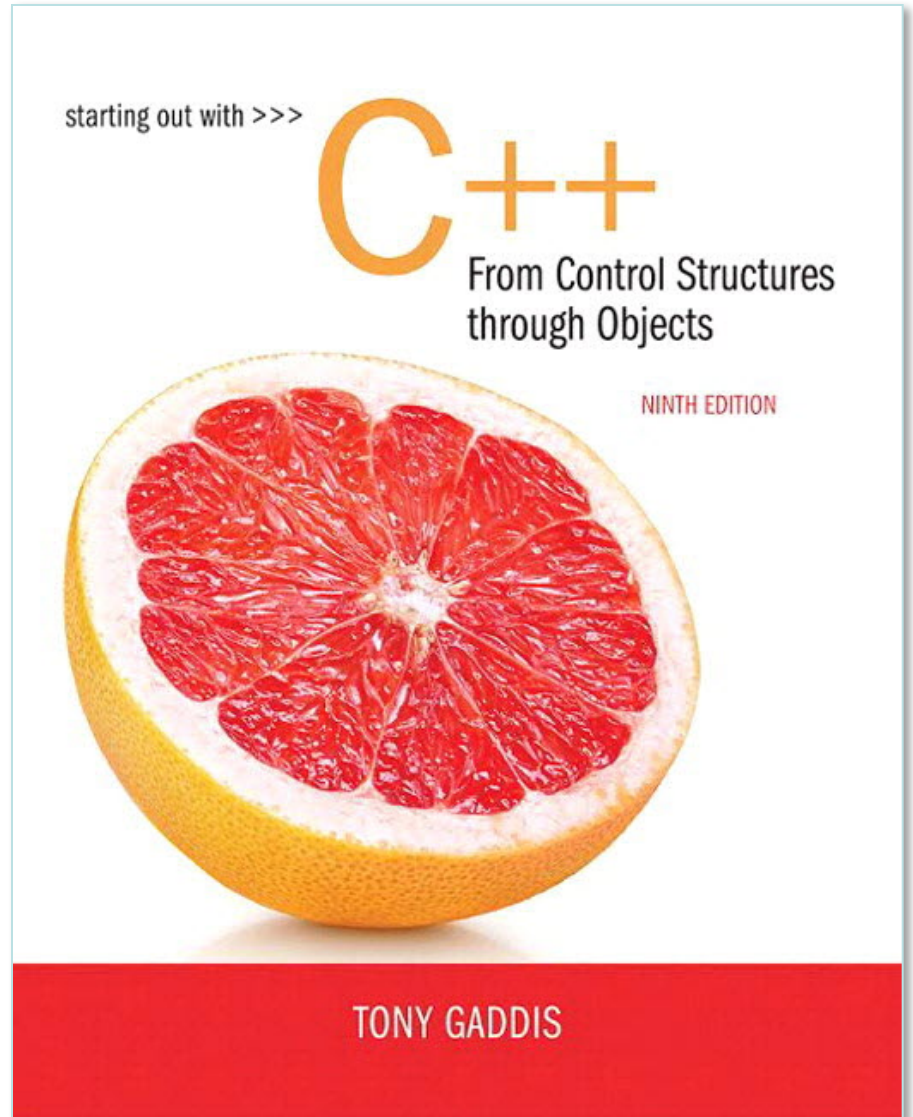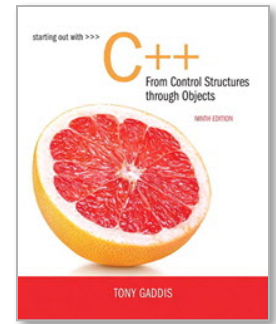# Chapter 19:

Stacks and Queues

19.1

Introduction to the Stack ADT

Introduction to the Stack ADT

- <u>Stack</u>: 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

Last plate in, first plate out → 5
4
3
2
First plate in, last plate out → 1
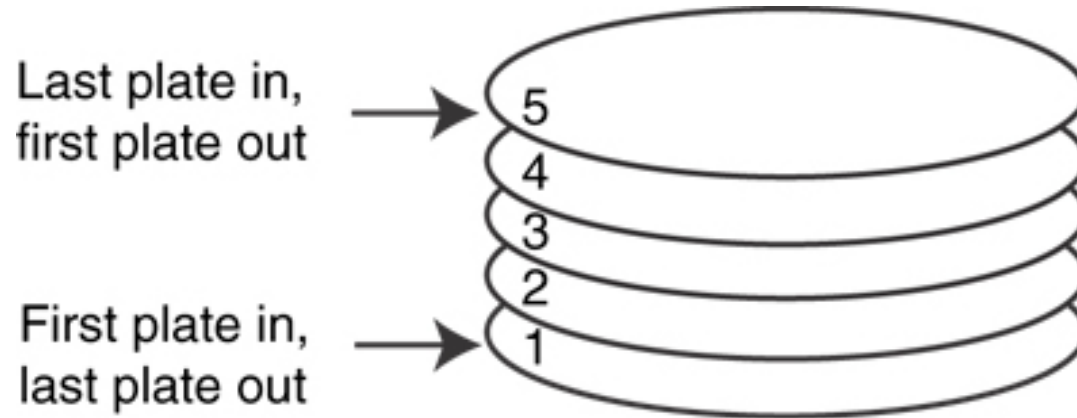
Stack Operations and Functions

