Lecture 16:
Queues and More
## Plan For The Rest Of The Term

<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tues</th>
<th>Weds</th>
<th>Thurs</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hank OH</td>
<td>11/25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/1</td>
<td>12/2</td>
<td>12/3</td>
<td>12/4 3C extension to this day</td>
<td>12/5</td>
<td>12/6</td>
<td>12/7</td>
</tr>
<tr>
<td></td>
<td>12/2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/9</td>
<td>12/9 Final, 8am-10am</td>
<td>12/10</td>
<td>12/11 Last day to turn in work, 3D due</td>
<td>12/12</td>
<td>12/13</td>
<td>12/14</td>
</tr>
</tbody>
</table>
Inspirational Message

• This class is hard.

• If you are surviving, then it means you can get a CS degree.

• What you learn will be very useful to you.
Mental Model

Why The Projects Are Hard

Project complexity

- Unix,
- vim,
- new language (C),
- memory,
- VirtualBox,
- edit-compile-execute
d neighbouring debuggers

Week 1

Week 5

Week 10
YouTube Video Ideas?
Project grades

It was my intent that the homeworks added up to 100 points. So far the homeworks have been 63 points. To me, 3B should be worth 8 points, 3C 9 points, and 3D 12 points. This would add up to 92 points. If I wanted to make it a clean 100, it would be 3B = 10, 3C = 11, 3D = 16.

There are two perspectives on this:

(1) if someone got dinged on the early projects, they may be banking on the later projects to bring up their score. In this case 10/11/16 is best.

(2) if someone did well on the early projects, then they wouldn't want to have so many points up for grabs in the final weeks. In this case 8/9/12 is best.
Proposal: Midterm “Curve”

\[ \text{MAX} ( \quad Q1 + Q2 + Q3 + Q4 + Q5, \quad 2 \times Q1 + 1 \times Q2 + 0.2 \times Q3 + 1 \times Q4 + 0.8 \times Q5) \]

\[
\begin{align*}
2 \times 5 &= 10 \\
1 \times 5 &= 5 \\
0.2 \times 5 &= 1 \\
1 \times 5 &= 5 \\
0.8 \times 5 &= 4 \\
\rightarrow 25 \text{ points}
\end{align*}
\]

\[\begin{array}{|c|c|}
\hline
\text{Score goes up by} & \text{# of people} \\
\hline
>3 & 7 \\
2.5-2.99 & 7 \\
2-2.49 & 14 \\
1.5-1.99 & 2 \\
1-1.49 & 7 \\
0.5-0.99 & 6 \\
0.01-0.49 & 10 \\
0 & 16 \\
\hline
\end{array}\]

<table>
<thead>
<tr>
<th>Question</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>strlen</td>
</tr>
<tr>
<td>Q2</td>
<td>strdup</td>
</tr>
<tr>
<td>Q3</td>
<td>memory, not 2D</td>
</tr>
<tr>
<td>Q4</td>
<td>memory, 2D</td>
</tr>
<tr>
<td>Q5</td>
<td>Unix</td>
</tr>
</tbody>
</table>
Queues

• British definition:
  – a line or sequence of people or vehicles awaiting their turn to be attended to or to proceed

• It is just a word for a “line.”
How does a line (queue) work?

• You go to McDonalds.
• At the beginning of the day, no one is in the line
• The first person of the day arrives; that person is first in line
• More people arrive. They join the line in an order.
• The first person in line is called to a register.
• The second person becomes the first person in line, the third person in line becomes second. And so on.
Queues Vs Stacks

- Queue: first in first out (FIFO)
- Stack: last in first out (LIFO)
Queue: methods (1/2)

• What methods?
  – Initialize
  – Enqueue (enter the queue)
    • Line up at McDonald’s
  – Dequeue (exit the queue)
    • Get called to the register
Queue: methods (2/2)

