Lecture 15: Queues and More
Plan For The Rest Of The Term

• 3C: assigned today -- ~100 lines of code beyond the starter code

• 3D: assigned Tuesday Nov. 20\textsuperscript{th}
  – Will be major project, worth 8%
  – Will be grade options for partial credit

• No projects after 3D.
Code Review

• Looking for feedback
Piazza Question: Course Grades

• In almost all of my classes, 90 is the cutoff for A-, 80 is the cutoff for B-, etc.

• That said, I look at the final distributions and occasionally place the cutoffs below these numbers. (They would never go higher, of course.) This is especially true when I am trying to determine what a "pass" should be. (I.e., A- might be 90 and B- might be 80, but C- might be less than 70.)
Piazza Question: What’s On The Final?

• The form of the test will be 5-9 questions. At least two will require you to write code. The code you write will be a few functions. The coding parts don't require cleverness to solve, just knowledge of how to code. One question will be a series of True/False statements, and will test your advanced knowledge.

• If you do all the homeworks on time and attend all the labs, then you should have close to 70/70 points entering the final. That would mean a 15/30 on the final would still put you at a solid B.

• Also, I will be doing a Finals review on the last day of class.
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

Week 1

Why The Projects Are Hard

Week 5

Week 10

Project complexity

Unix, vim, new language (C), memory, VirtualBox, edit-compile-execute debuggers
Sunday OH

• About 15 people attended on Sunday
• Good session, open to doing it again
• About 1/6\textsuperscript{th} of the class is lagging on projects, and was disappointed not to see those people
YouTube Video Ideas?
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 \rightarrow 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

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

| 10 | -5 | 14 | -8 | 0 | 0 | 0 | 0 | 0 | 0 |

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

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

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

- Enqueue: $O(1)$
- Dequeue: $O(n)$  $O(1)$

- Can we do better? ... no

- But the coding gets more complex
  - Have to manage two locations
  - Wraparounds!!
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 4% of your grade
• Assigned today, due in one week
BIG WEEK

• 3B due tomorrow
• 3A due Monday
• 3C due Tuesday

• 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; will ask on Thursday
For 3C: fgets

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);
}
```

Note: no ‘\n’
Misc. Topics
Unions

• Union: special data type
  – store many different memory types in one memory location

```c
typedef union
{
  float x;
  int   y;
  char  z[4];
} cis330_union;
```

When dealing with this union, you can treat it as a float, as an int, or as 4 characters.

This union has 4 bytes
Unions

Why are unions useful?

```c
128-223-223-72-wireless:330 hank$ cat union.c
#include <stdio.h>

typedef union
{
    float x;
    int y;
    char z[4];
} cis330_union;

int main()
{
    cis330_union u;
    u.x = 3.5; /* u.x is 3.5, u.y and u.z are not meaningful */
    u.y = 3;   /* u.y is 3, now u.x and u.z are not meaningful */
    printf("As u.x = %f, as u.y = %d\n", u.x, u.y);
}
128-223-223-72-wireless:330 hank$ gcc union.c
128-223-223-72-wireless:330 hank$ ./a.out
As u.x = 0.000000, as u.y = 3
```
typedef struct
{
    int firstNum;
    char letters[3];
    int endNums[3];
} CA_LICENSE PLATE;

typedef struct
{
    char letters[3];
    int nums[3];
} OR_LICENSE PLATE;

typedef struct
{
    int nums[6];
} WY_LICENSE PLATE;

typedef union
{
    CA_LICENSE PLATE ca;
    OR_LICENSE PLATE or;
    WY_LICENSE PLATE wy;
} LicensePlate;
typedef enum
{
    CA,
    OR,
    WY
} US_State;

typedef struct
{
    char *carMake;
    char *carModel;
    US_State state;
    LicensePlate lp;
} CarInfo;

int main()
{
    CarInfo c;
    c.carMake = "Chevrolet";
    c.carModel = "Camaro";
    c.state = OR;
    c.lp.or.letters[0] = 'X';
    c.lp.or.letters[1] = 'S';
    c.lp.or.letters[2] = 'Z';
    c.lp.or.nums[0] = 0;
    c.lp.or.nums[1] = 7;
    c.lp.or.nums[2] = 5;
}

typedef struct
{
    int firstNum;
    char letters[3];
    int endNums[3];
} CA_LICENSE_PLATE;

typedef struct
{
    char letters[3];
    int nums[3];
} OR_LICENSE_PLATE;

typedef struct
{
    int nums[6];
} WY_LICENSE_PLATE;

typedef union
{
    CALICENSE_PLATE ca;
    ORLICENSE_PLATE or;
    WYLICENSE_PLATE wy;
} LicensePlate;
# 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>++ -- () [] . -&gt; (type)[] list</td>
<td>Suffix/postfix increment and decrement Function call Array subcripting Structure and union member access Structure and union member access through pointer Compound literal(C99)</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>2</td>
<td>++ -- + - ! ~ (type) + &amp; sizeof _Alignof</td>
<td>Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT Type cast Indirection (dereference) Address-of Size-of Alignment requirement(C11)</td>
<td>Right-to-left</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;= &gt;=</td>
<td>For relational operators &lt; and ≤ respectively For relational operators &gt; and ≥ 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>Bitwise OR (inclusive or)</td>
<td></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></td>
</tr>
<tr>
<td>14</td>
<td>= += -= *= /= %= &lt;&lt;= &gt;&gt;=</td>
<td>Simple assignment Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, 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
  – \( \rightarrow \) 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, ...
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 nanoseconds</td>
<td>~10 milliseconds</td>
</tr>
<tr>
<td></td>
<td>5 orders of magnitude!!</td>
<td></td>
</tr>
</tbody>
</table>

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 2G