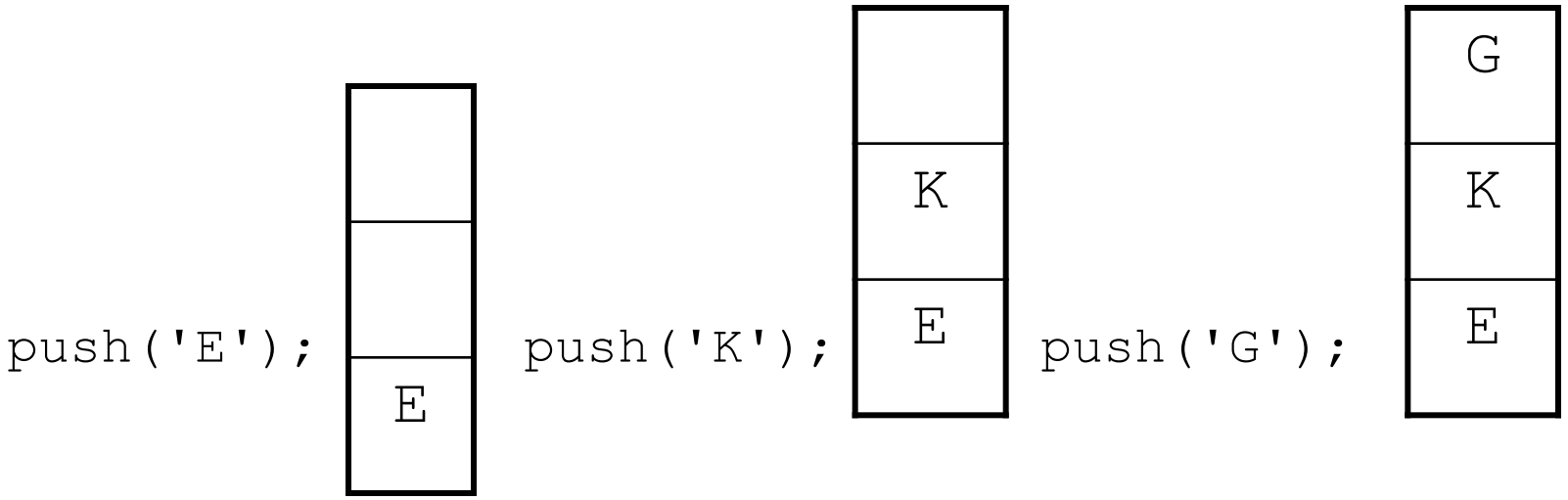


Stack Operations and Functions

- Operations:
  - push: add a value onto the top of the stack
  - pop: remove a value from the top of the stack
- Functions:
  - `isFull`: `true` if the stack is currently full, *i.e.*, has no more space to hold additional elements
  - isEmpty: `true` if the stack currently contains no elements

Stack Operations - Example

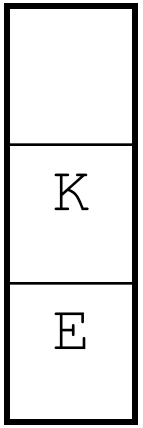
- A stack that can hold `char` values:



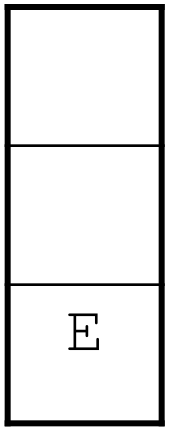
Stack Operations - Example

- A stack that can hold `char` values:

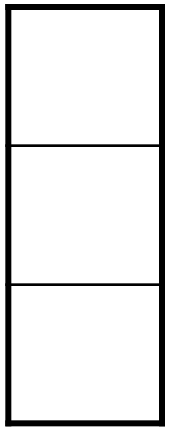
`pop();`
(remove `G`)



`pop();`
(remove `K`)



`pop();`
(remove `E`)



Dynamic Stacks

- Grow and shrink as necessary
- Can't ever be full as long as memory is available
- Implemented as a linked list

Introduction to the Queue ADT

Introduction to the Queue ADT

- Queue: a FIFO (first in, first out) data structure.
- Examples:
  - people in line at the theatre box office
  - print jobs sent to a printer
- Implementation:
  - **static**: fixed size, implemented as **array**
  - **dynamic**: variable size, implemented as **linked** list

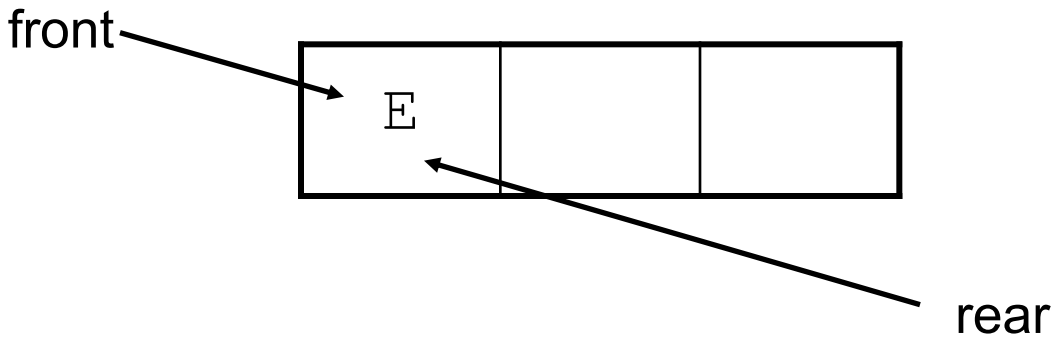
Queue Locations and Operations

- **rear**: position where elements are added
- **front**: position from which elements are removed
- **enqueue**: add an element to the rear of the queue
- **dequeue**: remove an element from the front of a queue