• What methods?
  – Sometimes: Front
    • Look at the first person in line, but don’t get them out of line
    • Also useful to see if there is anyone in line
      – Or: IsEmpty
#define MAX_ELEMENTS 10

typedef struct
{
    int num_elements;
    int elements[MAX_ELEMENTS];
} Queue;

void Initialize(Queue *q)
{
    q->num_elements = 0;
}

void Enqueue(Queue *q, int val)
{
    if (q->num_elements+1 >= MAX_ELEMENTS) { /* ERROR */ }
    q->elements[q->num_elements] = val;
    q->num_elements++;
}
Enqueue In Action

```c
void Enqueue(Queue *q, int val) {
    if (q->num_elements+1 >= MAX_ELEMENTS) { /* ERROR */ }
    q->elements[q->num_elements] = val;
    q->num_elements++;
}

int main() {
    Queue q;
    Initialize(&q);
    int x;
    Enqueue(&q, 10); /* q.elements[0] = 10; q.num_elements = 1; */
    Enqueue(&q, -5); /* q.elements[1] = -5; q.num_elements = 2; */
    Enqueue(&q, 14); /* q.elements[2] = 14; q.num_elements = 3; */
    Enqueue(&q, -8); /* q.elements[3] = -8; q.num_elements = 4; */
```
What About Dequeue?

```c
int Dequeue(Queue *q) {
    if (q->num_elements == 0) { /* ERROR */ }
    int rv = q->elements[0];
    for (int i = 0 ; i < q->num_elements-1 ; i++)
        q->elements[i] = q->elements[i+1];
    q->num_elements--;  
    return rv;
}
```
Dequeue In Action

```c
int Dequeue(Queue *q)
{
    if (q->num_elements == 0) { /* ERROR */ }
    int rv = q->elements[0];
    for (int i = 0 ; i < q->num_elements-1 ; i++)
        q->elements[i] = q->elements[i+1];
    q->num_elements--; 
    return rv;
}

int main()
{
    Queue q;
    Initialize(&q);
    int x;
    Enqueue(&q, 10); /* q.elements[0] = 10; q.num_elements = 1; */
    Enqueue(&q, -5); /* q.elements[1] = -5; q.num_elements = 2; */
    Enqueue(&q, 14); /* q.elements[2] = 14; q.num_elements = 3; */
    Enqueue(&q, -8); /* q.elements[3] = -8; q.num_elements = 4; */
    /* Now: q.elements = { 10, -5, 14, -8, ...}; q.num_elements = 4; */
    x = Dequeue(&q); /* X == 10 */
    /* Now: q.elements = { -5, 14, -8, ...}; q.num_elements = 3; */
    x = Dequeue(&q); /* X == -5 */
    /* Now: q.elements = { 14, -8, ...}; q.num_elements = 2; */
```
What is the time complexity?

• Enqueue: $O(1)$
• Dequeue: $O(n)$

• Can we do better?
Queue: Try #2

front: an integer. The index of the element that is next to be dequeued
back: an integer. The index of the element that the next enqueue should write to.

Observation: front == back → nothing in the queue
Queue: Try #2

```
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
```

front    back

Enqueue(10)
Queue: Try #2

Enqueue(10)
Enqueue(-5)
Queue: Try #2

| 10 | -5 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

front  back

Enqueue(10)
Enqueue(-5)
Enqueue(14)
Queue: Try #2

Enqueue(10)
Enqueue(-5)
Enqueue(14)
Enqueue(-8)
Queue: Try #2

Enqueue(10)
Enqueue(-5)
Enqueue(14)
Enqueue(-8)
Dequeue()
Queue: Try #2

<table>
<thead>
<tr>
<th>Front</th>
<th>Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-5</td>
</tr>
<tr>
<td>14</td>
<td>-8</td>
</tr>
</tbody>
</table>

Enqueue(10)
Enqueue(-5)
Enqueue(14)
Enqueue(-8)
Dequeue()
Dequeue()
What is the time complexity?

