ECE 120 Midterm 1

HKN Review Session

Time: 8:30-10:00 pm (Arrive at 8:15 pm)
Location: Your Room on Compass
What to bring: iCard, pens/pencils, Cheat sheet (Handwritten)
Overview of Review

- Binary
- IEEE floating point
- Hex & ASCII
- Operators
- Bitmasks
- C programming
- Practice Exam
Binary Representation

● Unsigned - k bits can represent \([0, 2^k)\) values
  ○ Can only represent non-negative integers
  ○ Overflow Condition: If the most significant bit has a carry out
  ○ Zero Extended - pad the front with zeros when moving to larger data type
  ○ \((10101) = (16+4+1) = 21\)

● Signed Magnitude - \((-2^{(k-1)}, 2^{(k-1)})\) values
  ○ First bit determines sign of number (1 = negative, 0 = positive)
  ○ \((10101) = (-1) \times (4+1) = -5\)
Binary Representation - Part II

- 2’s Complement - k bits represents \([-2^{(k-1)}, 2^{(k-1)} - 1]\)
  - Sign Extended - pad the front of the number with the sign bit when moving to larger data type
  - Overflow Condition: if adding numbers of the same sign yields a result of the opposite sign
  - If signed bit is 0, magnitude is same as if the number were treated as unsigned
  - If signed bit is 1, to determine the magnitude, bitwise NOT the bits, and add 1 before finding magnitude
  - 2’s complement provides a greater range of values and cleaner binary arithmetic operations
    - ie. the same logic circuit used to add binary unsigned and 2’s complement numbers
IEEE 754 Floating Point Representation

- Great for approximations & expressing very large/very small decimal values
- 1st bit is sign bit, 0 for positive and 1 for negative
- 2nd through 9th bit is exponent
  - Special Cases: When exponent is 0 or 255, then denormalized or NAN/inf forms respectively.
  - Denormalized format has normal sign, and magnitude: 0.mantissa * 2^(-126)
  - NAN/inf form is NAN when mantissa is non-zero, and infinity (inf) when mantissa is all zeros
- 10th through 32nd bit is 23 bit mantissa
- Normal Form has magnitude of 1.mantissa * 2^(exponent - 127)
- (-1)^sign * 1.fraction * 2^(exponent - 127)
Hexadecimal & ASCII Representations

- 7-bit ASCII table will be provided if necessary
- Although 7 bits, chars in C are 1 byte of memory and allocate 8 bits per character anyway
- Hexadecimal is base 16 number system (0-9 & A-F)
- All numbering systems that have a power of 2 base can have individual characters mapped to binary strings
- A=1010 B=1011 C=1100 D=1101 E=1110 F=1111
- Converting 76 (01001100) to hex = 0x4C
- 011010, 0110011, 0101001 converting to ASCII -
Hexadecimal & ASCII Representations

- 7-bit ASCII table will be provided if necessary
- Although 7 bits, chars in C are 1 byte of memory and allocate 8 bits per character anyway
- Hexadecimal is base 16 number system (0-9 & A-F)
- All numbering systems that have a power of 2 base can have individual characters mapped to binary strings
- A=1010 B=1011 C=1100 D=1101 E=1110 F=1111
- Converting 76 (01001100) to hex = 0x4C
- 0111010, 0110011, 0101001 converting to ASCII -
  - 0x3A 0x33 0x29
  - :3)
Operators

- **AND**
  - Returns 1 if both inputs are 1

- **OR**
  - Returns 1 if any of the inputs are 1

- **XOR**
  - Returns 1 if either a or b is 1, not both

- **NOT**
  - Returns opposite of input (0→1, vice versa)
Bitmasks

- Great for looking at only certain bits
- A bitmask using AND as the bitwise operator will selectively mask bits to 0
- A bitmask using OR as the bitwise operator will selectively mask bits to 1
C programming

- Variables
  - Double, Long (8 bytes)
  - Int, Float (4 bytes)
    - Int may be 2 bytes on some systems
  - Short (2 bytes)
  - Char (1 byte)

```c
#include <stdio.h>
#define PI = 3.1415926;

int main(){
    int r = 12;
    int float area;
    area = (PI * r * r);
    return 0;
}
```
# Format Specifiers and Escape Sequences

