We will be programming in C in lab and in class.

C is different from most programming languages in that it is useful for writing very complex software, but it can also be used to control minute hardware details of a processor, and thus interact with the real world efficiently.

In lab we will use it to control hardware. In class you will learn higher-level capabilities.

In lab, I will only discuss a very small subset of the language.

If you know C, feel free to use more than is described here. You will also, eventually, learn more C in class.
A simple C program

```c
void main(void) {
    int i, j, k;
    i=3;            //Set i=3
    j=4; //Set j=4
    k = i * j / 2;  //Calculate k
}
```

Every program needs a "main" routine (between braces)

Declare i, j and k to be integer variables. All variable must be declared. (In this lab we will use only integers)

Give the value 3 to the variable "i"

Give the value 4 to the variable "j"

Calculate k = 3 * 4 / 2 = 6

Anything after "//" is a comment. It is ignored by the computer, but describes what the code is doing.

Individual lines end with ";"

Note:
• "int" variables have a range of -32768→32767 (16 bit, -(2\(^{15}\)) →2\(^{15}-1\))
• "unsigned int" variables have a range of 0→65535 (0→ 2\(^{16}-1\))
# C Operators

## (Arithmetic)

<table>
<thead>
<tr>
<th>Arithmetic Operator</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic assignment</td>
<td>a = b</td>
</tr>
<tr>
<td>Addition</td>
<td>a + b</td>
</tr>
<tr>
<td>Subtraction</td>
<td>a - b</td>
</tr>
<tr>
<td>Unary minus</td>
<td>-a</td>
</tr>
<tr>
<td>Multiplication</td>
<td>a * b</td>
</tr>
<tr>
<td>Division</td>
<td>a / b</td>
</tr>
<tr>
<td>Bitwise NOT</td>
<td>~a</td>
</tr>
<tr>
<td>Bitwise AND</td>
<td>a &amp; b</td>
</tr>
<tr>
<td>Bitwise OR</td>
<td>a</td>
</tr>
<tr>
<td>Bitwise XOR</td>
<td>a ^ b</td>
</tr>
</tbody>
</table>

*Use with great caution!*

## C Comparisons

### (Result is logical - True or False)

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal to</td>
<td>a == b</td>
</tr>
<tr>
<td>Not equal to</td>
<td>a != b</td>
</tr>
<tr>
<td>Greater than</td>
<td>a &gt; b</td>
</tr>
<tr>
<td>Less than</td>
<td>a &lt; b</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>a &gt;= b</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>a &lt;= b</td>
</tr>
</tbody>
</table>

This is just a subset. If you know “C” feel free to use others, but they won’t be necessary.

## C Operators

### (logical – True or False)

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical negation (NOT)</td>
<td>!a</td>
</tr>
<tr>
<td>Logical AND</td>
<td>a &amp;&amp; b</td>
</tr>
<tr>
<td>Logical OR</td>
<td>a</td>
</tr>
</tbody>
</table>
Making Simple Decisions in C

if...then...else
    if (condition)
        statement;
    else
        statement;

Find absolute value of difference between i and j.
    if (i>j)
        k = i - j;
    else
        k = j - i;

The “else” clause is optional. An equivalent code segment.
    k = j - i;
    if (i>j)
        k = i - j;
Statements in C

Statements
• a simple statement is a single statement that ends in a ";"
• a compound statement is several statements inside braces:

```c
{  
  statement;  
  ...  
  statement;  
}
```

Find absolute value of difference between i and j, and set lower one to zero (both if equal).

```c
if (i>j) {  
  k = i - j;  
  j = 0;  
} else {  
  k = j - i;  
  if (j>i) i=0;  
  else {  
    i=0;  
    j=0;  
  }  
}
```
Iteration in C

**While loop**
```
while (condition) {
    statement;
}
```

// calculate k = j * i by repetitive addition
k = 0;
while (j != 0) {
    k = k + i;
    j = j - 1;
}

**for loop**
```
for (statement 1; condition; statement 2) {
    statements...;
}
```

**for loop is equivalent to:**
```
statement 1;
while (condition) {
    statements...;
    statement 2;
}
```

Common use of “for loop” is to count through values:
```
for (i = 0; i < 10; i = i + 1) {
    statements...;
}
Making Complicated Decisions in C

**switch (one choice of many)**

```
switch (<expression>) {
    case <label1> :
        statements 1;
        break;
    case <label2> :
        statements 2;
        break;
    case <label3> :
    case <label4> :
        statements 3;
        break;
    default :
        statements 4>
}
```

- `<expression>` is compared against each label, and appropriate statements are executed, until a “break” statement is encountered.
- All labels must be unique.
- If expression matches no label, the “default” statements are executed.
- Don’t forget “break” statements, otherwise execution continues from one label to the next.
Indenting in C

Indenting

There are no rules about indenting code, but if you don’t adopt a standard style, your code becomes unreadable.

Choose a style and stick to it. Your code will be graded partly on readability.

Three possibilities

```c
while (x == y) {
    something();
    somethingelse();
    if (some_error)
        do_correct();
    else
        continue_as_usual();
}
```