- 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
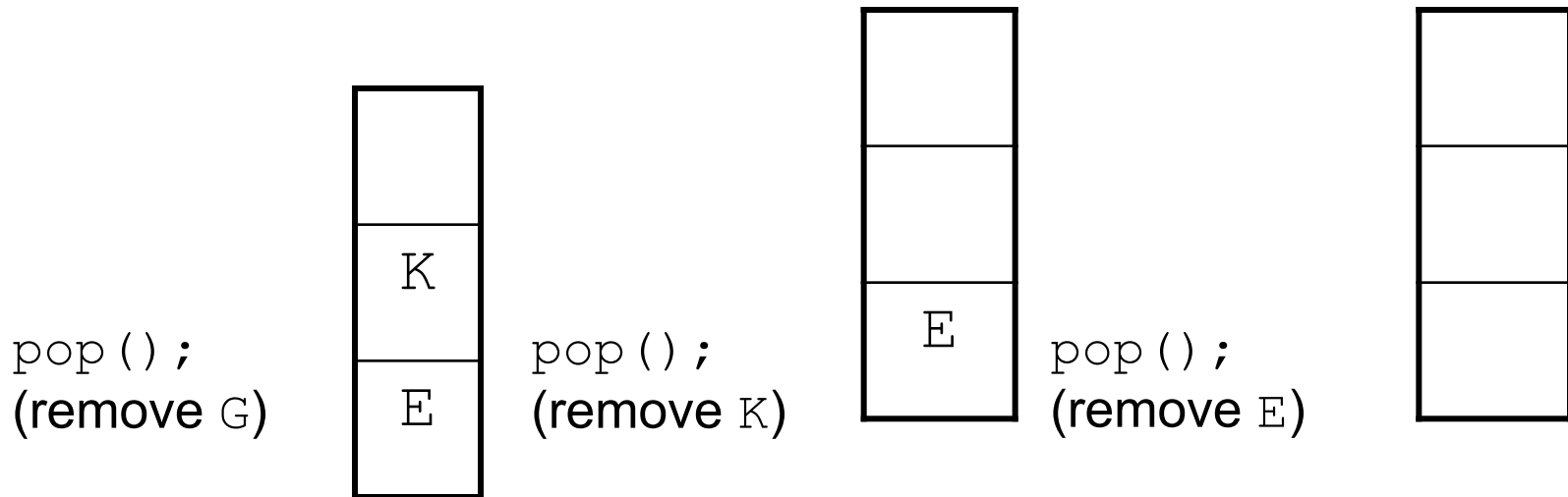 - isEmpty: `true` if the stack currently contains no elements

Stack Operations - Example

- A stack that can hold `char` values:

```
push('E');    push('K');    push('G');
```

Stack Operations - Example

- A stack that can hold `char` values:

```
pop();
(remove G)
```
```
pop();
(remove K)
```
```
pop();
(remove E)
```