- Enqueue: $O(1)$
- Dequeue: $\Theta(n) \quad O(1)$

- Can we do better? ... no

- But the coding gets more complex
  - Have to manage two locations
  - Wraparounds!!
Project 3B

- Implement stacks
- Will use 2C number parsing code
- There is a different syntax:
  - Not: “2 x 3”
  - Instead: “2 3 x”
- Stacks are good for this
- Worth 8% of your grade
- Assigned today, due on Wednesday (6 days)
Project 3C

• Implement queues – the O(1) kind
• Solve a specific problem: matching up surgeries between donors, recipients, and hospitals
• Everything works based on queues. Get in line to get a heart, etc.
• Worth 9% of your grade
• Assigned today, due on Sunday (10 days), but can turn in the following Weds (13 days) for full credit
BIG WEEK

• 3B is significant
• 3A will require looking at lecture 13 slides a lot, compiler errors
• 3C is more code than you have done before
• Sunday OH is possible
For 3C: fgets

FGETS(3) BSD Library Functions Manual FGTS(3)

NAME
fgets, gets -- get a line from a stream

LIBRARY
Standard C Library (libc, -lc)

SYNOPSIS
#include <stdio.h>

char *
fgets(char * restrict str, int size, FILE * restrict stream);

char *
gets(char *str);

DESCRIPTION
The fgets() function reads at most one less than the number of characters
specified by size from the given stream and stores them in the string str. Reading stops when a newline character is found, at end-of-file or
error. The newline, if any, is retained. If any characters are read and
there is no error, a \0' character is appended to end the string.
fgets example

```c
#include <stdio.h>

int main()
{
    FILE *f_in = fopen("fgets.c", "r");
    char line[256];
    int linenum = 0;
    while (fgets(line, 256, f_in) != NULL)
    {
        printf("%d: %s", linenum++, line);
    }
}
```
Misc. Topics
## Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+ -</td>
<td>Suffix/postfix increment and decrement</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>()</td>
<td>Function call</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[]</td>
<td>Array subscripting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td>Structure and union member access</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt;</td>
<td>Structure and union member access through pointer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(type){list}</td>
<td>Compound literal(C99)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>++ --</td>
<td>Prefix increment and decrement</td>
<td>Right-to-left</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td>Unary plus and minus</td>
<td></td>
</tr>
<tr>
<td></td>
<td>! ~</td>
<td>Logical NOT and bitwise NOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(type)</td>
<td>Type cast</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>Indirection (dereference)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>Address-of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sizeof</td>
<td>Size-of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>__alignof</td>
<td>Alignment requirement(C11)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>* / %</td>
<td>Multiplication, division, and remainder</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>4</td>
<td>+ -</td>
<td>Addition and subtraction</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Bitwise left shift and right shift</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>For relational operators &lt; and &lt;= respectively</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For relational operators &gt; and &gt;= respectively</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>== !=</td>
<td>For relational = and != respectively</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>&amp;</td>
<td>Bitwise AND</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>^</td>
<td>Bitwise XOR (exclusive or)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>Bitwise OR (inclusive or)</td>
</tr>
<tr>
<td>11</td>
<td>&amp; &amp;</td>
<td>Logical AND</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>Logical OR</td>
</tr>
<tr>
<td>14</td>
<td>=</td>
<td>Simple assignment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+= -=</td>
<td>Assignment by sum and difference</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*= / %=</td>
<td>Assignment by product, quotient, and remainder</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;= &gt;&gt;=</td>
<td>Assignment by bitwise left shift and right shift</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp; = ^ =</td>
<td>Assignment by bitwise AND, XOR, and OR</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>,</td>
<td>Comma</td>
<td>Left-to-right</td>
</tr>
</tbody>
</table>

DRAM vs NV-RAM

• DRAM: Dynamic Random Access Memory
  – stores data
  – each bit in separate capacitor within integrated circuit
  – loses charge over time and must be refreshed
  – → volatile memory

• NV-RAM: Non-Volatile Random Access Memory
  – stores data
  – information unaffected by power cycle
  – examples: Read-Only Memory (ROM), flash, hard drive, floppy drive, ...
Seagate Expansion 5TB Desktop External Hard Drive USB 3.0 (STEB5000100)
by Seagate

$133.99 $169.99
Get it by Friday, Nov 20

More Buying Choices
$133.99 new (68 offers)
$117.24 used (1 offer)

Crucial Ballistix Sport 16GB Kit (8GBx2) DDR3 1600 MT/s (PC3-12800) UDIMM Memory BLS2KIT8G3D1609DS1S00/ BLS2CP8G3D1609DS1S00
by Crucial

$74.99 $159.99
Get it by Thursday, Nov 19

More Buying Choices
$69.95 new (73 offers)

