Class Announcements

1. How many of you attended the WACM tutorial and found it useful?

2. Assignment 1 released - due before 9AM on Sep 30.
   - You can find partners using Piazza too.
   - Start Early! Much much harder than Assign 0!

3. Make sure you don’t change your files to add very small changes like formatting, comments etc. after deadline. You get points deducted even if it is a small change.
Lecture Overview

1. Doubly Linked Lists
2. Data Representation (Unsigned, 2’s complement)
3. Signed <-> Unsigned Conversions
4. Integer Arithmetic (Addition)
Example C Program on Singly Linked List
Deleting a node in a Singly Linked List without copying
Doubly Linked List

1. In order to delete a node in a singly linked list without copying values, a pointer to the previous node is also needed.

2. Doubly linked lists allow inserts and deletes in constant number of operations with only the node’s address.

3. Doubly linked lists are easier to manipulate they allow fast and easy sequential access to the list in both directions.
struct node {
    int theint;
    struct node *next;
    struct node *previous;
};

Doubly Linked
For convenience, name this user-defined type:

typedef struct node {
    int theint;
    struct node *next;
} Node;

Now, declarations have less (keyboard) typing:

Node one, two, three;
Node *head;
Some code, to show pointers and such.

```c
one.theint = 1;
one.next = &two;
one.next->next = &three;
three.next = NULL;
head = &one;
```

```
    □
    □    □
one    two    three
```
Some code, to show pointers and such.

```c
one.theint = 1;
one.next = &two;
one.next->next = &three;
three.next = NULL;
head = &one;
```

![Diagram of linked list with three nodes: one, two, and three, where one points to two and two points to three, with head labeled as the starting point.](image)
Some code, to show pointers and such...

```c
one.theInt = 1;
one.next = &two;
one.next->next = &three;
three.next = NULL;
head = &one;
```

```
+---+      +---+      +---+      +---+   
|   |      |   |      |   |      |   |   
|   |      |   |      |   |      |   |   
|   |      |   |      |   |      |   |   
|   |      |   |      |   |      |   |   
|   |      |   |      |   |      |   |   
one       two       three
+---+      +---+      +---+      +---+   
```
Some code, to show pointers and such...

```c
one.theInt = 1;
one.next = &two;
one.next->next = &three;
three.next = NULL;
head = &one;
```

![Diagram](attached-diagram)
int value = 1;
Node *ptr;

ptr = head;
while (ptr != NULL) {
    ptr->theInt = value * 11;
    value++;
    ptr = ptr->next;
}
Some code, to show pointers and such.

```
one.theint = 1;
one.next = &two;
one.next->next = &three;
three.next = NULL;
head = &one;
```
int value = 1;
Node *ptr;

ptr = head;
while (ptr != NULL) {
    ptr->theInt = value * 11;
    value++;
    ptr = ptr->next;
}
Some code, to show pointers and such...

```c
one.theint = 1;
one.next = &two;
one.next->next = &three;
three.next = NULL;
head = &one;
```
int value = 1;
Node *ptr;

ptr = head;
while (ptr != NULL) {
    ptr->theint = value * 11;
    value++;
    ptr = ptr->next;
}
Some code, to show pointers and such...

```c
one.theint = 1;
one.next = &two;
one.next->next = &three;
three.next = NULL;
head = &one;
```

![Diagram of linked list](image)
int value = 1;
Node *ptr;

ptr = head;
while (ptr != NULL) {
    ptr->theint = value * 11;
    value++;
    ptr = ptr->next;
}

Why is this now incorrect?
With the correct code, what happens when this code is executed?

```c
ptr = three.next;
ptr = ptr->next;
```
Some code, to show pointers and such...

```c
one.theint = 1;
one.next = &two;
one.next->next = &three;
three.next = NULL;
head = &one;
```

![Diagram of linked list with pointers](image)
With the correct code, what happens when this code is executed?

```c
ptr = three.next;
ptr = ptr->next;  ← Runtime error: NULL pointer dereference
```

In Linux:
Segmentation fault (core dumped)
Example C Program on Doubly Linked List
Detailed Example C Program on Singly Linked List
Bits, Nibbles, Bytes, Words

1. Bits represented using “high & low voltages”, “magnetic domain oriented clockwise or anticlockwise” etc.

2. 4 Bits == Nibble ; 8 Bits == Byte ; 16/32/64 Bits == Word (depending on architecture);

3. Group of bits collected together with some interpretation is more useful than individual bits.
Word size

1. It is the nominal size of integers and pointer data

2. Determines the maximum size of virtual address space

3. $w$ bit word can address a virtual memory of size $(2^w)$ ranging from 0 to $2^w-1$. 