K
E

| |
|:-:|
| |
| |
| E |
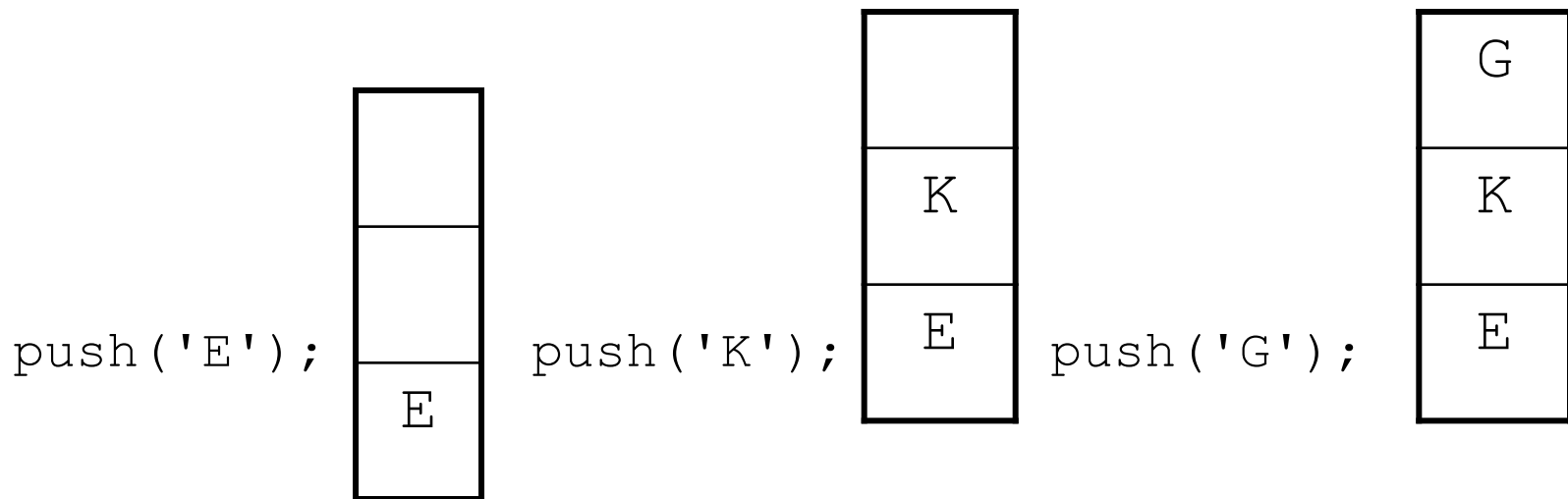
Contents of `IntStack.h`

```
1   // Specification file for the IntStack class
2   #ifndef INTSTACK_H
3   #define INTSTACK_H
4
5   class IntStack
6   {
7   private:
8       int *stackArray;   // Pointer to the stack array
9       int stackSize;     // The stack size
10      int top;           // Indicates the top of the stack
11
12  public:
13      // Constructor
14      IntStack(int);
15
16      // Copy constructor
17      IntStack(const IntStack &);
18
19      // Destructor
20      ~IntStack();
21
22      // Stack operations
23      void push(int);
24      void pop(int &);
25      bool isFull() const;
26      bool isEmpty() const;
27  };
28  #endif
```

*(See `IntStack.cpp` for the implementation.)*
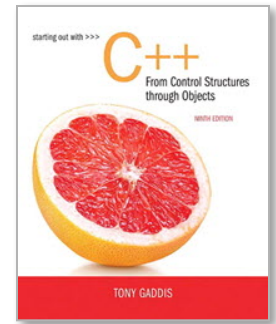
19.2

Dynamic Stacks

Dynamic Stacks

- Grow and shrink as necessary

- Can't ever be full as long as memory is available

- Implemented as a linked list

Implementing a Stack

- Programmers can program their own routines to implement stack functions

- See `DynIntStack` class in the book for an example.

- Can also use the implementation of stack available in the STL
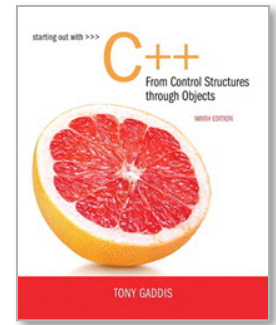
19.3

The STL `stack` Container

The STL `stack` container

- Stack template can be implemented as a `vector`, a linked list, or a `deque`
- Implements `push`, `pop`, and `empty` member functions
- Implements other member functions:
  - `size`: number of elements on the stack
  - `top`: reference to element on top of the stack

Defining a `stack`

- Defining a stack of `char`s, named `cstack`, implemented using a `vector`:

  ```
  stack< char, vector<char>> cstack;
  ```

- implemented using a list:
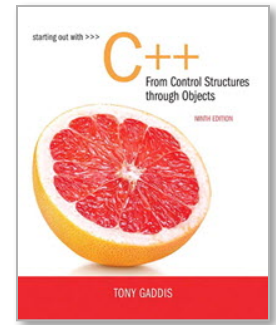
  ```
  stack< char, list<char>> cstack;
  ```

- implemented using a `deque`:

  ```
  stack< char > cstack;
  ```

- When using a compiler that is older than C++ 11, be sure to put spaces between the angled brackets that appear next to each other.

  ```
  stack< char, vector<char> > cstack;
  ```