Crucial 16GB Kit (8GBx2) DDR3/DDR3L-1600 MHz (PC3-12800) CL11 204-Pin SODIMM Memory for Mac CT2K8G3S160BM / CT2C8G3S160BM
by Crucial

$72.99 $166.99
Get it by Thursday, Nov 19

More Buying Choices
$71.29 new (99 offers)
$62.00 used (8 offers)
Relationship to File Systems

• File Systems could be implemented in DRAM.
• However, almost exclusively on NV-RAM
  – Most often hard drives
• Therefore, properties and benefits of file systems are often associated with properties and benefits of NV-RAM.
DRAM vs NV-RAM properties

<table>
<thead>
<tr>
<th>Property</th>
<th>DRAM</th>
<th>NV-RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>~10GB</td>
<td>~10TB</td>
</tr>
<tr>
<td>Cost</td>
<td>$5/GB</td>
<td>$0.03/GB</td>
</tr>
<tr>
<td>Latency</td>
<td>&lt;100 ns</td>
<td>~10 ms</td>
</tr>
</tbody>
</table>

5 orders of magnitude!!

What does 100000:1 mean?

- Distance: a 20” map of Oregon is 1:100,000 scale
- Time: 1 second to 27 hours is 1:100,000 scale
- Time: 1 minute to 69 days is 1:100,000 scale
- Time: 1 hour to 11 years is 1:100,000 scale
- Time: 1 day to 273 years is 1:100,000 scale
Implement strdup
Tricky Memory Question
How a Computer is **Not** Like Monopoly

- **Actions during cycle/turn:**
  - Monopoly: roll, buy, build, trade
  - Computer: other

- **Who’s turn?**
  - Monopoly: passes between players
  - Computer: always the computer’s turn

- **Time spent per cycle/turn:**
  - Monopoly: variable
  - Computer: fixed

- **Duration of turn:**
  - Monopoly: sometimes over a minute
  - Computer: much faster!!!
“First” computer: ENIAC

• Year: 1946
• Location: Pennsylvania
• Purpose: military
• Cost: $487K
  – ($6.9M today)
• Technology:
  – very different than today
  – ... but still the same
Vacuum Tubes

- Vacuum tubes:
  - Glass tubes with no gas
  - Used to control electron flow in early computers

- Occasionally, a bug would get stuck in the tube and cause the program to malfunction

- We no longer have vacuum tubes, but the term bug has remained with us...
An ENIAC Computation

• Used for military calculations:
  – A-bomb design
  – Missile delivery

• ENIAC could do ~5000 calculations in one minute

• In one case:
  – ENIAC did a calculation in 30 seconds
  – Human being took 20 hours
  – 2400x increase in speed

source: wikipedia
Hertz (Hz) = unit of measurement for how fast you do something

• 1 Hertz = do something once per second
• KHz = 1024 Hz
• MHz = 1024 KHz
• GHz = 1024 MHz

• The ENIAC machine ran at 5000 Hertz, or about 5 KHz.
  – Vocab term: “clock speed” → the number of cycles per second
    • the clock speed of the ENIAC was 5 KHz
Today’s Desktop Computers Are Fast!

- Most computers run at ~1-3 GHz
- i.e., operate billions of instructions each second

- This is about one million times faster than the ENIAC
  - ... and the ENIAC was 2400X faster than humans
  - (at least at tasks computers are good at)
1 million-fold increase!
How does this happen?

- Moore’s Law (old timer’s version)
  - Clock speed doubles every 18 months
- Moore’s Law (newer version but still for old timers)
  - Clock speed doubles every 24 months
Moore’s Law

• Moore’s Law (actual version)
  – Number of transistors doubles every 24 months
  – And clock speed is a reflection of number of transistors

• So what is a transistor?
  – Semiconductor device for amplifying or switching electronic signals/power
  – Fundamental building block of modern electronics
  – Replacement for vacuum tube
Moore’s Law – The number of transistors on integrated circuit chips (1971-2016)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.

