CS 106B, Lecture 6

Queues

reading:

*Programming Abstractions in C++*, Chapter 5.3
What is the Big O?

```cpp
Vector<int> v;
for (int x = 1; x <= N; x += 2) {
    v.add(x);
}
while (!v.isEmpty()) {
    cout << v.remove(0) << endl;
}
```
Plan for Today

• Another collection: the **Queue**
Queue

• What if we want to remove from the bottom instead of the top?
• We want "First In First Out" – FIFO

• Real World
  – Lines at the dining hall (No cutting!)
  – Escalators
  – Anything first-come first-serve

• Computers
  – Sending jobs to a printer
  – Call services (being put on hold)
Queue

• Queue: ADT that retrieves elements in the order they were added.
  – There are no indexes (just like a stack)
  – Can only add to the end of the queue and remove from the front.

• Operations
  – enqueue: add an element to the back
  – dequeue: remove the front element
  – peek: examine (but do NOT remove) the front element
Queue Syntax

```cpp
#include "queue.h"

Queue<string> strs;
strs.enqueue("Hello");
strs.enqueue("World");
cout << strs.peek() << endl; // "Hello"
cout << strs << endl; // {"Hello", "World"}
strs.dequeue(); // strs = {"World"}
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q.dequeue()</td>
<td>O(1)</td>
<td>removes front value and returns it; throws error if queue is empty</td>
</tr>
<tr>
<td>q.enqueue(value)</td>
<td>O(1)</td>
<td>places given value at back of queue</td>
</tr>
<tr>
<td>q.isEmpty()</td>
<td>O(1)</td>
<td>returns true of queue has no elements</td>
</tr>
<tr>
<td>q.peek()</td>
<td>O(1)</td>
<td>returns front value without removing; throws an error if queue is empty</td>
</tr>
<tr>
<td>q.size()</td>
<td>O(1)</td>
<td>returns number of elements in queue</td>
</tr>
</tbody>
</table>
Queue Question?

Queue<int> queue;
for (int i = 1; i <= 6; i++) {
    queue.enqueue(i);
}
for (int i = 0; i < queue.size(); i++) {
    cout << queue.dequeue() << " ";
}
cout << queue << " size " << queue.size() << endl;

A. 1 2 3 4 5 6 {} size 0
B. 1 2 3 {4, 5, 6} size 3
C. 1 2 3 4 5 6 {1, 2, 3, 4, 5, 6} size 6
D. none of the above
Exercise

Write a function `repeat` that accepts a queue of integers and replaces every element with two copies of itself. For example: `{1, 2, 3}` becomes `{1, 1, 2, 2, 3, 3}`
Solution

```cpp
void repeat(Queue<int>& q) {
    int size = q.size();
    for (int i = 0; i < size; i++) {
        int n = q.dequeue();
        q.enqueue(n);
        q.enqueue(n);
    }
}
```
Queue Tips

• You cannot access a queue's elements by index.
• Instead, you dequeue elements out of the queue one at a time.

// process (and empty!) an entire queue
while (!q.isEmpty()) {
    do something with q.dequeue();
}

• Be careful iterating over a queue if you are changing it.

// Save the size before changing the queue
int size = q.size();
for (int i = 0; i < size; i++) {
    // do something with q.dequeue();
}
Mixing Stacks and Queues

How can we reverse the order of elements in a queue?
**Mixing Stacks and Queues**

How can we reverse the order of elements in a queue?

```cpp
Queue<int> q {1, 2, 3};  // q={1, 2, 3}
Stack<int> s;
while (!q.isEmpty()) {
    s.push(q.dequeue());  // q={} s={1, 2, 3}
}
while (!s.isEmpty()) {
    q.enqueue(s.pop());  // q={3, 2, 1} s={}
}
cout << q << endl;
```
Exercise

Write a function **mirror** that accepts a queue of strings and appends the queue’s contents to itself in reverse order. For example:

{“a”, “b”, ”c”} becomes {“a”, “b”, “c”, “c”, “b”, ”a”}
void mirror(Queue<string>& q) {
    Stack<string> s;
    int size = q.size();
    for (int i = 0; i < size; i++) {
        string str = q.dequeue();
        s.push(str);
        q.enqueue(str);
    }
    while (!s.isEmpty()) {
        q.enqueue(s.pop());
    }
}
Deque

• Deque ("deck"): double-ended queue
  – Can add/remove from either end

• Basic Operations
  – enqueueFront, enqueueBack
  – dequeueFront, dequeueBack
  – peekFront, peekBack

• Get queue and stack functionality in one data structure!
Look Ahead

• Assignment 1 (Game of Life) due Wednesday, July 3, at 5PM. You can work in a pair.

• No class on July 4th
  – There is no section on July 4th either. This means section attendance for this week is optional. We will record a section on Wednesday, right after class in the same room.
  – We recommend if you have a section on Wednesday to still attend, and if you have a section on Thursday to watch the taped section online or stay after lecture on Wednesday.