19.4

Introduction to the Queue ADT

Introduction to the Queue ADT

- <u>Queue</u>: 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
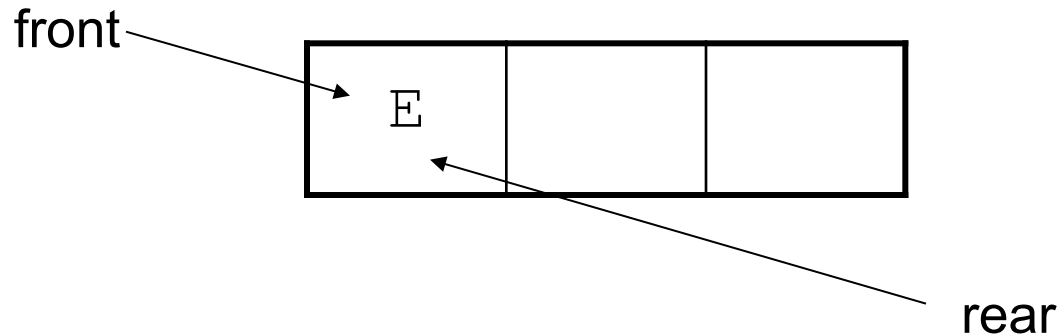
Queue Locations and Operations

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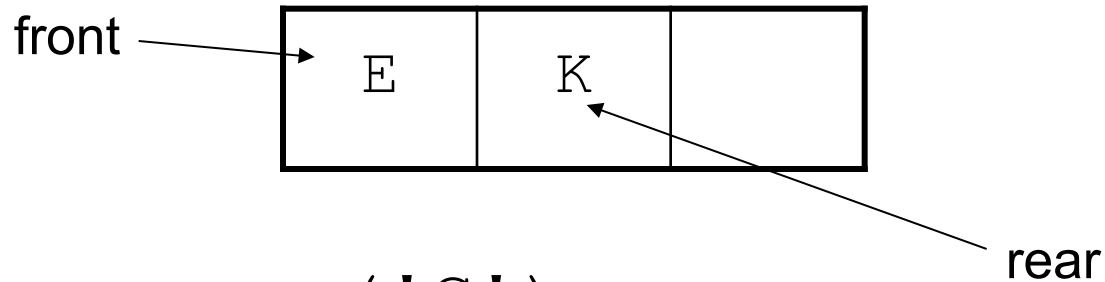
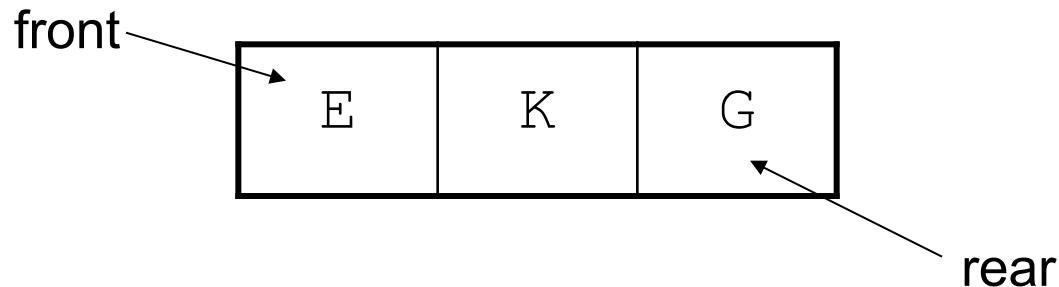- A currently empty queue that can hold `char` values:

- `enqueue('E');`

front

E

rear

Queue Operations - Example

- `enqueue('K');`

front ──→ | E | K | |

rear

- `enqueue('G');`

front ──→ | E | K | G |

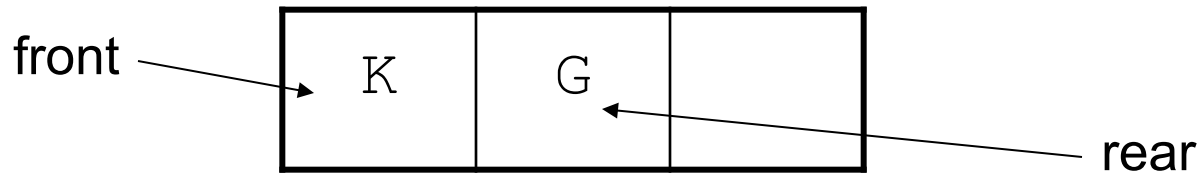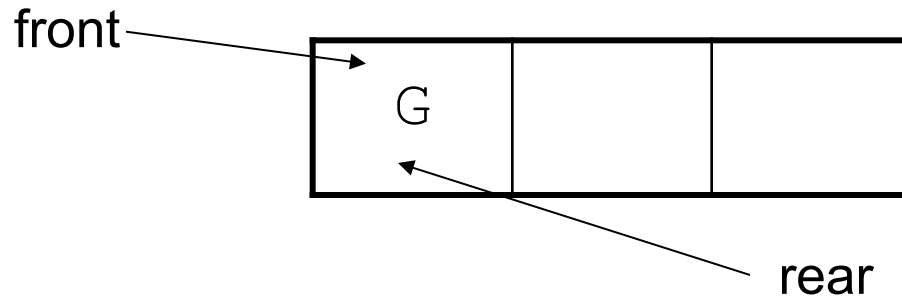rear