The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic. Licensed under CC-BY-SA by the author Max Roser.
But actually…

Source: maximumpc.com
The reason is power

• Desktop computer takes ~200W
  – There are multiple components that consume the power:
    • CPU
    • Monitor
    • Disk
    • Memory

• 200W * 1 year → ~$70
Relationship Between Power and Clock Speed

• Clock goes twice as fast → Power goes up by factor of 8
  – (Increase of X in clock speed → Increase of $X^3$ in power)

• Clock speeds haven’t changed in 12 years

• What if they had doubled every 2 years?
  • Then 64X faster
    – → 262144X more power (for the CPU)
    – → power bill now $18M
New vocab term: "core"

What Changed?
- We are getting double the transistors every two years
- … but clock speed is the same
- … so what is changing?

### CHOOSE YOUR OPTIMIZATION POINT

<table>
<thead>
<tr>
<th></th>
<th>INTEGRATED</th>
<th></th>
<th>POWER²</th>
<th>RECOMMENDED CUSTOMER PRICING³</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CORES</td>
<td>GHZ</td>
<td>MEMORY</td>
<td>FABRIC</td>
</tr>
<tr>
<td><strong>7290¹</strong></td>
<td>72</td>
<td>1.5</td>
<td>16GB</td>
<td>Yes</td>
</tr>
<tr>
<td>Best Performance/Node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7250</strong></td>
<td>68</td>
<td>1.4</td>
<td>16GB</td>
<td>Yes</td>
</tr>
<tr>
<td>Best Performance/Watt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7230</strong></td>
<td>64</td>
<td>1.3</td>
<td>16GB</td>
<td>Yes</td>
</tr>
<tr>
<td>Best Memory Bandwidth/Core</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7210</strong></td>
<td>64</td>
<td>1.3</td>
<td>16GB</td>
<td>Yes</td>
</tr>
<tr>
<td>Best Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹Available beginning in September  
²Plus 15W for integrated fabric  
³Pricing shown is for parts without integrated fabric. Add additional $278 for integrated fabric versions of these parts. Integrated fabric parts available in October.

---

Intel Xeon Silver  | Intel Xeon Gold  | Intel Xeon Platinum  
|-------------------------------------|---------------------|------------------------|
| Intel Xeon Silver  | Intel Xeon Gold  | Intel Xeon Platinum  

M - an optional SKU is available with support for up to 1.5TB memory per CPU socket

F - an optional SKU is available with integrated 100Gbps Intel Omni-Path fabric
How To Use Multiple Cores?

• Answer: parallel programming
  – Write computer programs that use all the cores
  – Ideally the coordination between the cores is minimal
Parallel Programming Concepts

• Usual goal:
  – if program takes N seconds to run with one core
  – then take N/2 seconds to run with two cores
  – and N/M seconds to run with M cores

Let’s consider an example outside of computers
Example: paint a house

- One person: 6 days (1 day = 10 hours)
- Two people: 3 days
- Three people: 2 days
- Six people: 1 day

- Sixty people: 1 hour?
- Six hundred people: 6 minutes?
Example: paint a house, plan #3

- One person: paint one house in 6 days
- Two people: paint two houses in 6 days
- Three people: paint three houses in 6 days
- One thousand people: paint 1000 houses in 6 days?

Parallel programming is hard, and smart people spend their whole careers figuring out how to make parallel programs be efficient
GPUs: Graphical Processing Units (graphics cards)

• Historical:
  – Introduced to accelerate graphics (gaming!)
  – Boom with desktop PCs late 90s onward
  – Mid-2000’s: people started hacking interface to program a GPU to make it do things besides graphics
  – Late 2000’s: GPU makers jump on board and start encouraging folks to program GPUs directly
  – GPGPU: General-purpose GPU programming
  – Mid 2010’s: GPUs used for *lots* of computing problems.
  – Machine learning workhorse!
Why Are GPUs So Good?

Market summary > NVIDIA Corporation
NASDAQ: NVDA - Mar 5, 7:59 PM EST

235.65 USD ↓0.89 (0.38%)
After-hours: 236.50 ↑0.36%

<table>
<thead>
<tr>
<th>1 day</th>
<th>5 day</th>
<th>1 month</th>
<th>3 month</th>
<th>1 year</th>
<th>5 year</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12.82 Mar 8, 2013</td>
</tr>
</tbody>
</table>

This level of increase in computation is not just a quantitative change, it is a qualitative one too.
Relationship Between C and Parallel Programming: Very Strong