4. Modern computers have 64 bit words. (Theoretically: $2^{64} = 16$ Exabytes.)
Byte encodings

- **Byte = 8 bits**
  - Binary 00000000₂ to 11111111₂
  - Decimal: 0₁₀ to 255₁₀
  - Hexadecimal 00₁₆ to FF₁₆
    
    - Base 16 number representation
    - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
    - Write FA1D37B₁₆ in C as
      
      - 0xFA1D37B
      - 0xfa1d37b
Byte Ordering/Endianness

1. Ordering of bytes within a word
2. Little endian – least significant byte comes first
3. Big endian – most significant byte comes first
Representations

1. Unsigned encodings – positive integers
2. Two’s complement – signed integers
3. Floating point – real numbers
4. Because of limited number of bits to encode a number, some operations can “overflow” when results are too large.
Arithmetic Operations

- Arithmetic Operations
  - addition
  - subtraction
  - multiplication
  - division

- Each of these operations on the integer representations:
  - unsigned
  - two's complement
Addition

One bit of binary addition

carry in

a + b

sum bit

carry out
## Addition Truth Table

<table>
<thead>
<tr>
<th>Carry In</th>
<th>a</th>
<th>b</th>
<th>Carry Out</th>
<th>Sum Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>


Unsigned Representation

\[ \text{B2U}_w(x_{vec}) = \sum_{i=0}^{w-1} x_i \cdot 2^i \]

\[ \text{B2U}_4([0101]) = 0.2^3 + 1.2^2 + 0.2^1 + 1.2^0 = 5 \]

\( \text{B2U}_w \) is a bijection:
- associates a unique value to each bit vector of length \( w \)
- each integer between 0 and \( 2^w - 1 \) has a unique binary representation as a bit vector of length \( w \)
Unsigned Addition

Of two unsigned w bit values X & Y

X + Y equals:
- X+Y, if \((X+Y) < 2^w\)
- X+Y-2^w, if \(2^w \leq (X+Y) < 2^{w+1}\)
Addition

- Unsigned and 2's complement use the same addition algorithm
- Due to the fixed precision, throw away the carry out from the msb

```
00010111
+ 10010010
```

```
Addition

- Unsigned and 2's complement use the same addition algorithm
- Due to the fixed precision, throw away the carry out from the msb

\[
\begin{array}{c}
00010111 \\
+ 10010010 \\
\hline
10101001
\end{array}
\]
Two’s complement Representation

\[ B2T_w(x_{vec}) = -x_{w-1}2^{w-1} + \sum_{i=0 \rightarrow w-2} x_i2^i \]

\[ B2T_4([1011]) = -1.2^3 + 0.2^2 + 1.2^1 + 1.2^0 = -5 \]

\[ B2T_w \] is a bijection:
- associates a unique value to each bit vector of length \( w \)
- each integer between \(-2^{w-1}\) and \(2^{w-1}-1\) has a unique binary representation as a bit vector of length \( w \)
# Range of Values for Unsigned and 2’s Complement (16 bits)

<table>
<thead>
<tr>
<th></th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMax</td>
<td>65535</td>
<td>FF FF</td>
<td>11111111 11111111</td>
</tr>
<tr>
<td>Tmax</td>
<td>32767</td>
<td>7F FF</td>
<td>01111111 11111111</td>
</tr>
<tr>
<td>Tmin</td>
<td>-32768</td>
<td>80 00</td>
<td>10000000 00000000</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>FF FF</td>
<td>11111111 11111111</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>00 00</td>
<td>00000000 00000000</td>
</tr>
</tbody>
</table>

#include <limits.h> declares constants, e.g., ULONG_MAX, LONG_MAX, LONG_MIN
(Values platform specific)
## 4-bit Unsigned and 2’s complement Integers

<table>
<thead>
<tr>
<th>$X$</th>
<th>B2U$(X)$</th>
<th>B2T$(X)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>–8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>–7</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>–6</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>–5</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>–4</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>–3</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>–2</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>–1</td>
</tr>
</tbody>
</table>