Queue Operations - Example

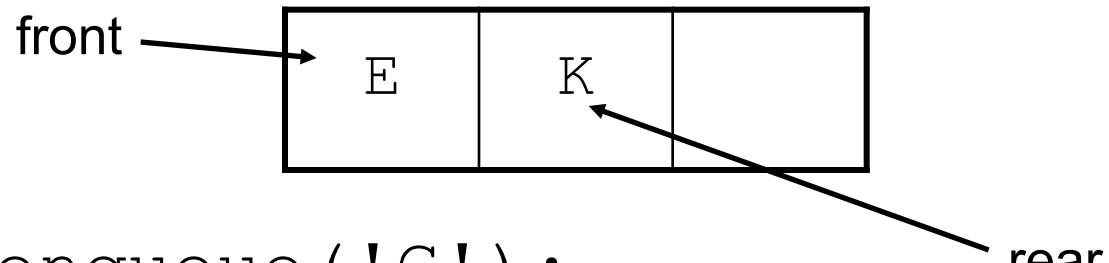
- A currently empty queue that can hold `char` values:

- `enqueue('E');`

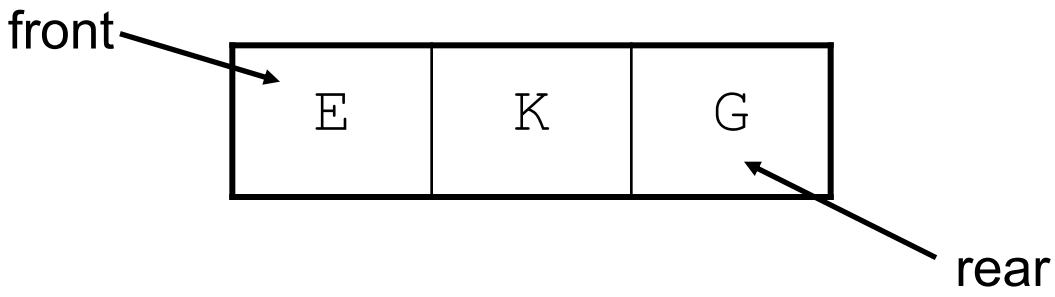


Queue Operations - Example

- `enqueue('K');`

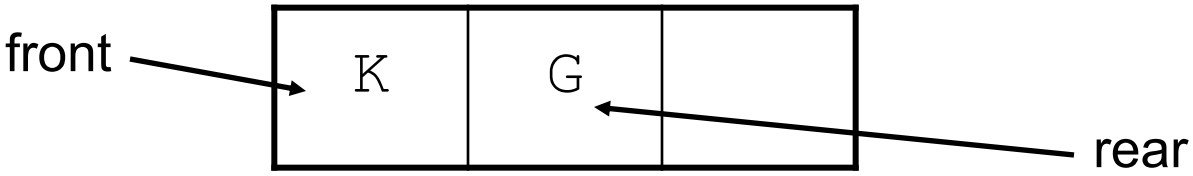


- `enqueue('G');`

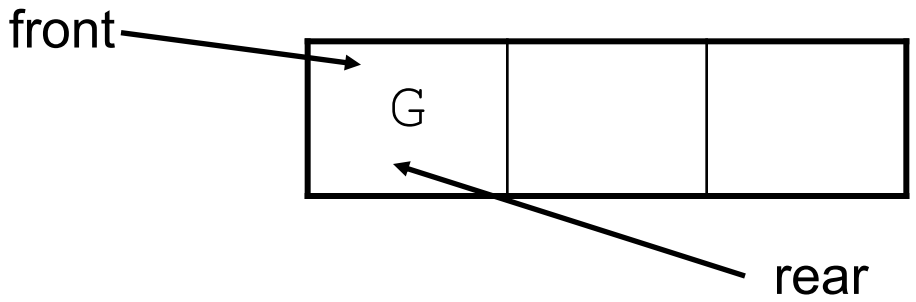


Queue Operations - Example

- `dequeue(); // remove E`



- `dequeue(); // remove K`



`dequeue` Issue, Solutions

- When removing an element from a queue, remaining elements must shift to front

- Solutions:
  - Let front index move as elements are removed (works as long as rear index is not at end of array)
  - Use above solution, and also let rear index "wrap around" to front of array, treating array as **circular** instead of linear

Dynamic Queues

- Like a stack, a queue can be implemented using a linked list
- Allows dynamic sizing, avoids issue of shifting elements or wrapping indices