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 (more complex enqueue, dequeue code)

Contents of `IntQueue.h`

```
1   // Specification file for the IntQueue class
2   #ifndef INTQUEUE_H
3   #define INTQUEUE_H
4
5   class IntQueue
6   {
7   private:
8       int *queueArray;    // Points to the queue array
9       int queueSize;      // The queue size
10      int front;          // Subscript of the queue front
11      int rear;           // Subscript of the queue rear
12      int numItems;       // Number of items in the queue
```

Contents of `IntQueue.h` (Continued)

```
13   public:
14      // Constructor
15      IntQueue(int);
16
17      // Copy constructor
18      IntQueue(const IntQueue &);
19
20      // Destructor
21      ~IntQueue();
22
23      // Queue operations
24      void enqueue(int);
25      void dequeue(int &);
26      bool isEmpty() const;
27      bool isFull() const;
28      void clear();
29   };
30   #endif
```

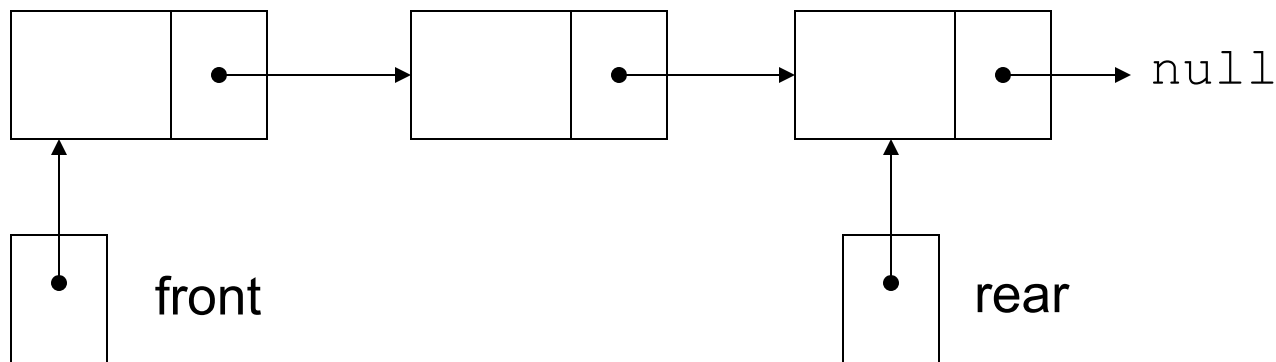(See `IntQueue.cpp` for the implementation)
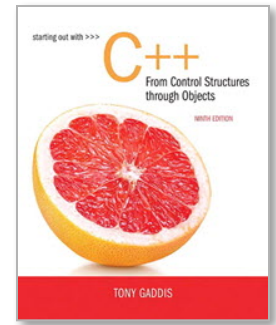
19.5

Dynamic Queues

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

front

rear

Implementing a Queue

- Programmers can program their own routines to implement queue operations

- See the `DynIntQue` class in the book for an example of a dynamic queue

- Can also use the implementation of queue and dequeue available in the STL
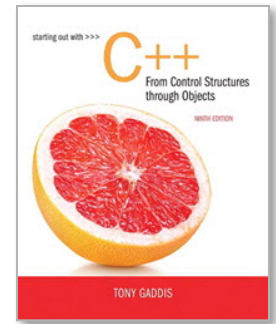
19.6

The STL `deque` and `queue` Containers

The STL `deque` and `queue` Containers

- `deque`: a double-ended queue. Has member functions to enqueue (`push_back`) and dequeue (`pop_front`)
- `queue`: container ADT that can be used to provide queue as a `vector`, list, or `deque`. Has member functions to enque (`push`) and dequeue (`pop`)

Defining a `queue`

- Defining a queue of `char`s, named cQueue, implemented using a deque:

  ```
  deque<char> cQueue;
  ```

- implemented using a queue:

  ```
  queue<char> cQueue;
  ```

- implemented using a `list`:

  ```
  queue<char, list<char>> cQueue;
  ```