```c
while (x == y)
{
    something();
    somethingelse();
}
finalthing();
```

```c
if (x < 0)
{
    printf("Negative");
    negative(x);
}
else
{
    printf("Positive");
    positive(x);
}
```
Manipulating Bits in C

<table>
<thead>
<tr>
<th>Hex</th>
<th>Bit 7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT0 = 0x01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>BIT1 = 0x02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>BIT2 = 0x04</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BIT3 = 0x08</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BIT4 = 0x10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BIT5 = 0x20</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BIT6 = 0x40</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BIT7 = 0x80</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Setting bits

\[ x = \text{BIT0} \mid \text{BIT3} \mid \text{BIT6} = 0100\ 1001_{\text{binary}} = 49_{\text{hex}} = 0x49 \ (C \ syntax) \]

\[ x = \text{BIT0} + \text{BIT3} + \text{BIT6} = 0100\ 1001_{\text{binary}} = 49_{\text{hex}} \]

\[ x = x \mid \text{BIT4} = 0101\ 1001_{\text{binary}} = 59_{\text{hex}} \]

Clearing bits (assume \( x = 0101\ 1001_{\text{binary}} = 59_{\text{hex}} \) to start = 0x59 (C syntax))

\[ y = x \& \sim\text{BIT3} = 0101\ 0001_{\text{binary}} = 51_{\text{hex}} \quad \sim\text{BIT3} = 1111\ 0111_{\text{binary}} = F7_{\text{hex}} \]

\[ y = x \& \sim(\text{BIT3} \mid \text{BIT4}) = 0100\ 0001_{\text{binary}} = 41_{\text{hex}} \]
Programming a microcontroller

Potential uses: smoke detector, touch screen interface, scale, glass breakage sensor, thermostat, ...

Notes:
- Pins have multiple functions.
- In this lab we only do digital I/O (the P1.0-P1.7 and P2.6 and P2.7 functions).
- We control the pins through registers.
# Simple Program to Blink an LED

P1.0 is attached to LED (BIT0 = 0x01)

```c
#include <msp430x20x2.h>

void main(void) {
  int i;
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
  P1SEL = 0;
  P2SEL = 0;
  P1DIR = P1DIR | 0x01; // Set P1.0 to output

  while (1) { //Do this forever
    P1OUT = P1OUT ^ 0x01; // Toggle P1.0 using exclusive-OR
    for (i=1; i<30000; i=i+1) {} //Delay
  }
}
```

- XOR with 1 inverts bit:
  - `1 ^ 1 = 0`
  - `0 ^ 1 = 1`

## Bit Table

<table>
<thead>
<tr>
<th>Bit#</th>
<th>P1.7</th>
<th>P1.6</th>
<th>P1.5</th>
<th>P1.4</th>
<th>P1.3</th>
<th>P1.2</th>
<th>P1.1</th>
<th>P1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1SEL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P1DIR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P1OUT</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0/1</td>
</tr>
<tr>
<td>P1IN</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Simple Program to Control LED

P1.0 is attached to LED (BIT0 = 0x01),
P1.4 is attached to push button (BIT4 = 0x10)

while (1) {  //Do this forever
    if (!(P1IN & 0x10)) {     // If button pushed (P1.4 is low)
        P1OUT = P1OUT | 0x01;  // ... turn on LED
    } else {
        P1OUT = P1OUT & ~0x01; // ... else turn it off.
    }
}

When button is pushed:
• P1IN bit 4 is 0,
• P1IN & 0x10 is 0,
• !(P1IN & 0x10) is true.

When button is not pushed:
• P1IN bit 4 is 1,
• P1IN & 0x10 is 0x10,
• !(P1IN & 0x10) is false.

<table>
<thead>
<tr>
<th>Bit#</th>
<th>P1.7</th>
<th>P1.6</th>
<th>P1.5</th>
<th>P1.4</th>
<th>P1.3</th>
<th>P1.2</th>
<th>P1.1</th>
<th>P1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1SEL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P1DIR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P1OUT</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0/1</td>
</tr>
<tr>
<td>P1IN</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0/1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

0 = digital I/O
1 = output
0/1 = Control LED
Sense Switch
Functions in C
(and #define)

Functions are a useful way to separate a program into functional units, and to reuse code.

```c
#include <msp430x20x2.h>

#define LED 0x01
#define PB 0x10

void LED_on(int ledval) {
    if (ledval) P1OUT = P1OUT | LED;
    else P1OUT = P1OUT & ~LED;
}

void main(void) {
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    P1SEL = 0;
    P2SEL = 0;
    P1DIR = P1DIR | LED; // Set P1.0 to output

    while (1) { //Do this forever
        if (!((P1IN & PB))) { // If button is pushed (P1.4 is low)
            LED_on(1); // ... turn on LED
        } else {
            LED_on(0); // ... else turn it off.
        }
    }
}
```

Define constants to make code easier to read (PB signifies PushButton)

Function is defined before “main” routine.

All input arguments must be defined.

Turn on LED if argument (ledval) is 1.

Turn off LED if argument (ledval) is 0.

Code is same as before, but function calls are used instead.
State Machines

A method to define behavior of time dependent systems
On to the lab!

• To do lab:
  – Go to my web page (in Firefox, type “Erik Cheever” into address bar).
  – Go to E15 Labs (http://www.swarthmore.edu/NatSci/echeeve1/Class/e15/e15_labs.html)
  – Go to lab 2
  – Start with tutorial, then do lab

• Lab write-up:
  – Due next time you come to lab.