<table>
<thead>
<tr>
<th>Format Specifiers:</th>
<th>Escape Sequences:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%c</code> char single character</td>
<td></td>
</tr>
<tr>
<td><code>%d (%i)</code> int signed integer</td>
<td><code>\n</code> - new line character</td>
</tr>
<tr>
<td><code>%e (%E)</code> float or double exponential format</td>
<td><code>\b</code> - backspace character</td>
</tr>
<tr>
<td><code>%f</code> float or double signed decimal</td>
<td><code>\t</code> - horizontal tab</td>
</tr>
<tr>
<td><code>%g (%G)</code> float or double use <code>%f</code> or <code>%e</code> as required</td>
<td><code>\’</code> - allows for storing and printing of <code>\</code> ASCII value in string</td>
</tr>
<tr>
<td><code>%o</code> int unsigned octal value</td>
<td><code>\&quot;</code> - allows for storing and printing of <code>\</code> ASCII value in string</td>
</tr>
<tr>
<td><code>%p</code> pointer address stored in pointer</td>
<td><code>\?</code> - allows for storing and printing of <code>?</code> ASCII value in string</td>
</tr>
<tr>
<td><code>%s</code> array of char sequence of characters</td>
<td>Ie. char x = <code>\&quot;</code>; (those are 2 single quotes)</td>
</tr>
<tr>
<td><code>%u</code> int unsigned decimal</td>
<td>will store the data byte associated with <code>\</code> into x</td>
</tr>
<tr>
<td><code>%x (%X)</code> int unsigned hex value</td>
<td></td>
</tr>
</tbody>
</table>
int integer = 5;

float decimal = 2.5;

printf(“This is an integer: %d
This is a float: %f
”, integer, decimal);
Format Specifier Example

int integer = 5;
float decimal = 2.5;

printf("This is an integer: %d\nThis is a float: %f\n", integer, decimal);

Prints:
This is an integer: 5
This is a float: 2.50000
C - operators

- **Order of precedence**
  - *, /, %, then +, - (Note, Modular Arithmetic (%) is not defined for floating point numbers)

- **Assignment operator**
  - = (takes a variable on the left and a value/expression on the right side)
  - Sets the variable on the left equal to the expression on the right
    - IE, you can’t do 5 = x; as 5 is not a variable (you can’t assign a value to 5!)

- **Relational**
  - ==, !=, <, >, <=, =>
  - Returns 1 for true, 0 for false

- **Bitwise**
  - &, |, ~, ^ (AND, OR, NOT, XOR)

- **Logical**
  - !, &&, || (NOT, AND, OR)
C operators - bitwise examples

```
int a = 12;
int b = 20;
int c;
c = (a|b);
```

```
int a = 13;
int b = 6;
int c = 17;
int d;
d = (((a|b)&c) | b);
```

```
int a = 7;
int b = 16;
int c = 21;
int d = 12;
int e;
e = (((a^b)&c) & ((d^a)|b));
```
C operators - bitwise examples

```c
int a = 12;
int b = 20;
int c;
c = (a|b);
```

```c
int a = 13;
int b = 6;
int c = 17;
int d;
d = (((a|b)&c) | b);
```

```c
int a = 7;
int b = 16;
int c = 21;
int d = 12;
int e;
e = (((a^b)&c) & ((d^a)|b));
```

- c = 28
- e = 17
- d = 7
Basic Input / Output

```c
float i = 2.15;
int j;

scanf("%d", &j);

printf("%d and %f", j, i);
```
Conditional Constructs

```c
int a = 4;
if (a < 10){
    printf("a is less than 10");
}

int a = 4;
int b = 8;
if (a < 10){
    printf("a is less than 10");
} else if (b < 20){
    printf("b is less than 20");
} else{
    printf("b is greater than 20");
}
```
Conditional Constructs - Example

```c
int a = 20;
int b = 13;
int c = 6;

if ((a-b > 10) || ((c*a)%10 == 0)){
    if(b+c < a){
        printf("a = %d", a);
    }else{
        printf("Error");
    }
}else{
    printf("b = %d", b);
}
```
Iterative Constructs

```c
int i = 0;

    do {
        /* do things */
    } while(i < 10);
```

```c
int i;

for(i = 0; i<=10; i++) {
    printf("%d", i);
}
```

```c
int i = 0;

while (i < 10) {
    /* do things*/
}
```
Iterative - Example

```c
int i;

for(i = 0; i<=10; i++){
    if (i%3 == 0){
        printf("%d\n", i);
    }
}
```
HW2.9

HW2.9. Masking Bits

Define \( X \text{ NAND} Y = \text{ NOT}(X \text{ AND} Y) \) and \( X \text{ NOR} Y = \text{ NOT}(X \text{ OR} Y) \).

Select a logical operation and enter a bitmask that convert the following input bit strings to the corresponding output bit strings.

\[
\begin{align*}
00100101 & \rightarrow 00111101 \\
11001010 & \rightarrow 11111110 \\
10111011 & \rightarrow 10111111
\end{align*}
\]

Logical operation

- AND
- OR
- NAND
- NOR

Bitmask (8 bits) [ ]
Past Exam Review

- https://wiki.illinois.edu/wiki/display/ece120/Midterm+1
- Fall 2016, Midterm 1
- Solutions available