Lecture 16: Linked Lists
Plan For The Rest Of The Term

• 3C: assigned Tuesday -- ~100 lines of code beyond the starter code
• 3D: assigned Tuesday Nov. 20th
  – Will be major project, worth 8%
  – Will be grade options for partial credit
• No projects after 3D
Sunday OH

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

- 3C overview
- gdb
- More?
Can you provide more insight into how the pointers for functions within structs works in a more clear way. I have looked over the example from YT and that did help but I am very confused at the exact way that this is working and want to ensure that its used correctly.

Potentially a internet resource that already exists or a youtube video?
Linked Lists: Motivation

• Will introduce two data structures: linked lists and doubly linked lists
• Linked lists will be used in 3D – you will implement a “map” using linked lists
• Doubly linked lists will tie into a future data structure (Deques)
Linked List (from Sventek)

• A linked list is linear collection of data elements, in which the linear order is not given by their physical placement in memory.
• Instead, each node in the list points at the next element in the list.
• Such a structure enables insertion or removal of elements without reallocation or reorganization of the entire structure, since the data items do not need to be stored contiguously in memory.
Linked List Example/Analogy
Example (from Sventek)

```c
struct node {
    struct node *next;
    char *value;
};

struct node *head;

int search(char *university, struct node *theList) {
    struct node *p;

    for (p = theList; p != NULL; p = p->next)
        if (strcmp(p->value, university) == 0)
            return 1;
    return 0;
}
```
How Do Linked Lists Compare With Arrays?
Comparison With Arrays: Hank’s (Wikipedia’s) Answers

- Can do insertions -- beginning, middle, end
- Indexing much slower
- Extra storage

<table>
<thead>
<tr>
<th></th>
<th>Linked list</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexing</td>
<td>Θ(𝑛)</td>
<td>Θ(1)</td>
</tr>
<tr>
<td>Insert/delete at beginning</td>
<td>Θ(1)</td>
<td>N/A</td>
</tr>
<tr>
<td>Insert/delete at end</td>
<td>Θ(1) when last element is known; Θ(𝑛) when last element is unknown</td>
<td>N/A</td>
</tr>
<tr>
<td>Insert/delete in middle</td>
<td>search time + Θ(1)[6][7]</td>
<td>N/A</td>
</tr>
<tr>
<td>Wasted space (average)</td>
<td>Θ(𝑛)</td>
<td>0</td>
</tr>
</tbody>
</table>
Let’s Implement Linked Lists
Doubly Linked List

```c
struct node {
    struct node *next;
    char *value;
};
```

Doubly Linked List

```c
typedef struct node {
    struct node *next;
    struct node *prev;
    void *value;
} Node;
```
(Singly) Linked List Vs Doubly Linked List

I am asked such question and I have my own sayings but I am not really sure what to say about cons and pros? Microsoft asked this question to one of its candidates.

Singly linked list allows you to go one way direction. Whereas doubly linked list has two way direction next and previous.

It all comes down to usage. There's a trade off here.

Singly linked list is simpler in terms of implementation, and typically has a smaller memory requirement as it only needs to keep the forward member referencing in place.

Doubly linked list has more efficient iteration, especially if you need to ever iterate in reverse (which is horribly inefficient with a single linked list), and more efficient deletion of specific nodes.
Example of Doubly Linked List

• iPod play list
  – Need reverse to get to previous song
Let’s Implement Doubly Linked Lists
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

#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);
}

As u.x = 0.000000, as u.y = 3
Unions Example

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;
}
## 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; (type)[list]</td>
<td>Structure and union member access through pointer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sizeof _Alignof</td>
<td>Compound literal(C90)</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) + &amp; sizeof _Alignof</td>
<td>Type cast, Indirection (dereference), Address-of, Size-of, 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 ≤ respectively</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>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></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></td>
</tr>
<tr>
<td>13[note 1]</td>
<td>:?</td>
<td>Ternary conditional</td>
<td>Right-to-Left</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>=</td>
<td>Assignment by bitwise AND, XOR, and OR</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 ✔️Prime
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 ✔️Prime
Get it by Thursday, Nov 19
More Buying Choices
$69.95 new (73 offers)

Corsair Vengeance 16GB (2x8GB) DDR3 1600 MHz (PC3 12800) Desktop Memory (CMZ16GX3M2A1600C10)
by Corsair
$83.90 $148.79 ✔️Prime
Get it by Thursday, Nov 19
More Buying Choices
$72.50 new (101 offers)
$74.99 used (3 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 ✔️Prime
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, 5 orders of magnitude!!</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